Blockchain-based Data Provenance

Filipe Lautert[1], Daniel F. Pigatto[1], Luiz Gomes-Jr.[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA - UTFPR)

[filipelautert@alunos., pigatto@, lcjunior@]utfpr.edu.br

Abstract. *Data provenance tracks the origin of information with the goal of improving trust among interested parties. One of the key aspects provided by data provenance is transparency, which allows stakeholders to follow all the changes applied to the information (e.g. a document). Blockchains, a recent technological development, allow transparency in a distributed application context without the need for a trusted centralized entity. The approach presented here aims to use blockchain as a secure, shared and auditable storage providing transparent data provenance. Our proposal builds upon the well established W3C Prov Model, which simplifies adoption of the framework. An application consisting of a client and a REST API service that is able to store provenance information using open standards in a blockchain has been developed. Here we report the results of several stress tests to validate the practicability of our approach.*

1. Introduction

Provenance is the process or techniques utilized to track the origin, authorship and history of any given object. It was originally used in the context of works of art to make sure that an object was created by the claimed author. Data Provenance aims to track the history of a piece of data, starting from its original source and accounting for all the transformations.

Important requirements of data provenance systems are transparency and immutability of a version of the data. Blockchains offer characteristics that meet these requirements: Blockchains are distributed immutable ledgers that track all transactions and register them on blocks of information that are tied together (chains). One block is linked to another using its hash, thus ensuring the immutability and security of the blockchain.

The work presented here enables clients to store provenance records that will be shared and tracked by the blockchain structure. This approaches reduces the complexity of the data provenance system by delegating tracking to the blockchain whilst helping to secure information by preventing data tampering. The main contributions of this paper are: (i) the implementation of a distributed service to demonstrate the practicality of the solution, and (ii) tests and analyses that demonstrate the practicability of our approach.

2. Related work

Due to their flexibility, blockchains are being used for many different applications. For example, Azaria et al. [2016] propose MedRec, a blockchain for medical data access and permission management. This solution is a good example of a blockchain that is available on the market (Ethereum) that was explored as a backend to a complete solution that has APIs, security and linking protocols all built relying on the blockchain as its safe storage.

Greenspan [2016] cites provenance tracking as one of the main usages for blockchain and emphasizes how a blockchain can be more secure than a database for provenance information. And Kamath [2018] reports on Walmart's blockchain solution using IBM Hyperledger developed to track the supply chain (food provenance of Pork in China and Mangoes in USA). No details are provided about the implementation but this is an example of business's using blockchain to track provenance.

Data provenance has diverse application scenarios. Simmhan et al. [2005] surveyed many research efforts in the data provenance field and created a taxonomy to categorize those efforts. The work demonstrates that provenance systems can be built to support a number of uses, having different characteristics and different ways of working.

Bauer and Schreckling [2013] define characteristics and requirements for data provenance used on the Internet of Things (IOT), which is a recent use for provenance. Those characteristics can be applied to the internet and any other recent usages for provenance, and for this work they will be used as the baseline for data provenance requirements. Comparing those requirements with blockchain's characteristics it is possible to see how a blockchain solution fits in a provenance scenario: **(i) Integrity**: Blockchain is decentralized, information written to it is immutable, and even in a private implementation no one is able to change what has been recorded. **(ii) Availability**: As it is distributed and transactions can be signed, anyone with access to the blockchain can hold a copy of the entire chain and verify all of the transactions on it, thus being available and verifiable as required. **(iii) Completeness**: Data can be stored in any way that is required, so every action and information can be stored on the chain. **(iv) Confidentiality**: Blockchain can be executed in a private way and cryptography can be used to store provenance data if required. **(v) Efficiency**: consensus mechanisms as Proof of Work (POW) may be a problem to achieve efficiency on a blockchain, but there are options as Proof of Stake (POS) and others that try to fix this problem. Efficiency can be a point of failure and needs to be well reviewed before a blockchain is selected to track provenance.

W3C PROV Standard, a provenance protocol, has been defined by the W3C. Groth and Moreau [2013] define a set of documents called the "Prov Family" that was developed after an extensive research including use case cataloging, requirements elicitation and a literature survey. The last version of the set was released in 30 April 2013. It aims to define a model and related operations to enable the inter-operable interchange of provenance information that can be used as a reference to design provenance representation, especially in heterogeneous environments such as the Web. Libraries have been implemented to generate and handle those documents as published on the PROV Implementation Report[1] which is utilized in this work.

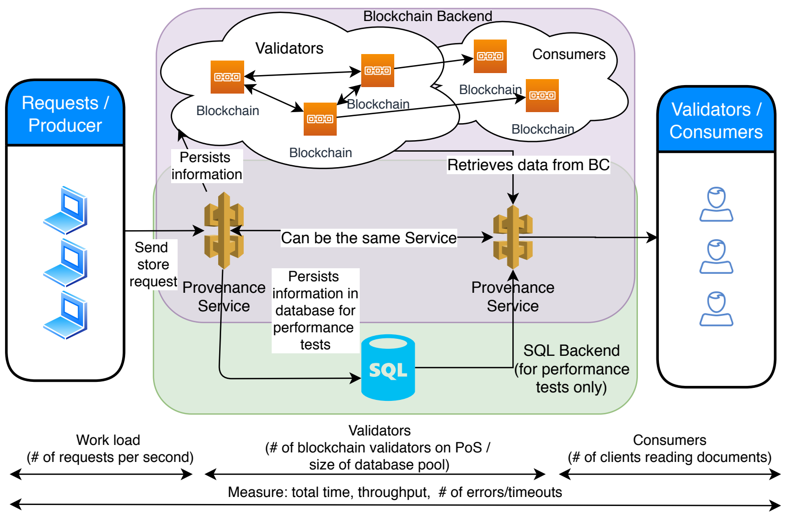
3. Use case and Architecture

A common use for provenance was selected to illustrate the architecture: a supply chain. In this scenario, a company needs to track a food supply chain as this information needs to be shared with health agents outside the company. The use case would be: the producers use an application that store the data regarding the steps of the production as harvest, processing and dispatching of the food. This application is a client of our service, and

[1]https://www.w3.org/TR/prov-implementations/, visited on 2020-04-13

it stores the relevant information using the W3C prov format. Health agents access the blockchain information using our service and can audit the retrieved information related to a specific product. The servers that are acting as blockchain validators are managed and executed by the supply chain owners.

Architecture: The architecture is composed by a client that handles the task of getting the required data and storing them in a provenance system that would be accessible by the auditors and anyone else that requires access to that information. Figure 1 outlines the proposed architecture. There are 2 actors in this figure: the "producer" operated by the supply chain users and the "Consumers" that is the final destination of the provenance data generated.



Blockchain Backend
Validators
Consumers
Blockchain
Blockchain
Blockchain
Blockchain
Blockchain
Requests / Producer
Validators / Consumers
Persists information
Retrieves data from BC
Send store request
Can be the same Service
Provenance Service
Provenance Service
Persists information in database for performance tests
SQL
SQL Backend (for performance tests only)
Work load (# of requests per second)
Validators (# of blockchain validators on PoS / size of database pool)
Consumers (# of clients reading documents)
Measure: total time, throughput, # of errors/timeouts

Figure 1. Architecture

Provenance service appears twice to demonstrate that it can be deployed for each actor. The left one represents the service responsible for receiving REST calls and persisting the information in a blockchain. The right one is responsible for receiving a request from the consumers, accessing the blockchain and interpreting the provenance data. The encoding and decoding of the data required by the other actors is handled here.

The **Blockchain backend** represents the cloud of blockchain nodes. The provenance service executes calls to the Blockchain API requesting to store or retrieve information. It is split in 2 clouds: the "validators" cloud represent the nodes that have voting power and are involved in the process of creating new blocks; and the "Consumers" with instances that will only retrieve copies of the blockchain. In contrast the **SQL backend** represents the traditional approach were data would be stored in a database. This approach is outlined in the proposed architecture as it is used as an alternative route for performance comparison tests. In practice, it is not used in our approach.

4. Experiments planning and configuration

To validate the practicability of the approach, two applications were developed: a Provenance Service that is able to persist information in the blockchain or in a database for performance comparison; a client that generates metrics for different test scenarios. The Blockchains were executed over Docker containers to allow for multiple instances.

The API presented in this work was modeled after ProvStore, a service that is used as a repository to store and manage documents in W3C's PROV Data Model as described by Huynh and Moreau [2014]. ProvStore is available at Open Provenance umbrella[2], and there are client implementations[3] that can be used to access the REST API and manage PROV documents. Python provstore-api library was used to create a client that handles provenance operations on the original ProvStore and in the blockchain version.

The chosen Blockchain implementation for the service API was Tendermint Kwon [2014]. It was selected for two main reasons: 1) simple REST API that allows to store and retrieve data in the blockchain; 2) fast set up and execution of many nodes using docker instances. Tendermint's consensus protocol is a byzantine fault tolerance (BFT) Proof of Stake protocol that has a peculiarity when comparing with other PoS protocols: validator nodes are defined statically in the genesis file instead of being selected based on some kind of value (coin, processing power, etc).

Testing scenarios and details: To benchmark the performance of the application, three test scenarios were developed and executed. Those tests were performed by sending requests containing a W3C Prov document of around 1kb as a payload to the API. Three backend configurations were used in the service API: Blockchains with 1 (BC1) and 4 (BC4) validator nodes, as 1 node is the minimum configuration for Tendermint and 4 nodes is a configuration that allows 1 node to fail while the other 3 will continue to generate blocks; and a **Local Database server (DB)**, specifically a PostgreSQL 11 running on a Ubuntu 19.04 host.

To test the backends under different loads, 3 throughput levels were defined: 1 request per second (1 rps), 20 requests per second (20rps) and 40 requests per second (40 rps), until it reaches the maximum request number of 1000 requests. Each one of those combinations (backend and requests per second) was executed 10 times in a round robin fashion, meaning that each combination was invoked 10.000 times. The test execution combinatorial formula is: $10 * ((BC1 + BC4 + DB) * (1rps + 20rps + 40rps) * 1000r)$

All of the tests were done in a dedicated machine with a Core i7-5557U CPU (4 physical cores), SSD storage and 16GB of RAM. The memory was monitored during the execution and it never used more than 4GB of RAM and swap was never used. Actions were taken to ensure that the machine was not overloaded during the executions.

5. Results and discussion

Table 1 shows the results of the executions logged by the scripts. The table demonstrates that execution time for the same number of requests per second is similar, so it does not matter if the program is run using a blockchain with 1 or 4 nodes or against the Database, the application takes around the same time to execute. But there is a big difference when we look at the blocked requests and system load means. Figure 2 shows a view of the blocked requests and figure 3 shows the overall system load. It is important to note that the system load is mainly influenced by the server side, as the client is mostly blocked waiting for the responses. Correlating the two figures:

- Blockchain with 1 node has the lowest load but the average number of requests blocked: figure 3 shows that load confidence intervals are consistently below 1,

[2]https://openprovenance.org/store/, visited on 2019-09-07
[3]https://pypi.org/project/provstore-api/, visited on 2019-09-10

and it is explained as Tendermint will wait 1 second before committing a block upon receiving the first data.

- Blockchain with 4 node has the highest load and the highest average number of requests blocked: as it waits 1 second before committing the block and the requests are made to the nodes in a round-robin manner, the nodes are constantly negotiating the block. Because of that less requests are added in a block, generating a longer queue and consuming more computational power.
- Database has the average load and the lowest average number of requests blocked as it deals with each request as a separated transaction, not waiting for timeouts.

Type	Req /sec	Total time	Client store (s)	Client retrieve (s)	Load	Blocked requests	Server retrieve (s)	Server store (s)
BC1	1	02:48:05	1.5959	0.0138	0.1615	2.1104	0.0027	1.5634
BC1	20	00:09:03	1.7615	0.1131	0.5470	42.5346	0.0070	1.6667
BC1	40	00:05:00	2.0207	0.2277	0.6547	89.4222	0.0073	1.8761
BC4	1	02:48:13	2.5429	0.0160	0.7476	3.0429	0.0028	2.4979
BC4	20	00:09:19	2.8668	0.1751	2.1306	63.1977	0.0076	2.5550
BC4	40	00:05:17	3.3688	0.5000	3.4800	131.6167	0.0080	2.5535
DB	1	02:48:07	0.0158	0.0258	0.0924	0.9990	0.0086	0.0050
DB	20	00:09:47	0.0577	0.1423	1.1548	16.9392	0.0342	0.0124
DB	40	00:05:58	0.0831	0.2004	1.9488	22.8653	0.0462	0.0183

Table 1. Execution averages

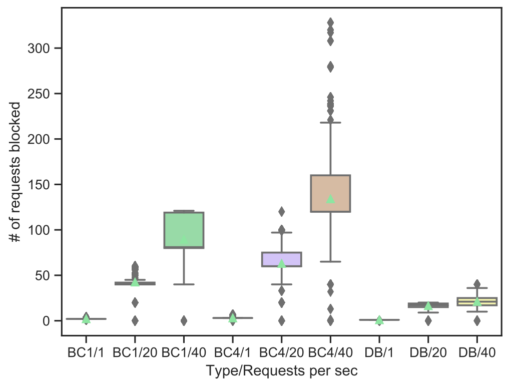


Figure 2. Client Blocked Threads

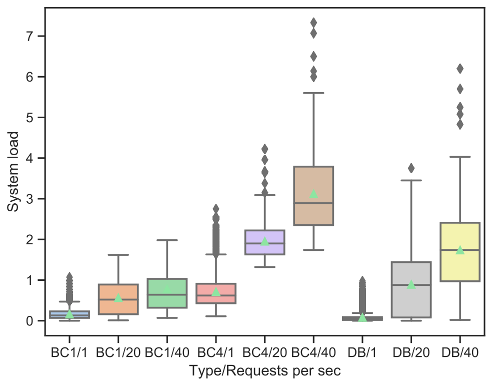


Figure 3. Client Load (1 min)

Server side performance is illustrated in figures 4 and 5. Looking at the figures it is possible to understand better the time it takes to store the documents and why the clients had different numbers of blocked requests: the Blockchain with 1 node takes around 1.56 seconds to store a document while the blockchain with 4 nodes takes 2.49 seconds to handle the same task, and the database handles it in 0.005 seconds. As Tendermint waits 1 second before committing the block and the rest of the time is due to the overhead generated by negotiating the block content. On the other hand, the retrieve time from blockchain is faster than the database.

The same figures also demonstrates that the confidence intervals under different loads are similar for each backend. Comparing it with the load on figure 3 that increases when the number of requests increases, we can imply that the tests are extracting the maximum from the hardware that does not impact the testing results.

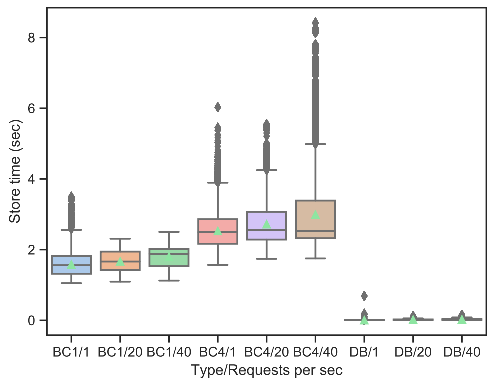


Figure 4. Server store

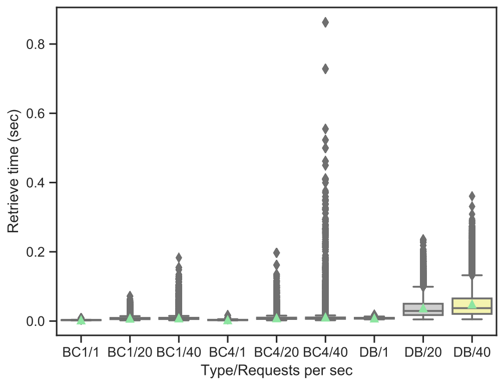


Figure 5. Server retrieve

Discussion: Our blockchain was configured to store information, providing what is required to store provenance information using W3C Prov standard and take all the advantages of the blockchain storage, suiting it to the user needs.

Before testing we expected the Blockchain's performance to be far worse than the Database's. The results show that overall performance is similar, although performance for a single document being slower on the blockchain due to Tendermint's timeout in block creation. When running the blockchain with 4 validators, the system's load was higher as expected, but it has the advantage of having the data replicated in 4 nodes; besides, in a real scenario this load would be split between 4 separated servers.

References

Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30. IEEE, 2016.

Sabine Bauer and Daniel Schreckling. Data provenance in the internet of things. In *EU Project COMPOSE, Conference Seminar*, 2013.

Gideon Greenspan. Four genuine blockchain use cases. Technical report, 2016. URL `https://www.multichain.com/blog/2016/05/`.

Paul Groth and Luc Moreau. Prov-overview. an overview of the prov family of documents. w3c working group note. *World Wide Web Consortium*, 2013.

Trung Dong Huynh and Luc Moreau. Provstore: a public provenance repository. In *5th International Provenance and Annotation Workshop (IPAW'14)*, June 2014.

Reshma Kamath. Food traceability on blockchain: Walmart's pork and mango pilots with ibm. *The Journal of the British Blockchain Association*, 1(1):3712, 2018.

Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1:11, 2014.

Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance techniques. *Computer Science Department, Indiana University*, 47405:69, 2005.