UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
POSTGRADUATE PROGRAM IN COMPUTER SCIENCE

João Vicente Meyer

A PROPOSAL OF A UNIFIED MODEL FOR BLOCKCHAIN DATA

Florianópolis, Santa Catarina – Brazil
2023

João Vicente Meyer

A PROPOSAL OF A UNIFIED MODEL FOR BLOCKCHAIN DATA

Master's Thesis submitted to the Postgraduate Program in Computer Science of Universidade Federal de Santa Catarina for degree acquirement in Masters of Science degree in Computer Science.
Supervisor: Ronaldo dos Santos Mello, Phd.

Florianópolis, Santa Catarina – Brazil
2023

Legal Notes:

There is no warranty for any part of the documented software. The authors have taken care in the preparation of this thesis, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained here.

Cataloging at source by the University Library of the Federal University of Santa Catarina.
File compiled at 17:25h of the day Sunday 10th March, 2024.

João Vicente Meyer
A Proposal of a Unified Model for Blockchain Data / João Vicente Meyer; Supervisor, Ronaldo dos Santos Mello, Phd. - Florianópolis, Santa Catarina - Brazil, 14 of December of 2023.
58 p.

Master's Thesis - Universidade Federal de Santa Catarina, INE - Department of Informatics and Statistics, CTC - Technological Center, Postgraduate Program in Computer Science.

Includes references

1. blockchain, 2. data, 3. model, I. Ronaldo dos Santos Mello, Phd. II. Postgraduate Program in Computer Science III. A Proposal of a Unified Model for Blockchain Data

CDU 02:141:005.7

João Vicente Meyer

A PROPOSAL OF A UNIFIED MODEL FOR BLOCKCHAIN DATA

This Master's Thesis was considered appropriate to get the Masters of Science degree in Computer Science, and it was approved by the Postgraduate Program in Computer Science of INE – Department of Informatics and Statistics, CTC – Technological Center of Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brazil, 14 of December of 2023.

Documento assinado digitalmente
Marcio Bastos Castro
Data: 11/03/2024 08:33:41-0300
CPF: ***.707.720-**
Verifique as assinaturas em https://v.ufsc.br

Márcio Bastos Castro, Phd.
Coordinator of Postgraduate Program in Computer Science

Examination Board:

Documento assinado digitalmente
RONALDO DOS SANTOS MELLO
Data: 12/03/2024 16:41:14-0300
CPF: ***.693.660-**
Verifique as assinaturas em https://v.ufsc.br

Ronaldo dos Santos Mello, Phd.
Supervisor
Universidade Federal de Santa Catarina – UFSC

Prof. Renato Fileto, PhD.
Unifersidade Federal de Santa Catarina – UFSC

Prof. Jean Everson Martina, PhD.
Unifersidade Federal de Santa Catarina – UFSC

Prof. Helena Grazziotin Ribeiro, PhD.
Unifersidade de Caxias do Sul – UCS

Documento assinado digitalmente
JOAO VICENTE MEYER
Data: 10/03/2024 17:34:02-0300
CPF: ***.816.500-**
Verifique as assinaturas em https://v.ufsc.br

To my family and friends.

ABSTRACT

The popularity of blockchain-based systems has been steadily rising, and a wide range of studies have emerged: from consensus algorithms to the management of huge and immutable data structures. We have also seen the spawn of dozens of cryptocurrencies, hundreds of Initial Coin Offerings (ICO), as well as the rise of big blockchains, such as Bitcoin and Ethereum, and their use for digital crime. In this context, enthusiasts are facing problems such as scalability, fast data retrieval for blockchain analytics and fraud detection. Some surveys discuss the challenges faced by blockchain networks and other discuss easier ways to make blockchain easily queriable and analysable. However, the literature lacks an analysis of data models in such systems. Like, how blocks are represented in most blockchains? Is there common ground between each different chain? In order to fill this gap, this work researches the state-of-the-art on blockchain data modelling, and proposes a suitable and expressive one in order to tackle the mentioned problems. Differently from past works, the present work tries to focus on a unified data model for different blockchains.

Keywords: blockchain. data. model.

RESUMO

A popularidade de sistemas baseados em blockchain vem crescendo e uma variedade de estudos têm sido produzidos: desde algoritmos de consenso ao gerenciamento de gigantescas e imutáveis estruturas de dados. Pode-se mencionar também o surgimento de dezenas de criptomoedas, centenas de *Initial Coin Offerings* (ICO), assim como o surgimento de grandes blockchains, como Bitcoin e Ethereum, e seus usos para o crime digital. Neste contexto, entusiastas estão enfrentando problemas como escalabilidade, rápido acesso a dados de blockchains para *analytics* e detecção de fraudes. Alguns *surveys* discutem os desafios enfrentados pelas redes de blockchain e outros discutem maneiras de tornar os dados gerados pelas blockchains fácilmente analisáveis. Contudo, a literatura não faz uma análise dos modelos de dados nestes sistemas. Por exemplo, como blocos são representados na maioria das redes? Existem definições comuns entre as diferentes redes? De maneira a preencher esta lacuna, este trabalho investiga o estado da arte em modelagem de dados para dados de blockchains e propõe um modelo de dados adequado para resolver os problemas mencionados. Diferentemente dos trabalhos já existentes, o presente trabalho busca focar no modelo de dados unificado para diferentes blockchains.

Palavras-chaves: blockchain. dados. modelo.

RESUMO EXPANDIDO

INTRODUÇÃO

Com o aumento do interesse em tecnologias baseadas em blockchain, o mundo testemunhou a criação de novas e diferentes platafomas. A blockchain teve sua origem como uma forma de descentralizar finanças, sendo *Bitcoin* o exemplo mais bem-sucedido. No entanto, com a chegada das blockchains baseadas em contratos inteligentes, surgiram organizações descentralizadas autônomas, como Aragon, Maker e The DAO, além de *exchanges* descentralizadas, como Uniswap, e até mesmo mundos virtuais inteiros, como Decentraland. Contudo, novas tecnologias podem trazer novas e inovadoras formas de enganar seus usuários. A pseudo-anonimidade concedida por blockchains as torna úteis para atividades ilegais, como ransomware (KSHETRI; VOAS, 2017) e esquemas Ponzi (CHEN et al., 2018).

No entanto, a pseudo-anonimidade não é anonimato total e há discussões em andamento sobre desanonimização de contas (TURNER; IRWIN, 2018). Um modelo de dados especificamente projetado para blockchains poderia ajudar a tornar esse problema menos complicado. Por exemplo, um algoritmo de identificação poderia pesquisar um conjunto de dados com mais precisão se estivesse de acordo com o modelo de dados. Isso também poderia ser associado a algoritmos de aprendizado de máquina para fazer isso. Dessa forma, os pesquisadores gastariam menos tempo limpando os dados e mais tempo analisando-os. Além disso, há um tópico de pesquisa chamado análise de blockchain (VO; KUNDU; MOHANIA, 2018), onde problemas novos (antigos), como consulta em grandes volumes de dados gerados por redes. Este é um dos principais pontos deste trabalho, ou seja, compreender os modelos de dados usados em sistemas baseados em blockchain e as técnicas aplicadas para aprimorar a sua eficiência.

OBJETIVOS

O objetivo geral que esta pesquisa espera alcançar é desenvolver um modelo de banco de dados que pode ser usado para mapear dados de vários blockchains em um modelo unificado. Este modelo, usado em combinação com soluções de armazenamento conhecidas, como sistemas de gerenciamento de bancos de dados, pode fornecer uma maneira muito mais fácil de extrair informações das redes blockchain. Além deste objetivo geral, alguns objetivos específicos também devem ser alcançados: (1) Identificar os modelos atualmente utilizados pelas redes mais conhecidas, como Bitcoin e Ethereum; (2) Propor um modelo de dados que englobe os principais componentes do modelos nessas redes; (3) Aplicar e avaliar este modelo de dados para testar sua expressividade e utilidade ao extrair informações.

METODOLOGIA

O protocolo de pesquisa utilizado foi baseado na metodologia de revisão sistemática da literatura de (KITCHENHAM, 2004). O protocolo envolve a definição de questões de pesquisa, uma string de busca adequada e critérios de inclusão e exclusão. As perguntas de pesquisa incluem investigações sobre modelos de dados para blockchains e técnicas de consulta de dados de blockchain. A string de busca foi refinada para garantir resultados relevantes. Critérios de inclusão e exclusão foram estabelecidos para filtrar pesquisas irrelevantes, e o ano de publicação não é considerado como critério de exclusão devido à novidade do tópico de gestão de dados de blockchain.

Após encontrar os artigos relevantes e revisá-los, avançamos para a fase de desenvolvimento e validação. Nesta etapa, definimos um modelo de dados para os dados de blockchain e realizamos testes para validar sua eficácia. Os testes incluíram a avaliação da integridade e desempenho do modelo em diferentes cenários de uso. Os resultados demonstraram que o modelo proposto é capaz de lidar com diversas operações de dados de blockchain de forma eficiente e precisa. Além disso, identificamos áreas de melhoria e refinamento para otimizar ainda mais o modelo. Este processo de desenvolvimento e validação foi crucial para garantir a robustez e confiabilidade do modelo de dados para aplicações de blockchain.

RESULTADOS E DISCUSSÃO

Ao comparar os resultados, observamos que o MongoDB, em alguns casos, apresenta desempenho superior ao cliente oficial Ethereum. Por exemplo, o MongoDB mostrou resultados melhores nas consultas "fáceis", mesmo em comparação com o cliente oficial Ethereum. O PostgreSQL leva mais tempo do que o cliente Ethereum, mesmo que índices sejam criados e utilizados pelo planejador de consultas.

No entanto, observamos que o PostgreSQL e o MongoDB se destacam quando precisamos realizar as consultas "difíceis". Ambos os bancos de dados executam as consultas difíceis duas ordens de magnitude mais rápido do que o cliente Ethereum. O MongoDB tem a vantagem aqui também, recuperando os dados mais de 10 vezes mais rápido que o PostgreSQL em alguns casos. Isso se deve ao fato de que os dados, no PostgreSQL, são armazenados em duas tabelas distintas. Portanto, sempre que queremos consultar dados que exigem informações de ambos, blocos e transações, precisamos realizar uma "junção", que não é barata. O MongoDB, por outro lado, devido à sua arquitetura orientada a documentos, nos permite obter facilmente ambos os dados, blocos e transações, sem a necessidade de junções caras. Por exemplo, ao consultar timestamps para transações, informações contidas na entidade "bloco", simplesmente encontramos a transação desejada e obtemos seu bloco do mesmo documento. Enquanto para o PostgreSQL, precisamos realizar uma "junção"entre o atributo "bloco"da tabela de transações com o atributo "hash"da tabela de blocos.

CONSIDERAÇÕES FINAIS

O gerenciamento e análise de dados de blockchain ainda são questões em aberto, embora seus casos de uso sejam muito relevantes. Detecção de fraudes, desanonimização de carteiras, transações e análises entre cadeias são apenas alguns exemplos. Os clientes nativos de blockchain não estão bem equipados para realizar essas tarefas. Nesse contexto, este trabalho apresenta o estado-da-arte em relação à modelagem de dados de blockchain, destacando a importância de combinar o melhor desses mundos para projetar soluções de gerenciamento de dados de blockchain. Além disso, o trabalho apresenta uma pesquisa pioneira sobre modelagem de dados para informações de blockchain, bem como uma proposta de um modelo de dados unificado usado para mapear informações de diferentes blockchains sob o mesmo esquema, facilitando a criação de ferramentas para interagir com esses dados. No entanto, ainda não encontramos um trabalho que detalhe um processo de modelagem de dados de blockchain, incluindo o uso de uma metodologia de design de banco de dados típica, começando por um modelo conceitual e avançando para um modelo lógico para ser implementado posteriormente em um SGBD. Este estudo também resultou em uma publicação na iiWAS 2022, demonstrando o interesse acadêmico na área de modelagem de dados de blockchain, e futuros trabalhos incluem o uso de técnicas de modelagem de dados mais avançadas, como o uso de bancos de dados baseados em grafos para entender melhor o movimento de fundos nessas redes.

Palavras-chaves: blockchain. dados. modelo.

LIST OF FIGURES

LIST OF TABLES

LISTINGS

CONTENTS

1 INTRODUCTION

With the surge in popularity of blockchain technologies, the world has seen the imagination of developers flourish with its many applications. Blockchain originated as a way to decentralise finances, Bitcoin as its biggest and most successful example. However, with the advent of smart contract-based blockchains, we have witnessed the creation of decentralised autonomous organisations, like Aragon[1], Maker[2] and The DAO[3], as well as decentralised exchanges such as Uniswap[4], and even full virtual worlds, like Decentraland[5].

Unfortunately, new technologies come with new and innovative ways to trick its users. The pseudo anonymity given by blockchains make them useful for illegal activities, like most commonly ransomware (KSHETRI; VOAS, 2017), and ponzi schemes (CHEN et al., 2018). Yet, contrary to popular belief, pseudo anonymity is not full anonymity and there are ongoing discussions on the topic of de-anonymization of accounts (TURNER; IRWIN, 2018; NICK, 2015). An specific designed data model for blockchains could make this problem less cumbersome. It could help, for example, an identification algorithm to search with more precision a dataset that is in accordance to the data model. It could also be coupled with machine learning algorithms to do so. In such a way, researches would spend less time cleaning the data and more time performing analysis on it.

Additionally, there is a research topic, called *blockchain analytics* (VO; KUNDU; MOHANIA, 2018; AKCORA et al., 2018), where (old) new problems, such as data querying, arise from the volume of data generated by networks. This is one of the main points of this work, i.e., to understand data models used in blockchain systems and the applied techniques to improve query efficiency in order to make analysis easier, faster and cheaper. With that, a fast enough system could be used to make market predictions based on token movements on the network, prevent frauds and identify wallets. Besides, a well-designed data model provides an easier way to manipulate data in order to fit a broad set of applications. For example, cross-chain analytics could be made more viable by providing a data pattern used by most blockchain networks, and even help integrate blockchain with existing systems, like Open Banking.

With new networks spawning every day, it is natural to see each network creating its own data model to represent its real-world concepts. We need not to go any further than the bigger networks to realise differences in their models. Some of them occur

---

[1] Aragon: `https://aragon.org`

[2] Maker: `https://makerdao.com`

[3] The DAO: `https://github.com/blockchainsllc/DAO`

[4] Uniswap: `https://uniswap.org`

[5] Decentraland: `https://decentraland.org`

mainly in terms of naming, e.g., “transactionsRoot” vs “merkleroot” in Ethereum and Bitcoin’s block, respectively. Yet, some differences are more complex, such as the inclusion of “logsBloom” and “uncles”, in Ethereum’s blocks, which refers to Ethereum’s specific features. This heterogeneity raises the need for a clearer understanding in the subject.

1.1 OBJECTIVES

The general objective that this research hopes to achieve is to develop a data model that can be used to map data from multiple blockchains under one unified model. This model, used in combination with known storage solutions, such as DBMS, could provide a much easier way to extract information from the blockchain networks. In order to achieve this general objective, some specific ones must be achieved as well: (1) Identify the models currently used by most well-known networks, such as Bitcoin and Ethereum; (2) Propose a data model that encompasses the main components of the models in those networks; (3) Apply and evaluate this data model in order to test its expressiveness and usefulness when extracting information from it.

1.2 CONTRIBUTIONS

The main contributions of this work is the creation of a data model, together with an extract, transform, load (ETL) process to be able to apply the data model to publicly available blockchain data in order to evaluate the the model expressiveness and usefulness.

1.3 METHODOLOGY

The methodology of this project follows the guidelines from the Wazlawick’s book (WAZLAWICK, 2017), which comprises 3 steps: (1) A systematic review of the literature; (2) Development of the solution; (3) Evaluation of the developed solution.

1.4 STRUCTURE

This document is organised as follows: Chapter 2 briefly covers some basic concepts related to blockchains, like consensus, storage approaches and common networks properties and models. Chapter 3 shows an overview of the state-of-the-art in order to gather current understandings and challenges on the subject. Chapter 4 describes the proposal and its implementation. Finally, Chapter 5 evaluates the proposed model.

2 BASIC CONCEPTS

2.1 BLOCKCHAIN

A blockchain is a distributed hash list of blocks, also commonly called ledger. Those blocks are usually composed of a set of transactions, in this case, of crypto coins[1], sent from one user to another. The first published work that made use of blockchain-like structure[2] was in 1990 (HABER; STORNETTA, 1990). It devised trustless time-stamping service (TTS) where a linked list of hashed objects was used to guarantee non tampering of timestamp of files. The big difference between the modern blockchains and the one devised in 1990 is its use cases. Whereas the old one was used to assure reliable timestamps, the newer ones are used to transfer digital currencies between members.

Every node in the network[3] has an identical copy of the ledger. Every change performed in a node is propagated to every other node to keep the structure in sync. Unfortunately, a big problem arises from it. How to guarantee that the changes made are valid/trustworthy? This is famously known as the Byzantine Fault (WENSLEY et al., 1978). To solve this problem, consensus algorithms were introduced.

2.1.1 Consensus Algorithms

Consensus algorithms are used to achieve, as the name indicates, consensus by the parties involved upon something. Be it a transaction, timestamp or anything communicated between nodes. Nakamoto (2008) famously published the incentive-based consensus algorithm together with the Bitcoin proposal. The basic idea is that there must be incentives given for the peers in the network to play by the rules. It does not sound like a perfectly safe idea, in theory. However:

> ...we observe consensus working, but have not developed the theory to fully explain why it works (NARAYANAN, 2016, p. 35).

There are many consensus algorithms already devised (NGUYEN; KIM, 2018; ALSUNAIDI; ALHAIDARI, 2019; SANKAR; SINDHU; SETHUMADHAVAN, 2017; FERDOUS; CHOWDHURY; HOQUE, 2021). Arguably, the most famous one is Bitcoin's Proof of Work (PoW) and in use by many other networks. It boils down to a race between all members of the network in the search of a partial hash based on the hash of the latest block of the blockchain. When found, this value is used to validate a new block which is appended to the ledger and broadcasted to the whole network. In other

---

[1] Usually referred as simply "crypto".

[2] Although it was not called "blockchain" when it was published.

[3] Usually a peer-to-peer (P2P) network.

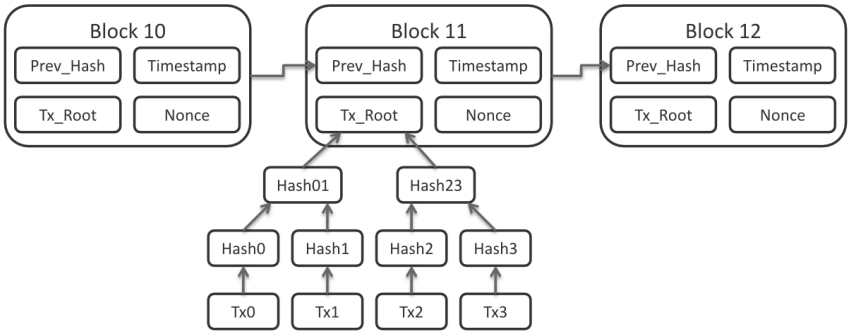


Figure 1 – An illustration of a basic blockchain ledger structure (MURCH, 2013).

words, it means that a node in the network spent computation power in order to find the hash. Because of that, it receives an incentive, some Bitcoins, in the case of the Bitcoin network. This way, this consensus incentivizes nodes to play by the rules and also makes very expensive for malicious nodes to affect the network.

2.1.2 Storage

Blockchains were not developed with generic data access in mind. Most networks use a key value store solution, as its underlying storage. Most notably, LevelDB[4], a fast key-value storage developed by Google. However, Hyperledger Fabric also allows the use of Apache's CouchDB[5], a JSON document-based database.

The choice for a key-value store was influenced by the fact that most data on a blockchain is identified and linked by hashes of the data itself. Figure 1 illustrates this idea. Each block has, included in itself, the hash of the previous block, together with the hash of the root node of a Merkle Tree and some metadata. The key to most data is the hash of the data itself. Therefore, faster key lookups were essential for performance and influenced the choice for LevelDB (PODGORELEC et al., 2020).

2.2 COMMON BLOCKCHAIN NETWORKS

This section will present some popular blockchain networks. Here, we will report a brief history of each of the networks while also describing the most important concepts of each one, mainly how the blocks, transactions and accounts are modelled in each. Finally, a brief description of the blockchain's topology will be shown.

2.2.1 Bitcoin

Bitcoin (NAKAMOTO, 2008) was the first popular blockchain network, launched in late 2008, early 2009. It is fully open source under the MIT license. Undoubtedly, it

[4] `https://github.com/google/leveldb`

[5] `https://couchdb.apache.org/`

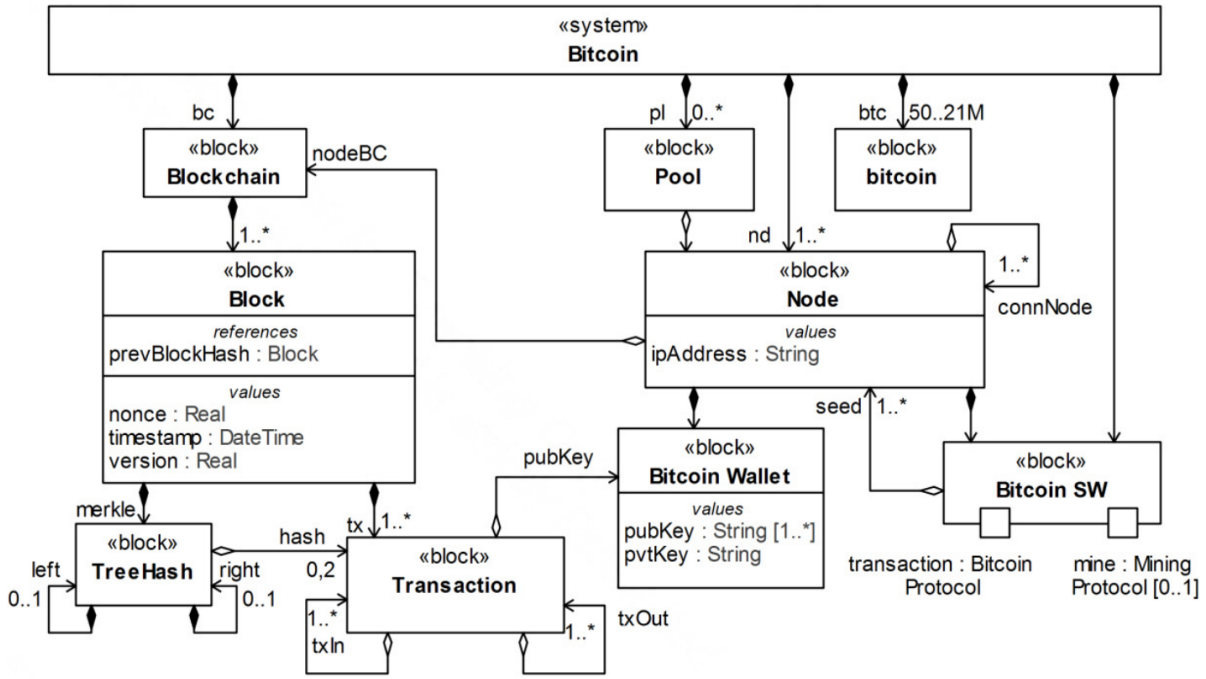
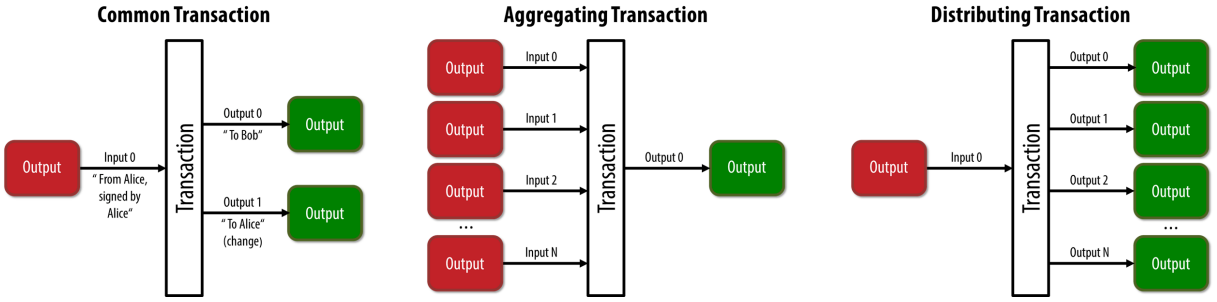


Figure 2 – Bitcoin's data model (ROTH, 2015).

is the most famous network. Envisioned by someone under the pseudonym of Satoshi Nakamoto. The real name of the author has yet to be known. It uses the Proof of Work (PoW) consensus algorithm and it is widely criticised by the network's enormous energy consumption. Today, the network is mostly used as a distributed and (quite) unregulated banking system. Its users can transact between themselves freely and without limits.

Bitcoin's data model is widely used as the basis when describing how blockchains work. Figure 2 shows a *highish* level UML diagram containing the basic components that are part of the network. Some of the key takeaways from it are the relations between blocks, transactions and wallets. Those relations and the data of each entity are the key components that this work will focus on. The other components of the figure get too specific for different networks and it will not be covered by this research.

Transactions are the most important part of Bitcoin's model. It is "where things happen". In this case, sending/receiving bitcoins. However, transactions are represented differently in Bitcoin. Transactions are composed of inputs and outputs. If you have 20 bitcoins and want to send 5 to another user you will end up generating (sort of) 2 transactions. One of 5 bitcoins for the user you want to send the money and one of 15 bitcoins for yourself. These are called inputs and outputs. In the previous example, 20 bitcoins is the input and the 15 and 5 are called outputs. One input may generate more outputs and one output may come from many inputs. Figure 3 shows a visualisation of some types of transactions. Figure 3 (a) is the most common transaction of sending some amount to someone. It generates 2 outputs, for the other user, the receiver, and one to itself, the sender. This is a bit counter intuitive from the classic "*user A sending amount X to user B*". Yet, in the end, most applications make all of this transparent to its



(a) Common transaction. (b) Aggregated transaction. (c) Dist transaction.

Figure 3 – Bitcoin transaction's input and outputs (ANTONOPOULOS, 2017, Ch. 2).

users. Figure 3 (b) and Figure 3 (c) show other types of transactions where many users are sending some bitcoin to a single user, generating a single output, and one user sending bitcoin to many users, generating lots of outputs, respectively. This system is known as Unspent Transaction Output (UTXO).

A consequence of the above is that, in transactions with multiple inputs and multiple outputs, you cannot know for certain who sent how much to whom. Let's say we have a transaction with 2 inputs, one input X of 2 bitcoins and one input Y for 3; and 5 outputs of 1 bitcoin each. You cannot map any of the inputs to any of the outputs directly. The only thing that is known is that both inputs contributed to all of the outputs in the transaction.

Blocks give structure to the set of all transactions in the network. A block is composed of two parts. A header, which holds metadata for the block, and a set of transactions. The average number of transactions in a bitcoin block is around 1.900. The transactions are stored in a Merkle tree and the block holds the hash of the root of said tree in its header. Besides, the block header holds important information regarding the block itself, like the timestamp of mining, difficulty, nonce and the previous block header's hash.

An account can be identified by its address, which is a hash of its public cryptographic key. Accounts are the real "players" of the network, the ones who make transactions. For its functionality and capacity to interact with the network, they are usually represented as a pair of keys, private and public, like in Figure 2. However, for the purposes of this work, we will only consider their addresses as a way to identify it and its transactions.

2.2.2 Ethereum

Ethereum (BUTERIN et al., 2014) is a blockchain network and distributed computing system. It was released in 2015 under the GPL v3 license and it is fully open-source. It used to use the Proof of Work (PoW) consensus algorithm, although a different one from Bitcoin. Now, it has since changed to use a Proof of Stake (PoS) consensus, an

event called “The Merge” by the community. It also differs from Bitcoin on block time, the average time the network takes to mine a block. Whereas in Bitcoin a block takes around 10 minutes[6] to be mined, in Ethereum a block takes around 12 seconds[7].

Yet, the biggest difference between Ethereum and Bitcoin is its capacity to execute so called *Smart Contracts*. Those “contracts” are simply computer programs that are executed, and validated, by the nodes in the network and allow the introduction of complex logic when transacting *ether*, the currency of the network. Those contracts are written in Solidity, a high level programming language based on JavaScript.

Smart contracts were first conceptualised by Szabo (1997). It works on the idea that if a node network can agree in some random pieces of data to be added in the list of blocks, it can also validate computer programs executions and its outputs. As the execution is limited to only use data from inside the chain itself, the execution of these computer programs should be the same in every node that has the same chain. This logic is used to validate smart contracts in the network. As a consequence, every single state that the contract will ever have will be permanently stored in the blockchain. Smart contracts provide a new paradigm where participants do not need to rely on any third parties to “force” the execution of a program. The whole network of nodes will act as the “enforcer”. The Ethereum network is the most famous example of a smart contract-based blockchain.

Ethereum, as most blockchains, has a data model similar to Bitcoin. It is composed of blocks, transactions and accounts[8]. Figure 4 shows an UML diagram of those components and its relations, and Figure 5 shows what composes each primitive of Ethereum’s blockchain. As we can see, there are many differences between Bitcoin and Ethereum’s blocks and transactions. First, the block’s header is much more complex, containing way more entries. Second, the body of the block contains a second data structure from the transactions, called *uncles*[9]. Finally, the transactions contain more information as well. This information is related to gas and smart contracts, which is not present on Bitcoin’s.

Even though there are many differences between blockchains, the basic building blocks remain very similar: blocks, transactions and accounts. These are the basic components that most blockchains use and should be the focus of the proposed work.

---

[6] Bitcoin is designed to always take around 10 minutes to mine a block. No matter the computational power of the network.

[7] Etherscan block times: `https://etherscan.io/chart/blocktime`

[8] Accounts are also called *Wallets*.

[9] Uncles are sometimes called *Ommers*.

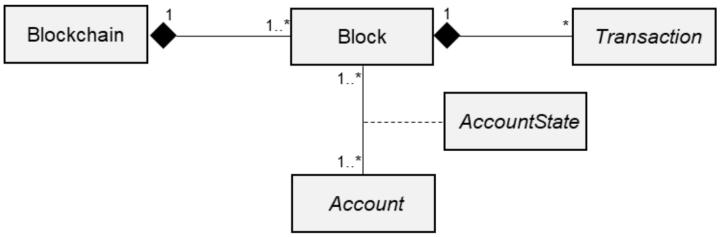


Figure 4 – UML diagram of Ethereum's components (OLIVÉ, 2020).

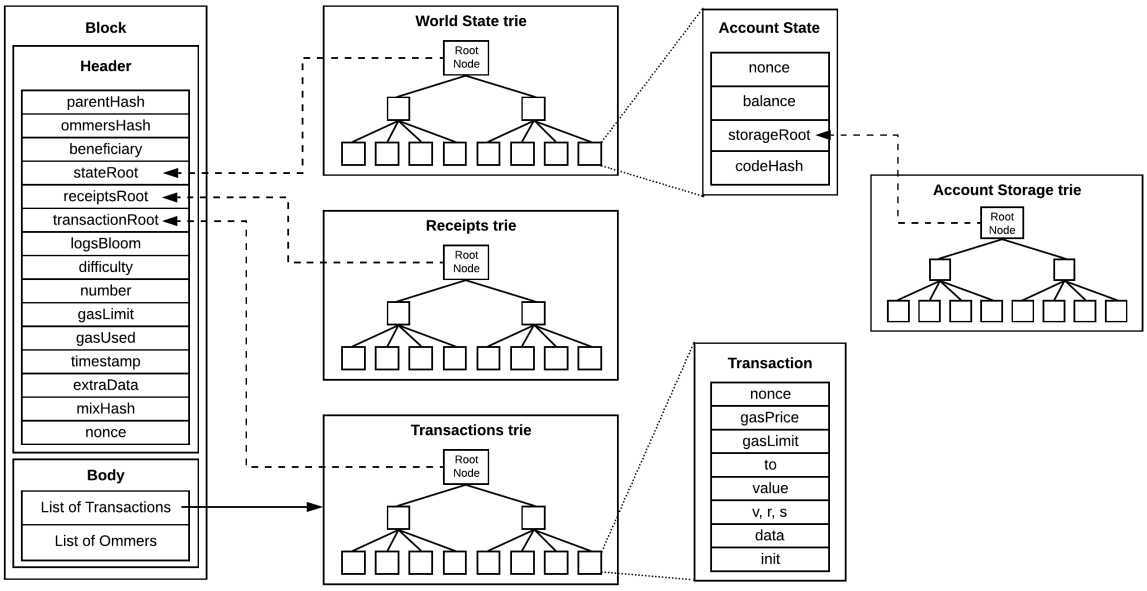


Figure 5 – Data composition of Ethereum's primitives (SALDANHA, 2019).

3 STATE-OF-THE-ART

This chapter elaborates on related works regarding blockchain data modelling. However, even with the big interest in blockchain, no previous study with specific focus on blockchain data modelling had been found by the authors.

The work of (HUANG et al., 2021), for example, performs a thorough analysis of many blockchain surveys and its subjects. However, even though there are plenty of surveys about blockchain data analytics and graph models, none of them highlights the considered blockchain data model.

(XIE et al., 2019), in turn, performs a survey on blockchain scalability challenges. This study does acknowledge the need for scalable data storage utilities, such as the Inter Planetary File System, by (BENET, 2014), and an off-chain distributed hash table by (ZYSKIND; NATHAN, et al., 2015). Both approaches discuss the removal of storage off the blockchain and into separated services in order to make the networks lighter. Nevertheless, none of the analysed approaches discusses scalability of data accessing based on a data model for blockchain as our work intends to do.

3.1 RESEARCH PROTOCOL

This section describes the considered research protocol and its results. The chosen protocol is the broad used systematic literature review (SLR) methodology by (KITCHENHAM, 2004). The SLR protocol comprises a set of ordered steps:

1. Definition of research questions;
2. Definition of the search string;
3. Definition of inclusion and exclusion criteria.

These steps are performed in a way to direct the research and provide a well-defined process for the review. The first step defines a set of questions that we hope to find answers to. These questions are the starting point of the SLR. We propose the following ones:

- What are the data models for blockchains?
- What are the techniques used for querying blockchain data?

Once defined the research questions, the next step is to define a search string. For this work, we initially attempted a few sets of keywords, such as “blockchain data model” and “blockchain model”. However, strings with the keyword “model” resulted in works related to modelling *systems* based on blockchain and not in the data model

itself, which is the focus of this research. Nonetheless, the search string “blockchain AND (query OR model)” was chosen because its results were more promising.

Usually, queries performed on bibliographic DBs return too many results. This way, on following the SLR protocol, we provide a set of inclusion and exclusion criteria in order to filter out researches that are not related or important to our work. Our proposed criteria are shown in Table 1. We do not consider the year of publication as an exclusion criterion because blockchain data management is a relatively new research topic. So, there are not old-fashioned works.

Table 1 – Inclusion and exclusion criteria

Inclusion	**Exclusion**
English and Portuguese only	Focuses on the economic aspect of crypto coins
Discusses performance of data access in popular blockchains	Has no performance considerations about the described approach
Discusses use cases for fast blockchain data querying	
Describes entities and models for blockchain data	

We submitted the search string to a set of bibliographic DBs. Results are shown in Table 2.

Table 2 – Search results for each bibliographic DB

DB	**Total results**
DBLP	24
Google Scholar	28.000
IEEE Xplore	150
Springer Link	2.013

Next, we narrowed down the result set to make the research viable. Based on the inclusion and exclusion criteria, we applied the following filters:

1. Selection of the first 25 results of each DB;
2. Title filtering;
3. Abstract filtering;
4. Fully reading.

The remaining papers were fully read. Table 3 shows the 13 selected papers after applying the aforementioned filters.

Table 3 – Resulting papers after filtering each DB

DB	Start	Title	Abstract	Full
DBLP	24	22	13	9
Google Scholar	25	8	3	3
IEEE Xplore	25	4	1	1
Springer Link	25	4	1	0

3.2 SELECTED WORKS

This section gives an overview of the 13 selected works by our research protocol.

(BARTOLETTI et al., 2017) proposes a tool, using Scala, for general purpose analytics on the blockchains of Bitcoin and Ethereum. It allows the synchronisation of views of blockchain data to common DBMSs, MongoDB or MySQL, for analysis. The tool was tested using Bitcoin and both DBMS. However, the authors failed to compare the performance of their solution against other approaches.

(LI et al., 2017) introduces EtherQL. EtherQL is a REST service that synchronises Ethereum blockchain data, such as accounts, blocks and transactions, into a MongoDB instance. The authors validate their implementation by performing aggregate and range queries unavailable at LevelDB when used by the Quorum blockchain network. The results show that EtherQL is almost twice as fast in all tests.

(XU et al., 2017) discusses the creation of an Educational Certificate BlockChain (ECBC). ECBC has a very specific use case, i.e., the emission of educational certificated by trusted parties. The transactions are modelled with a sender, receiver and some extra metadata, such as issuance timestamp. The senders are usually educational institutions, and receivers are usually students. The authors deemed efficient querying as a requirement for ECBC. In order to reach this requirement, they created MPT-Chain, a combination of Merkle and Patricia Trees indexing data structures. The experimental evaluation suggests that queries are fast when using this approach. However, the authors did not compare their results with any other available solution.

(BRAGAGNOLO et al., 2018) proposes Ethereum Query Language (EQL), an SQL-like language, coupled with a tool to execute it. This tool allows its users to perform complex information fetching from Ethereum. In order to make queries faster, the authors describe the creation of a binary search tree index. However, they did not specify if this index is created on the fly by queries being performed or beforehand. The tests show a decrease in performance when using the tool. Yet, the expressiveness of possible queries is the main point here. It allows user to easily fetch ranged and aggregated data from the blockchain using SQL queries.

(PRATAMA; MUTIJARSA, 2018) leverages the work of (LI et al., 2017). The authors developed an improved REST interface for Ethereum data that is synchronised

with MongoDB. It describes models for accounts, blocks and transactions. The authors argue that their main focus is to make development and use of the system from (LI et al., 2017) easier for others. The authors provided load tests, however, there is no comparison with either (LI et al., 2017) or Ethereum's default interfaces for aggregate queries and select queries.

On contrasting with previous works, (HAN et al., 2019) develops a Quorum[1] client that stores smart contract transactions in an embedded SQLite DB. It allows the use of complex SQL queries. Their evaluation against LevelDB made use of range queries, queries not available in LevelDB. The results show an increase around 16 times for fetching data when comparing the performance of the oficial client. However, the authors do not mention that their approach appears to double the disk usage because it stores duplicated data, one copy in LevelDB and other one in SQLite.

(LINOY et al., 2019) proposes a tool to query Ethereum block data by using SQL and Hadoop as the underlying processing layer. It uses an in-memory B^+Tree that indexes the block's and transaction's ids to find the Hadoop File System file paths for the original data. It uses Hadoop's MapReduce tasks to find and return the data concurrently and in a distributed way. It allows range and aggregate queries. The experimental evaluation suggests that the tool performance improves linearly with the addition of more nodes to the system.

(PENG et al., 2019) presents a middleware, called Verifiable Query Layer, that reorganises Ethereum blocks and transactions into a verifiable MongoDB instance. It makes the data malleable according to MongoDB capabilities. The authors show that the throughput of MongoDB is higher than Ethereum's client, but they did not perform more complex queries not available in such clients, like range and aggregate queries.

(QU et al., 2019) introduces a novel approach to enable spatio-temporal queries over a blockchain. Blocks are defined as a set of transactions where each one contains geographical coordinates (latitude and longitude) together with timestamps and the public key of the entity that generated the data. It stores the transactions as a Merkle KD tree that enables fast geographical queries such as range query, K-nearest neighbours, bounded K-nearest neighbours, and ball-point query. Tests were performed in comparison with "scan time space" and "scan space time", full scan on the data filtering by time and space and vice-versa, respectively. The results show that the custom data structure outperforms the scans, being three times faster.

In a similar way to (LINOY et al., 2019), (TRIHINAS, 2019) introduces Datachain, an open source framework for querying and manipulating data over blockchain networks. The author's implementation works with Hyperledger and BigchainDB[2]. Unlike Bitcoin, that only considers a data model based on the cryptocurrency, the "asset" is the smallest

[1] An Ethereum based blockchain with enterprise features, such as customizable consensus algorithms.

[2] `https://www.bigchaindb.com`

data unit of Datachain. An asset can be anything, tangible or intangible. This way, the models are entirely customizable for the use case of the network. However, the author defines the minimum requirements for common blockchain transactions: sender address, receiver address and asset address. The implementation allows its users to query data using easier and well-established interfaces, such as common Python libraries (numpy, pandas, etc.) and SQL. The evaluation shows that Datachain does not perform better than querying data directly from Hyperledger and BigchainDB. This is not a demerit, because the framework makes the data handling much easier. Yet, the framework implements an *async* mode that lets many parts of the underlying queries be performed in parallel, which then incurs in a bigger throughput compared to using the original clients directly.

(ZHOU et al., 2019) presents Ledgerdata Refiner (LR). LR is a library that extracts data from Hyperledger and parses them to be organised and stored in common DBMSs. The tests used Postgres as the DBMS. They demonstrate that the synchronisation between the blockchain and the DBMS is quite fast. However, the authors did not provide performance analysis on the executed queries.

The work of (OZDAYI; KANTARCIOGLU; MALIN, 2020) is the result project of a competition hosted by Multichain. Multichain is a Bitcoin compatible blockchain with extra features such as data streams. These data streams can be used to store arbitrary data into the blockchain. For this competition, the data placed in the streams, together with the common Bitcoin block data, is a set of activity logs consisting of a timestamp, node, id, ref-id, user, activity and resource. Although this data is not necessarily related to the blockchain itself, it shows that arbitrary data could be stored into blockchain systems. The query methods described by the authors use a set of reverse indexes and bucketization techniques. The reverse index is used to quickly find all data related to a user, for example. The bucketization, in turn, is used for range queries. At last, the implemented approach proved to be less efficient than using a common relational DBMS, like SQLite.

At last, (REN et al., 2020) describes a custom-built dual combination bloom filter[3] (DCOMB) index that can be constructed using the hash power of available hardware created for mining. This enables the conversion of all the computing power of blockchains, specially proof of work (PoW) ones, to be used for query processing. This approach focuses on Internet of Things applications. However, the authors say that it could be adapted for more generic cases. The implementation uses DCOMB to quickly find the correct block of the blockchain and query data inside that block based on indexed fields. They also present an experimental evaluation showing that their implementation is much faster than other techniques, such as indexing blockchain data into a known DBMS, MySQL. A drawback of this work is the lack of a data model description. It only

[3] "A probabilistic data structure for efficient insertion and query", (REN et al., 2020).

emphasises that the model contains timestamps.

3.3 WORK ANALYSIS

This section analyses the selected papers and compare related solutions. We organize it into two parts: comparison and discussion.

3.3.1 Comparison

Most works agree that blockchain querying is not optimal. Some of them propose solutions that make use of readily available DBMSs, such as MongoDB, PostgreSQL, MySQL, which run besides the original blockchain and duplicate its data in a structured manner. Table 4 shows, when applicable, the considered DBMSs.

Table 4 – DBMS usage

Work	**DBMS**
(BARTOLETTI et al., 2017)	MongoDB, MySQL
(LI et al., 2017)	MongoDB
(XU et al., 2017)	
(BRAGAGNOLO et al., 2018)	
(PRATAMA; MUTIJARSA, 2018)	MongoDB
(HAN et al., 2019)	SQLite
(LINOY et al., 2019)	
(PENG et al., 2019)	MongoDB
(QU et al., 2019)	
(TRIHINAS, 2019)	
(ZHOU et al., 2019)	PostgreSQL
(OZDAYI; KANTARCIOGLU; MALIN, 2020)	
(REN et al., 2020)	

Table 4 reveals that MongoDB is the most used DBMS. Most authors explain their choices arguing that MongoDB is highly scalable and good for rapid development. However, some works prefer relational DBMS.

We also notice the usage of custom and complex data indexing structures within blockchain networks to increase query performance. Table 5 shows the works that consider them.

Table 5 – Indexing data structures

Work	**Data structure**
(BARTOLETTI et al., 2017)	
(LI et al., 2017)	

(XU et al., 2017)	Merkle and Patricia tree
(BRAGAGNOLO et al., 2018)	Binary tree
(PRATAMA; MUTIJARSA, 2018)	
(HAN et al., 2019)	
(LINOY et al., 2019)	B+ tree
(PENG et al., 2019)	
(QU et al., 2019)	Merkle KD-tree
(TRIHINAS, 2019)	
(ZHOU et al., 2019)	
(OZDAYI; KANTARCIOGLU; MALIN, 2020)	Bucketization and reverse index
(REN et al., 2020)	Bloom filter

In some cases, the authors decided to implement data indexing structures into the blockchain software itself. This makes the networks independent of a DBMS and avoid duplicating data. However, it increases complexity of the network, in particular, when we try to add this feature to current networks, such as Ethereum.

Table 6 summarises the querying strategies adopted by the selected works. Some authors had developed libraries that only concern to make blockchain data easier to query and in a more expressive way, like the usage of SQL or SQL-like languages, as well as and REST interfaces. Additionally, they state that performance is not the main point of the project. Instead, they just intend to facilitate data querying.

Table 6 – Querying strategies

Work	**Querying Strategies**
(BARTOLETTI et al., 2017)	
(LI et al., 2017)	EtherQL (REST API)
(XU et al., 2017)	
(BRAGAGNOLO et al., 2018)	Ethereum Query Language (SQL)
(PRATAMA; MUTIJARSA, 2018)	REST API
(HAN et al., 2019)	
(LINOY et al., 2019)	SQL + Hadoop
(PENG et al., 2019)	
(QU et al., 2019)	
(TRIHINAS, 2019)	SQL + Python libraries
(ZHOU et al., 2019)	
(OZDAYI; KANTARCIOGLU; MALIN, 2020)	
(REN et al., 2020)	

Unfortunately, not all works define the entities explicitly, i.e., blockchain data mod-

Table 7 – Considered data entities.

Work	Block	Transaction	Account	Smart Contract	L
(BARTOLETTI et al., 2017)	X	X	X	X	Bitc Eth
(LI et al., 2017)	X	X	X	X	Eth
(XU et al., 2017)	X	X	X		Cus
(BRAGAGNOLO et al., 2018)	X	X	X	X	Eth
(PRATAMA; MUTIJARSA, 2018)	X	X			Eth
(HAN et al., 2019)		X		X	Quo
(LINOY et al., 2019)	X	X			Eth
(PENG et al., 2019)	X	X			Eth
(QU et al., 2019)	X	X	X		Cus
(TRIHINAS, 2019)	X	X	X		Hyp
(ZHOU et al., 2019)	X	X			Hyp
(OZDAYI; KANTARCIOGLU; MALIN, 2020)	X	X			Mul
(REN et al., 2020)	X	X			Cus

els are presented in a tangential way. For some of them, we had to search for the source code, if available, and check what entities were implemented there. However, many works do not provide the source code. In this case, we had to infer entities based on the broad descriptions or tests reported. Table 7 shows the entities considered by each work. As stated before, blocks and transactions are the basic ones. Accounts, in turn, are usually an address with a balance count, and can be derived from the whole list of blocks. However, for performance issues, some blockchain data models keep an updated list of all accounts and their balances separately.

It is worth mentioning that all studies make use of the transaction entity. This is due to the fact that the transaction contains the core information of the distributed ledger, which usually is an account sending assets to another account. Some studies also consider smart contracts as an entity (BARTOLETTI et al., 2017; BRAGAGNOLO et al., 2018; HAN et al., 2019; HECTOR; BORIS, 2020; KRUIJFF; WEIGAND, 2017; LI et al., 2017). This is the case of some blockchains, like Ethereum and Quorum, where smart contracts are widely used.

Finally, we compare the data models considered by the authors. Unfortunately, not all of them define the model directly, but many define the data they want to search as being building blocks of common networks, mostly blocks, transactions and accounts. For the works that do not specify a data model, it is possible to infer some of the data entities based on the used blockchain.

Table 8 – Data models categorised by ledger or customized

Work	Ledger	Customized
(BARTOLETTI et al., 2017)	Bitcoin	

(LI et al., 2017)	Ethereum	
(XU et al., 2017)		X
(BRAGAGNOLO et al., 2018)	Ethereum	
(PRATAMA; MUTIJARSA, 2018)	Ethereum	
(HAN et al., 2019)	Quorum	
(LINOY et al., 2019)	Ethereum	
(PENG et al., 2019)	Ethereum	
(QU et al., 2019)		X
(TRIHINAS, 2019)		X
(ZHOU et al., 2019)	Hyperledger	
(OZDAYI; KANTARCIOGLU; MALIN, 2020)	Multichain	
(REN et al., 2020)		X

Table 8 shows that Ethereum is the blockchain of choice in this topic. As one of the most popular networks today, it is natural that many works tend to consider it. Besides, its smart contract notion raised several smart contract specific studies, such as (HAN et al., 2019), which is performed on Quorum, an Ethereum-based blockchain. Another important point is its ease of use. It is very easy to boot up a local node and start prototyping ideas. Finally, its community is very active and documentation is plenty. We did not include BigchainDB in the Table 8 because, different from common blockchains, BigchainDB tries to emulate properties of blockchains, like immutability, within a database. This way, it works very differently from common blockchains and is out of the scope of this project.

3.3.2 Discussion

From the previous comparison of the selected works, we see that blockchain querying is a hot topic in the academy. Based on the proposed solutions, we can classify the approaches into three categories:

1. Blockchain to DBMS data synchronisation;
2. Built-in indexing structures;
3. Query interfaces.

Category 1, as shown in Table 4, is a technique used by several works. In case of relational DBMSs, it forces developers to use a rigid data model for blockchain data in order to insert data into tables and sometimes create tables for relationships. Coupling it with works like (OLIVÉ, 2020), where the authors proposed a well-defined schema for blockchain data, this can become a powerful combination. In the case of NoSQL

DBMSs, they relax the data model definition. Nevertheless, it still maintains some common concepts between approaches, such as blocks, transactions and accounts, which are the basis for building a useful DB.

Also, the use of largely used DBMSs seems like a good approach to make data easily accessible by interested parties. But, besides making the data easy to query, this approach has a pitfall. It requires duplicating data from the blockchain key-value store, usually LevelDB, to another DBMS. Of course, storage is a cheaper resource. However, a blockchain, as an append-only data structure, is always growing, making storage management a future problem.

Regarding category 2, we argue that the advantage of this approach is a bit more complicated. The usage of built-in index structures for blockchain data is, of course, positive as it provides faster querying as a native feature. However, index management on existing networks may be very hard. Some blockchain networks, like Bitcoin, Ethereum and Ripple, already have mature code bases. Developers should debate, very carefully, if the advantages of having faster queries make the effort to insert this feature worth it, or if they should let it to be managed off chain, like the DBMS approaches of category 1.

Finally, category 3 shows us that there is a need for easier data accessing. The selected works claim that native clients lack interfaces for expressive queries when fetching useful data from the network. Besides, we notice a lack of useful queries in the key-value storage used by blockchains. This kind of storage is fast and reliable for the network to work properly. However, it does not serve well when users want to perform analysis on its data, and a new interface layer between users and the key-value storage could be a promising research issue.

Even though we identified some works where data can be queried by using common DBMS, the approaches still lack a well-defined data model for blockchains.

4 PROPOSAL

4.1 PROBLEM

Blockchains were not developed with generic data access in mind. They use efficient key-value stores in order to write and read specific data, mostly transactions, blocks and accounts identified by their own hashes. Even though this approach is fast and efficient for the networks' use case, it lacks useful data manipulation features, such as generic queries, like temporal queries and aggregations, such as "How many transactions were made to an specific account, X, last week?". It is completely possible to iterate over all entries in the blockchain and filter out what is wanted or needed, but this approach is terribly inefficient and slow, as the sizes of blockchains are getting bigger and bigger. Besides, the lack of proper tooling, like common query APIs, such as SQL, also makes this approach cumbersome.

Yet, not being able to access the data effectively and efficiently is only part of the problem. There is a lack of a standard data model for representing blockchain components. Each chain uses its own definition of a block and transaction. Usually, this definition only makes sense for the inner workings of the chain itself, but most of its data is not useful when one would like to perform analytics on it. For example, the concepts of "logsBloom", used to store a filter to identify logs emitted by a transaction, and "gas", used to describe how much the transaction did cost to be executed in the network, only make sense for the Ethereum blockchain, as others do not have those features.

4.2 PROPOSED SOLUTION

In order to resolve the lack of a standardised data model that encompasses different blockchains and its main components, such as blocks, transactions, accounts and others, i.e., we propose a unified data model. With a good data model in hands, it should be straightforward to extract, transform and load (ETL) data into any system with common query interfaces.

We first need to set the requirements for this data model:

1. The model should map the 3 main components of a blockchain: blocks, transactions and accounts;
2. The model should contain all of the original data extracted from the blockchain;
3. The model should be easy to import to any DBMS;
4. The model should allow us to perform queries that are hard/inefficient to do using the official clients for the networks;

For (1), we want our model to map the basic constructs of blockchains, which are blocks, transactions and accounts; For (2), we want our model to not loose extra data that is specific to a blockchain implementation. For example, Ethereum's "gas" and Bitcoin's "bits" attributes. With this data included, we can still query the original data for more specialised analysis. We included item (3) as a way to make our model generic enough to not be bounded by relationships, for relational databases, or other database concepts such as documents, for document-based databases. This way, our model should be easily importable to multiple types of DBMS, such as PostgreSQL or MongoDB. For (4), our model should allow us to perform aggregated queries. Those queries are usually hard, or non performant, to do in the blockchains itself. This is due to the key-value stores usually used by blockchains and also the nature of the data.

The main focus of the proposed model is that we should be able to query information regarding transactions. Who sent? What amount? To whom? When? So our model will focus on that.

It is important to that some questions are better suited for different types of databases. For example, aggregation and time sensitive queries, such as "Which account made the most transactions last week?", are easily answerable when the data is on a relational or document based database. Yet, questions which want to find out where the values of transaction are ending up, "follow the money" questions, are better suited for graph based databases. An example question would be "From starting transaction X, which accounts possibly received money from it on later transactions?".

4.3 BASE DATA MODEL

Based on the requirements previously defined, we start by first defining the conceptual model. The conceptual model defines the entities and its relationships. Afterwards, we will define the logic model. The logical model will be the one imported to databases.

Figure 6 shows the UML diagram for the mapped entities. We will be considering only the Block, Transaction and Account entities. Those are the main entities observed in the survey shown on Chapter 3. A block contains a set of transactions, while a transaction can be between multiple accounts. In the case of Bitcoin, for example, multiple accounts can send tokens to multiple other accounts. In the case of Ethereum, a single account can send to only one other account.

Figure 7 shows our proposed logical model based on the conceptual model from Figure 6. In order to link a transaction to its respective block, we need to include the block's hash as an attribute[1]. Transactions do not need a timestamp, as all transactions in a block are considered to be made at the exact same time, the block's time. Also, as

[1] This is needed if our model is to be loaded in a relational database. For document based databases, this is unnecessary as the transactions would be part of the same document.

Figure 6 – UML diagram showing the minimal proposed conceptual model.

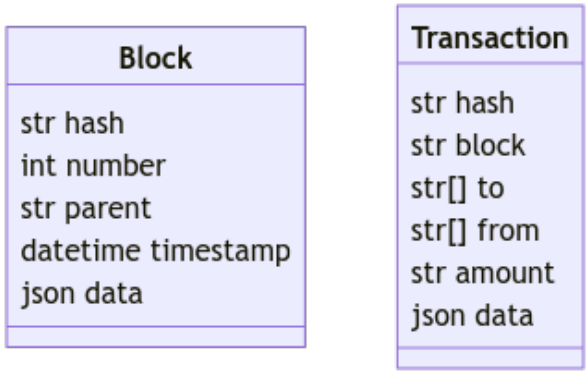


Figure 7 – Proposed logical data model.

a blockchain is a linked list of blocks, each block will contain the hash of the previous block. Table 9 shows details of each of the attributes.

4.4 EXTRACT, TRANSFORM AND LOAD (ETL)

With the data model in hands, we implement an ETL process which extracts data from network nodes, transform it to the model defined in Section 4.3 and load it to a DBMS. Finally, when the data is loaded, an user can perform analysis on it by using the DBMS's interface, such as SQL. Figure 8 illustrates this process.

In the extract step, for each blockchain, we will create a tool, or use existent solutions, to extract the data from it. This process needs to be performed on a per

Table 9 – Table showing the modelled attributes and their meanings.

Attributes	Details
Block.hash	The hash of the block
Block.number	The number of the block
Block.parent	The hash of the parent/previous block
Block.timestamp	The time when the block was mined
Block.data	Extra chain specific data
Transaction.hash	The hash of the transaction
Transaction.block	The hash of the block where this transaction was placed
Transaction.from	The senders of the transaction
Transaction.to	The receivers of the transaction
Transaction.amount	The amount transacted
Transaction.data	Extra chain specific data

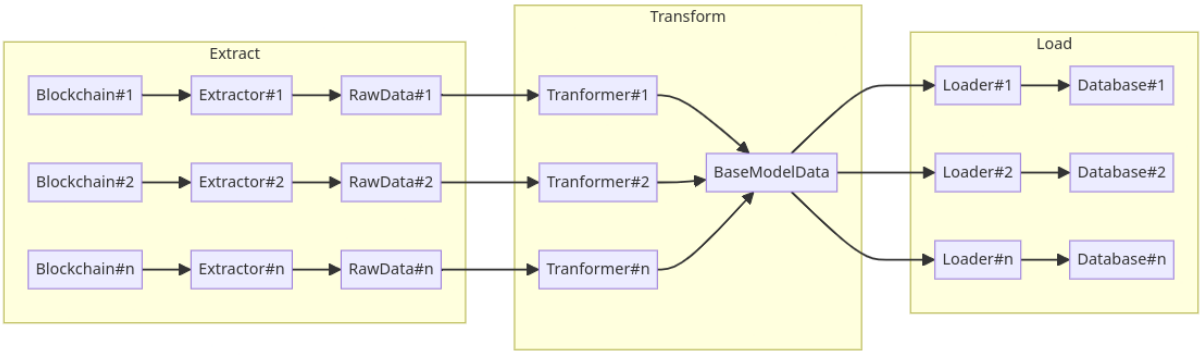


Figure 8 – Proposed extract, transform and load process.

chain basis, because there is no unified interface to interact with all blockchains. For the transform step, we will create transformers, again, for each blockchain in order to map the raw data from the extract step to our defined model. Finally, in the load step, we will load the data to databases using readily available tools to do so.

The rest of this chapter will go as follows, first we will describe the Extract and Transform steps for Ethereum, than Bitcoin. As the Load step will only be different on a per DBMS case, we will discuss it after showing the Extract and Transform steps for each blockchain.

4.4.1 Ethereum

This section explains the ETL steps taken for the Ethereum blockchain.

4.4.1.1 Extract

The first step of the ETL process is to extract the blockchain data from the Ethereum network. In order to do that, we first start a node, locally, that will sync up with Ethereum's mainnet, the main network used by all nodes in the public Ethereum blockchain. This process can be done in multiple ways, however, we decided to simply start a Docker container by the official Ethereum implementation[2], Geth. We booted up the node and waited a a few hours until it had, at least, the first 1 million blocks synced. This way, we have enough data to work with.

Once the syncing was finished, we used the ethereum-etl (ETHEREUM-ETL, 2023) open source project to extract the data from our local node into JSON files. This way, the data is easier to work with. We executed the following command in order to extract the first 1 million blocks, together with its transactions:

```
ethereumetl export_blocks_and_transactions \
    --start-block '0' \
    --end-block '1_000_000' \
    --provider-uri 'http://localhost:8545' \
    --max-workers 8 \
    --blocks-output 'eth_blocks.json' \
    --transactions-output 'eth_transactions.json'
```

Listing 1 – Command used to extract the first 1 million blocks and transactions from Ethereum network.

Once it was done, we were left with two files, containing information about the first 1 million blocks and its transactions. The models exported by ethereum-etl can be seen in Figure 9.

On creating an entity relationship diagram, we can see that the Ethereum chain

[2] Ethereum client Docker: `https://hub.docker.com/r/ethereum/client-go`.

ETH_BLOCK	
int	number
str	hash
str	parent_hash
str	nonce
str	sha3_uncles
str	logs_bloom
str	transactions_root
str	state_root
str	receipts_root
str	miner
int	difficulty
int	total_difficulty
int	size
str	extra_data
int	gas_limit
int	gas_used
DateTime	timestamp
int	transaction_count
int	base_fee_per_gas
str	withdrawals_root
str	withdrawals

ETH_TRANSACTION	
str	hash
str	nonce
str	block_hash
str	block_number
int	transaction_index
str	from_address
str	to_address
int	value
int	gas
int	gas_price
str	input
DateTime	block_timestamp
int	max_fee_per_gas
int	max_priority_fee_per_gas
int	transaction_type

Figure 9 – Models exported by ethereum-etl.

is very simple. It contains blocks and each blocks contains 0 or more transactions. All transactions are always related two accounts, one sending funds, while the other is the one receiving the funds. Figure 10 illustrates these relationships.

As we can see, the model extracted by ethereum-etl is very similar to the base model defined in Section 4. The only difference is that there is no “Account” entity. The account is merely an identifier in the transaction and blocks. Because of that, accounts do not have a state in itself, such as the current balance. However, the balance can be calculated by iterating over the blocks and checking all of the transactions.

4.4.1.2 Transform

Now that we have the data, we need to transform it. The transformation step is a conversion from the model extracted to our base model.

For Ethereum, this process is rather simple, as the base model is very similar to Ethereum’s. So, we will need to move and rename some of its attributes. Table 10 shows the mapping for blocks. The Ethereum column shows the names of the block attributes extracted from Ethereum, the Base Model column shows where did we map all of the attributes from Ethereum extracted to the base model. As discussed, not all attributes from all blockchains make sense in the base model, but we add those into the “data” attribute, which will contain custom chain data on a per chain basis. For example,

Figure 10 – Relation of extracted models from Ethereum.

"gas" related attributes only makes sense for Ethereum, so they will be added there along with other attributes not shown in Figure 7.

Table 11 shows the mapping for the transactions. It follows the same structure as Table 10, the Ethereum column shows the attributes extracted from Ethereum, and the Base Model column shows where we are mapping those attributes to the base model.

4.4.2 Bitcoin

This section explains the steps taken for the Bitcoin blockchain.

4.4.2.1 Extract

Similar to what was done for Ethereum in Section 4.4.1.1, we will start a local node for Bitcoin. We decided to use the official Docker image for Bitcoin[3], bitcoin-sv.

Once the syncing was finished, we used the bitcoin-etl (BITCOIN-ETL, 2023) open source project to extract the data from our local node into JSON files. Here is the command used to export the data:

```
1 bitcoinetl export_blocks_and_transactions \
2     --start-block '0' \
3     --end-block '100_000' \
4     --provider-uri 'http://bitcoin:password@localhost:8332' \
```

[3] Ethereum client Docker: `https://hub.docker.com/r/bitcoinsv/bitcoin-sv`.

Table 10 – Table showing the mapping of block attributes from Ethereum model to Base Model.

Ethereum	Base Model
number	number
hash	hash
parent_hash	parent
nonce	data.nonce
sha3_uncles	data.sha3_uncles
logs_bloom	data.logs_bloom
transactions_root	data.transactions
state_root	data.state_root
receipts_root	data.receipts_root
miner	data.miner
difficulty	data.difficulty
total_difficulty	data.total_difficulty
size	data.size
extra_data	data.extra_data
gas_limit	data.gas_limit
gas_used	data.gas_used
timestamp	timestamp
transaction_count	data.transaction_count
base_fee_per_gas	data.base_fee_per_gas
withdrawals_root	data.withdrawals_root
withdrawals	data.withdrawals

Table 11 – Table showing the mapping of transaction attributes from Ethereum model to Base Model.

Ethereum	Base Model
hash	hash
nonce	data.nonce
block_hash	block
block_number	data.block_number
transaction_index	data.transaction_index
from_address	from
to_address	to
value	amount
gas	data.gas
gas_price	data.gas_price
input	data.input
block_timestamp	timestamp
max_fee_per_gas	data.max_fee_per_gas
max_priority_fee_per_gas	data.max_priority_fee_per_gas
transaction_type	data.transaction_type

```
    --chain 'bitcoin' \
    --max-workers 8 \
    --blocks-output 'btc_blocks.json' \
    --transactions-output 'btc_transactions.json'
```

Listing 2 – Command used to extract the first 100.000 blocks and transactions from Bitcoin network.

However, exporting all the data needed from Bitcoin is a two step process. After extracting the transactions, we need to enrich them. The raw transaction data does not contain all the information needed for us to construct our base model. For example, we cannot know “who sent amount X to who”. The bitcoin-etl project contains a function to help us with that, called “enrich_transactions”. This function takes the transactions we already exported and enriches with more info from our node. So we execute the following command to do so:

```
bitcoinetl enrich_transactions \
    --provider-uri 'http://bitcoin:password@localhost:8332' \
    --chain 'bitcoin' \
    --max-workers 8 \
    --transactions-input 'btc_transactions.json' \
    --transactions-output 'btc_transactions_enriched.json'
```

Listing 3 – Command used to enrich the transactions from Bitcoin network.

Once the export finished, we were left with 2 files, containing information about all the blocks and transactions for first 100 thousand blocks in the chain. The models exported by bitcoin-etl can be seen in Figure 11.

Differently from Ethereum, Bitcoin has a different relationship between its entities. For example, each transaction contains a set of inputs and outputs, as explained in Section 2.2.1.

4.4.2.2 Transform

With the data exported, we start the transforming step in order to convert the Bitcoin model extracted to our base model.

The process is similar to what was done to Ethereum, we need to map the attributes from Bitcoin to our base model. However, because the model for Bitcoin is considerably different, we need to perform some further processing. For example, bitcoin-etl does not export the parent block’s hash in the current block. So, using the block’s number, we add this information. Similar to Ethereum’s, we move and rename some of the attributes. Table 12 shows the mapping from Bitcoin blocks to our base model. Table 13 shows the mapping of transactions.

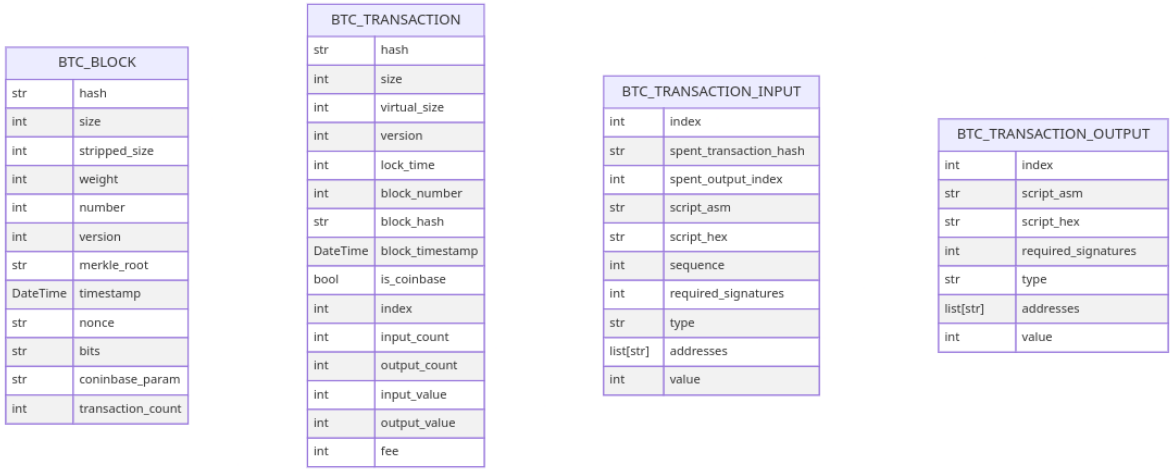


Figure 11 – Models exported by bitcoin-etl.

Table 12 – Table showing the mapping of block attributes from Bitcoin model to Base Model.

Bitcoin	Base Model
hash	hash
size	data.size
stripped_size	data.stripped_size
weight	data.weight
number	number
version	data.version
merkle_root	data.merkle_root
timestamp	timestamp
nonce	data.nonce
bits	data.bits
coninbase_param	data.coinbase_param miner
transaction_count	data.transaction_count

Table 13 – Table showing the mapping of transaction attributes from Bitcoin model to Base Model.

Bitcoin	**Base Model**
hash	hash
size	data.size
virtual_size	data.virtual_size
version	data.version
lock_time	data.lock_time
block_number	data.block_number
block_hash	block
block_timestamp	data.block_timestamp
is_coinbase	data.is_coinbase
index	data.index
outputs[].addresses	to
inputs[].index	data.inputs[].index
inputs[].spent_transaction_hash	data.inputs[].spent_transaction_hash
inputs[].spent_output_index	data.inputs[].spent_output_index
inputs[].script_asm	data.inputs[].script_asm
inputs[].script_hex	data.inputs[].script_hex
inputs[].sequence	data.inputs[].sequence
inputs[].required_signatures	data.inputs[].required_signatures
inputs[].type	data.inputs[].type
inputs[].addresses	from
inputs[].value	data.inputs[].value
outputs[].index	data.outputs[].index
outputs[].script_asm	data.outputs[].script_asm
outputs[].script_hex	data.outputs[].script_hex
outputs[].required_signatures	data.outputs[].required_signatures
outputs[].type	data.outputs[].type
outputs[].value	data.outputs[].value
input_count	data.input_count
output_count	data.output_count
input_value	data.input_value
output_value	amount
fee	data.fee

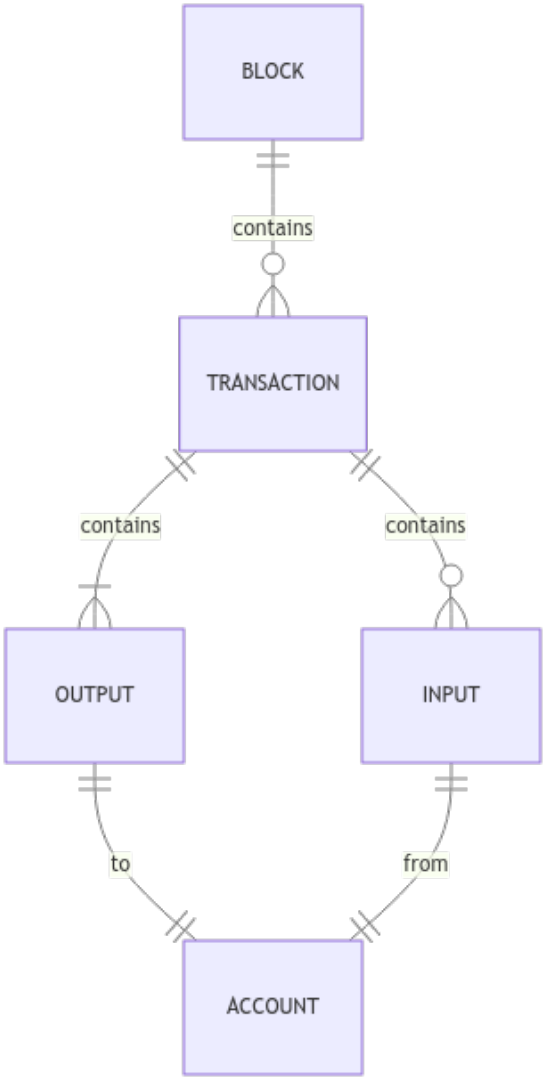


Figure 12 – Relation of extracted models from Bitcoin.

4.4.3 Load

The load step of the ETL process has its own section because the process is the same for every blockchain. However, it differs between databases. We've decided to load the data into three different DBMS: PostgreSQL, a relational database; and MongoDB, a document based database.

This section is organised as follows: we have a section for how we mapped the data to each of the DBMS chosen. Starting with PostgreSQL, followed by MongoDB.

4.4.3.1 PostgreSQL

For PostgreSQL we have created a schema based on two tables, a "blocks" table and a "transactions" table. The blocks table contains all the information about each block. The transactions table contains all the information about each transaction. For the blocks table, the primary key is the hash of the block, as it guarantees that each

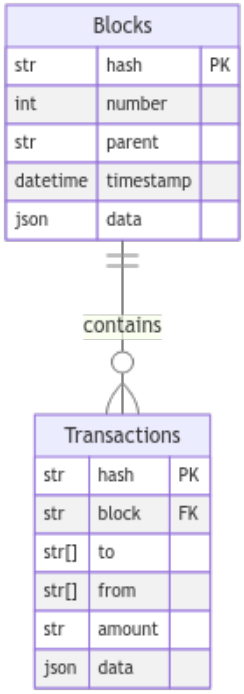


Figure 13 – Relational model used for PostgreSQL.

block will have a unique hash. The same can be said for the transactions table, where each transaction contains its own unique hash, so it is also used as a primary key. Finally, the use the block's hash as the foreign key in order to create the relationship between blocks and transactions. Figure 13 shows the relationship model used for PostgreSQL.

4.4.3.2 MongoDB

For MongoDB, the stored model differs a bit from the PostgreSQL one. Instead of having two distinct collections, we have a single collection of documents, the "blocks" collection. Each block document contains the transactions of said block in itself. This goes more in line with data modelling for document based databases.

Figure 14 shows the relationship for MongoDB. The only difference between the MongoDB model and the PostgreSQL model is that the transaction entity is contained inside the block entity.

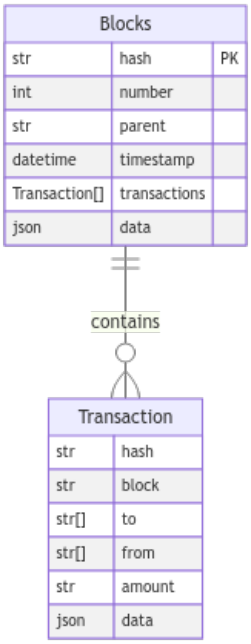


Figure 14 – Document model used for MongoDB.

5 ANALYSIS

In order to do so, we have developed two sets of questions. The first set contains questions that are easily answerable using the official Ethereum blockchain. While the second set is questions that are hard to answer. For example, queries that perform some sort of aggregation.

The following questions were devised based on the remote procedure calls (RPCs) available to interact with Ethereum nodes. Those questions are considered easy to answer as the data is stored in key value stores enabling the nodes to quickly return this data. These questions will be henceforth named “easy” queries or questions:

1. How many transactions were in block X?
2. Which block contains transaction Z?
3. When was the block X mined?
4. Who sent transaction Z?
5. Who received transaction Z?
6. How much was sent in transaction Z?

The following questions were devised based on what we cannot easily answer using a Ethereum node. Mostly, queries that perform some aggregation on the data. In order to perform aggregations on Ethereum directly, we would need to perform a scan as the data is not indexed for these types of queries. These questions will be henceforth named “hard” queries or questions:

1. Which account made the most transactions?
2. Which account made the most transactions last week?
3. Which day had the most transactions? Or blocks?
4. Which block has the most transactions?
5. What are the biggest transactions?
6. What is the biggest transaction?

All the queries used for each of the databases and Ethereum can be found in the appendix.

The evaluation was executed in a Linux computer running Arch Linux, kernel version 6.5.9-arch2-1. The CPU is a Intel i7-8550U and the machine has 20Gb of memory. The data extracted from the networks, using the steps shown in Chapter 4,

has already been inserted in the databases and the proper indexes were created for each database.

However, first we need to have baselines in order to be able to compare the performance of our queries with the network client. For this, we are going to use the set of easy questions from the previous section. The code used to interact with the Ethereum node is listed in Appendix E. The queries used for PostgreSQL and MongoDB are listed in Appendices B and D, respectively. Figure 15 shows a comparison of execution times. As we can see, PostgreSQL takes the longest to answer. So much that the values for MongoDB and Ethereum are almost invisible in the graph.

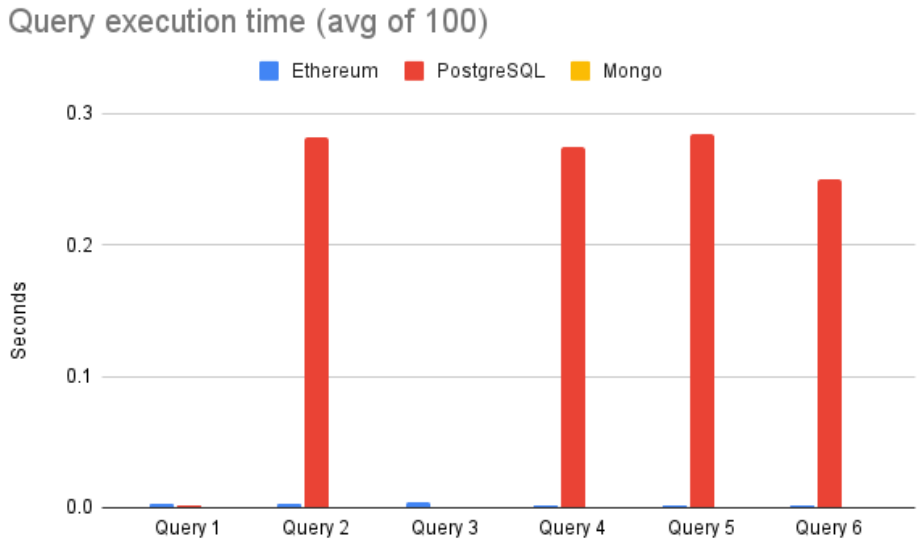


Figure 15 – Easy queries execution times for Ethereum, PostgreSQL and MongoDB. Data in Appendix G.

In order to show the performance comparison between MongoDB and Ethereum, Figure 16 shows only MongoDB and Ethereum query execution times. As we can see, even though Ethereum uses a fast key value store, MongoDB was still able to answer the queries faster than Ethereum.

Now, in order to have a way to compare the query execution performance for the hard queries from DBMS with the Ethereum client, we will create a baseline measure. As our hard queries contain aggregations, we will perform a “scan” on the Ethereum chain. The scan is used as there is no other way to perform those queries, while using the official clients, without having to scan the whole chain. We scan the first 100.000 blocks from Ethereum. No data analysis, filtering, or aggregation will be performed. This way, we will be able to see how fast we can fetch the data from the chain. It is safe to assume that any other steps added, like filtering or aggregations, would only impact the time negatively. Table 14 shows the times taken to get the first 100 thousand blocks from a local Ethereum node. It is safe to assume that to extract all of the 1 million blocks we loaded into our databases would take approximately 10x the time.

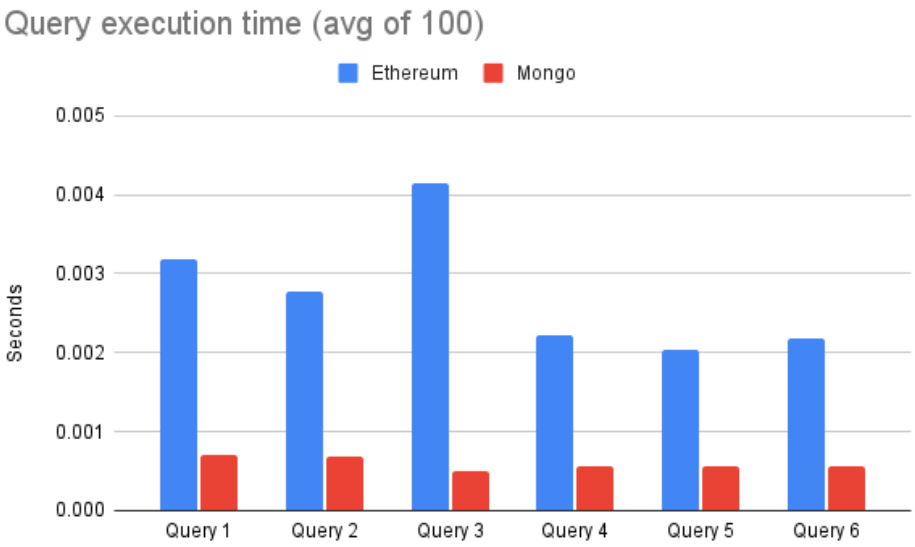


Figure 16 – Easy queries execution times for Ethereum and MongoDB.

Type	Time (s)
Sequential	228.98
Concurrently (1024 workers)	184.97
Concurrently (100k workers)	155.73
Parallel (4 CPUs)	89.32
Parallel (8 CPUs)	66.32

Table 14 – Time to perform a manual scan for the first 100 thousand blocks from Ethereum.

With the baseline metrics in hand, we can start making queries to the data loaded in DBMSs. For PostgreSQL, we will use SQL to query the data. All queries are in Appendix A. For MongoDB, we will use MongoDB's custom query syntax, which is based on JavaScript. All queries are in Appendix C. Figure 17 shows the execution time comparison between MongoDB and PostgreSQL.

When comparing the results shown in Figure 15 and Figure 16, we can see that MongoDB, in some cases, is more performant than the official Ethereum client. For example, MongoDB showed better results on the "easy" queries, even compared with the official Ethereum client. PostgreSQL takes longer than the Ethereum client, even though indexes are created and used by the query planner.

Yet, we see the PostgreSQL and MongoDB shine when we need to perform the "hard" queries. See Figure 17. Both databases perform the hard queries 2 orders of magnitude faster than the Ethereum client. MongoDB has the advantage here as well, fetching the data more than 10 times faster than PostgreSQL on some cases. This is due to the fact that the data, in PostgreSQL, is stored in two distinct tables. So any time we want to query for data which requires information from both, blocks and transactions, we need to perform a "join", which is not cheap.

MongoDB, on the other hand, because of its document oriented architecture,

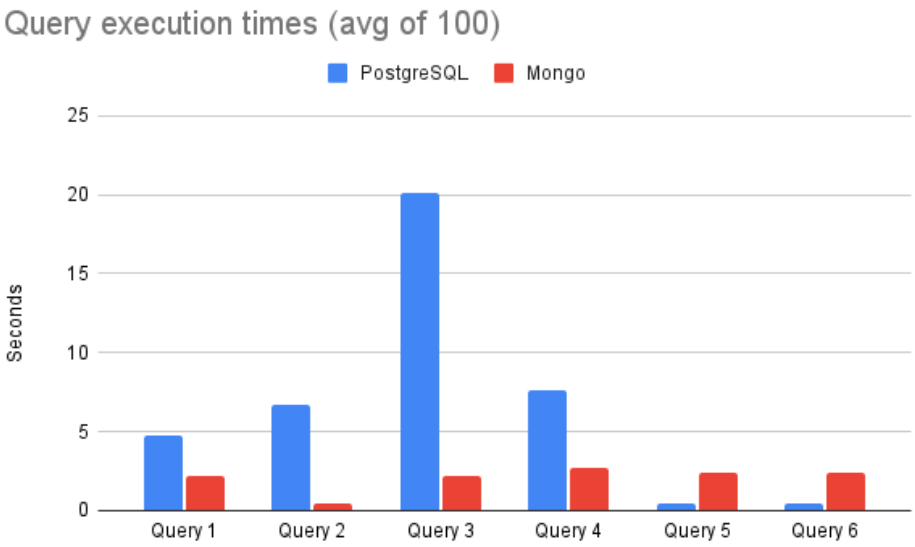


Figure 17 – Hard queries execution times for PostgreSQL and MongoDB. Data in Appendix F.

allows us to easily get both data, blocks and transactions, without the need of expensive joins. For example, when querying timestamps for transactions, information that is contained in the “block” entity, we simply find the transaction we want and get its block from the same document. While for PostgreSQL, we need to perform a “join” between the table transactions’ “block” attribute with the “hash” attribute of the block table.

6 CONCLUSION

Blockchain data management and analysis still is an open issue. However, its use cases are very relevant. Fraud detection, wallet de-anonymization, transactions and inter-chain analysis are just the simpler ones. Native blockchain clients are not well equipped to perform such tasks.

Following this context, this work presents the state-of-the-art regarding blockchain data modelling. We see the usage of well-established technologies, such as different DBMSs, to better store, index and query blockchain-related data, as well as common interfaces, like REST and SQL, to facilitate interaction instead of using native clients, which are not well equipped to handle such tasks. It highlights the importance of combining the best of those worlds to design blockchain data management solutions.

This work also present a pioneer survey on data modelling for blockchain information, as well as a proposal of a unified data model used to map information from different blockchains under the same schema, making it easier to create tools to interact with these data. A set of ETL tools were developed in order to map data from different blockchain networks to the proposed model and load it into multiple database technologies.

Nevertheless, we failed to find a work that details a *blockchain data modelling process*, including the usage of typical DB design methodology starting by a conceptual model and moving to a logical model to be further implemented over a DBMS. As described in Section 3.2, DBMSs with different data models are considered (e.g., document, key-value and relational), but there is no focus on how their corresponding logical and physical schemas are designed. One promising research direction, in this context, is to propose a *polystore design methodology* so that different logical models, as well as related DBMSs, could be considered for maintaining parts of a conceptual schema for blockchains. The choice for one or more DB models could be guided by the representation of entities and relationships, as well as the expected workload over them.

There is also the lack of a *common conceptual data model for blockchains*, i.e., a consensual set of concepts that properly represent blockchain data. As stated before, we have some blockchain network schemas, like Bitcoin and Ethereum, that are used as baselines for many works. However, these works do not worry about the design of a single interface for querying data for multiple blockchains and provide inter-chain relations based on a consensual data model. One example could be to link different accounts in different networks. It would be useful for fraud detection programs to identify possible inter-chain frauds.

Another relevant research issue that was tackled in this project is a *comprehensive comparison of DB technologies* for storing and querying blockchain data. Many selected

works consider relational DBMSs, but we also see MongoDB as a valid choice, proving itself as more performant than the official clients for the networks, even for queries that are commonly thought to be more performant to key-value stores. For complex queries, we have showed that both DBMSs used are multiple times faster than the official clients of the networks when executing more complex queries.

Finally, this study resulted in a publication to iiWAS 2022 (MEYER; SANTOS MELLO, 2022), which shows the academic interest in the area of blockchain data modelling. Future works on this topic includes the use of more advanced data modelling techniques, such as using graph-based DBs to better understand the movement of funds in those networks. Further publications are planned for 2024 where we will show the results of this research to the academy.

REFERENCES

AKCORA, Cuneyt Gurcan et al. Blockchain data analytics. **Intelligent Informatics**, p. 4, 2018.

ALSUNAIDI, Shikah J; ALHAIDARI, Fahd A. A survey of consensus algorithms for blockchain technology. In: IEEE. 2019 International Conference on Computer and Information Sciences (ICCIS). [S.l.: s.n.], 2019. P. 1–6.

ANTONOPOULOS, Andreas M. **Mastering Bitcoin: Programming the open blockchain**. [S.l.]: " O'Reilly Media, Inc.", 2017.

BARTOLETTI, Massimo et al. A general framework for blockchain analytics. In: 1ST Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers. [S.l.: s.n.], 2017. P. 1–6.

BENET, Juan. Ipfs-content addressed, versioned, p2p file system. **arXiv preprint arXiv:1407.3561**, 2014.

BITCOIN-ETL. **Bitcoin ETL**. [S.l.]: GitHub, 2023. `https://github.com/blockchain-etl/bitcoin-etl`.

BRAGAGNOLO, Santiago et al. Ethereum query language. In: 1ST International Workshop on Emerging Trends in Software Engineering for Blockchain. [S.l.: s.n.], 2018. P. 1–8.

BUTERIN, Vitalik et al. A next-generation smart contract and decentralized application platform, 2014.

CHEN, Weili et al. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology. In: WORLD Wide Web conference. [S.l.: s.n.], 2018. P. 1409–1418.

ETHEREUM-ETL. **Ethereum ETL**. [S.l.]: GitHub, 2023. `https://github.com/blockchain-etl/ethereum-etl`.

FERDOUS, Md Sadek; CHOWDHURY, Mohammad Jabed Morshed; HOQUE, Mohammad A. A survey of consensus algorithms in public blockchain systems for crypto-currencies. **Journal of Network and Computer Applications**, Elsevier, v. 182, p. 103035, 2021.

HABER, Stuart; STORNETTA, W Scott. How to time-stamp a digital document. In: SPRINGER. CONFERENCE on the Theory and Application of Cryptography. [S.l.: s.n.], 1990. P. 437–455.

HAN, Jongbeen et al. Enabling SQL-query processing for ethereum-based blockchain systems. In: 9TH International Conference on Web Intelligence, Mining and Semantics. [S.l.: s.n.], 2019. P. 1–7.

HECTOR, Ugarte-Rojas; BORIS, Chullo-Llave. BLONDiE: Blockchain Ontology with Dynamic Extensibility. **CoRR**, abs/2008.09518, 2020.

HUANG, Huawei et al. A survey of state-of-the-art on blockchains: Theories, modelings, and tools. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 2, p. 1–42, 2021.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1–26, 2004.

KRUIJFF, Joost T. de; WEIGAND, Hans. Understanding the Blockchain Using Enterprise Ontology. In: 29TH Int. Conference on Advanced Information Systems Engineering, CAiSE. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science), p. 29–43.

KSHETRI, Nir; VOAS, Jeffrey. Do crypto-currencies fuel ransomware? **IT professional**, IEEE, v. 19, n. 5, p. 11–15, 2017.

LI, Yang et al. EtherQL: a query layer for blockchain system. In: SPRINGER. INTERNATIONAL Conference on Database Systems for Advanced Applications. [S.l.: s.n.], 2017. P. 556–567.

LINOY, Shlomi et al. Scalable privacy-preserving query processing over Ethereum blockchain. In: IEEE. 2019 IEEE International Conference on Blockchain (Blockchain). [S.l.: s.n.], 2019. P. 398–404.

MEYER, João Vicente; SANTOS MELLO, Ronaldo dos. An Analysis of Data Modelling for Blockchain. In: SPRINGER. INTERNATIONAL Conference on Information Integration and Web. [S.l.: s.n.], 2022. P. 31–44.

MURCH. **Can someone explain how the Bitcoin blockchain works?** [S.l.: s.n.], Sept. 2013. Available from:

<https://bitcoin.stackexchange.com/questions/12427/can-someone-explain-how-the-bitcoin-blockchain-works>.

NAKAMOTO, Satoshi. Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008.

NARAYANAN, Arvind. **Bitcoin and cryptocurrency technologies : a comprehensive introduction**. Princeton, New Jersey: Princeton University Press, 2016. ISBN 0691171696.

NGUYEN, Giang-Truong; KIM, Kyungbaek. A survey about consensus algorithms used in blockchain. **Journal of Information processing systems**, Korea Information Processing Society, v. 14, n. 1, p. 101–128, 2018.

NICK, Jonas David. **Data-driven de-anonymization in bitcoin**. 2015. MA thesis – ETH-Zürich.

OLIVÉ, Antoni. The conceptual schema of Ethereum. In: SPRINGER. INTERNATIONAL Conference on Conceptual Modeling. [S.l.: s.n.], 2020. P. 418–428.

OZDAYI, Mustafa Safa; KANTARCIOGLU, Murat; MALIN, Bradley. Leveraging blockchain for immutable logging and querying across multiple sites. **BMC Medical Genomics**, Springer, v. 13, n. 7, p. 1–7, 2020.

PENG, Zhe et al. VQL: Providing query efficiency and data authenticity in blockchain systems. In: IEEE. 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW). [S.l.: s.n.], 2019. P. 1–6.

PODGORELEC, Blaž et al. A Brief Review of Database Solutions Used within Blockchain Platforms. In: SPRINGER. INTERNATIONAL Congress on Blockchain and Applications. [S.l.: s.n.], 2020. P. 121–130.

PRATAMA, Fariz Azmi; MUTIJARSA, Kusprasapta. Query support for data processing and analysis on ethereum blockchain. In: IEEE. 2018 Int. Symposium on Electronics and Smart Devices (ISESD). [S.l.: s.n.], 2018. P. 1–5.

QU, Qiang et al. On spatio-temporal blockchain query processing. **Future Generation Computer Systems**, Elsevier, v. 98, p. 208–218, 2019.

REN, Yongjun et al. Data query mechanism based on hash computing power of blockchain in internet of things. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 20, p. 207, 2020.

ROTH, Nicholas. An architectural assessment of bitcoin: using the systems modeling language. **Procedia Computer Science**, Elsevier, v. 44, p. 527–536, 2015.

SALDANHA, Lucas. **Ethereum Yellow Paper walkthrough (2/7)**. [S.l.]: Lucas Saldanha, Mar. 2019. Available from: <https://www.lucassaldanha.com/ethereum-yellow-paper-walkthrough-2/>.

SANKAR, Lakshmi Siva; SINDHU, M; SETHUMADHAVAN, M. Survey of consensus protocols on blockchain applications. In: IEEE. 2017 4th international conference on advanced computing and communication systems (ICACCS). [S.l.: s.n.], 2017. P. 1–5.

SZABO, Nick. Formalizing and securing relationships on public networks. **First Monday**, v. 2, n. 9, 1997.

TRIHINAS, Demetris. Datachain: a query framework for blockchains. In: 11TH International Conference on Management of Digital EcoSystems. [S.l.: s.n.], 2019. P. 134–141.

TURNER, Adam; IRWIN, Angela Samantha Maitland. Bitcoin transactions: a digital discovery of illicit activity on the blockchain. **Journal of Financial Crime**, Emerald Publishing Limited, 2018.

VO, Hoang Tam; KUNDU, Ashish; MOHANIA, Mukesh K. Research Directions in Blockchain Data Management and Analytics. In: EDBT. [S.l.: s.n.], 2018. P. 445–448.

WAZLAWICK, Raul. **Metodologia de pesquisa para ciência da computação**. [S.l.]: Elsevier Brasil, 2017. v. 2.

WENSLEY, John H et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. **Proceedings of the IEEE**, IEEE, v. 66, n. 10, p. 1240–1255, 1978.

XIE, Junfeng et al. A survey on the scalability of blockchain systems. **IEEE Network**, IEEE, v. 33, n. 5, p. 166–173, 2019.

XU, Yuqin et al. ECBC: A high performance educational certificate blockchain with efficient query. In: SPRINGER. INTERNATIONAL Colloquium on Theoretical Aspects of Computing. [S.l.: s.n.], 2017. P. 288–304.

ZHOU, Ence et al. Ledgerdata refiner: a powerful ledger data query platform for hyperledger fabric. In: IEEE. 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). [S.l.: s.n.], 2019. P. 433–440.

ZYSKIND, Guy; NATHAN, Oz, et al. Decentralizing privacy: Using blockchain to protect personal data. In: IEEE. 2015 IEEE Security and Privacy Workshops. [S.l.: s.n.], 2015. P. 180–184.

Appendix

APPENDIX A – POSTGRESQL "HARD" QUERIES

```
1  -- (1) Which account made the most transactions?
2  SELECT t.from, count(t)
3  FROM eth_transactions t
4  GROUP BY t.from
5  ORDER BY count(t) DESC
6  LIMIT 5;
7
8  -- (2) Which account made the most transactions last week?
9  SELECT t.from, count(t)
10 FROM eth_blocks b LEFT JOIN eth_transactions t
11 ON b.hash = t.block
12 WHERE
13     b.timestamp >= '2015-10-01'
14     AND b.timestamp < '2015-10-08'
15 GROUP BY t.from
16 ORDER BY count(t) DESC
17 LIMIT 5;
18
19 -- (3) Which day had the most transactions? Or blocks?
20 SELECT date_trunc('day', b.timestamp) AS day, count(t)
21 FROM eth_blocks b LEFT JOIN eth_transactions t
22 ON b.hash = t.block
23 GROUP BY day
24 ORDER BY count(t) DESC
25 LIMIT 5;
26
27 -- (4) Which block has the most transactions?
28 SELECT b.number, b.hash, count(t)
29 FROM eth_blocks b LEFT JOIN eth_transactions t
30 ON b.hash = t.block
31 GROUP BY b.hash, b.number
32 ORDER BY count(t) DESC
33 LIMIT 5;
34
35 -- (5) What is the biggest transaction?
36 -- (6) What are the biggest transactions?
37 SELECT t.hash, t.amount
38 FROM eth_transactions t
39 ORDER BY t.amount DESC, t.hash
```

```
40 LIMIT 5;
```

APPENDIX B – POSTGRESQL "EASY" QUERIES

```sql
-- (1) How many transactions were in block X?
SELECT COUNT(t)
FROM eth_blocks b LEFT JOIN eth_transactions t
ON b.hash = t.block
WHERE b.number = 123

-- (2) Which block contains transaction Z?
SELECT b.hash
FROM eth_blocks b LEFT JOIN eth_transactions t
ON b.hash = t.block
WHERE t.hash = '0x...'

-- (3) When was the block X mined?
SELECT b.timestamp
FROM eth_blocks b
WHERE b.number = 123

-- (4) Who sent transaction Z?
SELECT t."from"
FROM eth_transactions t
WHERE t.hash = '0x...'

-- (5) Who received transaction Z?
SELECT t."to"
FROM eth_transactions t
WHERE t.hash = '0x...'

-- (6) How much was sent in transaction Z?
SELECT t.amount
FROM eth_transactions t
WHERE t.hash = '0x...'
```

APPENDIX C – MONGODB "HARD" QUERIES

```
# (1) Which account made the most transactions?
db.blocks.aggregate([
    {'$project': {'_id': 0, 'transactions': 1}},
    {'$unwind': '$transactions'},
    {'$group': {'_id': '$transactions.from', 'total': {'$sum': 1}}},
    {'$sort': {'total': -1}},
    {'$limit': 5},
])

# (2) Which account made the most transactions last week?
db.blocks.aggregate([
    {'$match': {'timestamp': {'$gte': '2015-10-01', '$lt':
       '2015-10-8'}}},
    {'$project': {'_id': 0, 'transactions': 1}},
    {'$unwind': '$transactions'},
    {'$group': {'_id': '$transactions.from', 'total': {'$sum': 1}}},
    {'$sort': {'total': -1}},
    {'$limit': 5},
])

# (3) Which day had the most transactions? Or blocks?
db.blocks.aggregate([
    {'$project': {
        'tx': {'$size': '$transactions'},
        'date': {'$dateToString': {
            'format': '%Y-%m-%d', 'date': '$timestamp'}}}},
    {'$group': {'_id': '$date', 'total': {'$sum': '$tx'}}},
    {'$sort': {'total': -1}},
    {'$limit': 5},
])

# (4) Which block has the most transactions?
db.blocks.aggregate([
    {'$project': {'_id': 0, 'hash': 1, 'transactions': 1}},
    {'$unwind': '$transactions'},
    {'$group': {'_id': '$hash', 'total': {'$sum': 1}}},
    {'$sort': {'total': -1}},
    {'$limit': 5}
])
```

```
39
40 # (5) What is the biggest transaction?
41 # (6) What are the biggest transactions?
42 db.blocks.aggregate([
43     {'$project': {'_id': 0, 'transactions': 1}},
44     {'$unwind': '$transactions'},
45     {'$replaceRoot': {'newRoot': '$transactions'}},
46     {'$sort': {'amount': -1, 'hash': 1}},
47     {'$limit': 5}
48 ])
```

APPENDIX D – MONGODB “EASY” QUERIES

```
# (1) How many transactions were in block X?
db.aggregate([
    {'$match': {'number': 123}},
    {'$project': {'size': {'$size': '$transactions'}}},
])

# (2) Which block contains transaction Z?
db.aggregate([
    {'$match': {'transactions.hash': '0x...'}},
])

# (3) When was the block X mined?
db.aggregate([
    {'$match': {'number': 123}},
])

# (4) Who sent transaction Z?
db.aggregate([
    {'$match': {'transactions.hash': '0x...'}},
    {'$unwind': '$transactions'},
    {'$replaceRoot': {'newRoot': '$transactions'}},
    {'$match': {'hash': '0x...'}},
    {'$project': {'from': '$from'}},
])

# (5) Who received transaction Z?
db.aggregate([
    {'$match': {'transactions.hash': '0x...'}},
    {'$unwind': '$transactions'},
    {'$replaceRoot': {'newRoot': '$transactions'}},
    {'$match': {'hash': '0x...'}},
    {'$project': {'from': '$to'}},
])

# (6) How much was sent in transaction Z?
db.aggregate([
    {'$match': {'transactions.hash': '0x...'}},
    {'$unwind': '$transactions'},
    {'$replaceRoot': {'newRoot': '$transactions'}},
```

```
40      {'$match': {'hash': '0x...'}},
41      {'$project': {'amount': '$amount'}},
42  ])
```

APPENDIX E – ETHEREUM "EASY" QUERIES

Queries are represented as calls to a local Ethereum node using the web3 package[1] for the Python language.

```
# (1) How many transactions were in block X?
def q1(w3: Web3) -> int:
    return w3.eth.get_block_transaction_count(123)

# (2) Which block contains transaction Z?
def q3(w3: Web3) -> str:
    tx = w3.eth.get_transaction('0x...')
    return tx['blockHash']  # or blockNumber

# (3) When was the block X mined?
def q4(w3: Web3) -> int:
    blk = w3.eth.get_block(123)
    return blk['timestamp']

# (4) Who sent transaction Z?
def q5(w3: Web3):
    tx = w3.eth.get_transaction('0x...')
    return tx['from']

# (5) Who received transaction Z?
def q6(w3: Web3):
    tx = w3.eth.get_transaction('0x...')
    return tx['to']

# (6) How much was sent in transaction Z?
def q7(w3: Web3):
    tx = w3.eth.get_transaction('0x...')
    return tx['value']
```

[1] web3.py - https://github.com/ethereum/web3.py

APPENDIX F – EXECUTION TIMES FOR "HARD" QUERIES

Data for execution times for hard queries. Data used to generate Figure 17. Average of 100 executions. Data is in seconds.

	PostgreSQL	Mongo
Query 1	4.77676739	2.228310444
Query 2	6.67E+00	4.86E-01
Query 3	20.1019508	2.238149748
Query 4	7.588967261	2.677318439
Query 5	0.4637488818	2.448379524
Query 6	0.4637488818	2.448379524

APPENDIX G – EXECUTION TIMES FOR "EASY" QUERIES

Data for execution times for easy queries. Data used to generate Figure 15. Average of 100 executions. Data is in seconds.

	Ethereum	PostgreSQL	Mongo
Query 1	0.003174932003	0.001934921741	0.0007137560844
Query 2	0.002773206234	0.2822695255	0.0006787109375
Query 3	0.004147675037	0.0009716916084	0.0005009746552
Query 4	0.00221817255	0.2741855526	0.0005585432053
Query 5	0.002046205997	0.2841969633	0.0005664920807
Query 6	0.002186026573	0.2505286384	0.0005709385872