Incorporating Blockchain into RDF Store at the Lightweight Edge Devices

Anh Le-Tuan[1,3(✉)], Darshan Hingu[1], Manfred Hauswirth[1,2], and Danh Le-Phuoc[1]

[1] Open Distributed Systems, TU Berlin, Berlin, Germany
anh.letuan@tu-berlin.de
[2] Fraunhofer Institute for Open Communication Systems, Berlin, Germany
[3] Insight Centre for Data Analytics, NUI Galway, Galway, Ireland

Abstract. RDF stores provide a simple abstraction for publishing and querying data, that is becoming a norm in data sharing practice. They also empower the decentralised architecture of data publishing for the Web or IoT-driven systems. Such architecture shares a lot in common with blockchain infrastructure and technologies. Therefore, there are emerging interests in marrying RDF stores and blockchain to realise desirable but speculative benefits of blockchain-powered data sharing. This paper presents the first RDF store with blockchain that enables lightweight edge devices to control of the data sharing processes (personal, IoT data). Our novel approach on the deep integration of the storage design for RDF store enables the ability to enforce controlling measures on access methods and auditing policies over data elements via smart contracts before they fetched from the sources to the consumers. Our experiments show that the prototype system delivers an effective performance for a processing load of 1 billion triples on a small network of lightweight nodes which costs less than a commodity PC.

Keywords: Blockchain · Linked Data · RDF store

1 Introduction

User-generated and sensor data is being widely used by various applications to make them smarter and better. While such applications are using the data for nearly free, such consumer data from both public and private sources is incredibly valuable to corporations, marketers, investors, and individuals. For example, American companies alone are estimated to have spent over $19 billion in 2018 acquiring and analysing consumer data, according to the Interactive Advertising Bureau.[1] There are recent recurrent questions of how users can take control and make the benefit out of it. It is obvious that in order to take control

[1] https://www.iab.com/news/2018-state-of-data-report/.

A. Le-Tuan and D. Hingu—These authors Contributed equally to the work.

M. Acosta et al. (Eds.): SEMANTiCS 2019, LNCS 11702, pp. 369–375, 2019.
https://doi.org/10.1007/978-3-030-33220-4_27

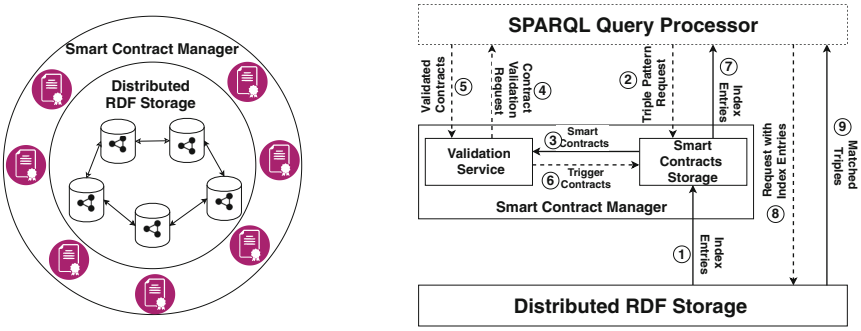
of one's data, there must be an ability to accurately account for ownership, and similarly account or keep a record of all transactions, exchanges and permissions while in a secure and tamper-proof manner.

The arrival of blockchain technologies gives the promise to get your data out of a corporation's centralised database to store your own devices or your encrypted storage of your choice at the edges of networks. This will be critical in helping develop transparency and accountability in data sharing as we take ownership of our data. Blockchain provides a number of substantive benefits. It provides a layer of transparency and accountability for data ownership and transactions. It can also minimise the influence of data middlemen in any consumer data transaction while putting the data back in the hands of the consumer. This means users can regain control of who uses our data, when our data is used and our compensation for it.

In parallel, the recently emerging trend of edge computing paradigm for IoT-driven information systems makes it much more feasible to push computation and data management operations closer to the data sources. Being able to store and query data at the edges of networks offers opportunities to improve performance and to reduce network overhead, but also flexibility for the continuous integration of new IoT devices and data sources. This motivates us to build a novel distributed RDF store which leverages blockchain benefits for data publishers at the edges of networks. To the best of our knowledge, our system is the first of this kind. The system design will be presented in Sect. 2, the implementation report will be followed in Sect. 3. Our experimental results in Sect. 4 show that a small cluster of Raspberry Pis (cost less than a commodity workstation) can efficiently handle 1 billion RDF triples of IoT data.

2 System Overview

To store and to share RDF data on the edge with the guarantee of the data ownership and the compensation for shared data, we marry a distributed database system and blockchain. Confronting the decentralised edge networks, the distributed database system allows any members to share their data, resources



(a) System overview

(b) General procedure

Fig. 1. System overview and general procedure

while maintaining their autonomy and independence from centralised servers. A feature of blockchain technology is the smart contract which allows the data owners to control the access to their data (e.g., who can have the access or how much to pay to gain the access).

Our system (Fig. 1a) consists of two subsystems: a *Distributed RDF Storage (DRS)* and a *Smart Contract Manager (SCM)*. The DRS takes responsibility to store RDF data among connected devices in the network physically. Meanwhile, to secure the ownership of the data, the access to the data in the distributed storage is encrypted into smart contracts which are published and managed by the SCM. The basic principle of the system is that when a provider wants to trade his/her data, she/he publishes the data partitions (e.g., a set of sensor readings) associated with smart contracts. A smart contract contains the meta information of a published data partition, for example index key, and a contract that can be used to specify terms like price scheme and access control policy on data to be fetched to clients.

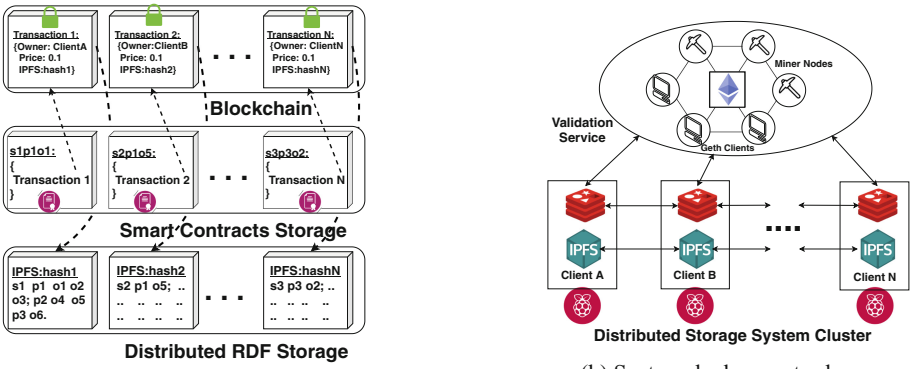
Considering that to answer a SPARQL query, the SPARQL query processor performs graph pattern matching over RDF datasets. The graph matching operator executes join operations between RDF triples that match the triple query patterns. Therefore, to retrieve query pattern matched data, we organise RDF data in the similar way as RDF4Led [4] does. We store RDF triples in three storage layouts as sorted permutations of triples: SPO (Subject - Predicate - Object), POS and OSP. Each layout is a sorted list and is partitioned into chunks called data blocks. The first triple of each data block and the physical address of the data block are formed an index entry which is kept in the main memory. The triples of the index entries are the keys for searching the data blocks that potentially contain the matched triple of a query pattern. However, instead of storing the data blocks in the local file systems as in RDF4Led, we use distributed file systems to create the DRS subsystem. The distributed file systems provide scalable data distribution and sharding featured associated with distributed data structures like DHT [1]. In such data structure, each data block that is stored is mapped to a unique identifier. The identifier space is partitioned among the nodes. Each node is responsible for storing all the data blocks that are mapped to identifiers in its portion of the space. Hence, data is distributed and resource consumption is balanced among the edge nodes. Furthermore, instead of keeping the unique access identifiers of data blocks in index entries openly, we allow publishers to encrypt them into smart contracts and keep them in the *Smart Contracts Storage (SCS)* of the SCM (see Fig. 1b). In the Smart Contracts Storage, the smart contracts are kept sorted by the triple, therefore, the corresponding smart contracts that hold the query pattern matched triples can be searched as described in [4]. Finally, a data block can be fetched only when the access identifier is revealed, in consequence, only when the smart contract that holds identifier of the block is triggered.

The general workflow of the system is visualised in Fig. 1b. When a client starts sharing her/his RDF dataset, the system indexes the data, partitions the indexes into data chunks, stores the data chunks in the DRS and sends the index entries to the SCM (1). The SCM encodes the arriving index entries into

smart contracts and stores these contracts in the Smart Contracts Storage. From the given triple requests (2), the SCM searches in the SCS for the contracts that hold the accesses to the requested data. Later, these contracts are sent to the *Validation Service* which developed with blockchain technology (3). The Validation Service verifies the if contracts are validated, and navigates validation requests to the SPARQL Query Processor (4). When the contracts are validated (5), the Validation Service triggers these contracts (6), returns the opened index entries with access identifiers the SPARQL Query Processor (7). With these index entries and access identifiers, the SPARQL Query Processor should be able to fetch the matched triples from the DRS to join operators (8) (9).

3 Implementation and Deployment

To implement the system, we extended the Java code base of RDF4Led [4] that allows RDF data processing tasks (e.g., parsing, indexing) to execute on the edge node. The DRS subsystem is implemented by re-engineering the Physical Layer to store the data blocks (which contain RDF molecules) in the p2p file system IPFS [1]. IPFS is a secure, high-throughput, distributed block storage model with content-addressed hyper links. It allocates a unique hash for each stored block of data. In IPFS, the data blocks are identified and retrieved by their IPFS hashes. In SCM, the SCS is stacked on top of the Buffer Layer. The smart contract and related features are implemented with Ethereum[2]. However, instead of using an in-memory caching of RDF4Led, the index entries which also keep the address of their corresponding smart contracts are stored in Redis[3]. Redis is a distributed in-memory key-value data that allows data is clustered in memory of multiple devices. When a devices join the network, it also contributes its computational



(a) Physical data organisation

(b) System deployment schema

Fig. 2. Physical data organisation and System deployment schema

[2] https://www.ethereum.org/.
[3] https://redis.io/.

resources to the whole system. To communicate the Validation Service with the Ethereum's blockchain network, we use Web3j of Ethereum.

Figure 2a illustrates the physical data organisation of SPO index layout in our system. In the DRS layer, sorted list of triples are partitioned, compressed as molecules and stored in IPFS as byte arrays. The associated IPFS hash of RDF molecules are packed with addresses of the smart contracts. The smart contracts are stored in Ethereum blockchain and are programmed to return the stored IPFS hashes if the transactions are triggered. In the SCS, each entry contains the first triple of each molecule and its smart contract id. These entries are kept in sorted list of Redis which provides the key-range search that allows index lookup in the same fashion as in RDF4Led.

Figure 2b depicts our system's deployment strategy. The nodes are installed with IPFS and Redis which make our SCS and DRS layer respectively. A node's IPFS and Redis clubs with another node's IPFS and Redis to form a cluster respectively within the network to provide a decentralised distributed data storage. The *Validation Service* Cluster is implemented using Ethereum's Private Blockchain using Geth which in turn is configured to use Proof-of-Authority consensus mechanism. The remaining nodes are set up as full-nodes clients, which provide additional storage to the service.

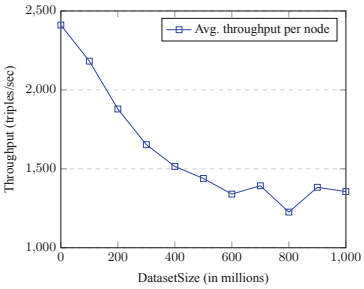
4 Evaluations

This section presents our evaluation on the deployment as presented in Fig. 2b. We created a network of 15 Raspberry Pi 3 model B (ARMv7 Quad-Core 1.2 GHz CPUs, 1 GB RAM, 64 GB SD card, Raspbian OS), each node costs approximately €50. *Note that the total cost of the whole setup is less than a commodity PC.* The RDF dataset for the evaluation is generated by mapping sensor readings from NOAA[4] dataset to RDF using SSN/SOSA ontology [3]. The schema of our data set is provided a long with the implementation in our Github repository[5].

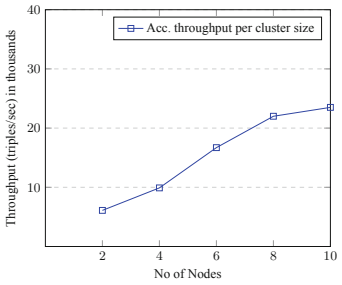
We evaluated our system with two experiments. Experiment 1 consists of two tests on two system settings. Firstly, we fixed the cluster size at 10 nodes and measured the average throughput per node when inserting more data. Secondly, we observed the increasing of the throughput when adding more node. A billion and 100 million triples dataset was used respectively in the first experiment. In Experiment 2, we measured the response time for searching the matched triples of single query patterns on 1 billion triples dataset.

Figure 3a represents the first setting's result with the accumulated throughput of a fixed number of nodes. As predicted, the throughput gradually decreases when data stored in the storage increases. After 500 million inserts, we can see a stagnant flow in the graph. The result of the accumulated throughput of a varying number of nodes in the cluster is presented in Fig. 3b. Here, we plot throughput (triples/seconds) in thousands against the number of nodes participating. It appears as the number of processing nodes in the cluster increases, the

[4] https://www.ncdc.noaa.gov/.
[5] https://github.com/anhlt18vn/Semantic2019.



(a) Acc. Throughput on 10-nodes Cluster Sizes



(b) Acc. Throughput on Varying Cluster Sizes

Fig. 3. Insert throughput results

throughput also increases. After reaching a peak point, the growth has minimised due to the decentralised nature of the system.

From both the tests, we can observe that the system is competent in handling a large dataset due to its decentralised ecosystem. We have also seen that the system's performance becomes stagnant after attaining peak. It is mainly because the decentralised system adheres to the network latency, which occurs due to its replication and distribution strategies. With the current architecture, the system has proven highly scalable, though it is acknowledged that there has been a trade-off in terms of performance to achieve this nature of the degree of scalability.

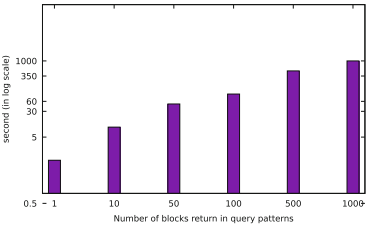


Fig. 4. Query response time

Figure 4 shows the result of experiment 2. We can see that the system deliver very good performance for the query that needs data from 1–10 blocks on the 1 billion dataset. However, the response time increases linearly to the number of fetched blocks. From our analysis, the delay is mainly contributed by the throughput/delay of processing multiple transactions on Ethereum.

5 Conclusion and Outlook

This paper presents the first implementation of a novel RDF distributed store with blockchain technology for the decentralised edge network. The experiments proved that the system can be deployed on a network of lightweight edge devices such as Raspberry Pis. With a network of fifteen nodes of Raspberry Pi, the system is able to host a dataset up to a billion RDF triples. The cost for deploying such system is quite competitive and flexible as a Pi node costs less than €50 and the number of nodes can be elastically increased or decreased at runtime. Next step, we will increase tenfold in the number of nodes to study the scalability and limitations. For the shortcomings shown in Fig. 4, the processing

throughput causing long delays on the queries that use a large number of data blocks can be increased by new achievement of transaction throughput, e.g. 20 k transactions/second in [2].

Acknowledgements. This work was funded in part by the German Ministry for Education and Research as BBDC 2 - Berlin Big Data Center-Phase 2 (ref. 01IS18025A), Irish Research Council under Grant Number GOIPG/2014/917 and Marie Skodowska-Curie Programme H2020-MSCA-IF-2014 (SMARTER project) under Grant No. 661180.

References

1. Benet, J.: Ipfs - content addressed, versioned, p2p file system. arXiv:1407.3561 (2014)
2. Gorenflo, C., Lee, S., Golab, L., Keshav, S.: Fastfabric: scaling hyperledger fabric to 20,000 transactions per second. arXiv:1901.00910 (2019)
3. Haller, A., et al.: The modular SSN ontology: a joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. Semant. Web **10**(1), 9–32 (2019)
4. Le-Tuan, A., Hayes, C., Wylot, M., Le-Phuoc, D.: Rdf4led: an rdf engine for lightweight edge devices. In: IOT 2018 (2018)