Indicus: "Unchaining" Byzantine serializable databases

PI: Lorenzo Alvisi

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

The context The ability to share data online offers exciting opportunities. In the medical domain, for instance, cloud-based solutions for managing health record offer doctors increased fault-tolerance at lower cost (they are adopted by 83% of doctors in NYC), and offer patient an easier path to share their medical history with their entire treatment teams, even when on the road. Opportunities abound in other areas too. 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 privacy and trust. Even when medical records are encrypted or anonymized [23, 24, 26], cloud providers or dishonest applications may be able to acquire sensitive information: for example, tracking the charts accessed by an oncologist can reveal not only whether a patient has cancer, but also, depending on the frequency of accesses (e.g., the frequency of chemotherapy appointments), indicate the cancers type and severity. Banking institutions must currently place their trust in the centralized SWIFT's network to issue payment orders. Sometimes there is 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 both these possibilities and these challenges, much research over the last decade has focused on enabling shared computation between mutually distrustful parties, most recently in the context of byzantine fault tolerance (BFT), and blockchains. Systems based on these technologies 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 be misbehaving. Each participant executes transactions that may touch one to multiple objects in the log

The opportunity This proposal argues that there exists a fundamental mismatch between the abstraction of a totally ordered log and the reality of much large-scale distributed processing. Many large-scale distributed systems consists primarily of unordered and unrelated operations. For example, Alice's surgery need not be ordered with respect to Bob's X-rays; likewise, 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 [31]. Callinicos [25] and Omniledger [13], 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.

What we propose Existing research, in essence, is attempting to add database-like transactions and sharding functionality to a Byzantine-fault tolerant totally ordered log. We propose instead to flip the problem on its head by adding BFT to an efficiently shardable replicated database. Specifically, *we propose to leverage the decades of transactional research on serializability to efficiently provide the illusion of a centralized shared log, rather than the non-scalable reality of a totally ordered log.*

We identify five requirements to the success of our agenda:

1. The database should be serializable: participants should be able to run arbitrary ACID transactions on top of sharded data. Given the sensitive nature of banking or supply chain operations, we expect that programmers will favor strong guarantees over weaker, more error-prone consistency models.
2. The database should have moderate latency: international banking, supply chains, or deed acquisition systems span multiple continent. An effective BFT database should aim to reduce as much as possible the number of round-trips per transaction.
3. The database should support up to f Byzantine replica failures out of n, any number of Byzantine clients, and any number of shards.
4. The database should support true partial replication: the overheads per transaction should depend on the number of objects accessed by that transaction, not the total number of shards. Further, it should aim to decouple trust from replication: acknowledging that a transaction committed should not require storing a full copy of the data.
5. The database should provably bound the impact of Byzantine clients and/or Byzantine replicas. Specifically, Byzantine clients and replicas should not be able to force honest transactions to abort.

Plan of Work To address these challenges, we propose to develop *Indicus* [1]. Indicus is a BFT, shardable database that takes as its starting point, TAPIR [31], a replicated serializable database that co-designs the ordering and replication logic of distributed transactions. In TAPIR, a first layer replicates operations into an unordered set for fault-tolerance, while a second layer orders transactions according to the shards they access. This mechanism allows TAPIR to commit general purpose transactions in a single round-trip in the common case, and in up to two round-trip otherwise, across an arbitrary number of shards.

TAPIRs good performance in geo-replicated settings is driven by the inherent optimism of its protocol: TAPIR optimistically executes operations out of order, and allows transactions to read from their local replicas directly. TAPIR then trusts clients to drive the optimistic concurrency control protocol that it executes to guarantee serializability.

A key research question in Indicus will be to explore the degree by which this optimistic stance can be maintained in the presence of Byzantine clients. To answer, we will need to address challenges in a variety of dimensions:

Integrity TAPIR, like most other geo-replicated systems [8], serves reads from individual local replicas to minimize latency; transactions that read stale values are aborted by the commit protocol. In Indicus, however, a Byzantine replica may supply the client not simply a stale value, but an incorrect one, causing the client to crash—for example, by returning a made-up string when the client is expecting an integer. Dealing with this threat requires us to aim for stronger guarantees than even serializability can provide. We will aim instead for *virtual-world consistency* [12]. Virtual world consistency requires that committed transactions be serializable, but add guarantees also for aborted transactions: in particular, it requires *no* transaction read from an inconsistent global state. Enforcing virtual world consistency should not, however, come at the cost of a prohibitive increase in latency for read operations; thus, running a full instance of BFT for both read and write operations is a non-starter. We will explore an alternative strategy: guaranteeing integrity for reads only, but not ordering. This requirement is sufficient to ensure that transactions indeed read value that existed for the objects they request, without insisting that this value be the latest one.

[1] The Malayan tapir (*Tapirus Indicus*), also called the Asian tapir, is the largest of the five species of tapirs and the only one native to Asia

Liveness Executing a transaction in a distributed database requires a mechanism by which to atomically commit transactions across multiple shards. This requires shards to vote on the outcome of conflicting transactions: "yes" if no conflicting transaction is ongoing, "no" otherwise. A key requirement for Indicus is that Byzantine replicas should not be able to force transactions that would have otherwise committed to abort. This is a hard problem, which may require a combination therapy. As a first step, we plan to require replicas that vote no to produce *aborts certificates*, which must demonstrate the existence of a conflicting transaction that already committed. Of course, we need to prevent Byzantine replicas from recycling old abort certificates as a way to spuriously abort transactions. It may be possible to leverage properties of concurrency control mechanism to prevent such replay attacks: OCC, for example, has a notion of forward progress in which old transactions cannot abort newer transactions. Even with abort certificates, however, Byzantine clients could hurt the progress of non-Byzantine clients by carefully scheduling their own transactions to conflict with honest clients requests. To sidestep this issue, we will also investigate concurrency control mechanisms that, unlike OCC, do not allow for aborts—deterministic databases [28] have such a property. Bounding the ability of Byzantine clients to impede the progress of correct clients may ultimately involve the ability to federate multiple concurrency controls, and switch among them dynamically: thankfully, our group has significant experience with such federations [27, 29]

Separating trust from replication Finally, we aim to design a system where replicas can contribute to establishing trust in about the state of the database without being required to maintain a full replica of it. Our goal is to extend to serializable transactions the ethos of replica witnesses [17, 14].

Preliminary results We have developed BFT version of the TAPIR protocol that tolerates f Byzantine replicas (but still depends on correct clients). We are actively working on addressing the challenges above; to address them, we hope to be able to build on our groups expertise in Byzantine-fault-tolerance [1, 2, 3, 4, 5, 6, 7, 14, 15, 16, 18, 19, 20, 21, 22, 30] and privacy-preserving systems [9, 10, 11].

Prior awards In 2017, we received support to investigate Dolorean, a new transactional geo-replicated database that addresses the performance challenges of geo-distributed concurrency through speculation. The work is proceeding. We have since developed and proved correct Dolorean's core algorithm: it allows replicas to fork multiple concurrent branches of execution, each corresponding to a different ordering of transactions, and, at commit time, prune branches that do not correspond to the decided-upon serialization order. In our proposal, we identified three main challenges: (1) How can we keep the cost of speculation under control? (2) How can the client API be modified to support transactions executing on multiple branches? (3) How can branches be efficiently represented? We detail below our progress:

Controlling the cost of speculation Since transactions within a transaction manager are likely to follow the same ordering, we do not speculate on those: instead, we assign a timestamp to these transactions and order them according to their timestamp. Across transaction managers, however, we do speculate on ordering. We do so only at the transaction level, not the operation level: we track dependencies and greedily prune branches that are not serializable. We leverage small programmer hints to determine which transactions are abortable: we do not speculate on non-abortable transactions.

API We leverage the support that new languages are adding for asynchronous programming/calbacks and lambdas, and ask users to rephrase their transactions as a series of callbacks. We use the new scoping functionality available in modern C++ to prevent users from modifying shared state. Instead, we ask users to encompass all state in a TransactionState object that exists for all possible branches of execution. Users never see or observe that a transaction is executing on multiple branches.

Branching We efficiently represent branches by expressing them in terms of the partial order that they encode. Branches consist of pairs of ordering decisions (e.g. {T1/T2, T4/T5}). We have built the

client and transaction manager for Dolorean, as well as support for sharding. We are in the process of implementing the critical step of supporting branching.

References

[1] A. S. Aiyer, L. Alvisi, and R. A. Bazzi. Byzantine and multi-writer k-quorums. In *DISC '06*, volume 4167, pages 48–62, Stockholm, Sweden, September 2006. Springer-Verlag.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. Oct. 2005.

[3] L. Alvisi, D. Malkhi, E. Pierce, and M. K. Reiter. Fault detection for Byzantine quorum systems. *IEEE Trans. Parallel Distrib. Syst.*, 12(9):996–1007, 2001.

[4] L. Alvisi, E. T. Pierce, D. Malkhi, M. K. Reiter, and R. N. Wright. Dynamic Byzantine quorum systems. In *DSN*, pages 283–292, 2000.

[5] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, New York, NY, USA, 2009. ACM.

[6] A. Clement, H. C. Li, J. Napper, J.-P. Martin, L. Alvisi, and M. Dahlin. Bar primer. In *DSN*, pages 287–296. IEEE Computer Society, 2008.

[7] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin. Making Byzantine fault tolerant systems tolerate Byzantine faults. Apr. 2009.

[8] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '12, pages 251–264.

[9] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi. Obladi: Oblivious serializable transactions in the cloud. In *OSDI*, pages 727–743. USENIX Association, 2018.

[10] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and Private Media Consumption with Popcorn. 2016.

[11] T. Gupta, H. Fingler, L. Alvisi, and M. Walfish. Pretzel: Email encryption and provider-supplied functions are compatible. In *SIGCOMM*, pages 169–182. ACM, 2017.

[12] D. Imbs and M. Raynal. Virtual world consistency: A condition for stm systems (with a versatile protocol with invisible read operations). *Theor. Comput. Sci.*, 444:113–127, July 2012.

[13] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE Symposium on Security and Privacy*, pages 583–598. IEEE, 2018.

[14] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39, Jan. 2010.

[15] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Flightpath: Obedience vs. choice in cooperative services. In R. Draves and R. van Renesse, editors, *OSDI*, pages 355–368. USENIX Association, 2008.

[16] H. C. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. Bar Gossip. In *Proc. 7th OSDI*, 2006.

[17] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, and L. Shrira. Replication in the Harp file system. pages 226–238, 1991.

[18] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst.*, 29(4):12:1–12:38, Dec. 2011.

[19] J.-P. Martin and L. Alvisi. A framework for dynamic Byzantine storage. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 04), DCC Symposium*, Florence, Italy, June 2004.

[20] J.-P. Martin and L. Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, July 2006.

[21] J.-P. Martin, L. Alvisi, and M. Dahlin. Minimal Byzantine storage. In *16th International Conference on Distributed Computing, DISC 2002*, pages 311–325, Oct. 2002.

[22] J.-P. Martin, L. Alvisi, and M. Dahlin. Small Byzantine quorum systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 02), DCC Symposium*, pages 374–383, June 2002.

[23] A. Narayanan and V. Shmatikov. Robust De-anonymization of Large Sparse Datasets. 2008.

[24] A. Narayanan and V. Shmatikov. Myths and fallacies of "personally identifiable information". *Commun. ACM*, 53(6):24–26, June 2010.

[25] R. Padilha, E. Fynn, R. Soulé, and F. Pedone. Callinicos: Robust transactional storage for distributed data structures. In *USENIX Annual Technical Conference*, pages 223–235. USENIX Association, 2016.

[26] R. Singel. Netflix spilled your *Brokeback Mountain* secret, lawsuit claims. *Wired*, Dec. 2009. `http://www.wired.com/images_blogs/threatlevel/2009/12/doe-v-netflix.pdf`.

[27] C. Su, N. Crooks, C. Ding, L. Alvisi, and C. Xie. Bringing modular concurrency control to the next level. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 283–297, 2017.

[28] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 1–12, 2012.

[29] C. Xie, C. Su, C. Littley, L. Alvisi, M. Kapritsos, and Y. Wang. High-performance acid via modular concurrency control. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 279–294, 2015.

[30] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. Oct. 2003.

[31] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 263–278, 2015.