Efficient Data Sharing across Trust Domains

Natacha Crooks
UC Berkeley
ncrooks@berkeley.edu

Cross-Trust-Domain Processing. Data is now a commodity. We know how to compute and store it efficiently and reliably at scale. We have, however, paid less attention to the notion of *trust*. Yet, data owners today are no longer the entities storing or processing their data (medical records are stored on the cloud, data is shared across banks, etc.). In fact, distributed systems today consist of many different parties, whether it is cloud providers, jurisdictions, organisations or humans. *Modern data processing and storage always straddles trust domains.*

From Technical to Socio-technical. Enforcing and maintaining distributed trust is challenging [4]. Trust is as much a social and legal concept as it is a technical one. Consensus protocols [2, 8, 10] or privacy-preserving algorithms [3, 5, 6, 11] offer *technical solutions* to enforcing trust for data availability/data confidentiality. SLAs in modern cloud storage systems provide *econonomic incentives* towards building trust. Likewise, *regulatory compliance* with laws such as GDPR also help bootstrap trust. One should always consider all three: cloud storage should, for instance, be evaluated using throughput-per-dollar as a metric. Similarly, one should explicitly ask: is the dollar cost necessary to enforce confidentiality higher than the potential financial and legal pitfalls?

In Log We Trust. We will focus here on technical solutions for guaranteeing data integrity. There exists a mismatch between the current technology stack and what is necessary to offer efficient and expressive data sharing functionality. These systems, which include permissioned blockchains, rely on Byzantine Fault Tolerance (BFT) protocols [2, 10]. They offer the appealing abstraction of a totally ordered log that is distributed across a group of disjoint trust domains. Together, protocol participants guarantee that the set of operations submitted to the log cannot be altered even as some parties may behave maliciously. This log can then easily be consumed to materialise the resulting shared state.

Limitations. We submit that implementing a physical, decentralized log suffers from two significant limitations: a scalability bottleneck, and a mismatched API. Concerning scalability, all replicas in the log must order every operation, and execute each such operation sequentially. While some systems introduce sharding to alleviate this issue [1, 7], each operation within a shard remains totally ordered and cross-shard transactions remain limited and costly. Concerning the API mismatch, most applications, BI tools and web development frameworks rely on the ability to execute interactive, serializable, transactions and SQL queries on application state. Current decentralized systems instead restrict the API to a key-value store API with stored-procedures, and traditionally offer no guarantees on read operations.

Towards a Decentralized Database. These two limitations are fundamental with today's architecture. Decentralized systems are designed with a rock hard separation between a log-based *ordering layer* and a state-based *materialization* layer. The ordering layer focuses on trust: how can one consistently order a sequence of bytes. The materialization layer instead focuses on semantics: how can one execute those bytes to generate the corresponding state. In practice, applications care about the materialized state, not the log. The log is simply the mechanism through which one can consistently materialize the state. Yet, without semantics, the ordering layer cannot do away with totally ordering all operations. In fact, one need only order *conflicting* operations, operations whose relative order affects the final state. This is the very premise that allows database systems to scale.

Our Wishlist. Most permissioned chains are, we argue, actually trying to build a database. Merging the ordering and the materialization layers together into a single entity that is *explicitly* a database is key to scalability. Such a system should support transactions and query processing at performance comparable to that of a traditional database. Achieving these properties when both replicas and clients may misbehave is challenging: 1) what do isolation guarantees mean in this setup? 2) how can clients efficiently verify the result of a complex query? Our initial results are promising: Basil [9], our BFT transactional key-value store prototype, achieves up to 5x better performance than prior work. Basil introduces the notion of *Byzantine Isolation* and *Byzantine Independence*, two key notions for characterising the behaviour of a decentralized database. Basil's merged architecture allows it to commit transactions in a single round-trip almost always. Moreover, conflicting transactions do not need to be ordered: under low contention, throughput scales linearly with the number of shards. We are now actively working on transforming Basil into a fully SQL compliant database by adding efficient query processing to the system.

REFERENCES

[1] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis. Chainspace: A sharded smart contracts platform, 2017.

[2] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[3] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi. Obladi: Oblivious serializable transactions in the cloud. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 727–743, Carlsbad, CA, Oct. 2018. USENIX Association.

[4] E. Dauterman, V. Fang, N. Crooks, and R. A. Popa. Reflections on trusting distributed trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, page 38–45, New York, NY, USA, 2022. Association for Computing Machinery.

[5] E. Dauterman, V. Fang, I. Demertzis, N. Crooks, and R. A. Popa. Snoopy: Surpassing the scalability bottleneck of oblivious storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 655–671, New York, NY, USA, 2021. Association for Computing Machinery.

[6] S. Eskandarian and M. Zaharia. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, oct 2019.

[7] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. Cryptology ePrint Archive, Paper 2017/406, 2017. `https://eprint.iacr.org/2017/406`.

[8] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4), jan 2010.

[9] F. Suri-Payer, M. Burke, Z. Wang, Y. Zhang, L. Alvisi, and N. Crooks. Basil: Breaking up bft with acid (transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[10] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 347–356, New York, NY, USA, 2019. Association for Computing Machinery.

[11] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, Mar. 2017. USENIX Association.