2017 13th European Dependable Computing Conference

# A Prototype Evaluation of a Tamper-resistant High Performance Blockchain-based Transaction Log for a Distributed Database

Leonardo Aniello, Roberto Baldoni, Edoardo Gaetani and Federico Lombardi
*CIS Sapienza - University of Rome*
{*aniello;baldoni;gaetani;lombardi*}*@dis.uniroma1.it*

Andrea Margheri and Vladimiro Sassone
*University of Southampton*
{*a.margheri;vsassone*}*@soton.ac.uk*

***Abstract*—As data is having an increasingly relevant role in different business fields, ensuring integrity has become fundamental. Modern databases rely on transaction history written on *redo logs* to allow for data restore. However, if redo logs are (maliciously) forged, data can actually be lost or altered. Due its strong data integrity guarantees, blockchain technology can be employed to ensure log integrity, but its current performance limitations hinder actual exploitations.**

**In previous work, we proposed a layered blockchain-based architecture for distributed (federated) database redo logs: a *fast* first layer blockchain, anchored to a *secure* second layer blockchain, based on *proof-of-work* to achieve strong integrity. Here, we present an implementation and an experimental evaluation of a prototype of that architecture, which employs a total consensus algorithm on the first layer blockchain. Finally, to improve availability and scalability, we refine our solution by investigating, respectively, a Byzantine Fault Tolerant consensus and a Distributed Hash Table solution to shard the first layer blockchain ledger among available nodes.**

***Keywords*-Blockchain, Cloud Federation, BFT, DHT**

## I. Introduction

As a critical component of many systems, data have an appealing target for cyber-attacks. Tampering with data can go undetected and drive malicious operations, e.g. data alteration and deletion. Most of all, differently from availability loss, data integrity can be hardly restored once lost. Modern database systems use logging mechanisms to track data changes, e.g. Write-Ahead Logging of PostgreSQL[1] and Redo Log of Oracle[2]. However, if such logging files are forged, recognising an attack or a failure is awkward as data integrity relies too intimately on the system itself. Typically, *Remote Data Auditing* mitigations are employed, but they come with high costs and rely on trusted third parties [10].

In recent years blockchain came to prominence through the Bitcoin system [7], and have emerged since as a powerful technology to provide strong data integrity guarantees in trust-less networks. Blockchain ensures integrity by means of *proof-of-work* (PoW), a consensus schema based on solving a crypto-puzzles that is considered computationally inviolable [5]. Blockchain also offers so-called *smart contracts*, immutable programs deployed and executed on top of a blockchain system, e.g. Ethereum [11], to realise decentralised applications where no involved party is in control neither of the code nor of the data.

Regardless of strong integrity guarantees, exploiting blockchain to strengthen (distributed) database systems implies coping with overwhelming performance penalties: viz., high latency and low throughput. Indeed, preliminary blockchain-based database solutions rationalise blockchain features: BigchainDB [1] replaces PoW (and its security) with a lightweight protocol; RSCoin [3] (re)introduces certain degree of centralisation in the system.

In [4] we proposed a blockchain-based architecture for distributed (federated) database transaction (or redo) logs, which permits achieving both high performance and adequate integrity guarantees. Specifically, we defined a layered architecture that makes use of a *first layer blockchain* assuring low latency and high throughput, anchored to a *second layer blockchain* featuring PoW to ensure data integrity.

In this paper, we present a prototype implementation of that architecture, which features a total consensus algorithm on the first layer blockchain, and its experimental evaluation. Then, to cope with faults and Denial-of-Service (DoS) attacks, we investigate a *Byzantine Fault Tolerant* (BFT) consensus. Finally, to improve scalability, we investigate a *Distributed Hash Table* (DHT) as redo log of the first layer.

As a case study, we address the distributed database underlying FaaS [8], a recent Cloud Federation solution put forward by the EU H2020 SUNFISH project. FaaS offers a democratic federation of Clouds which crucially relies on a distributed database to ensure that no federation member can tamper with federated data. Clearly, application requirements demand also for adequate performance. Deploying a fast and secure blockchain-based system is paramount to achieve both performance and democracy requirements of FaaS.

*Paper structure.* §II introduces the architecture; §III presents and evaluates the prototype. §IV illustrates the proposed extensions. §V concludes and discusses future work.

## II. Background

In [4] we posed some important open research questions on the issues of employing blockchain "as-is" in database

[1] postgresql.org/docs/9.1/static/wal-intro.html
[2] docs.oracle.com/cd/B19306_01/server.102/b14231/onlineredo.htm

978-1-5386-0602-5/17 $31.00 © 2017 IEEE
DOI 10.1109/EDCC.2017.31

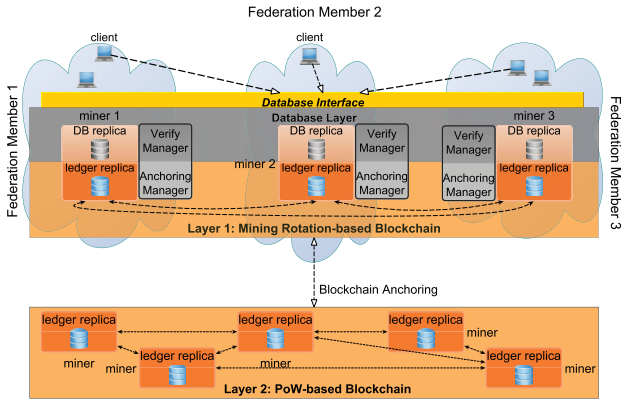



Figure 1. 2LBC architecture for a FaaS federation

settings, and pointed out practical research directions towards its effective employment: improving performance while ensuring data integrity. To this aim, we proposed a *2-layer blockchain* architecture (2LBC), enabling a replicated, trustworthy redo log for a distributed (federated) database.

2LBC consists of a permissioned blockchain on the first layer and a public permissionless blockchain on the second layer. The first layer features a fast consensus algorithm based on a *leader rotation* approach: the time is split into rounds each of which has a designed leader (i.e., a miner) chosen according to a deterministic fair policy. The leader defines the ordering of *transaction*s (txns) containing the operations tracked in the redo log (aka ledger). Transactions are stored, once signed with asymmetric cryptography by all members, in each ledger replica.

Figure 1 shows the 2LBC architecture in the context of a FaaS federation. Specifically, each federation member contributes to 2LBC with one miner –distinguished network nodes building the integrity of blockchain– keeping both database and ledger replicas. The second layer features PoW and is used to guarantee the integrity of the first layer. Periodically, the hash of the first layer blockchain is here stored via the *Anchoring Manager*. PoW ensures the immutability of such hashes. These hashes, so, act as forensic evidences of the first layer: if a malicious miner maliciously tries to alter the redo log, the first layer blockchain will result forged as the hash stored in the second layer will be different.

## III. Prototype Implementation and Evaluation

We developed the first layer blockchain in Java. Miners join a p2p network and communicate through txns composed by a *payload* (i.e., client *operation*, a *sequence number* and a *timestamp*) and a *certificate* with miners' signatures. Miners trade messages either directly through JavaRMI or in broadcast through JGroup (http://jgroups.org/). Clients can operate on the database by issuing to all miners, through JavaRMI, two kind of *operations*: (i) `set(k,v)` to assign a value `v` to a key `k` (it returns to the client a boolean confirmation); (ii) `get(k)` to obtain the value stored to key `k` by returning the txn related to the last `set` toward `k`.

*Consensus.* In case of `set(k,v)`, miners have to achieve consensus to give confirmation to the client. The consensus is based on a *three-phase commit* protocol [9]. Practically, a client broadcasts a $\texttt{set}_i$ operation to all miners which verify its correctness through a *Verify Manager* and add it to a queue. Once the current leader (who is in charge of ordering txns) proposes $\texttt{txn}_i$ related to $\texttt{set}_i$, other miners remove it from their queue and broadcast their signatures to build the corresponding certificate. When the certificate contains all the signatures, all miners *commit* the txn in their ledger replica and trigger the update in the database replica.

In case of `get(k)`, every miner answers to the client with the last txn related to a `set(k,v)`. The client can obtain the value `v` from the most recent txn among those received, and verify its correctness via the miners' signatures.

*Anchoring.* When the round time of a leader terminates, it triggers a *leader change*, which amounts to store a special txn, and the anchoring procedure. Specifically, the abdicating leader computes the SHA-1 hash of the first layer blockchain and sends a *witness txn* to a dedicated *witness smart-contract* on the second layer implemented with Ethereum.

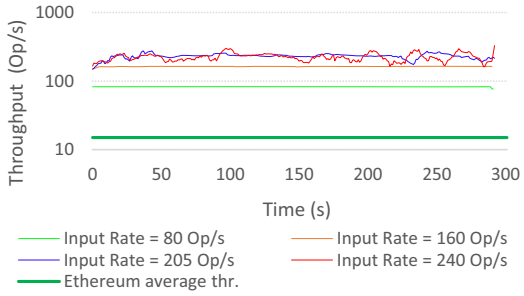
*Evaluation.* We evaluated our prototype on a private cluster composed by $N = 6$ Ubuntu 16.04 Virtual Machines (VMs) each one running a miner process, deployed on 4 blade servers IBM HS22, equipped with 2 Quad-Core Intel Xeon X5560 2.28 GHz CPUs and 24 GB RAM.

Network latency between miners is simulated according to a Poisson distribution in the range 5-20 ms. Operations `set` and `get` on random keys are injected with different rates via 2 multi-thread clients deployed on 2 further VMs. To evaluate 2LBC performances, we measured throughput and response time of the operations over time.

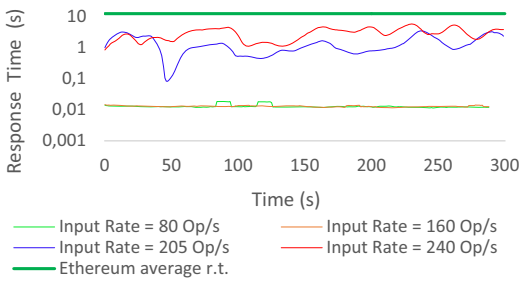
For `set` operations, Figure 2(a) (resp., 2(b)) reports throughput (resp., latency) results that, as expected, are much higher (resp., lower) than Ethereum. Above 240 Op/s, they become unstable for resource saturation due to exchanged messages: the total consensus algorithm requires miners to trade all their signatures for each operation. Additionally, leader rotation introduces an overhead on the enqueued operations during leader changes; we refer to *transient state* as the transitory period where latency exceeds 50% of the average latency. As shown in Figure 3, increasing the round time makes the transient state shorter. On the contrary, frequent leader changes increase such overhead.

For `get` operations, Figures 2(c) and 2(d) report throughput and latency results. As there is no transient state for `get` (i.e., miners just send their last local value), the results are comparable with and without leader rotation.

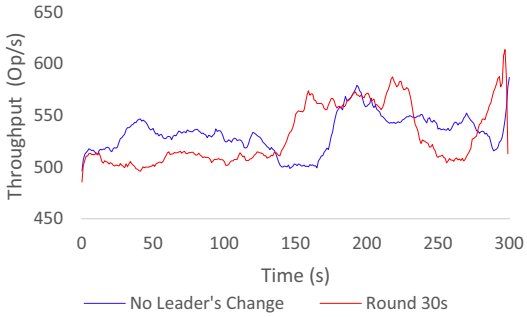
*Security Analysis.* We modelled three possible kinds of attack: *A*1 *tampering*, if a miner tries to modify the log; *A*2 *forgering*, if a miner tries to send a fake txn, i.e. by



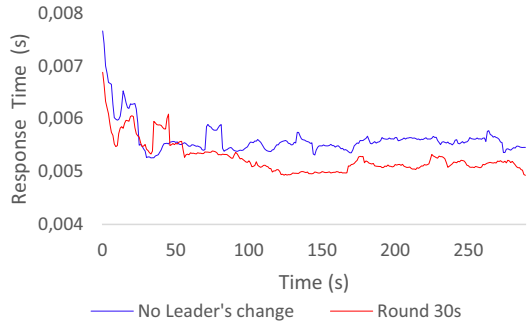

(a) Throughput of `set` operations for different input rates




(b) Response Time of `set` operations for different input rates




(c) Throughput for `get` operations while injecting ≈ 500 op/s without changing leader and with 30s leader round time




(d) Response Time for `get` operations while injecting ≈ 500 op/s without changing leader and with 30s leader round time

Figure 2. Throughput and Response Time evaluation of 2LBC for `set` and `get` operations

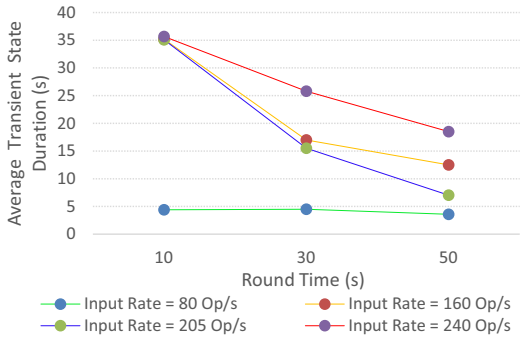



Figure 3. Analysis of the relation between round time (x axis) and transient state duration (y axis) for different input rates

forging a client operation; *A*3 *DoS*, either a miner does not sign a txn or a leader does not broadcast a txn.

In our approach a valid txn must contain a certificate with all miners' signatures. This avoids miners from maliciously update values without informing other members (*A*1), and from creating fake txns (*A*2) (unless it can obtain the private keys of all other miners). Moreover, hashes stored in the second layer blockchain are immutable, hence, though the attacker is able to compromise the first layer by stealing all miners' keys, *A*2 might be detected by comparing the hash of the first layer with the hash evidence stored on the second layer; to compromise also hashes in the second layer the attacker should obtain a significant, unfeasible computational power [5]. Furthermore, the leader cannot forward fake txns (*A*2), because miners sign only txns related to operations that they enqueued. Specifically, miners verify the correctness of txn fields being sure that are not forged.

Our approach is instead vulnerable to DoS attacks (*A*3): a single malicious miner can block `set` operations not sending its signature or a malicious leader (during its leading periods) can avoid to forward a txn. This cannot happen with `get` operations, as miners return their response independently. Indeed, if there is at least one honest miner, a client can obtain the value by its corresponding txn and verify the signatures in the certificate to prove the authenticity.

## IV. Improving Availability and Scalability

The main limitations of the current 2LBC prototype are related to *availability* issues (as explained in last section) and *scalability*. Indeed, overall system performance does not scale adding new nodes, as the used total consensus algorithm has lower performance with additional nodes. In this section we propose the solutions to cope with them.

*Availability.* To mitigate such limitations, we propose a Byzantine Fault Tolerant solution based on PBFT [2]. To tolerate up to $f$ Byzantine miners it requires $3f+1$ miners to provide both *safety* and *liveness*. This leads to higher availability level as honest miners need to wait just $f+1$ valid signatures to commit a txn. We can so tolerate up to $f$ *silent*

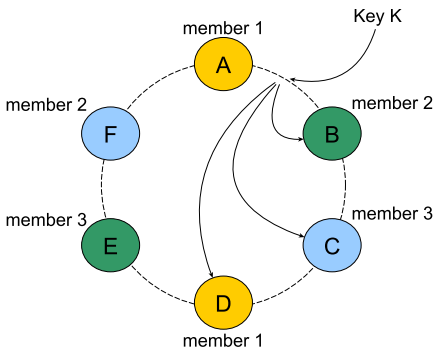



Figure 4. Example of the DHT-based ledger solution to achieve total replication in the FaaS scenario. The federation is composed by $M = 3$ members (each one marked with a proper color), each one exposing $N = 2$ miners (identified by a proper letter). Miners are disposed on the ring so as each member has all keys of the database sharded between its miners. The single miners keep only a subset proportional to the replication factor. In the example, the replication factor is $M = 3$ and each miner maintains only $1/N = 1/2$ of database keys.

miners at the cost of weaker integrity guarantees; indeed data corruption is possible with just $f + 1$ compromised miners, rather than all $N$ when total consensus is used. Anyway, the integrity of txns already witnessed in the second layer is still ensured thanks to the PoW.

*Scalability.* To improve scalability, we propose a *data sharding* solution for permissioned (federated) blockchain, similar to [6] for permissionless blockchain. Specifically, we introduce a DHT-based ledger in which each miner, on the base of a *keyspace partitioning*, only handles txns for specific subsets of keys. This approach permits tuning txn loads on miners and, consequently, makes the system more scalable. Furthermore, each key range has a configurable replication factor to enable fault tolerance. In our solution, contrarily to common DHT implementations, the miners involved in a `set` operation must achieve consensus before writing the operation in the local replica (hence in the local keyspace) of the redo log. This permits achieving strong consistency, hence avoiding consistency issues which mar well known DHT-based NoSQL databases, such as DynamoDB (aws.amazon.com/dynamodb/) and Cassandra (wiki.apache.org/cassandra/). Clearly, this comes at the cost of performance penalties, whose quantification depends on the technology and is beyond our scope here.

The solution to the FaaS scenario can e.g. change as follow: let $M$ be the number of federation members, each providing $N$ (rather than a single) miners. The DHT ring includes $M \cdot N$ nodes, the replication factor is set to $M$, and miners can be placed over the ring so that the miners of each federation member collectively manage all the keys. Figure 4 shows an example of a FaaS federation composed by $M = 3$ members, each exposing $N = 2$ miners.

## V. Conclusion

We evaluated a prototype of 2LBC, a 2-layered architecture for a blockchain-based database able to provide both high performance and strong data integrity guarantees in a totally decentralised environment. We proposed also a solution to cope with current availability issues and a DHT-based ledger to shard data and make the system scalable.

In the future, we will continue developing and evaluating our prototype by implementing the PBFT and DHT solutions. To compare outcomes, we plan to develop a metric for data integrity based on the effort required for an attacker to tamper with data without being noticed. This enables evaluating varying tradeoffs between system availability, integrity and performance. Finally, we aim to evaluate the architecture according to a varying number of miners and members exposing different pools of miners.

## Acknowledgment

This work has been supported by the EU H2020 SUNFISH project, grant N.644666.

## References

[1] BigchainDB GmbH. BigchainDB: A Scalable Blockchain Database, 2016. https://www.bigchaindb.com/whitepaper/.

[2] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[3] G. Danezis and S. Meiklejohn. Centrally Banked Cryptocurrencies. In *NDSS*. The Internet Society, 2016.

[4] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone. Blockchain-based database to ensure data integrity in cloud computing environments. In *ITA-SEC*, volume 1816. CEUR-WS.org, 2017.

[5] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, volume 9057 of *LNCS*, pages 281–310. Springer, 2015.

[6] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *CCS*, pages 17–30. ACM, 2016.

[7] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[8] F. P. Schiavo, V. Sassone, L. Nicoletti, and A. Margheri (Eds.). Faas: Federation-as-a-service. *CoRR*, abs/1612.03937, 2016.

[9] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, (3):219–228, 1983.

[10] M. Sookhak, A. Gani, H. Talebian, A. Akhunzada, S. U. Khan, R. Buyya, and A. Y. Zomaya. Remote Data Auditing in Cloud Computing Environments: A Survey, Taxonomy, and Open Issues. *ACM Comput. Surv.*, 47(4):65:1–65:34, 2015.

[11] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.