Towards Scalable Blockchain Analysis

Santiago Bragagnolo
Inria Lille-Nord Europe
Lille, France
santiago.bragagnolo@inria.fr

Matteo Marra
Vrije Universiteit Brussel
Brussels, Belgium
mmarra@vub.be

Guillermo Polito
Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189
CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille
Lille, France
guillermopolito@gmail.com

Elisa Gonzalez Boix
Vrije Universiteit Brussel
Brussels, Belgium
egonzale@vub.be

ABSTRACT

Analysing the blockchain is becoming more and more relevant for detecting attacks and frauds on cryptocurrency exchanges and smart contract activations. However, this is a challenging task due to the continuous growth of the blockchain. For example, in early 2017 Ethereum was estimated to contain approximately 300GB of data [4], a number that keeps growing day after day.

In order to analyse such ever-growing amount of data, this paper argues that blockchain analysis should be treated as a novel type of application for Big Data platforms. We also explore the application of parallelization techniques from the Big Data domain, in particular Map/Reduce, to extract and analyse information from the blockchain. We show that our approach significantly improves the index generation by 7.77 times, with a setup of 20 worker nodes, 1 Ethereum node and 1 Database node. We also share our findings of our massively parallel setup for querying Ethereum in terms of architecture and the bottlenecks. This should help researchers setup similar infrastructures for analysing the blockchain in the future.

CCS CONCEPTS

• **Computing methodologies** → **MapReduce algorithms**; • **Information systems** → **Query languages**; *Information retrieval query processing*; Computing platforms; Digital cash.

KEYWORDS

blockchain, smart contracts, big data, Map/Reduce

ACM Reference Format:
Santiago Bragagnolo, Matteo Marra, Guillermo Polito, and Elisa Gonzalez Boix. 2019. Towards Scalable Blockchain Analysis. In .... ACM, New York, NY, USA, 7 pages. https://doi.org/...

WETSWEB '19, 2019, Montreal, QC, Canada

ACM ISBN ...6...$15.00
https://doi.org/...

1 INTRODUCTION

Many existing work on blockchain analysis (on the domains of fraud, security, statistic analysis, etc.), base their algorithms and analysis strategies on analysing transactions [9, 13, 19]. For gathering transaction information and running their algorithms, they gather data by sequentially selecting and filtering information from different blockchain networks. This kind of usage is becoming outdated nowadays. Ethereum and other blockchain platforms store a massive amount of heterogeneous data: transactions, accounts, smart contracts, etc. In early 2017, Ethereum was estimated to have approximately 300GB of data [4], and it keeps growing at a high rate. Extracting information from this massive amount of data is not an easy task. Currently, there are two options to search for specific information stored in a block (e.g., account's data). Either the unique identifier of the block (or block hash) containing such information is known, or a sequential search in the blockchain is required starting from a block (e.g. the most recent one) and searching for every parent block for any related information. Hashes of the blocks are typically unknown (unless a secondary database is used to store all hashes of blocks deployed on the blockchain). As a result, a full scan of the blockchain may be required to be able to locate and analyse data.

In addition, Ethereum stores information (e.g. contracts, transactions) with a generic representation which does not include metadata describing the stored information. While this reduces the data size improving performance, it complicates the research of information once it is deployed in the blockchain.

The goal of our work is to ease the extraction and analysis of data from blockchain platforms like Ethereum. This is crucial to assist the process of debugging and fixing bugs in blockchain applications, particularly in smart contract applications. In prior work, we have proposed a SQL-like query language for Ethereum to ease the task of querying data in the blockchain [5]. Such a query language allows developers to quickly scan the blockchain to access individual elements and aggregate arbitrary values. To perform fast searches by semantic attributes, the usage of indexes is crucial. However, the setup and maintenance of indexes on the data stored in Ethereum is a highly consuming task. Some work has explored the performance problems to do blockchain analytics [4, 8] but either they work on early stages of the main cryptocurrency networks (when the

amount of data was still bearable)[4], or on smaller blockchains than Ethereum [8].

In this paper, we explore the usage of parallelizing techniques present in Big Data platforms, in particular Map/Reduce, to extract and analyse information from the blockchain. We focus on applying the Map/Reduce model to build indexes of blocks, transactions and contracts. We show that using our approach significantly improves the index generation time by 7.77x with a setup of 20 worker nodes running on a cluster of 10 Intel Xeon CPU E3-1240 @ 3.50GHz machines. These good results have been achieved with just a single blockchain data node running in the cluster.

In summary, our contributions are:

- to the best of our knowledge, we are the first using Big Data techniques to enable blockchain analytics.
- we present a Map/Reduce architecture that can effectively and efficiently query the blockchain.
- we identify the bottlenecks of such architecture and propose possible solutions for it.

2 STATE OF THE ART AND MOTIVATION

Prior work has identified the need of blockchain analysis for several use-cases such as the detection of attacks and security vulnerabilities [3, 7, 12, 13, 16, 18, 19], finantial frauds [14, 15, 19] or just retrieving statistics or economic indicators [9, 17]. Most of these works create their custom representation of the blockchain with all the data required for the desired analysis.

To cope with blockchain analysis in a more general way, several platforms and frameworks have been proposed in literature [4, 5, 8]. Kalodner et al. [8] propose BlockSci, a BlockChain analysis platform to analyze crypto currency based Blockchains like Bitcoin, Litecoin and others. BlockSci's main trait is its architecture: it imports the entire BlockChain data into memory, where queries can be executed fast. This makes BlockSci outperform other tooks by 15x-600x, and spends only around 60 seconds to setup their memory database because they access directly the blockchain data of big blockchains by directly accessing the raw data in disk instead of using the APIs designed for it. BlockSci, however, suffers two main drawbacks. First, BlockSci was not designed to work on smart contracts. The disregard for the smart contracts takes out their possibilities the contract based analysis. Many of the emerging blockchain technologies provide smart contract support (Ethereum[1], Hyperledger Fabric [2], NEM [3], Waves [4], etc). Second, it was not designed to work over a large amount of transactions. At the moment of writing (August 2017), the largest blockchain they had analysed was Bitcoin. Bitcoin was at the time 478,559 blocks large representing 130GB of storage, but their analysis were scoped only to the 22GB of transactions. Ethereum contains, at the moment of writing this paper (January 30 2019), more than 7 million blocks representing 266GB of storage. It is not clear for us how Ethereum's constraints and its smart-contracts can easily fit in such a memory strategy.

Bartoletti et al. [4] propose a general framework to access blockchain data previously stored in a database. In their approach, they export all blockchain data (around 500k blocks, 300GB of Bitcoin) into a NoSQL (mongoDB) and a relational (MySQL) database. Such imports take around 9 hours in each implementation, and querying them takes between 50 minutes and 3.5 hours, respectively. While they claim support for smart contracts and Ethereum, they did not report any benchmarks on it.

In prior work, we have also proposed the creation of a SQL-like query language to extract arbitrary data and analyse changes in user-defined smart contracts [5]. Like other researchers [20], we have identified how the performance problems of analysing the blockchain gets worse day after day. To perform fast searches by semantic attributes in such a query language, the usage of indexes is crucial. These indexes must be built by doing a full scan of all blocks, transactions, accounts and smart contracts in the blockchain, and building from them a smart data structure that can be quickly accessed. For instance, if we want to be able to access to all blocks that were created between two dates, an index data structure could be created to quickly discard all blocks that are outside the desired range. This kind of index data structure can be used for this and many other analyses, for example to access all the transactions that modified a given contract, or all the blocks where a given user/account received money. To build such indexes, however, a fast scanning infrastructure needs to be setup. Furthermore, such infrastructure needs to be kept up-to-date with the status of the blockchain. In our previous work we lacked such support.

3 BACKGROUND

Hardware advances in storage capacity and CPU processing have given rise to the concept of Big Data, characterized by the so-called 3 Vs (Volume, Velocity and Variety). As a result, novel software platforms have emerged to analyze and store such large data sets in a scalable way. The two most prominent programming models are Hadoop Map/Reduce [6] and Apache Spark [2], which typically embrace a batch-oriented data processing to achieve a high parallelisation of data analysis.

Current trends indicate that the volume, velocity and variety of data are increasing quickly due to an explosion on diversity and number of sources of information (as a result of the digitalization of data, e.g. smart objects and sensors, interconnectivity of data and popularity of social media data [11]). In this paper we identify blockchain technology as a new type of sources of information in the Big Data revolution. Blockchain platforms like Ethereum contain a high amount of heterogeneous data growing at a high rate, presenting a clear example of the three Vs of BigData: Volume, Variety, Velocity. In this paper, we explore the usage of Big Data technology to enable blockchain analysis. A recent publication on research directions in blockchain analytics has also suggested the use of Big Data technology but, to the best of our knowledge, this is the first approach in realising such a vision [20]. In this section, we provide some details on the programming model we employed in this work, namely the Map/Reduce model [6], and the programming platform in which we conducted this research.

3.1 The Map/Reduce Model

The Map/Reduce model can be used in a variety of different data analysis, and offers a simple API to developers. In this model, the

[1] https://www.ethereum.org/
[2] https://www.hyperledger.org/projects/fabric
[3] https://nem.io/
[4] https://wavesplatform.com/

developer structures her program in two functions: a *map* and a *reduce*. The *map* function transforms an initial set of values to an intermediate result, encoded in key/value pairs. The *reduce* function aggregates data to compute a final result. In particular, *reduce* transforms the intermediate results generated by *map*, grouped by their key, into to a final result for each of the keys. A Map/Reduce platform will take as input some collection of data, partition it and run the developer's *map* and *reduce* functions on each partition in different machines or processes.

As a concrete example, consider a *grep* which searches the lines containing a pattern in a file. The code snippet in Listing 1 implements such an operation on the Map/Reduce model. The map function checks if a particular line includes a pattern, and returns the line if the pattern is present, or NIL otherwise. The reduce function filters the mapped lines to remove the values that are NIL, returning all of the lines that contain the pattern.

Listing 1: Pseudocode of a grep in Map/Reduce.

```
1  map(fileLine, pattern) {
2    if (fileLine.contains(pattern))
3    then return fileLine
4    else return NIL}
5  reduce(mappedLines){
6    return mappedLines.filter( line => line != NIL)}
```

In this paper, we employed a Map/Reduce framework for Pharo Smalltalk called Port [10]. Port defines a Smalltalk API to define and execute Map/Reduce programs deployed on top of the Apache Hadoop platform. In particular, developers need to write the *map* and *reduce* function in Smalltalk and then Port manages the execution of such a Map/Reduce application using master and worker nodes deployed on Hadoop Yarn [1], and implements all communication details between them.

4 APPLYING BIG DATA TECHNOLOGY TO BLOCKCHAIN ANALYSIS

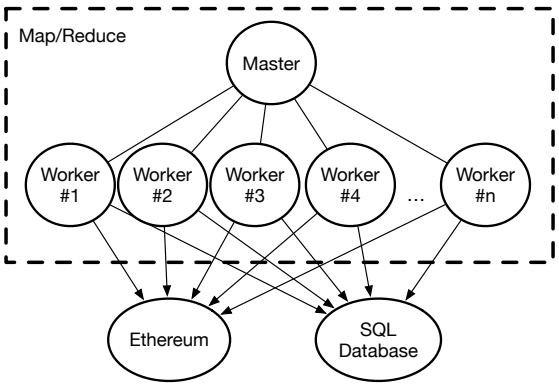

Figure 1: A Map/Reduce Architecture for Blockchain Analysis

In this section we describe how a blockchain analysis application can be created using the Map/Reduce model. Figure 1 illustrates our Map/Reduce architecture which has been adapted to work with the Ethereum platform. The Map/Reduce framework provides a master node that splits the data and assigns map and reduce jobs to different worker nodes. We also setup a single database instance and a single Ethereum client, accessible by all these nodes in the same network. Each *map* worker queries the Ethereum platform to access blockchain data and sends it back to the master node. Then, each *reduce* worker receives many map results together, and performs a bulk insert in the SQL database. Once the data is stored in the database, we can use the information that was indexed (e.g., the hash of the block, or a particular property of the block or contract, ...) to query the blockchain and gather the requested information.

We will now illustrate the different parts of our Map/Reduce architecture to implement an indexation algorithm that scans the entire blockchain and stores a block index into the centralized SQL database.

Block hash	Timestamp	ParentBlock
ca896d6	28/01/2019 ...	da6b261
da6b261	27/01/2019 ...	7aa96ae
7aa96ae	26/01/2019 ...	d6d3614
d6d3614	25/01/2019 ...	402d518

Figure 2: Example of a Block index table. Timestamp and ParentBlock columns are indexed.

4.1 An Indexation Algorithm

We have developed an indexation algorithm that uses a relational database to store indexed data. This choice was based in two reasons: to benefit from an existing and mature index implementation, and to focus on the problems of parallelism and high load. However, our overall architecture does not prevent alternative indexing data structures like a BST (Binary Search Trees) [5] to be implemented instead.

Our index has the structure of a relational table with standard database indexes. For example the table representing the block index has the block's hash, but also a timestamp and its parent block's hash as shown in Figure 2. The two latter columns are indexed, so we can do fast queries on blocks by both timestamp and their parent blocks.

To setup such an index, our core algorithm performs a full scan of the blockchain inserting all the corresponding values in our database. With a asynchronous I/O sequential scan approach, our algorithm takes about 30 milliseconds per indexed entry, meaning that the expected time for indexing the current load of Ethereum blocks, would take about 2 days and 11 hours.

To avoid a sequential scan, we decided to parallelize such processing using a Map/Reduce model as described below.

4.2 Parallelizing Blockchain Indexation

We now detail how we parallelized the indexation algorithm using a Map/Reduce model. This basically boils down to expressing the algorithm in terms of the *map* and *reduce* functions of the Map/Reduce model previously explained. Listing 2 illustrates our functions in pseudocode for the sake of clarity. In the next subsection, we detail the concrete implementation in Port.

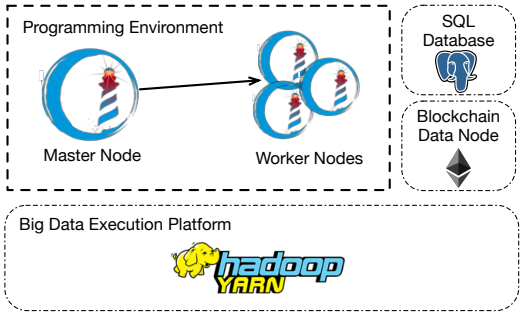

Figure 3: Our Map/Reduce Architecture

The `map` function queries the blockchain to obtain the data related to a block index. The `reduce` function takes the result of the map on several indexes (i.e. a partition of indexes), and stores them all in a centralized database with a bulk insert.

Listing 2: Pseudocode of Map/Reduce indexing algorithm.

```
1  map(blockIndex) {
2      return blockIndex->hash(blockchain.at(blockIndex))
3  }
4
5  reduce(pair){
6      storeInDatabase(pair)
7  }
```

Such an approach allows us to parallelize the mapping of the indexes and to execute in parallel different database stores, minimizing the calls by executing one for each partition of data (and not one for each index).

4.3 Implementation

Figure 3 illustrates the main different components of our Map/Reduce architecture. The application is programmed in Pharo Smalltalk, on top on top of the Port Map/Reduce framework, which handles the management of master and worker nodes. Port provides us integration with the Map/Reduce environment offering an API to implement a Map/Reduce by the means of *map* and *reduce* methods in Smalltalk. Port uses Hadoop Yarn [1] to manage the allocation of master and worker nodes in a cluster.

We use Geth [5] as blockchain data node. Internally, the communication with Geth is managed by the Fog Ethereum driver [6].

Listing 3 illustrates our application implementation in Port. Our Map/Reduce application functions accept an additional parameter, an `indexBuilder`, that provides the database connection and all of the information about the properties that are being indexed. Such parameter is identical in all of the parallelized map and reduce functions. The `map:parameter:` (lines 1 to 6) obtains a block given a block index, obtains the indexed property from the given block, and returns a key value pair in the form (block index, indexed value). For example, in the case of indexing blocks by timestamp, the indexed value is the timestamp of the obtained block. The `reduce:parameter:` (lines 8 to 9) receives a collection of pairs produced by the `map:parameter:` function and uses the builder to store those pairs in the corresponding database table.

Listing 3: Pharo implementation of indexing algorithm

```
1  MRIndexingApp >> map: blockIndex parameter: aBuilder
2      | ethereumBlock mappedProperty |
3      ethereumBlock := fogBlockChain at: blockIndex.
4      mappedProperty := ethereumBlock
5          get: aBuilder indexedProperty.
6      ↑ blockIndex -> mappedProperty
7
8  MRIndexingApp >> reduce: pairs parameter: aBuilder
9      aBuilder storeIndexedValues: pairs
```

Differently from a classical Map/Reduce framework, Port does not enforce the initial data to be in a key-value store. This allows us to map directly on simple indexes, generating an intermediate key-value store where each key has only one value. It also does not always apply a group-by between map and reduce, as happens in classical Map/Reduce platforms. In our case this avoids an unneeded shuffling of data over the network, that would only slow down the blockchain analysis application.

5 EXPERIMENTS

In this paper, we aim to tackle the difficult process of indexing the full blockchain by applying a Map/Reduce algorithm on top of Port, a Map/Reduce framework for Pharo. We conducted different experiments to check whether our Map/Reduce architecture and implementation is a scalable architecture for blockchain analysis. A first experiment, called *architecture*, tries to find the best architecture (in the terms of amount of workers) to index the full blockchain. A second one, called *analyzing the full blockchain*, analyses how our approach scales when trying to analyze the full Ethereum blockchain.

5.1 Setup

We run our experiments on a cluster of one *root* node and ten identical *slave* nodes. Each slave node presents the following specifications:

- Processor: Intel Xeon CPU E3-1240 @ 3.50GHz (4 cores, 8 threads)
- Ram: 32 GB
- Storage: 200 GB SSD

The root node has the same specification as the slave nodes, but it has enhanced storage. All the nodes are connected through a 1 GB/s local network.

In our configuration, the root node runs:

- A Postgres server that handles the database.
- An instance of Port to handle the external communication with the different nodes.

One of the slave nodes runs exclusively Geth, the blockchain data node. The rest of the slave nodes (9) are used to deploy Pharo containers through Yarn. There is always one container running a Yarn application, one running a master, and a global maximum of 70 containers available to run a worker. The number of actual

[5] https://github.com/ethereum/go-ethereum/wiki/geth
[6] https://github.com/smartanvil/Fog

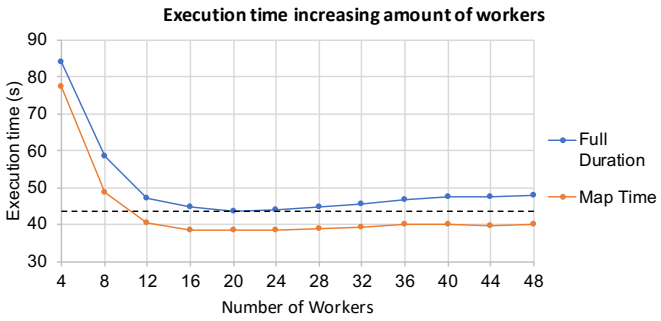

Figure 4: Execution time of indexing 10.000 block increasing the number workers

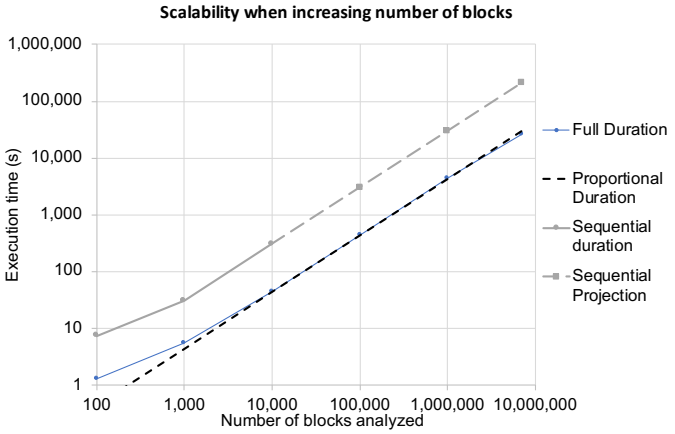

Figure 5: Execution time of indexing an increasing amount of blocks with 20 workers.

containers running a worker variates depending on the experiment that we are running.

5.2 Experiment 1: Architecture

Indexing the whole blockchain, even using a Map/Reduce approach, is not trivial. Not only does the performance of the indexing depends on the number of workers that can parallelize the work, but also on how these workers communicate with the blockchain data node (during the mapping phase) and with the database (during the reduce phase). While the employed database (i.e. Postgres) is designed for concurrent access and can scale well, the blockchain data node (i.e. Geth) allows us to query the blockchain using the RPC protocol, which can support only a limited amount of concurrent calls. Given our setup, Geth runs in a dedicated slave node with a total of 8 threads.

In this first experiment we test how the execution time changes when increasing the number of workers, while indexing a fixed amount of blocks (10.000 blocks). We increased the amount of workers from 4 to 48, 4 by 4. Each iteration is run 10 times, discarding the worst result.

Figure 4 shows the result of this experiment. The blue curve represents the average execution time of the full Map/Reduce, while the orange curve represents only the average time of the map phase. The black dashed line represents the averaged minimum amount of execution time (i.e. 43.6 seconds with 20 workers).

Looking at the total execution time, we observe that it quickly decreases from around 83 seconds (with 4 workers) to around 58 seconds for 8 workers. It then stabilizes between 47.7 seconds and 43.6 seconds when running with 12 to 48 workers, reaching its global minimum when running with 20 workers. As such, we conclude that 20 is the best amount of workers for this configuration.

Analyzing the relation between map time and total execution time, the graph shows that the two lines follow the same trend, being the map the biggest component of the whole execution time (between 83% and 92% of the total execution time). It is clear how the performance of the map is crucial to the performance of the whole indexation. We discuss more about the implications of this finding in Section 5.4.

5.3 Experiment 2: Analyzing the full blockchain

The results of experiment 1 show that the best amount of workers to use in our configuration is 20. In this second experiment, we use such configuration to check how our system scales to index the full blockchain. To do so, we gradually increase by a factor of 10 the number of blocks to index from 100 to the full Ethereum blockchain, that to the moment of our latest experiments (29/01/2019) consisted of 7.080.006 blocks.

Figure 5 shows how the execution time increases when increasing the number of analyzed blocks. The blue line represents the execution time, the grey line represents the execution time of the sequential implementation of the indexing algorithm. The values of the dashed gray line are proportionally projected, since the execution time would have been too high for experimenting it. The dashed black line represents what the proportional expected duration is, calculated on the result for 10.000 blocks. Both of the scales are logarithmic (log_{10}), hence the two axis are proportional. The graph shows that the execution time grows proportionally, as the black dashed line, except for 100 and 1.000 blocks, where the execution time is heavily impacted by the overhead of Port.

Using 20 parallel single threaded workers, we managed to index the full Ethereum blockchain (to the moment the experiment was executed) in 7 hours 18 minutes and 47 seconds. While the experiments on up to 100.000 blocks were repeated at least 10 times, due to time limitations the experiments over 1 million blocks were executed only 4 times and the one on the full blockchain only once.

If we compare such results to a sequential approach (that we estimated would take more than 2 days), it is clear how the execution time of the indexation is lower using our Map/Reduce approach.

5.4 Discussion

The first experiment, **Architecture** aims to give some light on the limits of the Geth client and, therefore, in our first general architecture. The second experiment **Analyzing the full blockchain**, leverages the results of the first experiment for showing a good

result on how a map reduce approach can improve the performance of blockchain analysis.

In the first experiment, we learned that the performance of the application does not scale to the number of workers, since between 12 and 48 workers the difference of execution time ranges in less than 10 seconds for 10.000 blocks. In fact, the map operation, which constitutes at least 83% of the execution time, performs calls to the blockchain data node (Geth), which does not handle optimally concurrent requests, since it is not designed to be used for analysis on the blockchain. This shows how making the analysis of the blockchain faster and scalable is not a mere question of parallelising, but also of finding the right configuration.

Since our first experiments we were sceptical about the responsiveness of Geth, the blockchain data node. To confirm quickly our hypothesis, we launched our algorithm with the maximum possible amount of workers deployed (70 worker nodes), one blockchain data node and one database node. Profiling the network performance exposed the limits of the blockchain data node towards the processing external requests, congestioning the network and overconsuming operating system resources. Hence, we conducted our first experiment starting from a little amount of nodes, to discover the limits of our architecture.

We believe that increasing the number of blockchain data nodes could further improve the map execution's time, heavily impacting the overall duration of the indexing. In our configuration, we found that the best ratio *worker/blockchain data nodes* is 20x, or, if we count on the physical threads the workers and the blockchain data nodes were using, around 2.5 workers for threads of the blockchain data node. However, further experiments are required to verify that such a ratio remains the same when increasing the number of blockchain data nodes (or the amount of cores it is using). In fact increasing such a number can increase the amount of network usage in the whole cluster, other than change the storage footprint of the blockchain that would need to be replicated in the different blockchain data nodes. We are planning to execute different experiments to investigate how such ratio can change, and what can be the impact on the overall execution time.

Overall, we believe that the experiments conducted on our approach shows how the use of a Big Data framework for parallelisation can improve the blockchain analysis.

6 CONCLUSION

In this paper, we explored the application of Big Data techniques to enable blockchain analytics. In particular, we applied Map/Reduce for indexing Ethereum blocks. To the best of our knowledge, this is the first work employing Big Data techniques to enable scalable blockchain analytics. Our experiments show that Map/Reduce can efficiently be used to index the blockchain, improving the run-time performances of the indexing process by 7.7 times with respect to a sequential implementation.

Our current architecture includes one node running a SQL database (i.e. Postgres), one node running the blockchain data node (i.e. Geth), one master node coordinating different worker nodes that run the map and reduce tasks. Our Map/Reduce application is written using Port, a Map/Reduce framework for Pharo Smalltalk. In such application, we parallelize the queries to the blockchain data node in the map function, trying to minimize the impact of such queries on the run-time. In the reduce function, we then perform a bulk insert of the indexed data in the database. By testing different setups of such architecture, we identified Geth as being a performance bottleneck, discovering the right ratio of worker nodes (that are consuming data from the blochckain) and blockchain data nodes.

Despite of the promising results shown in this paper, we believe it is too early to say that our architecture reached the limit of the Map/Reduce efficiency to enable blockchain analytics. Further research is needed to explore the best configuration for such an architecture. In particular, we are interested in exploring the following research directions:

- Exploring the advantages and limitations of using multiple data nodes.
- Exploring the utility of Map/Reduce on the resolution of index based queries.
- Exploring the limits of Map/Reduce as an online algorithm for continuous indexation.

ACKNOWLEDGMENTS

Matteo Marra is funded by a SB PhD grant at FWO - Research Foundation - Flanders. Project number: 1S63418N.

We would like to thank UTOCAT for incentivating this work.

REFERENCES

[1] Apache. [n. d.]. Apache Hadoop YARN. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. Accessed: 2017-08-24.
[2] Apache. [n. d.]. Apache Spark. http://spark.apache.org/. Accessed: 2017-05-12.
[3] Khaled Baqer, Danny Yuxing Huang, Damon McCoy, and Nicholas Weaver. 2016. Stressing out: Bitcoin stress testing. In *International Conference on Financial Cryptography and Data Security*. Springer, 3–18.
[4] Massimo Bartoletti, Stefano Lande, Livio Pompianu, and Andrea Bracciali. 2017. A General Framework for Blockchain Analytics. In *1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL '17)*. ACM, New York, NY, USA, 7:1–7:6. https://doi.org/10.1145/3152824.3152831
[5] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse. 2018. Ethereum Query Language. In *1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. 1–8. https://doi.org/10.1145/3194113.3194114
[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492
[7] Michael Fleder, Michael S Kester, and Sudeep Pillai. 2015. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657* (2015).
[8] H. Kalodner, S. Goldfeder, A. Chator, M. Möser, and A. Narayanan. 2017. BlockSci: Design and applications of a blockchain analysis platform. *ArXiv e-prints* (sep 2017). arXiv:cs.CR/1709.02489
[9] Matthias Lischke and Benjamin Fabian. 2016. Analyzing the bitcoin network: The first four years. *Future Internet* 8, 1 (2016), 7.
[10] Matteo Marra, Clèment Bèra, and Elisa Gonzalez Boix. 2018. A debugging approach for Big Data applications in Pharo. In *To Appear in Proceedings of the 13th Edition of the International Workshop on Smalltalk Technologies (IWST '18)*. ACM, New York, NY, USA.
[11] Cukier K. Mayer-Schönberger, V. 2013. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. London: John Murray.
[12] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 127–140.
[13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2016. A fistful of bitcoins: Characterizing payments among men with no names. *Commun. ACM* 59, 4 (2016), 86–93.
[14] Malte Moser, Rainer Bohme, and Dominic Breuker. 2013. An inquiry into money laundering tools in the Bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*. IEEE, 1–14.

[15] Malte Möser, Rainer Böhme, and Dominic Breuker. 2014. Towards risk scoring of Bitcoin transactions. In *International Conference on Financial Cryptography and Data Security*. Springer, 16–32.
[16] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. 2013. Structure and anonymity of the bitcoin transaction graph. *Future internet* 5, 2 (2013), 237–250.
[17] Dorit Ron and Adi Shamir. 2013. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*. Springer, 6–24.
[18] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. 2014. Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*. Springer, 457–468.
[19] Marie Vasek and Tyler Moore. 2015. There is no free lunch, even using Bitcoin: Tracking the popularity and profits of virtual currency scams. In *International conference on financial cryptography and data security*. Springer, 44–61.
[20] Hoang Tam Vo, Ashish Kundu, and Mukesh K Mohania. 2018. Research Directions in Blockchain Data Management and Analytics.. In *EDBT*. 445–448.