Chitti, P., Murkin, J., & Chitchyan, R. (2019). Data management: Relational vs blockchain databases. In H. A. Proper, & J. Stirna (Eds.), *Advanced Information Systems Engineering Workshops: CAiSE 2019 International Workshops, Rome, Italy, June 3-7, 2019, Proceedings* (pp. 189-200). (Lecture Notes in Business Information Processing; Vol. 349). Springer, Cham. https://doi.org/10.1007/978-3-030-20948-3_17

Peer reviewed version

License (if available):
Other

Link to published version (if available):
10.1007/978-3-030-20948-3_17

Link to publication record in Explore Bristol Research
PDF-document

This is the accepted author manuscript (AAM). The final published version (version of record) is available online via Springer Cham at https://doi.org/10.1007/978-3-030-20948-3_17 . Please refer to any applicable terms of use of the publisher.

University of Bristol - Explore Bristol Research

General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available: http://www.bristol.ac.uk/pure/user-guides/explore-bristol-research/ebr-terms/

Data Management: Relational vs Blockchain Databases ⋆

Phani Chitti, Jordan Murkin, and Ruzanna Chitchyan[0000−0001−6293−3445]

Department of Computer Science, University of Bristol, MVB Building, Woodland Road, Bristol, BS8 1UB
{sc18092,jordan.murkin,r.chitchyan}@bristol.ac.uk

Abstract. This paper presents an initial exploitative study of how the relational and blockchain databases compare in defining and deploying data structures,populating these with new entries, and retrieving the relevant data for further use. The aim of this study is to better inform the software developers in general and the distributed application developers in particular.

Keywords: data management · blockchain · RDBMS · smart contract · distributed ledger

1 Introduction

Traditionally software systems store data in relational database management systems (RDBMS), occasionally using other means of storing data like File systems (for storing documents). Off late No-SQL databases for storing discrete data like documents, search results etc. have gained popularity. Yet, the RDBMSs continue to be preferred for storing data generated out of enterprise systems. Under these solutions the responsibility for maintaining the database, ensuring data integrity and providing secured access to different users based on their roles, lies with a select set of people - called administrators - who are employed by the database owners. This centralized authority which maintains the database also ensures the administrative activities, like regular backups etc.

On the other hand, (public) distributed ledger technologies (a.k.a. blockchains) are gaining more recognition as decentralized data stores and as an alternative to the traditional data storage systems. Initially these technologies arose to support cryptocurrencies [14]. However, their distinguishing features - distributed consensus, non-dependency on a centralized authority, built-in trust, low entry barriers for joining the network - are attracting many diverse domains to adapt this technology. Yet, in order to confidently develop an enterprise application that would rely on a distributed ledger solution as it's main database, we must have confidence that all the necessary services supported by the current solutions

⋆ This research is funded by the UK EPSRC Refactoring Energy Systems fellowship (EP/R007373/1)

(primarily RDMSs) can be adequately delivered through the distributed ledger solutions.

To this end we present an initial exploitative study of how the relational and blockchain databases compare in defining and deploying data structures, populating these with new entries, and retrieving the relevant data for further use.

One of the most popular distributed ledger environments today is the blockchain-based Ethereum platform which supports building decentralized applications [1], and provides the flexibility of writing arbitrary state transition functions via scripts. The scripts are executed by the Ethereum Virtual Machine (EVM). In Ethereum the state is made of two types of accounts: External (private key) accounts, and Contract (contract code) accounts [2]. External accounts can send messages, by creating transactions, to Contract accounts. Contract accounts, upon receipt of the transaction request messages, execute their contract code with the given transaction parameters. Contract accounts also can send messages to other contracts. The contract accounts contain their contract code and store the values of persistent variables. Each action undertaken by a contract account when executing code, storing a data value, or calling another contract account, should be paid for by the initiator of the contract's transaction.

As Ethereum is the most used blockchian development platform today, we too use it for our exploitative study. As a relational database counterpart to Ethereum we use the MySQL RDBMS. MySQL provides a comprehensive set of advanced RDBMSs features, and, in its own right, is also one of the most well used RDBMSs.

This paper discusses an experiment where we set out to create a set o data structures for an application using both RDBMS and a blockchain-based solution. We aim to identify the correspondent activities needed to carry out a given task with these two alliterative technologies (as discussed in section 2). We then draw out the challenges and opportunities identified through this exploitative comparison (see section 3) and discuss these. The aim of this study is to better inform the software developers in general and the distributed application developers in particular on the issues that one has to address when using either of these technologies as a main database.

2 Experiment

This section describes the experiment that is conducted to evaluate each of above discussed database solution alternatives. The experiment is carried out using the context of a peer-to-peer energy trading scenario [13].

Energy Trading Scenario: A number of households (or farms, and other non-energy small businesses) generate renewable energy (e.g., via solar PV panels, wind turbines, etc.) for own use, and wish to sell the excess generation to other households. For this the excess generation is advertised through "sell requests" over a trading platform. Similarly, those wishing to purchase a certain amount of

energy advertise their "buy requests". A peer-to-peer energy trading algorithm (ETA) [13] that runs on the trading platform, works to match the sell and buy requests declared by the market participants for a given time period. The sell and buy requests are matched on the basis of scores calculated by the ETA for all trade participants. The scores are based on participants' preferences, which are provided by the participants when they join the trading platform (e.g., sell cheaper if buyer is local, e.g., within 2 miles, buy solar if possible, etc.).
The experiment consists of:

- Defining data structures to represent sellers and buyers: Each participant will have name, address, physical location details like longitude and latitude, contact details, as well as preferences on how (s)he wants to trade.
- Deploying the data structures: enabling their use either through a relational database or a blockchain;
- Storing the data generated per data structure: to replicate participation of a large number of users (here we use the Random() function to randomly generate values for user data).
- Retrieving the stored data to carry out transactions: data will be retrieved in order to form transactions between sellers and buyers. The transactions will be formed via the a Peer-To-Peer (P2P) energy trading algorithm (ETA).
- Storing trade transactions back into the databases.

Figures 1 and 3 depict the process of experiment realisation using a traditional relational database (MySQL RDBMS) and a blockchain-based solution (Ethereum). Both of these solutions are chosen due to their widespread popularity and availability of free access.

2.1 Implementing with MySQL RDBMS

As in any relational database, the data structures in MySQL are to be represented via relations (i.e., tables in a database). As shown in Fig. 1, the process starts with:

Fig. 1. Data storage and retrieval with MySQL

Defining tables and relationship among them. This requires that the data must be normalised and related, where:

- Normalization is the removal of the redundancy in data, leading to assured cohesion among entities and increased consistency.
- After normalization the data is arranged into a set of related entities (also called relations, tables). The relations for this study are depicted in Fig. 2. Unique identifiers (called keys) are defined for each record entry; these keys are referred in other records when a link is to be established between two pieces of data. Use of such keys ensures referential integrity between entities (i.e., making sure that the data integrity is preserved when two or more pieces of data need to refer to each other).

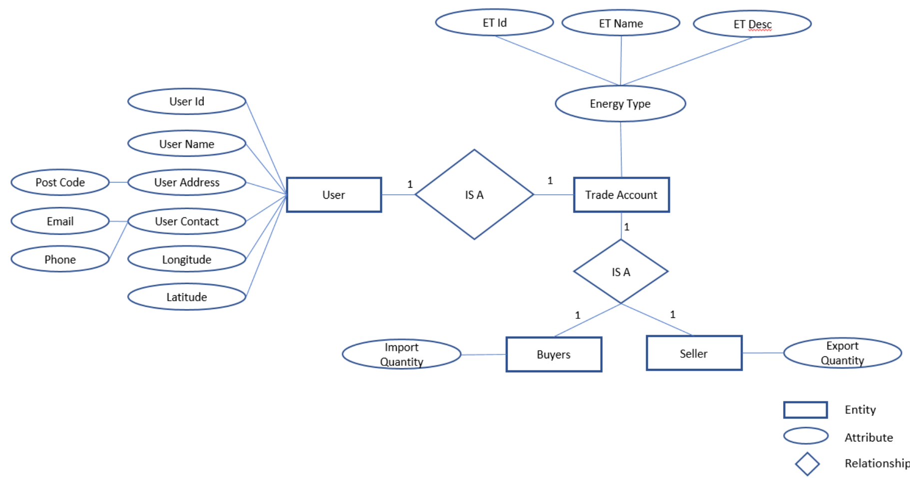

Fig. 2. Entities and Relationships identified after Normalization

With MySQL all the steps of our experiment (i.e., relation definition, deployment, data storage, and retrieval) are fully supported by the database management system itself, where a command line and/or a web interface tool/site can be used to carry out all the required actions. Thus:

- *Deployment* of the relations is reduced to CREATE TABLE command within the RDBMS (e.g., for a user table: CREATE TABLE IF NOT EXISTS User(UserId varchar(255) NOT NULL, UserName varchar(255) NOT NULL, UserAddress varchar(255) NOT NULL, Postcode varchar(255) NOT NULL, UserEmail varchar(255) NOT NULL, UserPhone varchar(255) NOT NULL, Longitude FLOAT NOT NULL, Latitude FLOAT NOT NULL, PRIMARY KEY(UserID));
- *Data storage* is reduced to INSERT command with reference to the specific table name, and a new record line is inserted and stored in the referenced table (e.g., INSERT INTO EnergyType VALUES (1,'PV','Solar Energy')).

– *Retrieval of particular records* is carried out through the SELECT command by referencing the unique keys within each data record, whereby the related data is identified by repeated use of this same key in other tables.

2.2 Blockchain based approach

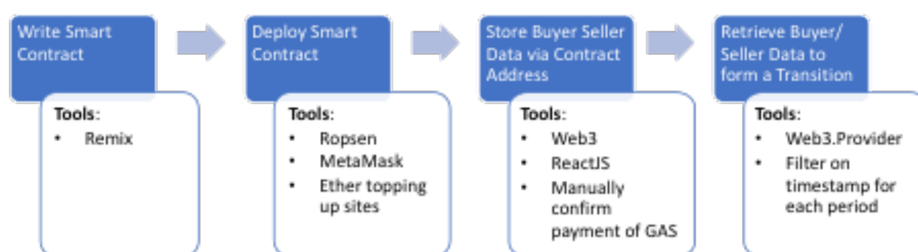

Fig. 3. Data storage and retrieval with Blockchain. Note: Tools listed here were used in this experiment, other tools could be chosen instead.

With the Ethereum alternative:

Defining the data structures is carried out through the creation of smart contracts (as shown in Listing 1.1) which will also contain the data to be stored in the blockchain. Hence, for this purpose, smart contracts are equivalent to tables in RDBMS.

Ethereum allows for the removal of duplication (akin to data normalisation in RDBMS) to be carried out by breaking data structures into multiple contracts then each refer to the other. As contracts may include references to other contracts via their contract addresses, this parallels to a kind of foreign key referencing. However, these references point to the whole of the contract (and its' entire data)and not to individual records; this is a contrast to the per individual record referencing of the RDBMSs.

Contracts may also emit events. An event affiliated to a contract, defines an indexing for quick filtering of the contract's data for outside of the blockchain access.

```
contract ETPAccount {
  bytes18 public postcode;
  bytes32 public latitude;
  bytes32 public longitude;
  uint16 public pricePreference;
  uint16 public distancePreference;

  event ElectricityExported(uint period, uint exports);
  event ElectricityImported(uint period, uint imports);

  constructor(bytes18 _postcode, bytes32 _lat,bytes32 _lng,
              uint16 _ppref, uint16 _dpref) public {
    postcode = _postcode;
    latitude = _lat;
    longitude = _lng;
    pricePreference = _ppref;
    distancePreference = _dpref;
  }

  function setPricePreference(uint16 _ppref) external {
    pricePreference = _ppref;
  }

  function setDistancePreference(uint16 _dpref) external {
    distancePreference = _dpref;
  }

  function storeExports(uint _period, uint _exports) external {
    emit ElectricityExported(_period, _exports);
  }

  function storeImports(uint _period, uint _imports) external {
    emit ElectricityExported(_period, _imports);
  }
}
```

Listing 1.1. Listing of Smart Contract for Seller's Data.

Deployment

Interaction with a blockchain is divided into two categories, *calls* - read-only interactions - and *transactions* - potentially state changing interactions. The reason for this distinction is the way each is processed. A call reads directly from a blockchain node and therefore does not require the involvement of the other parties in the network. Instead, a transaction must be broadcast, processed and included inside the distributed ledger. To account for this processing and storage requirement of other nodes in the network, transactions must be signed by an external account. This external account must have sufficient funds to pay for the costs incurred by the network.

Deployment of a smart contract therefore must take place via a transaction to a blockchain network which will record it into a permanent register (the chain) and validate that it concurs with the expected formats and rules (e.g., set by the consensus protocol). As noted before, our experiment was carried out on the Ethereum platform. Our contracts were deployed on the Ropsten test network [4] - an Ethereum test network that replicates the main network but provides free funds to accounts.

MetaMask – a Chrome browser extension – was used to create the external account in this experiment. The new account always contains zero funds balance. However, as noted before, use of the Ropsten network is free, so it provides test websites (https://faucet.metamask.io/ and https://faucet.ropsten.be/) to top up the testing pseudo-funds in the newly created external accounts.

Once the contract is deployed to the network, a contract address (e.g., 0xb0665117611910d3623358148dc6585f90ed7c3c5) will be provided that identifies it on the network.

Data Storage: Using the contract address, we can then interact with the public methods of the deployed smart contract. For example, in the smart contract (Listing 1.1) we could create transactions to supply data to our 'storeExports()' function which would permanently store the data in an accessible manner on the blockchain (in terms of an RDBMS, this corresponds to inserting rows of data into the newly created data structure).

In this experiment, to insert Buyer/Seller Data into the blockchain, we developed a single-page application (SPA, using HTML and JavaScript via React JS and Web3 tools). While inserting seller details (e.g.:"BS81UB", "-2.345672", "50.234098", "10", "12.31", "v10","1"), a new transaction would be created by the blockchain network. This transaction would be placed into a block, along with many other transactions so as the pre-defined size of the block is achieved. The block is then validated (or, in other words, *mined*) by the miners of the network. The validated block will then be inserted into the Blockchain under the contract's account space.

Prior to the transaction processing (mining) a fee would be levied on the contract user for the miners work. The user has to confirm the payment of the fee. In our case, the fee would be charged to the funds of the external account used. Only after confirming the proposed fee charging, and charge collection, would the transaction be inserted into a block, which would then be minded and stored within the block. Fig. 4 shows a number of data insertion transactions for some Seller data. (Note that in Fig. 4 each transaction is part of a discrete block indicated by block number, this is a peculiarity of our design, chosen for clear action traceability for this study; in a more general case many transactions would be grouped into the same block).

Retrieval: as each block in a Blockchain contains a set of discrete transactions, it is not inherently possible to retrieve a sub-set of data from the chain by simply referencing the per-record keys explicitly across several contracts (which is possible in case of RDBMS through per-record foreign keys).

Transactions | Code | Events

Latest 25 txns from a total Of 47 transactions

TxHash	Block	Age	From		To	Value	[TxFee]
0x43b9562b0fe42d6...	3985655	11 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099719
0x0379e6d9d58ed3...	3979262	23 hrs 41 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000098823
0x4ac36af5394a9a8...	3979245	23 hrs 45 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099719
0xeb41330022ec6e...	3979243	23 hrs 45 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099655
0x4a1281e88b0a6c...	3979109	1 day 15 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099591
0x5620a7922571d8...	3979105	1 day 17 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099591
0xe1fece4cc792865...	3979105	1 day 17 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099591
0x9fe5e186b0bc2f7...	3979085	1 day 23 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099655
0x1553d9e3ae401f6...	3979085	1 day 23 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099655
0x327878c60bdb98...	3979085	1 day 23 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099655
0xfa8047e54a37098...	3979073	1 day 26 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099591
0x5dffc45fe6cb26ec...	3979073	1 day 26 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099527
0xd04df6e5f086d20...	3979064	1 day 28 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099527
0xff99b85103782bf7...	3979060	1 day 29 mins ago	0xf3b7c1020d34e82...	IN	0x8e4a0038d59e0a...	0 Ether	0.000099591

Fig. 4. Data Insertion Transactions for a Seller

Data is retrieved from a blockchain using the address of the smart contract in which the data was stored. The method to retrieve this data is then dependant on the type of storage used inside the smart contract. If a smart contract emits events to the network, we can query these events using filters on any indexed parameters (similarly to SELECT statements in SQL). However, these filters are quite limited, with only three indexable parameters per event. Events may also be filtered upon their block number - essentially limiting the search space - but this requires some knowledge of the data stored to be used effectively. If, however, the data is stored directly in the smart contracts storage, this can be accessed directly using calls to read the variables inside the contract.

3 Technological Limitations and Discussion

Having implemented a backend database for a p2p energy trading domain with two alternative technologies, we observe a number of limitations and alternatives, as discussed below.

3.1 Observations on Use of MySQL

MySQL was simple and easy to install and use.

The Management System (MS) in any RDBMS creates a secured application layer around the database. The MS efficiently takes care of role based access on relations, consistency and reliability while inserting and retrieving data and simultaneous request for change-read-update-write operations.

The standard Query Language based on ANSI-SQL provides an efficient and sophisticated way of querying granular data from this database. Additionally, new relationships between existing tables can be defined in an "additive" way, without invalidating previously existing ones.

However, a number of limitations related to the use of the traditional RDBMS also arose:

- Transparency: As the data in the RDBMS is updated or changed, the changes are not visible to the users. This reduces the confidence in the data that is stored in the database.
- Immutability: After the data is inserted into the RDBMS, it can be changed at any time.
- Openness: The RDBMS are not open usually because enterprises own them.
- Data Formats: The data should be engineered (normalised and interlinked via foreign keys) before storing it in RDBMS. This reduces the flexibility. Disparate data sources having different data formats can not use RDBMSs to store the data.

3.2 Observations on Use of Blockchain

Blockchains offer an open, transparent, immutable and distributed platform for storing data, which addresses rather well the above mentioned shortcomings of the RDBMSs. Yet, we also observed a number of limitations of the Blockchain based approach:

- *Querying data from Blockchain*: blockchains don't have any sophisticated data query mechanism to interrogate the data that is stored in them. There are several ways in which a user can get data from a blockchain. Two of these require that the smart contract address is known.
  (a) In the first case "allEvents" quantifier provided by Web3 framework can be used, which returns all the events that are registered with the given Smart Contract since its deployment. Under this option, there is no control over the amount of data that can be obtained from it and on the time it takes to retrieve it. The user must retrieve all data and only then choose the entries relevant to her.
  (b) In the second case, the smart contract specifies data returning functions. This could work well for cases where all data of a particular type is to be returned for every call of this function (e.g., all users registered to this platform). Yet, the flexibility on what can be queried is very limited here (e.g., cannot return only users registered in the current year, unless this specific function is specified prior to the contract deployment). Clearly, inventive solutions could be thought to tackle such lack of flexibility (e.g., instantiate a new smart contract for user registration every year, then annual registrations can be accessed through the annual contract address).
  We have previously noted that if a smart contract emits events to the network, we can query these events using filters on any indexed parameters.

These, however, are all pre-designed solutions, as the deployed smart contracts are immutable, and the data to be returned and indexing to be used must all be set pre-deployment. These options lack flexibility of the general SQL-like queries over RDBMSs.
Other ways to get data require explicit knowledge of its transaction hash or block number. However maintaining the transaction hash or Block number for all transactions on a smart contract is not realistically possible.

- *Data Formats*: a smart contract along with its' functions is to be written per data type. However, data types in Solidity (language for smart contracts used in Ethereum) are limited and could cause difficulties if the data source were to evolve/change. However, some generic types also exist, e.g., "String" type could be used to store Json format, which is also a type of string. Nevertheless, this limited type availability requires careful pre-planning for evolution, which cannot always be foreseen.
- Cost: Our experience suggests that the data storage in a blockchain comes always with an incurred transaction cost. The cost is incurred in any network Public, Private or Consortium - whenever the state of the blockchian is changed via a new transaction, where any data insertion constitutes a new transaction. In our example the, however, publication of the data on generation and consumption, as well as registering as participants on the p2p network does not provide any financial benefits to the participants. It is only the actual trade recording (carried out upon retrial of the previously submitted data) that would lead to financial gains. The parties that publish the row data data may not even prosper from the matching. Thus, we suggest that a more flexible pricing model is necessary.
- Technological barriers: two main versions of Ethereum blockchain implementations are currently available: JavaScript based and GOLang based. This experiment is performed mainly using JavaScript based implementations. The tools that are used in both streams are different. JavaScript uses software packages from web3, where as GoLang has its own packages like solc, abigen etc. As of now work done in one stream is not interchangeable/reusable within the another. While doing this experiment we ourselves had frequent mis-communication as collaborators had backgrounds in different streams. We suggest that better integration of the technology streams would improve both the development experience and the quality of the end result.
- The Maturity of Technology: The Blockchain technology used in this approach Solidity, Web3.js, Ropsten test network, Metamask and Remix Editor are evolving and backward compatibility of features could be an issue.

4 Related Work and Conclusion

Since the emergence of Bitcoin [14] a wide range of research and development activities is under way on using blockchain as a data storage mechanism.

There are a number of areas which relate to the study presented in this paper, some focusing on comparing the utility of blockchain solutions to the centralised

databases, some on working on the unification of both technologies. A small selection of such related work is presented below.

In [16] authors analyse which blockchain properties are relevant to various application domains and problem requirements. They review three domains (supply chain management, inter-bank payments and decentralised autonomous organisations) and argue that various types of blockchain would best suite these various domains.

Greenspan [12] studies differences between private blockchains and SQL databases focusing on such issues as trust building and robustness.

A survey on literature describing scenarios where blockchain is applied is presented in [7]. Here the authors attempted to find what factors (i.e., why and how) contributed to the use of blockchain. By comparing blockchain vs a centralised database for such issues as robustness, fault tolerance, performance and redundancy, the paper proposes a decision tree to check whether the use case in hand is advantaged by using a blockchian solution.

Another area of research is in transferring properties of blockchain to centralised databases and/or database properties to blockchains. For instance , BigchainDB [6] integrates properties of a blockchain into a database, while using an Asset as a key-concept. Asset represents a physical or digital entity which starts its' life in BigchainDB with CREATE Transaction and lives further using TRANSFER Transaction. Each Asset contains Metadata to specify details about the Asset. The Life-cycle of a BigchainDB Transaction is described in [8]. Each node in BigchainDB will have a MongoDb [11] instance and maintains same set of data. The Tendermint [15] is used as consensus protocol, which ensures the data safety even if 1/3 of nodes are down in the network. Traditional database features, like indexing and query support are carried out through the underlying MongoDB.

Hyperledger [5] is a permissioned blockchain system focused on scalability, extensibility and flexibility through modular design. Different consensus algorithm can be configured to this solution (e.g., Kafka, RBFT, Sumeragi, and PoET) while smart contracts [2] can be programmed using platform called Chaincode with Golang [3]. Hyperledger uses Apache Kafka to facilitate private communication channels between the nodes. It can achieve up to 3,500 transactions per second in certain popular deployments. Work to support query [9] and indexing [10] on temporal data in blockchain using Hyperledger is also underway.

In conclusion, we reiterate that this study contributes to better understanding of the issues to be faced when selecting a database solution. We have carried out an exploratory implementation of a particular application (p2p energy trading) using both centralised database and a blockchian-based alternative, and reported the observations emerging from this experience.

References

1. Ethereum project. https://www.ethereum.org/greeter/ (2019 (accessed February 13, 2019))

2. Ethereum white paper: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2019 (accessed February 13, 2019))
3. The go programming language. https://golang.org/ (2019 (accessed February 13, 2019))
4. Testnet ropsten (eth) blockchain explorer. https://ropsten.etherscan.io/ (2019 (accessed February 13, 2019))
5. Androulaki, E., colleagues: Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference. pp. 30:1–30:15. EuroSys '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190538, http://doi.acm.org/10.1145/3190508.3190538
6. BigchainDB: Bigchaindb..the blockchain database. https://www.bigchaindb.com/ (2019 (accessed February 14, 2019))
7. Chowdhury, M.J.M., Colman, A., Kabir, M.A., Han, J., Sarda, P.: Blockchain versus database: A critical analysis. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). pp. 1348–1353 (Aug 2018). https://doi.org/10.1109/TrustCom/BigDataSE.2018.00186
8. Dhameja, G.: Lifecycle of a bigchaindb transaction - the bigchaindb blog. https://blog.bigchaindb.com/lifecycle-of-a-bigchaindb-transaction-c1e34331cbaa (2019 (accessed February 14, 2019))
9. Gupta, H., Hans, S., Aggarwal, K., Mehta, S., Chatterjee, B., Jayachandran, P.: Efficiently processing temporal queries on hyperledger fabric. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 1489–1494 (April 2018). https://doi.org/10.1109/ICDE.2018.00167
10. Gupta, H., Hans, S., Mehta, S., Jayachandran, P.: On building efficient temporal indexes on hyperledger fabric. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 294–301 (July 2018). https://doi.org/10.1109/CLOUD.2018.00044
11. MongoDB: Open source document database — mongodb. https://www.mongodb.com/ (2019 (accessed February 14, 2019))
12. MultiChain: Blockchains vs centralized databases. https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/ (2019 (accessed February 14, 2019))
13. Murkin, J., Chitchyan, R., Ferguson, D.: Goal-based automation of peer-to-peer electricity trading. In: Otjacques, B., Hitzelberger, P., Naumann, S., Wohlgemuth, V. (eds.) From Science to Society. pp. 139–151. Springer International Publishing, Cham (2018)
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
15. Tendermint: Blockchain consensus - tendermint. https://tendermint.com/ (2019 (accessed February 14, 2019))
16. Wust, K., Gervais, A.: Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 45–54 (June 2018). https://doi.org/10.1109/CVCBT.2018.00011