Benchmarking LinkChains: Performance of Blockchain-backed Trustworthy Linked Data

Allan Third and Kevin Quick and Michelle Bachler and John Domingue

Knowledge Media Institute, Open University, Milton Keynes, MK7 6AA, UK
{allan.third,kevin.quick,michelle.bachler,john.domingue}
@open.ac.uk

Keywords: Knowledge graphs, Linked Data, provenance, blockchains, distributed ledgers

Abstract. Certain uses of knowledge graphs require strong guarantees of data integrity – for example, medical or financial applications – which can be verified by the end user automatically and reliably. Distributed ledger technologies based on blockchains are able to provide a trustworthy guarantee that records have maintained their integrity since publication. There are multiple different scenarios for providing such guarantees, varying in terms of where data is stored – from traditional quad stores through distributed file systems to directly on a blockchain itself – and the strength and scope of the integrity guarantee which can be offered – with or without reliable timestamping, and at the level of whole datasets or, novelly, of query results. In this paper we present the results of a number of experiments across these dimensions. In particular we use the lightweight Linked Data Fragments (LDF) standard as a frontend to each experimental setup, and benchmark their performance relative to a standard LDF server using a SPARQL backend, to evaluate each for performance as candidates to implement a trusted Semantic Web.

1 Introduction

Blockchain technology provides a possible means of securing any data, not just cryptocurrency transactions, against *post-hoc* tampering. In [20], we presented a range of scenarios using distributed and blockchain approaches to create "LinkChains" – decentralised sources of Linked Data which can guarantee the integrity of their contents. Here we describe the results of testing the performance of these scenarios across different sizes of dataset.

When querying any knowledge or data source, there is an element of trust placed in the host(s): one relies on a knowledge store to be provide the content it purports to accurately and consistently. In certain use cases, such as medical or financial applications based on knowledge graphs, it would be beneficial to provide a technical guarantee of that reliability where possible. Scientific data too can benefit from some certification of data integrity or lack of tampering – [12] illustrates the risks of potential political influence on science, against which a redundantly-replicated trustworthy data store could defend.

Clearly, trustworthiness of content is generally a matter for human judgement; what we are specifically addressing here is data *integrity* and *timestamping*, which together can serve to provide assurances that data can be preserved intact without tampering or corruption *after* content accuracy is validated by humans.

We evaluate implementations of Linked Data Fragments (LDF) knowledge bases [1] which make use of distributed technology to store knowledge graphs in a way which provides a verifiable source of data. It might be expected that a knowledge base using a blockchain would be slower to respond to queries compared to standard knowledge bases, much as we might expect a knowledge base using distributed storage would also be slower. It is unknown what effect various methods of computing a guarantee of integrity might have. We present the results of a number of experiments in query performance in a variety of scenarios displaying different degrees of data distribution and different forms of integrity guarantee. We use the lightweight Linked Data Fragments (LDF) standard as a frontend to each experimental setup, and benchmark their performance relative to a standard LDF server using a SPARQL backend, to evaluate each for performance as candidates to implement a trusted Semantic Web – that is to say, a decentralised and distributed Semantic Web with built-in support for trust in data integrity and timestamping. We also introduce the notion of data integrity checking at the level of query results, as opposed to at the level of an entire dataset.

2 Knowledge Graphs

One of the cornerstones of the Semantic Web is the ability to represent data *semantically* – such that the *meaning* of a piece of data can be read, by human or machine, from the data itself and not from its position in a structure such as a relational table. The most commonly used standard is the Resource Description Framework (RDF) [26], in which data are represented as triples – semantically, “subject predicate object” sentences – or quads – “graph subject predicate object” sentences – where graphs serve to group triples. In both cases, subject and predicate (and graph, where present) are URIs, and objects may be either URIs or (optionally typed) literal values, e.g., numbers or strings. RDF has a number of concrete representation formats such as RDF/XML [6], Turtle [7], NTriples [5] and NQuads [10], JSON-LD [18], and so on, but each implement the same semantic model.

Querying of Linked Data is designed around the idea that data from multiple sources can be linked easily no matter where it comes from, and without the need for coordination among data publishers beyond the use of common vocabularies and ontologies for common concepts. This model allows simple data integration and on-the-fly construction of relevant knowledge graphs from multiple sources by querying.

The dominant standard for querying Linked Data is SPARQL [25]. A data source publishes a SPARQL endpoint, an HTTP-accessible interface which responds to queries in the SPARQL language, a query language which is somewhat similar to SQL [13], with SELECT, DELETE and INSERT operations (among others), and a query pattern syntax adapted to the RDF data model. It supports sophisticated filtering and aggregation, and, crucially, via the SERVICE keyword, the ability to pass subqueries to other SPARQL endpoints to achieve federated querying – that is, querying of multiple data sources at once.

The downside of SPARQL is its comparative weight on the server side, and its complexity. Many of the advanced filtering and aggregation features can be computationally expensive for the endpoint server executing them, and the implementation of SERVICE-based federated querying requires all communication between other data sources to be carried out by the initially targeted endpoint.

To address these issues, the Linked Data Fragments (LDF) approach to querying has recently been proposed [1]. A data source for an LDF query may be a SPARQL endpoint, but can be any source which provides valid RDF. Currently the only query patterns supported are Triple Pattern Fragments (TPF), although there is scope for extension. The basic query structure it handles is the *Triple Pattern*, which is an RDF triple – subject predicate object – but where one or more terms are replaced by *variables* in the form of alphanumeric strings beginning with a question mark. So where

```
:RV_193 risk:has_risk_evidence_source cite:CI_37 .
:RV_193 risk:has_risk_evidence_ratio_value
                        "1.22"^^xsd:decimal .
:RV_193 risk:has_confidence_interval_max
                        "1.41"^^xsd:decimal .
:RV_11 risk:has_risk_evidence_source cite:CI_8.
:RV_11 risk:has_risk_evidence_ratio_value
                        "4"^^xsd:decimal .
:RV_11 risk:has_confidence_interval_max
                        "2.8"^^xsd:decimal .
```

are triples,

```
?rv risk:has_risk_evidence_source ?citation .
```

is a triple pattern, matching the first and fourth triples above by successively binding the variable `?rv` to `carre:RV_193` and `carre:RV_11` and `?citation` to `cite:CI_37` and `cite:CI_8`, respectively. The result of evaluating a Triple Pattern on a particular dataset is the set of all triples which occur in the dataset which match it – that is, a triple matches a pattern if, when each variable in the pattern is consistently replaced with a term occurring in the dataset, the resulting triple is present. Triple patterns are the basis of Basic Graph Patterns – multiple triple patterns interpreted conjunctively with shared variable bindings – which are a fundamental component of full SPARQL queries.

A TPF server is a lightweight node which returns sets of triples in response to patterns. A Linked Data Fragments client can be configured with multiple TPF (or future LDF) data sources, including SPARQL sources, and can execute

SPARQL queries in the client by retrieving the results of triple pattern queries from each data source and performing the more complex parts of querying (joins, for example) on the results in the client, sparing server resources.

Regardless of how a knowledge graph is stored or queried, the client places a certain amount of trust in the source to report the contents accurately without modification, either from technical issues such as data corruption or from social/financial/political/personal interests on behalf of the data sources. In contexts such as medicine, accurate *timestamping* of data can also be important – if an application recommends a particular clinical choice based on the best available knowledge at the time of recommendation, it matters in terms of accountability to be able to reconstruct that knowledge correctly and verifiably.

Alterations to data can occur for a variety of reasons: everyday updates by data authors, corruption or, potentially, deliberate data tampering or removal. For some use cases, this mutability is an advantage, if it is important always to have the most up-to-date data. For others, however, the opposite is true; it can be important to know that data has not been modified, or that it corresponds to a particular version of the dataset. It would be beneficial to find a way to guarantee integrity while maintaining the decentralised nature of Linked Data.

3 Distributed Ledgers and Blockchains

A distributed ledger is a shared datastore to which multiple parties may append data without a requirement for mutual trust, but in such a way that added data cannot later be edited, corrupted or modified. The most common means of implementing a distributed ledger is via a *blockchain*. The most prominent blockchain, at the time of writing, is Bitcoin [15], whose associated cryptocurrency can be securely spent with potential fraud such as double-spending prevented by the blockchain.

The integrity of a blockchain stems from the network on which it runs. Typically anyone may add a node to a blockchain network and every node has a full copy of the chain. Any node may verify the authenticity of any transaction on the chain at any time. The blockchain itself is a temporally-ordered list of *blocks* (collections of *transactions*) beginning with a "genesis block". Pending data append transactions are grouped into blocks and appended to the chain by a "miner". The right to "mine" the next block is awarded to nodes by a consensus mechanism: roughly, the whole network must agree on which node adds the next block, and no node should be generally privileged over any other. This design prevents any one agent having undue influence on the contents of the chain.

There are different ways to implement the consensus mechanism, with the most common being "proof of work", where nodes must demonstrate the solution to a hard computational problem. The main requirements are that there be a cost to adding a new block, and an incentive to resolve forks quickly according to a consensus. Once a block has been added to a chain, anyone wishing to rewrite its contents must convince the network as a whole to agree; the further back along the chain a block is, the harder and more expensive it is to do so. Provided a blockchain network is sufficiently diverse (no more than 50% of all nodes owned or controlled by a single entity or cartel), the contents of a blockchain are secure from malicious alterations, and blocks a sufficient distance back along the chain from the most recent block may be regarded as effectively immutable (see [9]) and containing transactions which the network as a whole regards as having really happened.

Blockchains make heavy use of cryptographic hash functions; one way functions which convert arbitrarily-sized data to strings of fixed length such that very similar inputs produce very different outputs, and, for practical purposes, it is very unlikely that two different inputs can produce the same hash. A hash therefore provides a fixed-size signature of a particular piece of data.

Bitcoin supports a scripting language, with the available scripts which can be executed on the platform being whitelisted for security. Newer distributed ledger platforms such as Ethereum [27] have developed scripting capabilities further, in the form of *smart contracts*. Any user of Ethereum may deploy a smart contract onto the platform, and anyone with a contract's deployed address and a description of its programming interface may invoke it by creating a transaction with the contract's address as recipient. On execution, every node on the Ethereum platform will run the contract code with the given inputs. Smart contracts are as difficult to edit, retroactively, as any other existing data on a blockchain, and so, given the source code and a means to compile it (relatively easy to do), it is possible to verify that a particular smart contract does indeed implement a particular behaviour and therefore its output can be trusted as living up to that behaviour. Smart contracts therefore enable trustworthy distributed computation. Ethereum runs smart contracts using its own virtual machine running bytecode compiled from languages such as Solidity [2] and Serpent [3], which are, respectively, Javascript-like and Python-like languages with built-in support for blockchain primitives, such as transactions, accounts and hash functions.

These languages are Turing-complete, and so we know that it is impossible to prove that an arbitrary contract will terminate. Computation involves resources, and to encourage well-behaved contracts, Ethereum imposes a cost per significant step of computation; the invoker must provide a sum of money on invocation which sets a bound on execution, which stops when the money is exhausted. Unused money is returned. In Ethereum, this money is called "gas". Some operations (such as reading the value of a variable) are free, others (such as storing data) cost gas.

Interfaces between distributed ledgers and the Semantic Web are in their infancy. FlexLedger [19] describes generic HTTP interfaces to blockchains, with a vocabulary implicit in the standardised names and responses of these interfaces. BLONDiE [24] and EthOn [16] are formalisations of blockchain concepts as ontologies, generic across Bitcoin and Ethereum blockchains and specific to Ethereum, respectively. [21] and [23] discuss initial approaches to Linked Data indexing of blockchains, and the certification of Linked Data temporal streams on blockchains, but both are very preliminary.

This paper seeks to advance the subject of Linked Data and distributed ledgers by addressing the ways in which distributed ledgers can be used to certify the integrity of Linked Data sets.

4 Data integrity

How can we be sure that data have not been lost, corrupted or tampered with, either in data storage and querying, and that data can be associated with particular versions or publication times of a dataset? The authors have previously worked on the CARRE Linked Data system for medical decision support [22], where it is important not only to know that the data representing medical knowledge are not modified or corrupted, but also, for legal accountability reasons, to know precisely what the contents of the relevant knowledge base was at the time any particular decision was recommended. In other words, there is a need for an immutable and trustworthy means of storing and querying data. Other domains to which these requirements can apply include safety-critical engineering and financial auditing and accounting.

Distributed ledger technologies with smart contracts have the potential to provide immutable, trustworthy data storage and querying. As noted earlier, trustworthiness has a human evaluation component, based on domain expertise; we do not claim an automated solution to the task of verifying this attribute. However, alone, it is not sufficient for a knowledge base to be accredited by human experts, as a variety of processes, from failures in data replication and storage to politically or financially motivated interference can lead to datasets being altered or corrupted after accreditation, affecting knowledge base reliability. One can protect against this by, for example, publishing a hash of a knowledge base of interest so that its integrity can be checked. However, while potentially a protection against corruption in data transmission, this method is vulnerable to alteration of the published hash; if an attacker can replace the hash, data can be modified without detection by this method.

5 Verifiable distributed Linked Data scenarios

[20] described scenarios based on the degree to which Linked Data storage and querying could be distributed, and on the approach taken to certify its data integrity. We consider broadly three options for data distribution: not distributed (i.e., data is in a single triple store), distributed via a content-addressable filesystem such as IPFS [8], and distributed by storing data directly in a blockchain smart contract. The latter two cases work differently; IPFS is a "copy-on-demand" network which allows files to be addressed by hashes of their contents, whereas a smart contract is accessed via its address. Both are in some sense immutable. If an IPFS file is changed, its hash changes, and thus becomes effectively a new file – anyone accessing the original hash will still receive the original data. A smart contract can be made non-writable after data is loaded, with the blockchain providing immutable timestamps for every data entry event

prior to that. Only blockchains guarantee data non-deletion; an IPFS file can be fully deleted if its original host deletes it and network-cached copies expire.

There are four options with regard to guarantees of data integrity: no guarantee, a guarantee relying on a hash of a dataset serving as an IPFS address, a similar guarantee with a dataset hash on a blockchain, and the guarantee provided by storing an entire dataset in a blockchain smart contract. Of course, for hash computation to be useful, data must be transformed into a normalised format, as even minor variations in input produce large variations in hash string.

By combining these options in different relevant ways, we get the following scenarios, each of which has been implemented behind an LDF server frontend, for consistency of evaluation.

5.1 Base case (no distribution, no integrity guarantee)

A standard triple store with a SPARQL endpoint, which an LDF server instance is configured to use as a backend. To provide a useful comparison for 5.2 and 5.3, we also consider the even simpler case in which an LDF server is configured to query a single file containing a dataset. In the discussion below, we refer to the SPARQL case as “Base” and the latter case as “File”. Both are helpful comparators, but we believe that the SPARQL scenario represents the more common normal case.

5.2 IPFS (on-demand distribution, hash-based guarantee with no timestamp)

A Turtle or JSON-LD file (or other format) containing a dataset is stored on IPFS and accessed via its hash. The hash itself must be stored externally and securely; any tampering with hash storage would affect the ability to check data integrity. This integrity guarantee is provided at the level of the *dataset*: either the entire dataset is known to be unmodified, or none of it is. Unverified datasets must therefore cause an error to be returned by the query engine.

5.3 IPFS+Blockchain (on-demand distribution, hash-based guarantee with immutable timestamp)

As above, but the hash is stored in a smart contract to provide immutability and reliable timestamping. The integrity guarantee is at the dataset level.

5.4 Base+Blockchain (blockchain distribution, triple level guarantee with immutable timestamp)

As in the base case, but the whole dataset is duplicated into a smart contract datastore, permitting the checking of individual triples. Query results from the base triple store (which in the case of an LDF Triple Pattern Fragments query are sets of triples) can therefore be checked individually against the contract to ensure integrity. Note that this scenario provides guarantees at the *triple* level; a query which returns results present in the triple store but not in the contract can still return partial results, provided they were accompanied by appropriate metadata.

5.5 Blockchain storage (blockchain distribution, triple level guarantee with immutable timestamp)

This scenario uses the same contract backend as the preceding one, but with no standard triple store. Instead, the LDF server is modified to retrieve results directly from the contract, avoiding the need to manage a separate triple store.

6 Evaluation

Datasets were created by selection of random subsets from DBpedia, containing 20000, 40000, 60000, 100000 and 1000000 triples, such that each larger dataset contains each smaller one. For each dataset, and for each position of subject, predicate and object within a triple, the frequency of occurrence of each distinct term was computed in order to select terms of high, medium or low frequency to use to build a suitable test set of queries, covering every combination of triple pattern containing $0-3$ variable terms, with each combination of high, medium or low frequency terms in each position.

From the test query sets, each query was executed against each dataset in turn, on the same server and with the same software environment. The execution time was recorded, alongside any errors and the number of results generated (as a correctness check). Each test was run five times, and the results averaged, to take account of variations in system background loads. Table 1 shows the mean execution times per scenario per dataset size alongside the overall mean execution times per scenario.

	20000	40000	60000	100000	1000000	All
Base	0.03	0.08	0.04	0.04	0.03	0.04
File	0.04	0.05	0.07	0.09	0.87	0.28
IPFS	0.04	0.05	0.07	0.09	1.04	0.33
IPFS+BC	0.03	0.05	0.07	0.09	1.08	0.34
Base+BC	0.52	0.50	0.36	0.39	0.34	0.42
BC	4.22	5.76	7.29	70.07	5960.37	1045.72

Table 1. Mean execution times in seconds for each scenario per dataset size, ordered by overall time

To verify which of the observed differences are significant, we applied the Shapiro-Wilk test [17] for normality, which showed that the results were not normally distributed, and therefore suggested the need for a non-parametric

Dataset size	χ^2	p
20000	79.796	3.936×10^{15}
40000	101.93	$<2.2\times10^{16}$
60000	95.896	$<2.2\times10^{16}$
100000	149.77	$<2.2\times10^{16}$
1000000	85.501	2.6×10^{16}
all	396.72	2.2×10^{16}

Table 2. Kruskal-Wallis results - Size vs. statistics

analysis of variance such as a Kruskal-Wallis test [14]. This showed, as can be seen in Table 2, for each case, that some of the observed means indeed differed significantly from each other, with $p < 0.001$ in each case . *Post-hoc* analysis using Dunn's test [11] showed the results in Table 3, which suggest a number of conclusions. Most obviously, the blockchain storage scenario performs significantly and considerably worse than all other scenarios at all dataset sizes, with average execution times at least two orders of magnitude greater than those for other methods, and up to 5 orders of magnitude difference. As expected, the base case was by far the fastest scenario, though it makes no attempt at data verification, with flat-file formats grouping together at similar execution times, even with the addition of distribution (IPFS) and blockchain-based hash verification, for smaller dataset sizes, but with execution time increasing with dataset size. This is presumably because the query engine needs to read a whole file in order to carry out a query in these cases. The addition of blockchain-based hash verification to the IPFS cases does not in any case seem to have noticably impacted performance.

The main surprising result, however, is that while the base case+blockchain scenario can be significantly different in overall performance' from the IPFS scenarios, the size of the difference is, overall, comparatively small (at worst, one order of magnitude). Given the apparent consensus that blockchains are not good for actual data storage, this result appears to cast some doubt on that belief. Considering across sizes of dataset illuminates this result: at larger dataset sizes (e.g., 1000000 triples), the base case+blockchain case is in fact significantly faster than the IPFS scenarios. It seems likely that this reflects the dual advantages both of the efficiencies of a triple store and the availability of triple-level verification. Querying a file stored on IPFS requires the whole dataset ultimately to be read into memory, and verification involves computing the hash of the entire file. A triple store, by contrast, can be more sophisticated in data access than file methods, and verification involves simply checking the *results* of a query, rather than the full dataset. The novelty here is not in the notion that a database can outperform a file, which is well-known, but in the use of blockchains to permit this query-level verification. Further experiments and analysis are needed to separate these two contributing factors and to test the limits of large dataset sizes usable with the blockchain verification model.

	20000		40000		60000		100000		1000000		all	
Comparison	Z	p(adj.)	Z	p(adj.)	Z	p(adj.)	Z	p(adj.)	Z	p(adj.)	Z	p(adj.)
IPFS and IPFS+BC	0.40	1.00	-0.52	1.00	-0.15	0.88	-0.00	1.00	-0.44	1.00	-0.28	1.00
IPFS and BC	**-6.87**	**0.00**	**-8.24**	**0**	**-7.24**	**0**	**-7.30**	**0**	**-6.46**	**0**	-15.43	**0**
IPFS+BC and BC	**-7.27**	**0.00**	**-7.71**	**0**	**-7.09**	**0**	**-7.30**	**0**	**-6.09**	**0**	**-15.16**	**0**
IPFS and Base	-0.94	1.00	-2.09	0.40	0.75	1.00	0.81	1.00	3.06	0.03	1.39	1.00
IPFS+BC and Base	-1.34	1.00	-1.56	1.00	0.89	1.00	0.81	1.00	**3.50**	**0.01**	1.67	0.95
BC and Base	**5.93**	**0.00**	**6.15**	**0**	**7.99**	**0**	**8.11**	**0**	**9.05**	**0**	**16.77**	**0**
IPFS and Base+BC	-2.98	0.04	**-3.37**	**0.01**	-1.95	0.62	-1.91	0.73	0.64	1.00	**-4.00**	**0**
IPFS+BC and Base+BC	**-3.38**	**0.01**	-2.85	0.06	-1.80	0.79	-1.91	0.68	1.08	1.00	**-3.72**	**0**
BC and Base+BC	**3.89**	**0.00**	**4.87**	**0**	**5.29**	**0**	**5.39**	**0**	**7.01**	**0**	**11.56**	**0**
Base and Base+BC	-2.04	0.45	-1.29	1.00	-2.70	0.11	-2.72	0.10	-2.42	0.17	**-5.39**	**0**
IPFS and File	-0.23	1.00	-0.44	1.00	0.16	1.00	-0.10	1.00	0.52	1.00	-0.11	1.00
IPFS+BC and File	-0.63	1.00	0.09	0.93	0.31	1.00	-0.10	1.00	0.95	1.00	0.17	1.00
BC and File	**6.64**	**0.00**	**7.80**	**0**	**7.40**	**0**	**7.19**	**0**	**6.90**	**0**	**15.32**	**0**
Base and File	0.71	1.00	1.65	0.99	-0.58	1.00	-0.91	1.00	-2.55	0.13	-1.50	1.00
Base+BC and File	2.75	0.08	2.94	0.05	2.11	0.45	1.81	0.78	-0.13	0.90	**3.89**	**0**

Table 3. Dunn's test statistics, showing which pairs of scenarios are significantly different ($p <= 0.01$, highlighted in bold) from each other. A scenario pair is highlighted if a difference was seen at any size of dataset. BC=Blockchain, File=dataset in a single (local) file.

7 Conclusion and Future Work

We have presented a performance evaluation of a range of scenarios for storing knowledge graph data in the form of RDF in a trustworthy and verifiable manner, making use of content-addressable filesystems and distributed ledgers based on blockchains, to provide a means of offering integrity guarantees which can be verified by any user of the data.

The scenarios are implemented as proof-of-concept via the interoperable Triple Pattern Fragments/Linked Data Fragments standards, which permit Linked Data to be queried via the same method as standard RDF stores.

The evaluation showed that, while performing both querying and data storage entirely using a blockchain datastore is impractical, it is not necessary to use a solely hash-based approach to verifying *datasets* in order to achieve a performant system. *Storing* data on a blockchain for verification, while carrying out querying using a standard triple store, can offer a comparable performance to hash-based approaches on distributed storage on smaller datasets, and a better performance on larger datasets. This scenario has the added advantage of permitting partially-verified query results; a system can choose to return only results known to be verifiably unmodified even if the contents of the standard triple store have been compromised. Of course, since there have so far been no means of enabling integrity verification at the level of query results, current query standards do not support the communication of such detailed status to querying clients. In future work, we intend to address this lack, and embed the notion of data integrity and (blockchain) verification more deeply into the LDF and SPARQL querying processes. There is also, as mentioned above, a need for a deeper analysis of the factors affecting performance of blockchain-based query systems.

The software, data and analysis scripts described here are available online at [4]

References

1. (June 2016), https://www.hydra-cg.com/spec/latest/linked-data-fragments/
2. (May 2017), https://solidity.readthedocs.io
3. (May 2017), https://github.com/ethereum/serpent
4. (2018), http://blockchain7.kmi.open.ac.uk/eswc2018-rdf-speedtest
5. Beckett, D.: RDF 1.1 NTriples. W3C recommendation (2014)
6. Beckett, D., McBride, B.: RDF/XML syntax specification (revised). W3C recommendation 10 (2004)
7. Beckett, D., Berners-Lee, T., Prud?hommeaux, E.: Turtle-terse RDF triple language. W3C Team Submission 14, 7 (2008)
8. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
9. Buterin, V.: http://ethereum.stackexchange.com/a/203 (2016)
10. Carothers, G.: RDF 1.1 NQuads. W3C recommendation (2014)
11. Dunn, O.J.: Multiple comparisons using rank sums. Technometrics 6(3), 241–252 (1964)

12. Eilperin, J.: Under Trump, inconvenient data is being sidelined (2017), https://www.washingtonpost.com/politics/under-trump-inconvenient-data-is-being-sidelined/2017/05/14/3ae22c28-3106-11e7-8674-437ddb6e813e_story.html
13. Groff, J., Weinberg, P.: SQL The Complete Reference, 3rd Edition. McGraw-Hill, Inc., New York, NY, USA, 3 edn. (2010)
14. Kruskal, W.H., Wallis, W.A.: Use of ranks in one-criterion variance analysis. Journal of the American statistical Association 47(260), 583–621 (1952)
15. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
16. Pfeffer, J., Beregszazi, A., Detrio, C., Junge, H., Chow, J., Oancea, M., Pietrzak, M., Khatchadourian, S., Bertolo, S.: EthOn - an Ethereum ontology. https://consensys.github.io/EthOn/EthOn_spec.html (2016)
17. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika 52(3/4), 591–611 (1965)
18. Sporny, M., Kellogg, G., Lanthaler, M., Group, W.R.W., et al.: JSON-LD 1.0: a JSON-based serialization for linked data. W3C Recommendation 16 (2014)
19. Sporny, M., Longley, D.: Flex Ledger 1.0. W3C Blockchain Community Group (2016)
20. Third, A., Domingue, J.: Linkchains: Exploring the space of decentralised trustworthy linked data. In: Decentralizing the Semantic Web workshop at ISWC 2017 (2017)
21. Third, A., Domingue, J.: Linked data indexing of distributed ledgers. In: Proceedings of the 26th International Conference on World Wide Web Companion. pp. 1431–1436. International World Wide Web Conferences Steering Committee (2017)
22. Third, A., Gkotsis, G., Kaldoudi, E., Drosatos, G., Portokallidis, N., Roumeliotis, S., Pafili, K., Domingue, J.: Integrating medical scientific knowledge with the semantically quantified self. In: International Semantic Web Conference. pp. 566–580. Springer International Publishing (2016)
23. Third, A., Tiddi, I., Bastianelli, E., Valentine, C., Domingue, J.: Towards the temporal streaming of graph data on distributed ledgers. In: Proceedings of the 14th Extended Semantic Web Conference (forthcoming 2017)
24. Ugarte, H.: BLONDiE. https://github.com/EIS-Bonn/BLONDiE (2016)
25. W3C: SPARQL. https://www.w3.org/TR/rdf-sparql-query/ (2008)
26. W3C: Resource Description Framework. https://www.w3.org/RDF/ (2014)
27. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)