Blockchain Data Replication

Roberto De Prisco[1][a], Sergiy Shevchenko[1,2][b] and Pompeo Faruolo[2]
[1] *Computer Science Department, University of Salerno, Salerno, Italy*
[2] *eTuitus s.r.l., Fisciano (SA), Italy*

www.etuitus.it

Keywords: Data Replication, Blockchain, Fault-Tolerance, Self-Sovereign Identity, KERI.

Abstract: We consider applications that write data over a blockchain. Such applications are based on the implicit assumption that the blockchain will work forever. Although blockchains are very fault-tolerant by construction, the event that a blockchain becomes totally unusable or disappears is not impossible. We consider the problem of making the applications fault tolerant against total blockchain failures by replicating the needed data over several blockchains. As specific use cases, we consider the implementation of Self-Sovereign identities and the implementation of a Key Event Receipt Infrastructure using data replicated over several blockchains.

1 INTRODUCTION

Blockchain technology has gained considerable importance in the last decades. Although the basic tools and the technology itself have been studied in cryptography and distributed communities well before, the blockchain revolution has been ignited by the success of Bitcoin.

Following Bitcoin, literally hundreds of other cryptocurrencies are available today, each one implementing a blockchain. Many of them leveraged the service to include more sophisticated tools, such as fungible and non-fungible tokens and smart contracts. Blockchain solutions have been proposed for managing healthcare data, to automate business transactions, manage logistics and shipping. The emerging concept of self-sovereign identity exploits a blockchain to implement identity verification.

The prominent property of a blockchain is that of being immutable. Moreover, a blockchain has strong resilience against failures. There are many approaches to the implementation of a blockchain, but the most common one is that of a fully decentralized distributed ledger. Such systems are very resilient to failures. As long as a sufficient number of the participating nodes are properly working, typically a majority, the system functions correctly. In real life, failures happen, but they can also get fixed and if there is enough redundancy, keeping active a sufficient number of nodes is doable. There is an implicit assumption in the use of a blockchain regarding this aspect: the failures never disrupt irreparably the system.

But is this always the case?

Technically any blockchain can disappear: if there are too many failures it is possible that the entire history of transactions gets lost. As an extreme case consider the one in which all nodes of the network fail at the same time. Although such an event can be considered extremely rare, it really depends on how robust is the underlying distributed system. Current estimates of the Bitcoin network count about 15,000 nodes spread all over the globe. Such a huge number makes unlikely the event that the network collapses. However not all the existing blockchains are so resilient; for example, Freecoin (Foundation, 2023) and Devcoin (Devcoin, 2023) have basically disappeared since the value of the corresponding cryptocurrency is near zero and there have been no transactions for a very long time.

In this paper we pose the following general problem: we consider applications that write overlay data over an underlying blockchain but we take into consideration the possibility that the blockchain might disappear. Thus we want to replicate the data over several blockchains in such a way that the collapse of one (or few) blockchains does not cause the loss of the overlay data and consequently does not block the applications. Ideally, we would exploit a set of n blockchains, and be able to write each piece of the

[a] https://orcid.org/0000-0003-0559-6897
[b] https://orcid.org/0000-0002-0864-2919

De Prisco, R., Shevchenko, S. and Faruolo, P.
Blockchain Data Replication.
DOI: 10.5220/0012121000003555
In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 746-751
ISBN: 978-989-758-666-8; ISSN: 2184-7711

overlay data into all the *n* blockchains. The specific properties that such writes have to satisfy are strongly dependent on the specific application. In simple cases, the applications might need to write only a piece of data, but in more complex cases the overlay data might contain more structured information or it might even be itself a ledger implementing an overlay blockchain.

Replicating an entire blockchain seems like overkill. And indeed it is. However if an overlay application uses a relatively small ledger, or not even a ledger but just a small amount of data, it might make sense to have it replicated over several blockchains in order to be resilient to the failure of one or a few of the blockchains. Moreover, the difficulty related to the replication can vary depending on the specific application. Data replication is already a difficult problem by itself; doing it over blockchains it is clearly a challenge. For some applications, it might be very difficult or even impossible. For example, replicating the entire blockchain of a cryptocurrency is very complex since there are several peculiar difficulties that would be exacerbated by the replication. Notwithstanding, if an application makes a simple use of the underlying blockchain, replicating all the transactions might be feasible and would provide resilience against total failures of (few of) the underlying blockchains. If we abstract from the fact that the write operation is on a blockchain, we would be left with a consensus problem. The difficulty of this problem depends on the specific setting and it can be easy if the underlying distributed system offers strong guarantees or quite difficult, or even impossible, if such guarantees are weak. The specific setting that we are considering, however, is atypical: the “nodes” of the distributed system are the blockchains themselves and there is no direct communication between the blockchains. If, on one hand, we can rely on the fact that a blockchain is reliable, in the sense that as long as it exists it works properly, on the other hand, the difficulties of maintaining consistency over replicas of the data remain. For simple cases, we can exploit a quorum based solution that allows to replicate data in a consistent manner. This, for some applications, is enough. For more complicated scenarios we are left with a consensus problem. In such cases, we need to exploit a consensus algorithm to maintain consistency.

Sovrin is a publicly available infrastructure for self-sovereign identities. It is based on a blockchain on which some relevant data gets written. Transactions in Sovrin do not happen often and although, by definition, the blockchain provides a global ordering on the transactions, the use of the data written in the blockchain is unrelated to the order in which the data appears in the ledger. This makes Sovrin a good use case to consider. We propose an enhancement of Sovrin, to make it resilient to a possible total failure of the underlying blockchain.

As a second use case, we discuss the application of blockchain data replication to KERI, a distributed for the management of key event receipts.

Related Work. To the best of our knowledge this is the first paper that considers the specific problem outlined above. However, blockchain interoperability and different forms of blockchain data replication have been considered, for example, in (Diallo et al., 2019; Li et al., 2020; Bacis et al., 2019; Pillai et al., 2020; Pillai et al., 2022; Westerkamp and Kupper, 2022).

2 PRELIMINARIES

2.1 Blockchain

A blockchain is a just list of blocks of data linked together using hash pointers,.

The very first block, obviously, will not contain any pointer to a previous block, nor any hash value. Such a block, in the blockchain jargon, is called the genesis block. The data structure implemented by a blockchain is tamper resistant because of the properties of the hash function. If an adversary tries to manipulate the data by changing something in a specific block then, to keep the data structure consistent, it needs to change also the hash value of the block that is stored in the subsequent one, and this modifies the subsequent block thus the adversary needs to change also all the subsequent hash values, up to the latest hash value. So, it is sufficient to keep safe the latest hash value in order to prevent malicious changes to the data.

Access to the blockchain can be restricted: we can classify blockchains into 3 categories: private, permissioned and public. A private blockchain is accessible only by some selected users, while public blockchain can be accessed by anyone. Permissioned blockchains are a sort of trade-off between these two: anyone can join upon verification of the identity and users are granted selected permissions, thus can perform only certain activities.

2.2 Consensus Problems

In order to implement a blockchain in a distributed manner it is necessary that nodes of the distributed system that stores the information, agree on the information being stored in the blockchain. Thus there must be some form of *consensus* that the nodes have to achieve.

A consensus mechanism is a method by which the nodes in a blockchain network reach agreement on the state of the ledger. There are several different types of consensus mechanisms, each with its own advantages and disadvantages. Consensus arises in many distributed applications and is a well-studied problem, see for example (Attiya et al., 1994; Ben-Or, 1983; Castro and Liskov, 2002; De Prisco et al., 2000; Dwork et al., 1988; Fischer et al., 1985; Lamport et al., 1982; Pease et al., 1980; Rabin, 1983). The difficulty of the problem depends on the specific setting considered. Each blockchain uses its own specific consensus mechanism to reach an agreement on the distributed ledger.

One of the most widely-used consensus mechanisms is proof of work (PoW). In a PoW-based system, nodes, in blockchain jargon, also known as "miners," compete to solve a complex mathematical puzzle in order to create a new block and earn a reward.

Another popular consensus mechanism is proof of stake (PoS). In a PoS-based system, nodes "stake" their own tokens in order to create new blocks and earn rewards. The probability of creating a new block by this specific node is proportional to the value of tokens staked by the node.

Another consensus mechanism is Delegated proof of stake (DPoS) which is a variation of PoS, where token holders vote for a limited number of nodes to validate transactions.

Consensus can be achieved also exploiting quorums. A quorum system is a set of subsets such that each subsets intersects with each other subsets. The use of quorum systems has been well studied (see for example (Gifford, 1979; Guerraoui and Vukolic, 2010; Malkhi and Reiter, 1998; Malkhi et al., 2001)).

3 REPLICATION OF DATA OVER BLOCKCHAINS

We assume that there is a set $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ consisting of n blockchains that can be utilized by a set of clients $\mathcal{C} = \{c_1, c_2, \ldots, c_z\}$. Each client c_i must interact with each blockchain in $\mathcal{B}$ through a function $write_i(d)$ that requests a write operation of the data d on a blockchain in $\mathcal{B}$; such an operation is peculiar to the blockchain b_i, that is, it follows the protocol of that specific blockchain. For example if b_i is the Bitcoin blockchain then d could be a Bitcoin transaction that spends bitcoins and the $write_i(d)$ operation is a request of inserting the transaction d into some block. Function $write_i(d)$ returns either `success`, if the operation is correctly completed and thus the data d is written into the blockchain b_i or `failure`, otherwise. Since we want to replicate the data over all available blockchains a write operation needs to loop over all blockchains. Although, ideally, this is the operation that we would like to perform, we have to take into account the possibility that some of the writes will not succeed. We assume that all clients involved are honest and the only kind of failures that we worry about are blockchain failures. An unsuccessful write is a normal possible event: it just means that the attempt of writing into the blockchain did not succeed so the client needs to try again. A blockchain failure instead means that the blockchain has disappeared forever and thus is no longer available, together with the data stored in the blockchain. We are abstracting from the specific blockchains B_i, and we assume that the type of data d that needs to be written, can actually be written in each B_i, perhaps making some kind of data conversion to make d compatible with the actual transactions of the specific blockchain.

We assume that the data $D = d_1, d_2, \ldots$, to be written, is an ordered sequence of pieces of data, that is, a ledger (an application ledger). As a boundary case, we might also have $k = 1$, which means that there is only a single piece of data and not a ledger.

We also abstract a $read_i(k)$ operation that allows a client to retrieve the data written in the k^{th} transaction on the blockchain B_i. The index k here refers to the indexing of the data $d_1, d_2, \ldots$, and not to the index of the actual transaction of the blockchain B_i that contains d_k.

By using each blockchain as one storage node, the problem is that of keeping consistent the data replicated over the blockchains. The difficulty of the problem depends on the specific scenario considered. Using a "distributed memory jargon", we can distinguish single-writer or multi-writer cases and single-reader or multi-reader cases. The writers and the readers are the clients of the system and a client can be a writer, a reader or both. What makes a big difference for the difficulty of the problem is the number of writers. If there is only one writer, maintaining consistency is easier. When there are multiple writers it becomes more problematic depending on the specific setting considered. Consistent data replication is achieved by having all copies reach consensus on the replicated data.

3.1 Single-Writer

The case of a single writer is easier to deal with since the data has only one source. Ideally, the single writer would be able to write each d_i over all n blockchains. If we do so, a $write(d)$ would be an iteration over all blockchains B_i, for $i = 1, 2, \ldots, n$, and the execution of $write_i(d)$. And for a read operation, it would suffice to read any (single one) of the available blockchains. However, depending on the characteristics of the single blockchains, a $write_i(d)$ operation might take a long time or even be unsuccessful. Thus the writing of the replicated data over the blockchains can be in different states on different blockchains. Although the writer can keep trying to write in all the (still available) blockchains, we can exploit a quorum system to avoid getting blocked by the impossibility of writing on some blockchains and to improve performance. A simple quorum system is provided by majorities. For simplicity, we consider such a quorum system.

A $write_i(d)$ operation is successful when the single writer has been able to execute successfully at least a majority of the single $write_i(d)$, $i = 1, 2, \ldots, n$.

To improve the performance of subsequent read operations, a writer, although considering successful the $write(d)$ after $n/2 + 1$ successful blockchain-writes, can keep writing d over all the other blockchains, as long as the blockchains exist and until the write of d in the blockchain is successful. However given a specific blockchain B_i, the write operation $write(d_{j_1})$ can be executed only before a write operation $write(d_{j_2})$ for any $j_2 > j_1$. In other words, once $write(j_2)$ is completed the writer cannot execute anymore $write(j_1)$, otherwise, the order of the application ledger gets disrupted. This means that a specific data d_j might not get written in all blockchains; however, it is guaranteed that it gets written on a majority of them. An alternative approach consists in not proceeding on $write_i(d_{j_2})$ for any $j_2 > j_1$ before completing $write_i(d_{j_1})$.

For the $read(k)$ operation a reader needs to query a majority of blockchains in order to be sure to obtain d_k.

Notice that after at most $n/2 + 1$ successful reads, it is guaranteed that the data d will be retrieved. It is also worth to remark that data consistency is guaranteed by the properties of the blockchains and that the only "failure" that we are considering is the disappearance of blockchains. Assuming that no more than $n/2$ blockchains disappear, a reader is guaranteed to get back the data.

3.2 Multi-Writer

In a real setting, there will be multiple writers. And this clearly creates a consistency problem over the sequence of actual data written in the various blockchains. If every writer just tries to write its own data in every blockchains the order in which the data will be written in each of the blockchains might vary, depending on the time each write takes on each of the blockchains. A possible solution is the following: instead of writing only its own data each writer executes a consensus protocol with all the other writers to decide a global order on $d_1, d_2, \ldots$ so that they all try to write the same sequence of pieces of data in all the blockchains. There are some drawbacks. First of all this solution requires that all the clients have to be active in order to execute the consensus protocol; this might not be always the case as in many applications clients come and go and they are active only when they have to perform some actions. Moreover, this solution poses a computational extra load on the clients for the execution of the consensus protocol.

We are not suggesting a specific consensus protocol: the choice of such a protocol needs to take into account the interaction functionalities of the underlying blockchains and also the specific properties of the communication network that connects the clients.

We remark that there can be applications for which the global order might not be strictly necessary and some other partial order might be sufficient.

4 APPLICATIONS

In this section, we discuss two possible use cases.

4.1 Replicating SSI Data

Self-sovereign identity systems are identity systems in which the control is moved from a centralized authority to a self-governed system in which the user has full control on its identity. Indeed, the Self-sovereign model is a model in which the identity and the claims related to the identity are given back to the user without the need of a central authority. Distributed ledgers and blockchain approaches have facilitated the path to such a model.

The Sovrin Hyperledger network is designed to manage digital credentials (the digital equivalent of things like identity cards, passports, driver licenses, but also college degrees, medical certificates, etc.) so that they can be securely stored, managed and shared online. Sovrin gives users digital identities and allows them to manage controllable, trustworthy, and verifiable digital credentials. Such credentials are stored in a digital form in what is called, in the Sovrin jargon, a (digital) wallet. The wallet is managed through an app on a smart device (e.g. smart phone). Using his wallet, the user can prove his identity, or other claims, and safely use private information online. Sovrin exploits a distributed ledger.

Users of Sovrin can get digital identities from all kinds of trusted organizations, like governments, banks, insurers, hospitals, universities, and any organization that is publicly listed in the Sovrin network can verify who you are. The trusted organizations, once they have verified the identity of the user, can issue credentials.

Verifiable credentials are the digital analogs of any physical document that can attest the identity of an individual such as identity cards, passports or driving licenses. In addition to provide the same functionality as their physical counterparts, verifiable credentials can be more tamper-evident and trustworthy by using digital signatures for example. There are three main actors in an ecosystem where verifiable credentials are used: Holder (the entity that possesses credentials), Issuer (the entity that creates the credentials), and Verifier (the entity that checks the credentials).

Sovrin exploits Decentralized Identifiers (DIDs) in order to identify the involved entities. DIDs are identifiers specifically designed for the management of cryptographically-verifiable digital identities that are fully under the control of the owner. Cryptographic operations exploit public key cryptography and an entity that needs to use the service generates a secret (master) key and the corresponding public key that in the Sovrin jargon is called a *Verkey*. A pair of secret and public keys is relative to a specific DID. A DID can identify, for example, an issuer.

The main data that Sovrin needs to write in the blockchain are the following:

- DID and VerKey of the issuers,
- Schema definitions,
- Credential definitions,
- Revocation registries,
- Custom data.

These data constitute the sequence $d_1, d_2, \ldots$ that needs to be written in the blockchain.

Since the Sovrin infrastructure has multiple writers, which are the issuers, in order to replicate the data over several blockchains and maintain consistency over the order we need to use the multi-writer approach outlined in Section 3.2. Actually, Sovrin allows only trusted nodes, called *endorsers*, to write on the blockchain. However, endorsers write on the blockchain on behalf of the issuers. Thus, from a logical point of view, the writers are the issuers.

Moreover, we observe that for this specific application, it might be sufficient to use the multi-writer setting in which each writer uses the simpler approach outlined in Section 3.1. In this case, we might lose the ordering over different blockchains but this might not be as bad as it seems. Indeed, on one hand, this ordering can be messed up when there are multiple transactions executed at the same time; in the Sovrin network transactions are executed very slowly, in the order of one every few days; thus the probability that the order gets messed up is low. On the other hand, the actual order of the data is not strictly necessary for the use of the data itself, as long as there is an efficient mechanism to retrieve the needed data. Indeed, the data on the blockchain is needed when a verifier needs to check a credential. To check the credential handed in by an owner, the verifier needs to retrieve from the chain the credential definition, the schema used by the credential definition and the VerKey of the issuers. It is not important in which order these three pieces of data are written in the blockchain, as long as they are all retrieved by the verifier. The Sovrin blockchain allows to efficiently recover the needed data. When writing the data on the replicated blockchain we need to implement an efficient mechanism to retrieve the needed data. Recall that the read operation that we assumed available, allows to retrieve the k-transaction where k is not the index in the real blockchain but the index of the overlay data.

4.2 Replicating KERI Data

KERI (Key Event Receipt Infrastructure) (Smith, 2023) is a decentralized key management infrastructure. Such infrastructure needs to manage public cryptography key pairs. The basic tasks are key reproduction (creation and derivation), key recovery and key rotation. Key pairs are associated to identities and, in the KERI jargon, identities are “controlled” by a *controller*. All the key management events include a signature from the controller of the associated identity. The KERI system is built on the concept of a root-of-trust, which creates a secure binding between a controller, a key pair, and an identity. The events, called *key management events*, or just key events in the KERI jargon, are “inception events“, that create an identifier and its associated keys and “rotation events“, that represents the rotation of the keys of an existing identifier. Moreover, there are special events called *key event receipts*, that represent the validation of the events. The function of KERI is based on a *key event receipt log* (KERL), that is an order set of all the key event receipts. In a decentralized implementation of KERI, such a log can be written in a blockchain. In (Smith, 2023) (see Section 10.2), an alternative approach that removes the need for a totally ordered distributed consensus ledger is suggested. Thus KERI can be an application that could benefit from the blockchain data replication that we have proposed.

5 CONCLUSIONS

In this paper, we have put forward the idea of replicating blockchain data over several blockchains to face the possibility that a blockchain suffers a total failure. Blockchains are very resilient to failures of participating nodes but only if such failures are contained within reasonable limits. If failures go beyond these limits the blockchain can become completely unreliable and even disappear.

We have also discussed two actual use cases of applications that make use of an underlying blockchain to implement their services.

Future work can go towards several directions. We have assumed a quite simple scenario where we have at our disposal a write and a read operation for all the blockchains. One could consider different settings in which the underlying operations allowed by the blockchains are less powerful. We have not considered temporary outages of the blockchains; dealing with such events is a future improvement.

We are also working on an actual prototype implementation over a few of the most popular blockchains.

ACKNOWLEDGEMENTS

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU-NGEU.

REFERENCES

Attiya, H., Dwork, C., Lynch, N. A., and Stockmeyer, L. (1994). Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41(1):122–152.

Bacis, E., Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2019). Securing resources in decentralized cloud storage. *IEEE Transactions on Information Forensics and Security*, PP:1–1.

Ben-Or, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30.

Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

De Prisco, R., Lampson, B. W., and Lynch, N. A. (2000). Revisiting the paxos algorithm. *Theoretical Computer Science*, 243(1-2):35–91.

Devcoin (Accessed: 2023). Devcoin. https://www.devcoin.org/.

Diallo, E., Laube, A., Agha, K., and Martin, S. (2019). Efficient block replication to optimize the blockchain resources. In *2019 3rd Cyber Security in Networking Conference (CSNet)*, pages 1–5.

Dwork, C., Lynch, N. A., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

Foundation, P. (Accessed: 2023). Freicoin. https://wiki.p2pfoundation.net/Freicoin.

Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the 7th ACM symposium on Operating systems principles*, pages 150–162.

Guerraoui, R. and Vukolic, M. (2010). Refined quorum systems. *Distributed Computing*, 23:1–42.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

Li, K., Tang, Y., Chen, J., Yuan, Z., Xu, C., and Xu, J. (2020). Cost-effective data feeds to blockchains via workload-adaptive data replication. In *Proceedings of the 21st International Middleware Conference*, Middleware '20, page 371–385, New York, NY, USA. Association for Computing Machinery.

Malkhi, D. and Reiter, M. (1998). Byzantine quorum systems. *Distributed Computing*, 11(4):203–213.

Malkhi, D., Reiter, M., Wool, A., and Wright, R. (2001). Probabilistic quorum systems. *Information and Computation*, 170(2):184–206.

Pease, M., Shostak, R., and Lamport, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234.

Pillai, B., Biswas, K., Hóu, Z., and Muthukkumarasamy, V. (2022). Cross-blockchain technology: Integration framework and security assumptions. *IEEE Access*, 10:41239–41259.

Pillai, B., Biswas, K., and Muthukkumarasamy, V. (2020). Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review*, 35.

Rabin, M. O. (1983). Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE Computer Society.

Smith, S. M. (Accessed: 2023). Keri: Key event receipt infrastructure. https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/KERI_WP_2.x.web.pdf.

Westerkamp, M. and Kupper, A. (2022). SmartSync: Cross-blockchain smart contract interaction and synchronization. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE.