New Approaches to Managing Metadata at Scale in Research Libraries

Timothy A. Thompson
Indiana University Bloomington
School of Informatics, Computing, and Engineering
Bloomington, Indiana 47408
timathom@indiana.edu

ABSTRACT

The analysis of big data often relies on distributed storage and computation; however, access to big data—and to the platforms capable of managing and processing it—continues to be largely centralized. Centralization is particularly evident in the case of the metadata produced, managed, and disseminated by academic and research libraries. Libraries typically create and share their catalog records by uploading them to a centrally managed database, which can then be searched by other libraries for records that can be copied and added to an institution's local catalog. This centralized approach, which operates on the basis of membership fees, has the advantage of scalability and availability, but it comes at the cost of a loss of autonomy. Although technical innovation is possible within the current paradigm, the growing maturity of peer-to-peer protocols and decentralized solutions points toward an alternative approach, one that would allow libraries to share their data directly without having to pay an expensive intermediary.

KEYWORDS

i523, hid-sp18-705, Research Libraries, Library Catalogs, Blockchain, BigchainDB

1 INTRODUCTION

The problem of entity resolution (also known as record linkage or data matching [10]) is one that has a direct impact on the work of information professionals in research libraries. In library units responsible for catalog management, many workflows center on a procedure known as copy cataloging, which aims to expedite the processing of new acquisitions. Copy cataloging involves searching a shared database for records created by another cataloging agency, but that describe identical publications that have been acquired by one's local institution [14]. In the current environment, a single company, the Online Computer Library Center (OCLC—http://www.oclc.org), is the only viable platform for global cooperative cataloging [44]. OCLC provides data aggregation and warehousing services that allow libraries to effectively share their data, but its business model does not encourage peer-to-peer interaction and innovation among individual libraries. This vendor-driven paradigm entails the acceptance of a business model that, in effect, charges libraries for serving their own data back to them, with some added value through quality control and normalization. Once a library's data has been sent to OCLC, it also becomes subject to potential licensing restrictions, as well as the expectation that future dissemination of the data will include attribution of OCLC [35, 37].

2 NEW APPROACHES TO METADATA MANAGEMENT

Libraries have a tradition of experience with record matching and automation [30], but now stand to benefit from the increasingly mainstream availability of algorithms and routines developed within the context of data science and machine learning. Sophisticated algorithms for string comparison and probabilistic record linkage have long been available, but are not widely used by libraries, with the exception of large-scale projects such as the Social Networks and Archival Context Project (SNAC) (http://snaccooperative.org/) and the Virtual International Authority File (VIAF) (http://viaf.org/). The former has employed methods based on Naive Bayes classification algorithms to aggregate and disambiguate data from across a wide range of libraries and archives (the reported accuracy of the approach fell with the range of 80-90 percent) [21]. More recent approaches to record matching have improved on probabilistic methods such as Naive Bayes by using Artificial Neural Networks, improving accuracy rates in some cases to 98 percent or more [39].

As machine learning tools and methods have become more accessible, however, large-scale, real-time access to library metadata has not necessarily followed suit. The catalog of a large academic library may contain around 10 million records [45]. By comparison, as of August 2018, the OCLC catalog database, WorldCat, contained 427,501,671 bibliographic records in 491 languages [36]. As long as service providers such as OCLC maintain centralized control over the aggregated metadata of research libraries, large-scale computational analysis—and the innovation it could produce—will remain proprietary and locked away.

The situation is further complicated by professional and cultural norms within libraries. Although decentralization may be appealing as an ideal, librarians who manage bibliographic metadata are also immersed in a discourse that centers on the idea of control: they use terms such as authority control, controlled vocabularies, and intellectual and physical control of collections [38]. The idea of control is closely related to the idea of trust: when workflows and systems are centralized, it becomes easier to enforce norms and standards, but it also becomes more likely that potential contributors may be excluded, especially when they are unable to afford the price of membership in a proprietary system.

New distributed technologies and protocols, including blockchains and distributed hash tables (DHTs), could allow research libraries to form robust peer-to-peer networks that would enable data sharing on a larger scale. Although public blockchains such as Ethereum and Bitcoin are limited in the amount of data that can feasibly

be stored on chain, alternative platforms that address this limitation have recently emerged. The blockchain-based database service BigchainDB, implemented in Python, provides a robust storage data solution while preserving the benefits of blockchain features such as data immutability and an asset-based transactional model. By running a consortium blockchain network of BigchainDB nodes [9], libraries could be empowered to abandon centralized models and begin managing their data collectively.

3 BLOCKCHAINS FOR RESEARCH LIBRARIES

Some in the library profession have been skeptical of blockchain applications for their domain, arguing that they have been overhyped as a panacea, when in reality they are nothing more than slow, expensive, append-only databases [1]. Even core developers working to support the Bitcoin blockchain have argued that the constraints imposed by blockchain technology, such as immutability and decentralized consensus, make it appropriate for a very limited set of applications—namely, currency and the exchange of value [41]. For individuals and organizations who are investigating blockchains as a technical solution, it is important from the outset to establish a framework for evaluating their applicability and appropriateness [40]. For example, a blockchain-based solution may be appropriate in a scenario in which there is a lack of trust among participants, or in which processes and collaboration would be more efficient if the need for trust were eliminated [40]. In the case of a shared catalog for research libraries, trust is an issue because not all participants can be trusted to provide data that conforms to expected levels of quality. A commercial, centralized solution mitigates these concerns by requiring participants to pay a membership fee. A blockchain solution addresses issues of trust by enforcing a decentralized consensus mechanism, which may take different forms, but which is designed to ensure that participants can trust the network to maintain a consistent state across all transactions [8].

The Proof-of-Stake consensus algorithm, employed by some blockchain networks as an alternative to Bitcoin's resource-intensive Proof-of-Work mechanism, is similar to the membership fee model in that validator nodes are elected based on their share of "stake" in the network, measured by their willingness to commit or stake an allocation of network tokens as a proof of honesty [23]. For research library applications, a variation of Proof-of-Stake known as Proof-of-Authority may be the most appropiate solution [9, 23]. In contrast to public blockchains such as Ethereum and Bitcoin, or fully private blockchains restricted to a single organization, so-called consortium blockchains may be the preferred approach, one in which consensus "is controlled by a pre-selected set of nodes" [9]. The model implemented by the BigchainDB project fits the parameters of a consortium blockchain that implements a Proof-of-Authority approach to consensus [27].

4 DESIGN REQUIREMENTS

A blockchain-based catalog for research libraries should support the creation of a decentralized marketplace for library metadata. Rather than paying a centralized exchange to distribute their catalog records, libraries could buy and sell records in a peer-to-peer exchange. Catalog records could thus become a source of revenue rather than a costly expenditure. Many blockchain systems support the creation of so-called smart assets, or the creation of tokens to represent real-word assets. A new token could be minted to facilitate the exchange of metadata objects, and payment and settlement channels could be created using smart contracts on a public blockchain such as Ethereum. However, a public blockchain solution does not fully satisfy the requirements of decentralization for this use case. A data asset cannot be represented exclusively by a token—it also needs to be stored in a decentralized system optimized for read and write transactions. Public blockchains such as Ethereum have been designed for exchange, not storage. At the current price of the Ethereum blockchain's native token, Ether (ETH), at approximately $200.00, storing 1 Gigabyte of data on the blockchain would cost over $7,000,000.00 [18]. A decentralized system for library metadata must be able to scale and store big data out of the box. BigchainDB is a production-ready solution that meets the requirements for this use case: it supports the creation of assets and the direct storage of metadata objects on its blockchain [5].

5 PROJECT SCOPE

The current project presents findings from an exploration of BigchainDB as a blockchain database solution for a shared library catalog. It includes a preliminary analysis of library metadata requirements and whether they can be satisified using BigchainDB.

6 BIGCHAINDB

6.1 Evolution

BigchainDB was created to address the scalability and storage limitations of traditional blockchains such as Bitcoin and Ethereum and to create a hybrid solution that builds a blockchain layer on top of an existing big data system [7]. Development of the BigchainDB framework initially focused on integration with the RethinkDB system, but now works exclusively with MongoDB [7, 20].

The early focus of BigchainDB development was to create an architecture that would allow existing big data databases to be "blockchainified" [29]. The original BigchainDB whitepaper, released in June 2016, focused on the scalability limitations of traditional blockchain networks such as Bitcoin and claimed that it should be possible to develop a blockchain-based distributed database that would enable "1 million writes per second throughput, storing petabytes of data, and sub-second latency"—in contrast to the storage restrictions and 7 transaction-per-second (tps) limit of the Bitcoin network [29]. The advantages of adding a blockchain layer to an existing distributed database would be to incorporate "decentralized control, immutability, and creation [and] movement of digital assets" [29].

The primary challenge in designing a decentralized system is how to defend against both arbitrary failure and malicious actors. In so-called Sybil attacks, an attacker attempts to generate false identities in order to gain majority control over a network [15]. To address Sybil attacks, BigchainDB proposes a governance model that would create a federation of trusted nodes. Because all participants are known, any attempt by one participant to gain control over the network would be obvious. A more pervasive vulnerability comes in the form of the so-called Byzantine Generals' Problem [29]. Nodes

in a distributed network must be able to reach consensus about the final order of transactions at each state of the system, even in the presence of node failure or malicious attempts to manipulate system state in order to gain an unfair advantage—for example, in double-spending, in which a transaction is replayed so that the same asset can be used again (a particular problem in the case of financial transactions) [2, 29].

In its original design, BigchainDB relied on the consensus algorithm of its underlying database to manage benign node failure and incorporated additional constraints to verify the integrity of the voting process by which nodes in the network approved transactions—and the blocks containing them—as valid [29]. However, in its initial version, BigchainDB did not claim to be Byzantine Fault Tolerant (or BFT—the term used to indicate that a system can withstand unexpected node behavior, whether benign or malicious, up to a certain threshold [29]). In the original design, all nodes belonged to a single logical database. This made the system overly centralized and vulnerable to attack: a malicious actor who gained control over a single node would have been able to drop the entire database, which was shared among all nodes in the network [7, 20]. BigchainDB 2.0, released in June 2018, underwent a complete redesign and incorporated full Byzantine fault tolerance through integration with Tendermint, an application for managing consensus and state machine replication in blockchain systems [25, 42]. As a result of implementing Byzantine fault tolerance through Tendermint, BigchainDB's original goal of supporting 1 million tps was no longer viable.

6.2 Benchmark

A recent benchmark of BigchainDB 2.0 throughput performed by the BigchainDB development team indicated that the system was able to process approximately 300 tps [17, 24]. Benchmarking was carried out on a four-node network on Microsoft Azure-hosted virtual machines located in the same data center. Three separate experiments were run to test different options, and a full report of configurations and results is available for review [17]. The primary experiment tested how long it would take to commit 1 million transactions of 765 bytes each to the BigchainDB blockchain under default settings. Results showed an average rate of 299.0 tps and a median rate of 309.0 tps. All 1 million transactions were finalized in 56 minutes with no failures [17].

6.3 Architecture

The architecture of a BigchainDB 2.0 network is shown in Figure 1, created by the BigchainDB development team. Each node in the network is self-contained and includes its own MongoDB database and Tendermint application server. Tendermint is used to manage consensus, communication, and state replication among nodes, whereas the software that is unique to BigchainDB is responsible for "registering and tracking the ownership of 'assets' " [25]. In BigchainDB 2.0, as is the case in general with systems that are Byzantine Fault Tolerant, $3f + 1$ nodes are necessary to run a network, where f is the number of faulty nodes to be tolerated [28]. Therefore, at least four nodes are required in order to run a BigchainDB network: if one of the four nodes becomes unresponsive or attempts to approve an invalid transaction, the network will continue to function based on the majority consensus of the other three nodes [28].

A BigchainDB client can potentially connect to any node in the network. Each MongoDB instance contains a full replication of the data stored in the network [28]. The BigchainDB project officially supports three client drivers to connect to a node server (in Python, Node.js, and Java) [4].

6.3.1 BigchainDB Server. The BigchainDB Server, written in Python, implements the logic to model, validate, and store transactions in the BigchainDB blockchain [25]. The server also incorporates a Python implementation of the Crypto-Conditions specification, which is a standard for enforcing complex boolean conditions for fulfillment (asset transfer) using cryptographic signatures [43].

All objects in BigchainDB are modeled as *assets*. Two transaction types are available for managing assets: CREATE and TRANSFER [26]. Each transaction must be cryptographically signed with the private key of its "owner" (the agent who created an asset through a CREATE transaction or to whom an asset was assigned through a TRANSFER transaction). Public/private keypairs are implemented using the Edwards-curve Digital Signature Algorithm Ed25519 [26]. A transaction is encoded using a dictionary or associative array that can be serialized as a JSON object. The BigchainDB Transactions Specification defines the structure and usage of a BigchainDB transaction object [26]. Figure 3 shows the key/value pairs that all valid BigchainDB transactions must include.

Conditions for fulfillment and asset transfer are defined in the values of the "inputs" and "outputs" keys. An object representing the asset itself is stored as the value of the "asset" key and cannot be modified once an asset has been created and committed to a block in the BigchainDB blockchain. The "metadata" key is used to store an arbitrary object that records additional information about the asset or its state: in contrast to the asset object, the metadata object *can* be modified with each TRANSFER transaction [26].

6.3.2 Tendermint. Tendermint provides an application interface and BFT consensus algorithm for replicating application state across the nodes in a decentralized network [42]. Tendermint Core implements the consensus algorithm, which ensures that all nodes agree on a single order for transactions. Tendermint's Application Blockchain Interface (ABCI) provides a language-agnostic interface for blockchain applications to use when validating and processing transactions [42].

Figure 2 is a sequence diagram, created by the BigchainDB development team, that illustrates the role of Tendermint in processing BigchainDB transactions. After a client prepares and signs a transaction, typically using a BigchainDB driver, the transaction is submitted to the BigchainDB server for initial validation. The server then sends the transaction to Tendermint, which includes it in a local memory pool. Tendermint returns its own validation request to the server and, upon confirmation, proposes a new block and begins a round of voting as part of its consensus algorithm. Each node in the network votes on the order and validity of transactions in the block, and if consensus is reached, the block is committed to the application's blockchain [7, 13]. BigchainDB stores a queryable copy of each block in MongoDB, while Tendermint appends each

block to its canonical blockchain, which is stored in an internal LevelDB database and used for replicating transaction state to network peers [7, 42].

6.3.3 MongoDB. MongoDB is an enterprise-grade NoSQL database optimized for storing JSON objects as documents. It supports both high availability (replication) and scalability (sharding) [31]. Early verions of BigchainDB used a single MongoDB replication set and were able to take advantage of these core MongoDB features. In BigchainDB 2.0, because each node maintains a separate MongoDB instance, replication and sharding are not supported out of the box [25]. In order to enforce practical immutability and decrease the likelihood of data tampering, the BigchainDB server limits access to MongoDB and does not expose any interfaces for deleting or making arbitary modifications to database documents. Although it is technically possible for a system administrator to modify the MongoDB database directly, each BigchainDB transaction is signed with a public/private cryptographic keypair—thus any tampering would result in a modified signature, which would be detectable by other nodes in the network and would violate its social contract [7].

BigchainDB does take advantage of MongoDB's query facility for read-only queries. It exposes a simple search interface through its HTTP API, but also allows node administrators to create custom indexes and leverage the full range of MongoDB query functionality [6].

7 DATASET

The dataset for this project is intentionally small and meant to test a potential use case for BigchainDB as a library catalog application. Currently, library catalog records are stored in a set of industry-specific formats maintained by the Library of Congress: the MAchine Readable Cataloging (MARC) formats for bibliographic and authority data (standardized as ISO 2709 and ANSI/NISO Z39.2) [16, 33]. In recent years, the Library of Congress has undertaken an effort to update library metadata standards and adopt standards and formats maintained by the World Wide Web Consortium (W3C)—specifically those related to linked data and the Semantic Web, such as the core data model known as the Resource Description Framework (RDF) [11, 22]. A new domain-specific data model and ontology for library metadata, expressed using the W3C's OWL standard for semantic ontologies, is currently being developed and implemented [19]. The data used here for testing with BigchainDB follows this model, known as the Bibliographic Framework Initiative (BIBFRAME) [22].

In the basic model proposed by BIBFRAME, descriptions of library resources are divided into three entity types or classes: Work (the abstract concept of the resource), Instance (the embodiment of a Work in a particular publication), and Item (a physical copy of an Instance) [22]. As an example for this project, a catalog record from the Indiana University Library catalog was chosen. This record describes the Lilly Library's partial copy of the Gutenberg Bible. The data is divided into six files:

```
ocm05084045.xml
gutenberg-iul-item.rdf
gutenberg-iul-instance.json
gutenberg-iul-item.json
gutenberg-iul-record.json
gutenberg-work.json
```

The file `ocm05084045.xml` represents the original MARC-format record, encoded as XML. The file `gutenberg-iul-item.rdf` provides an example of a partial conversion of the original MARC record to the BIBFRAME model using the RDF/XML serialization [32]. The remaining files represent the data used to create assets for storage in BigchainDB and are encoded in BIBFRAME using the JSON-LD serialization of RDF [32]. Several preprocessing steps of data conversion and cleanup were necessary. The original MARC/XML catalog record was converted to BIBFRAME RDF/XML using a suite of XSLT stylesheets provided by the Library of Congress [34]. The RDF/XML documents were then converted to JSON-LD using the Python RDFLib library (see the `convert_rdf.py` script in the `project-code` directory for a brief example). The JSON-LD produced by RDFLib was then broken into separate files to allow for the creation of individual assets in BigchainDB.

8 IMPLEMENTATION

Currently, most large library catalogs are stored in enterprise relational databases such as Oracle. The catalog is one module in a suite of services known as an Integrated Library System (ILS), which also includes modules for circulation and ordering or acquisitions. The cataloging module in an ILS includes a public-facing interface for search and retrieval and a staff-facing interface for data entry and management. One advantage of using a distributed system such as BigchainDB for library cataloging functions would be to allow libraries to share their data more easily with peer institutions. BigchainDB's asset-based data model might also allow libraries to perform inventory and lending functions more efficiently. However, many functional components would need to be considered before determining whether a blockchain platform such as BigchainDB would be appropriate for the library catalog use case. This project focuses on one such component: namely, the management of roles and permissions for data entry.

The Python component of this project implements an extension to BigchainDB that adds support for Role-Based Access Control (RBAC) functionality [12]. The code is based on a Node.js example created by the BigchainDB development team to demonstrate the RBAC extension [3]. Support for RBAC is important for library cataloging because library personnel roles are typically divided between professional librarians (catalogers) and paraprofessional technicians. Librarians are expected to create "original" descriptions of library resources, whereas paraprofessionals are responsible for copying existing data from a shared database such as OCLC WorldCat. Public blockchain systems do not usually impose write restrictions, so support for RBAC is an important consideration when evaluating BigchainDB.

The Python script `rbac_demo.py` connects to a BigchainDB server instance and populates it with the BIBFRAME data described above. All BigchainDB CREATE transactions must include a JSON-serializable object to represent the asset being recorded on the blockchain. The `asset` field of a CREATE transaction takes an object with the required key `data`. The content of the `asset` field is treated as immutable—it cannot be changed once a CREATE transaction has been committed, or when ownership of an asset

is subsequently changed using a TRANSFER transaction. Figure 4 shows how a Work asset might be represented in BigchainDB. Because this data cannot be changed, it makes sense to represent it simply using its RDF type (in this case, it is a BIBFRAME Work with a subtype of Text), as well as a human-readable label. Any BigchainDB transaction may also include an optional `metadata` key that takes as its value an arbitrary JSON object. This flexible design makes it possible to effectively "update" data by using a TRANSFER transaction to indicate that the state of an asset has changed—and recording that change in the metadata object.

The code demo in `rbac_demo.py` creates separate BigchainDB assets for the BIBFRAME types Work, Instance, and Item. The data prepared for this project also illustrates how JSON-LD named graphs may be used to include both descriptive and administrative metadata about the same BigchainDB asset in a single transaction. The `gutenberg-work.json` file comprises two named graphs: one representing the Work entity and one representing a separate Record entity (which is not part of the core BIBFRAME model). Within the named graph for the Record entity, there is an RDF property (`foaf:topic`) that links to the URI for the named graph representing the Work entity. Figure 7 illustrates this pattern, indicating how BigchainDB metadata objects may be used to create internal linkages among assets conforming to the BIBFRAME data model.

9 CONCLUSION

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sodales eleifend pharetra. Phasellus interdum augue nec sapien pretium accumsan. Proin dapibus massa in enim pulvinar facilisis. Proin venenatis nisl metus, nec tincidunt lorem viverra at. Quisque sagittis lectus a mi varius, a auctor tortor dapibus. Nam magna ex, rutrum et mauris et, eleifend iaculis enim. Sed non tortor quis ligula placerat lacinia. Proin consectetur, sapien quis molestie volutpat, velit lacus faucibus quam, id rutrum dui est et eros. Ut fermentum malesuada hendrerit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempus, ante at suscipit facilisis, mauris urna auctor urna, eget pretium mi massa accumsan massa. Duis semper, ex eu rhoncus elementum, est est tristique est, nec tristique orci nunc eget nibh. In vestibulum purus at nibh egestas, eget convallis mi molestie.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sodales eleifend pharetra. Phasellus interdum augue nec sapien pretium accumsan. Proin dapibus massa in enim pulvinar facilisis. Proin venenatis nisl metus, nec tincidunt lorem viverra at. Quisque sagittis lectus a mi varius, a auctor tortor dapibus. Nam magna ex, rutrum et mauris et, eleifend iaculis enim. Sed non tortor quis ligula placerat lacinia. Proin consectetur, sapien quis molestie volutpat, velit lacus faucibus quam, id rutrum dui est et eros. Ut fermentum malesuada hendrerit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempus, ante at suscipit facilisis, mauris urna auctor urna, eget pretium mi massa accumsan massa. Duis semper, ex eu rhoncus elementum, est est tristique est, nec tristique orci nunc eget nibh. In vestibulum purus at nibh egestas, eget convallis mi molestie.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed sodales eleifend pharetra. Phasellus interdum augue nec sapien pretium accumsan. Proin dapibus massa in enim pulvinar facilisis. Proin venenatis nisl metus, nec tincidunt lorem viverra at. Quisque sagittis lectus a mi varius, a auctor tortor dapibus. Nam magna ex, rutrum et mauris et, eleifend iaculis enim. Sed non tortor quis ligula placerat lacinia. Proin consectetur, sapien quis molestie volutpat, velit lacus faucibus quam, id rutrum dui est et eros. Ut fermentum malesuada hendrerit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempus, ante at suscipit facilisis, mauris urna auctor urna, eget pretium mi massa accumsan massa. Duis semper, ex eu rhoncus elementum, est est tristique est, nec tristique orci nunc eget nibh. In vestibulum purus at nibh egestas, eget convallis mi molestie.

Shown in Figure 7. Shown in Figure 8.

[Figure 1 about here.]

[Figure 2 about here.]

[Figure 3 about here.]

[Figure 4 about here.]

[Figure 5 about here.]

[Figure 6 about here.]

[Figure 7 about here.]

[Figure 8 about here.]

ACKNOWLEDGMENTS

The author would like to thank Dr. Gregor von Laszewski and the i523 and i516 teaching assistants for their support and suggestions in writing this report.

REFERENCES

[1] S. Alman. 2018. Blockchain: Apps and Ideas. Blockchains for the Information Profession Website. (Jul 2018). https://ischoolblogs.sjsu.edu/blockchains/blockchain-apps-and-ideas/ Accessed 2018.

[2] A.M. Antonopoulos. 2017. *Mastering Bitcoin: Programming the Open Blockchain* (2 ed.). O'Reilly, Sebastopol, CA, United States.

[3] BigchainDB Contributors. 2018. BigchainDB RBAC Sample. bigchaindb/project-jannowitz GitHub Repository. (Sep 2018). https://github.com/bigchaindb/project-jannowitz/tree/master/rbac Accessed 2018.

[4] BigchainDB Contributors. 2018. Drivers & Tools. BigchainDB Documentation: Revision a62cc4e1. (2018). http://docs.bigchaindb.com/projects/server/en/latest/drivers-clients/index.html Accessed 2018.

[5] BigchainDB Contributors. 2018. Key Concepts of BigchainDB. (2018). https://www.bigchaindb.com/developers/guide/key-concepts-of-bigchaindb/ Accessed 2018.

[6] BigchainDB Contributors. 2018. Querying BigchainDB. BigchainDB Documentation: Revision a62cc4e1. (2018). https://docs.bigchaindb.com/en/latest/query.html Accessed 2018.

[7] BigchainDB GmbH. 2018. *BigchainDB 2.0: The Blockchain Database.* Technical Report. BigchainDB GmbH, Berlin, Germany. 14 pages. https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf Accessed 2018.

[8] E. Buchman, J. Kwon, and Z. Milosevic. 2018. The Latest Gossip on BFT Consensus. (2018). arXiv:cs.DC/1807.04938 https://arxiv.org/abs/1807.04938v2

[9] V. Buterin. 2015. On Public and Private Blockchains. Ethereum Blog. (Aug 2015). https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ Accessed 2018.

[10] P. Christen. 2012. *Data Matching.* Springer, Berlin, Germany.

[11] R. Cyganiak, D. Wood, and M. Lanthaler. 2014. *RDF 1.1 Concepts and Abstract Syntax.* Technical Report. World Wide Web Consortium. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ Accessed 2018.

[12] G. Dhameja. 2017. Role Based Access Control for BigchainDB Assets. BigchainDB Blog. (Sep 2017). https://blog.bigchaindb.com/role-based-access-control-for-bigchaindb-assets-b7cada491997 Accessed 2018.

[13] G. Dhameja. 2018. Lifecycle of a BigchainDB Transaction. BigchainDB Blog. (Aug 2018). https://blog.bigchaindb.com/lifecycle-of-a-bigchaindb-transaction-c1e34331cbaa Accessed 2018.

[14] C. Doran and C. Martin. 2017. Measuring Success in Outsourced Cataloging: A Data-Driven Investigation. *Cataloging & Classification Quarterly* 55, 5 (2017), 307–317. https://doi.org/10.1080/01639374.2017.1317309

[15] J.R. Douceur. 2002. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*. Springer-Verlag, London, United Kingdom, 251–260. http://dl.acm.org/citation.cfm?id=646334.687813

[16] K. M. Ford. 2012. LC's Bibliographic Framework Initiative and the Attractiveness of Linked Data. *ISQ: Information Standards Quarterly* 24, 2/3 (2012), 46–50. http://www.niso.org/publications/isq/2012/v24no2-3/ford/

[17] A. Granzotto, T. McConaghy, and M. Khan. 2018. *Performance Study: Analysis of Transaction Throughput in a BigchainDB Network*. Technical Report. https://github.com/bigchaindb/BEPs/tree/master/23 Accessed 2018.

[18] T. Hess. 2016. [Reply on Ethereum Stack Exchange]. Ethereum Stack Exchange. (Feb 2016). https://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block Accessed 2018.

[19] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph. 2012. *OWL 2 Web Ontology Language Primer*. Technical Report Second Edition. World Wide Web Consortium. https://www.w3.org/TR/2012/REC-owl2-primer-20121211/ Accessed 2018.

[20] killerstorm. 2016. BigchainDB: A Prime Example of Blockchain Bullshit. r/bitcoin. (May 2016). https://www.reddit.com/r/Bitcoin/comments/4j7wjf/bigchaindb_a_prime_example_of_blockchain_bullshit/ Accessed 2018.

[21] R. R. Larson and K. Janakiraman. 2011. Connecting Archival Collections: The Social Networks and Archival Context Project. In *Research and Advanced Technology for Digital Libraries, TPDL 2011*. Springer, Berlin, 3–14. https://doi.org/10.1007/978-3-642-24469-8-3

[22] Library of Congress. 2018. Bibliographic Framework Initiative. (2018). https://www.loc.gov/bibframe/ Accessed 2018.

[23] G. Marin. 2018. Understanding the Value Proposition of Cosmos. Cosmos Medium Blog. (Apr 2018). https://blog.cosmos.network/understanding-the-value-proposition-of-cosmos-ecaef63350d Accessed 2018.

[24] T. McConaghy. 2018. And we're off to the races! BigchainDB Blog. (Sept 2018). https://blog.bigchaindb.com/and-were-off-to-the-races-1aff2b66567c Accessed 2018.

[25] T. McConaghy. 2018. BigchainDB 2.0 is Byzantine Fault Tolerant. BigchainDB Blog. (May 2018). https://blog.bigchaindb.com/bigchaindb-2-0-is-byzantine-fault-tolerant-5ffdac96bc44 Accessed 2018.

[26] T. McConaghy. 2018. *BigchainDB Transactions Spec v2*. Technical Report. https://github.com/bigchaindb/BEPs/blob/master/13/README.md Accessed 2018.

[27] T. McConaghy. 2018. [Reply in bigchaindb/bigchaindb Gitter Chat]. BigchainDB Gitter Chat. (Jun 2018). https://gitter.im/bigchaindb/bigchaindb?at=5b16ac9599fa7f4c0648cc13 Accessed 2018.

[28] T. McConaghy. 2018. [Reply in bigchaindb/bigchaindb Gitter Chat]. BigchainDB Gitter Chat. (May 2018). https://gitter.im/bigchaindb/bigchaindb?at=5b055eaf9ed336150ea41180 Accessed 2018.

[29] T. McConaghy, R. Marques, A. Miller, D. De Jonghe, T.T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto. 2016. *BigchainDB: A Scalable Blockchain Database*. Technical Report. ascribe GmbH, Berlin, Germany. 64 pages. https://github.com/bigchaindb/whitepaper Accessed 2018.

[30] J. McQueen. 1992. Record Matching: Computers Cannot See That Which Is Obvious to "Any Idiot" . . . and Vice Versa. *Information Today* 9, 11 (Dec 1992), 41–44.

[31] MongoDB Contributors. 2018. Introduction to MongoDB. MongoDB Manual. (2018). https://docs.mongodb.com/manual/introduction/ Accessed 2018.

[32] E.L. Morgan. 2013. RDF Serializations. LiAM: Linked Archival Metadata Website. (2013). https://sites.tufts.edu/liam/2014/01/31/serializations/ Accessed 2018.

[33] Network Development and MARC Standards Office, Library of Congress. 2013. The Library of Congress Network Development and MARC Standards Office. (Oct 2013). https://www.loc.gov/marc/ndmso.html Accessed 2018.

[34] Network Development and MARC Standards Office, Library of Congress. 2018. marc2bibframe2. lcnetdev GitHub Repository. (Oct 2018). https://github.com/lcnetdev/marc2bibframe2 Accessed 2018.

[35] OCLC. 2010. WorldCat Rights and Responsibilities. (Jun 2010). https://www.oclc.org/en/worldcat/cooperative-quality/policy.html Accessed 2018.

[36] OCLC. 2018. Inside WorldCat. (2018). https://www.oclc.org/en/worldcat/inside-worldcat.html Accessed 2018.

[37] OCLC. no date. WorldCat Data Licensing. (no date). https://www.oclc.org/content/dam/oclc/worldcat/documents/worldcat-data-licensing.pdf Accessed 2018.

[38] H.A. Olson. 2001. The Power to Name: Representation in Library Catalogs. *Signs* 26, 3 (2001), 639–668. http://www.jstor.org/stable/3175535

[39] O.F. Reyes-Galaviz, W. Pedrycz, Z. He, and N.J. Pizzi. 2017. A Supervised Gradient-Based Learning Algorithm for Optimized Entity Resolution. *Data & Knowledge Engineering* 112 (2017), 106–129. https://doi.org/10.1016/j.datak.2017.10.004

[40] B. A. Scriber. 2018. A Framework for Determining Blockchain Applicability. *IEEE Software* 35, 4 (Jul/Aug 2018), 70–77. https://doi.org/10.1109/MS.2018.2801552

[41] J. Song. 2018. Why Blockchain Is Hard. Cryptocurrency Medium Blog. (May 2018). https://medium.com/@jimmysong/why-blockchain-is-hard-60416ea4c5c Accessed 2018.

[42] Tendermint Contributors. 2018. Tendermint. Tendermint Documentation Website. (2018). https://tendermint.com/docs/ Accessed 2018.

[43] S. Thomas, R. Reginelli, and A. Hope-Bailie. 2017. *Crypto-Conditions*. Technical Report. IETF. https://tools.ietf.org/html/draft-thomas-crypto-conditions-02 Accessed 2018.

[44] A.H. Turner. 2010. OCLC WorldCat as a Cooperative Catalog. *Cataloging & Classification Quarterly* 48, 2-3 (2010), 271–278. https://doi.org/10.1080/01639370903536237 arXiv:https://doi.org/10.1080/01639370903536237

[45] Yale University Library. 2018. YUL Quicksearch Search Results. Yale University Library Quicksearch RSS Feed. (Oct 2018). http://search.library.yale.edu/catalog?commit=Search&format=atom&q=&search_field=all_fields

A CHKTEX

```
Already up to date.
Warning 2 in content.tex line 473: Non-breaking space (`~') should have been used.
the Work entity. Figure \ref{f:rbac} illustrates this pattern, indicating
                                ^
```

List of Figures

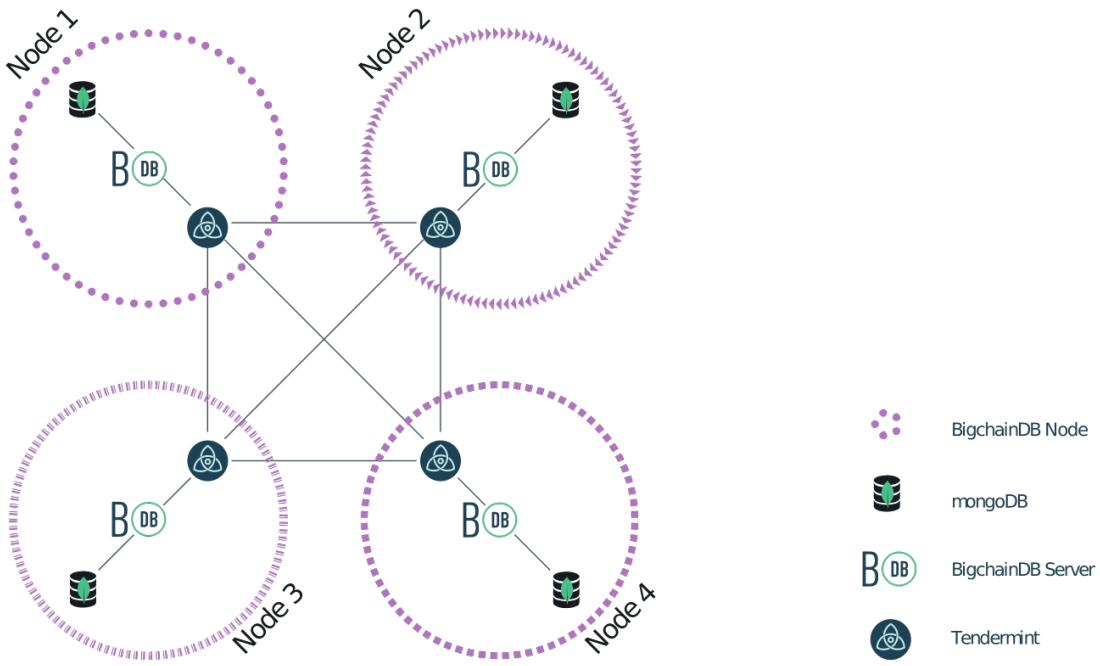

Figure 1: High-Level Architecture of BigchainDB 2.0 [6]

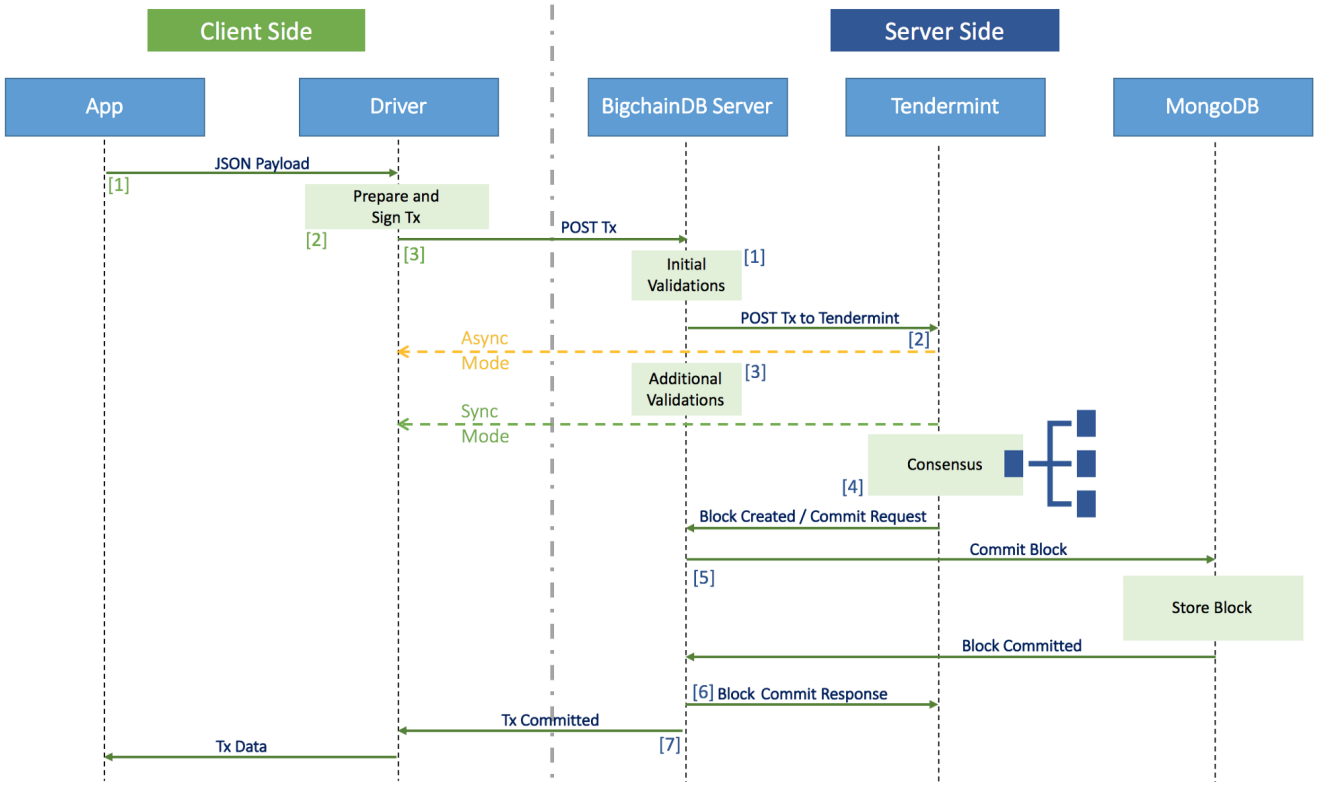

Figure 2: BigchainDB Sequence Diagram [13]

```
{
  "id": ctnull,
  "version": version,
  "inputs": inputs,
  "outputs": outputs,
  "operation": operation,
  "asset": asset,
  "metadata": metadata
}
```

Figure 3: Required key/value pairs in a valid BigchainDB transaction [26]

```
{
  "data": {
    "@context": {
      "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
      "schema": "http://schema.org/"
    },
    "@type": [
      "http://id.loc.gov/ontologies/bibframe/Work",
      "http://id.loc.gov/ontologies/bibframe/Text"
    ],
    "rdfs:label": "Bible. Latin. Vulgate. 1454."
  }
}
```

Figure 4: Representation of a Work asset in BigchainDB

```
<rdf:RDF xmlns:bf="http://id.loc.gov/ontologies/bibframe/"
  xmlns:bflc="http://id.loc.gov/ontologies/bflc/"
  xmlns:madsrdf="http://www.loc.gov/mads/rdf/v1#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <bf:Item rdf:about="http://example.org/ocm05084045#Item050-8">
    <bf:shelfMark>
      <bf:ShelfMarkLcc>
        <rdfs:label>BS75 1454</rdfs:label>
      </bf:ShelfMarkLcc>
    </bf:shelfMark>
    <bf:heldBy
      rdf:resource="http://id.loc.gov/vocabulary/organizations/inuli"/>
  </bf:Item>
</rdf:RDF>
```

Figure 5: Excerpt of Indiana University Library catalog record converted to RDF/XML

```
[
  {
    "@id": "http://example.org/ocm05084045#Item050-8",
    "@type": ["http://id.loc.gov/ontologies/bibframe/Item"],
    "http://id.loc.gov/ontologies/bibframe/heldBy": [{
      "@id": "http://id.loc.gov/vocabulary/organizations/inuli"
    }],
    "http://id.loc.gov/ontologies/bibframe/itemOf": [{
      "@id": "http://example.org/ocm05084045#Instance"
    }],
    "http://id.loc.gov/ontologies/bibframe/shelfMark": [{
      "@id": "_:Nb346817b2a894e1894d7c37a823ed84b"
    }]
  },
  {
    "@id": "_:Nb346817b2a894e1894d7c37a823ed84b",
    "@type": [
      "http://id.loc.gov/ontologies/bibframe/ShelfMarkLcc"
    ],
    "http://www.w3.org/2000/01/rdf-schema#label": [{
      "@value": "BS75 1454"
    }]
  }
]
```


Figure 6: Excerpt of Indiana University Library catalog record converted from RDF/XML to JSON-LD using the RDFLib library

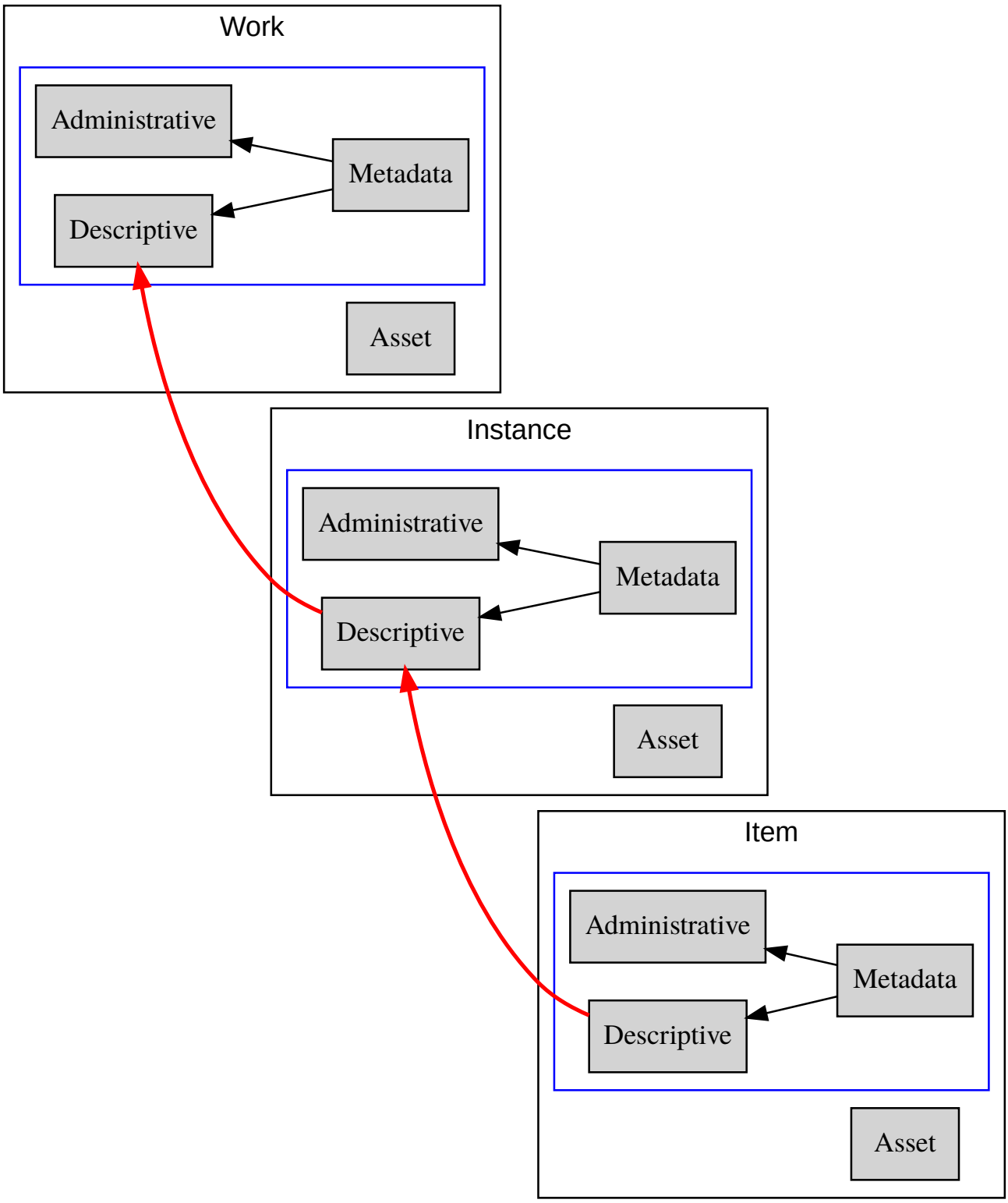

Figure 7: Graph of asset and metadata objects in BigchainDB

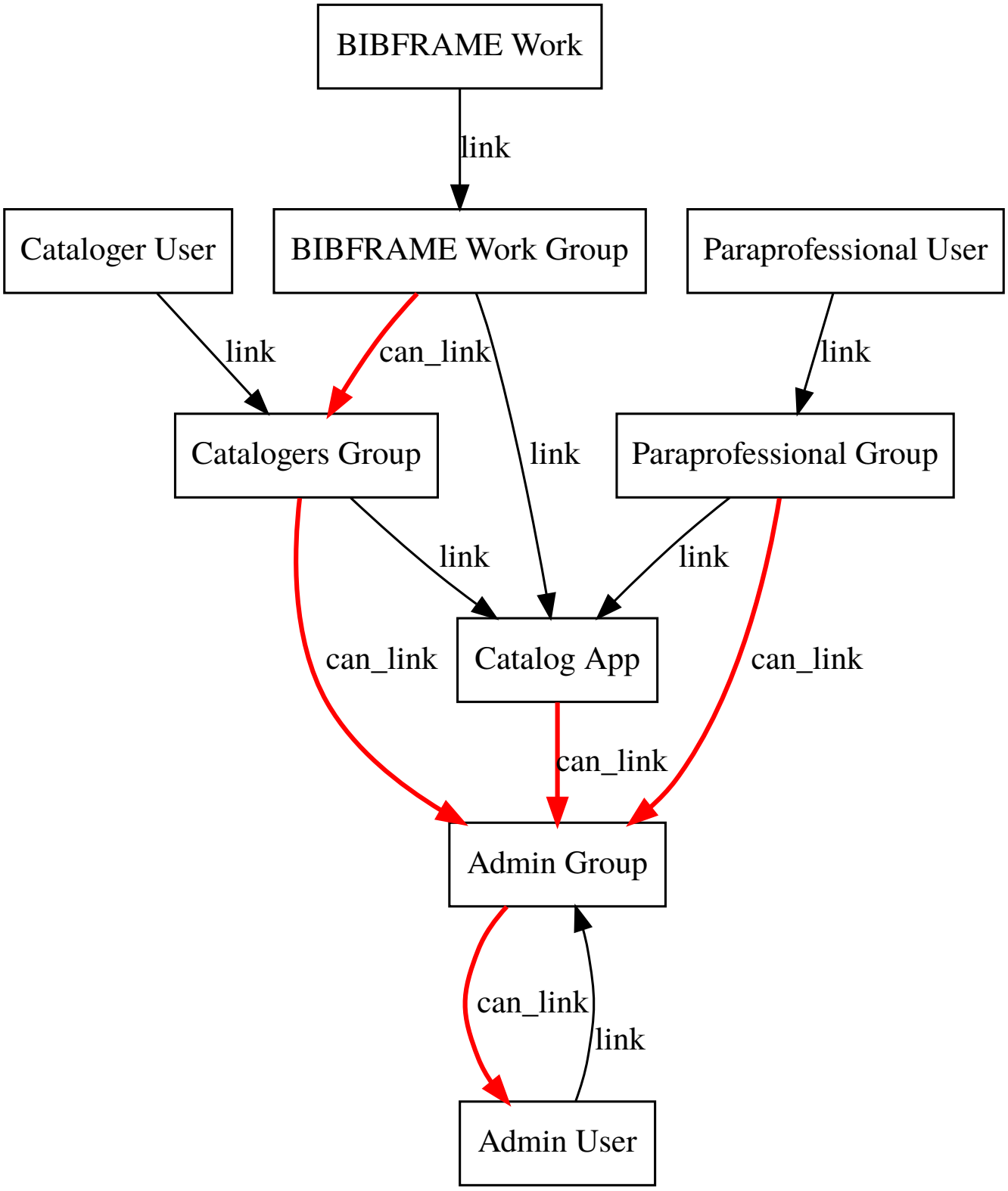

Figure 8: Graph of permissions in BigchainDB using Role-Based Access Control