Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka (INF)
specjalność: brak

Praca dyplomowa - inżynierska

An application for tracking the flow of resources for Bitcoin cryptocurrency

Marcin Pieczka

keywords:
bitcoin, blockchain, cryptocurrencies,
data availability, NoSQL databases,
configuration management

short summary:
In this work system allowing for fast and easy access to Bitcoin transactions was designed and implemented. The system also provides platform that enables implementation of analytic functionalities which use transactions data. This will let researchers to share their work and tools in order to increase efficiency of studying Bitcoin blockchain.

opiekun pracy dyplomowej	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

Ostateczna ocena za pracę dyplomową			
Przewodniczący Komisji egzaminu dyplomowego	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**
a) kategorii A (akta wieczyste)
b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)
* *niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław 2018

Contents

Streszczenie

Praca ta poświęcona jest zaprojektowaniu i implementacji systemu pozwalającego na szybki i prosty dostęp do historii transakcji przeprowadzonych w sieci Bitcoin oraz dodatkowego API pozwalającego na rozwój nowych funkcjonalności. Na początku pokrótce zostaje przedstawiona problematyka kryptowalut wraz z opisem podstawowych operacji kryptograficznych, które są w nich wykorzystywane. Następnie poruszony zostaje temat Bitcoina, wyjaśniony zostaje mechanizm działania jego sieci, pojęcia transakcji, bloku i blockchaina oraz sposobu autoryzacji transakcji. Po tych wstępnych rozdziałach opisane zostają alternatywne rozwiązania, które w pewnym stopniu realizują funkcjonalności oferowane przez implementowany system. Przedstawiane są ich zalety i wady w kontekście wymagań użytkowników. W Rozdziale 5 opisany jest proces projektowania systemu. Proces ten rozpoczyna się od opisu wymagań użytkowników poprzez opis docelowego użytkownika i jego wymagań. Od tego miejsca projektowanie zaczyna się od definicji wejść i wyjść systemu i podąża drogą nasuwających się pytań i problemów. W kolejnym rozdziale przedstawiony został proces implementacji systemu. Pokazane zostały najbardziej interesujące i najważniejsze elementy implementacji, w szczególności te które bezpośrednio pozwalały osiągnąć spełnienie potrzeb użytkownika. Po implementacji przedstawione zostały testy funkcjonalności, obrazujące zarówno wydajność jak i sposób użytkowania systemu. Ostatni rozdział przed podsumowaniem przybliża czytelnikowi w jaki sposób system był już wykorzystywany, z jakimi problemami się on boryka i jakie perspektywy przyszłego rozwoju ma on przed sobą. Praca zwieńczona jest podsumowaniem, w skrócie łączącym cały proces, od potrzeb użytkownika, do końcowych rezultatów.

Abstract

In this work, a system allowing fast and easy access to Bitcoin transactions' history, as well as an API allowing for new functionalities to be hosted was designed and implemented. At the beginning, the cryptocurrencies and cryptographic operations used by them are shortly described. Next, Bitcoin is being presented by describing mechanisms of its network, explaining concepts of transaction, block, the blockchain, and describing means of authorization. After those introductory chapters alternative software, which provides functionalities similar to those implemented in the system, is being described. Advantages and disadvantages of alternative software are being listed in context of user needs. In Chapter 5 process of design is being described. This process starts with describing user requirements by describing end user and listing user stories. From that point, design process starts with specifying system inputs and outputs and then follows questions and problems that arise. Most interesting and most important elements of implementation have been shown, specifically those which directly help to fulfill user needs. After implementation, functional tests had been performed, which serve a purpose of showing performance, as well as usage examples of the system. The chapter before summary shows usage of this system that had already occurred, what problems this system faces, and what are its future directions. Work ends with the summary that shortly binds the whole process from user needs to end results.

Chapter 1

Introduction

In a world of cryptocurrencies, Bitcoin is most widely accepted implementation of that concept. Bitcoin is based on the immutable distributed data structure called blockchain, which ensures security without a central authority. While Bitcoin is most often researched in the context of economics, it starts to be looked at as complex network by growing number of researchers today.

Every process of data analysis starts with accessing the data. When analyzing Bitcoin blockchain, the first step might be the hardest one. Currently, Bitcoin blockchain contains over 160GB of raw binary data and everyone who attempts to analyze it needs to have an efficient and reliable way of obtaining it. Moreover, users should have an obvious place for creating additional APIs that will be hosted on the same server as the data. Thanks to that, operations requiring big amounts of data, but returning results of significantly smaller size, could be implemented more efficiently.

The goal of this work is to create an application that fulfills the following requirements:

- allows fast access to blockchain data
- allows access over the network
- updates its data in a constant manner
- provides API for Python and R
- is easy to install

The structure of this thesis is as follows. Chapter 2 covers the introduction to cryptocurrencies. Chapter 3 shows the implementation of cryptocurrency concept in the example of Bitcoin. Chapter 4 lists alternative software providing similar features to software being created in this work. Chapter 5 goes through the process of designing the system. In chapter 6 implementation details of selected parts of the system are being shown. Chapter 7 covers functionality testing providing perspective on the performance of the system as well as usage examples. Next chapter describes real-life usage, problems and future directions of the system. At the end, the summary of whole work can be found.

Chapter 2

Cryptocurrencies

2.1 Overview

This chapter will explain the concept of cryptocurrencies, in some cases based on the example of Bitcoin. The connection between cryptography and cryptocurrencies will be shown, as well as means that are used to ensure correctness and safety of transactions.

2.2 Definition

"Cryptocurrency is a digital currency in which encryption techniques are used to regulate the generation of units of currency and verify the transfer of funds, operating independently of a central bank." [5]

2.3 Security

Every currency operates within its defined set of rules and has to have means to ensure that these rules are obeyed. For the regular money, as we know it today, security is provided by central authorities in multiple ways:

- banknotes that are hard to counterfeit
- a law that penalizes creating counterfeits
- banking systems for digital money transfer, that ensure specified rules and are secure from malicious actors

Cryptocurrencies have their own ways of ensuring their rules, but instead of central authority, those rules are ensured by cryptography and probability in a distributed system [31].

2.4 Cryptography

Cryptocurrencies are in big part based on cryptography to ensure the rules at which they operate. Cryptography helps not only to ensure that fraudulent and malicious operations can't have their place but as well to establish rules for creating new units of those currencies [12].

Cryptography is a field of knowledge that has its focus on protecting information from unauthorized access in communication channel [18], as well as providing means to authenticate and ensure integrity messages [11].

Cryptography can be classified as:

- Symmetrical - Information is encrypted and decrypted with the same key, with this type of encryption arises the problem of transferring the key [31].

- Asymmetrical - Information is encrypted with a different key than the key that is used for decryption. One key is usually called the public key and is used to encrypt messages. Because public key cannot be used to decrypt a message it can be shared freely without concern. The second key is called private key an is used to decrypt messages. The private key should never be shared, everyone possessing that key can decrypt messages encrypted with related public key [31].

Public key cryptography was developed in the 1970s and is widely used to ensure the security of information today. Mathematical functions were discovered that are easy to calculate in one way, but nearly impossible to calculate in reverse. Bitcoin uses elliptic-curve functions as a base for calculating public key [31].

Aside from encryption, Bitcoin uses cryptographic hash functions in its inner workings. Cryptographic hash functions map strings of arbitrary length to a constant number of bits called digest, typically from 128 to 512 [19].

One of the main properties of hash functions is that they work one-way only. This means that for given hash digest it is impossible to find original message, but creating the hash is easy. Another important property of hash functions is that for different messages, the output should be different. This property does not hold in theory. Because output space of hash function is finite, and input space being virtually infinite, there must exist such messages that while being different, their hashes are identical. In practice, output space is so large that finding so-called collisions is nearly impossible. Therefore when hash digest is identical for two messages, those messages are assumed to be identical as well [19].

Chapter 3

Bitcoin

3.1 Overview

This chapter explains Bitcoin protocol, communication between nodes, transactions, and creation of blocks.

3.2 Genesis of Bitcoin

Bitcoin is the first widely accepted cryptocurrency and was created by an anonymous author who published his work under pseudonym Satoshi Nakamoto [16]. Domain `bitcoin.org` was registered 18.08.2008, and the first block, called genesis block, was mined 3.01.2009. Since those times Bitcoin's popularity had grown substantially, and many other cryptocurrencies based on Bitcoin emerged [9].

3.3 Network

Bitcoin is distributed network and there are no authoritative nodes. The exception is servers that are used by new Bitcoin nodes to discover other nodes, but they will not be discussed. Reason for it is that essentially they are not part of Bitcoin network, and if those servers fail, nodes will use other methods for discovery [17].

There are two types of information that nodes share between them: transactions and blocks [17]. Those entities will be discussed in separate sections.

3.4 Transactions

Transactions in Bitcoin are used to transfer value between owners. Single transaction describes inputs from which resources are being transferred and outputs to which those resources are being transferred. All the resources from the inputs are being transferred to the outputs, unused resources are transferred to the miner which is known as the transaction fee.

Inputs are described by reference to the output of the previous transaction. To reference output of transaction, the hash of transaction is needed as well as the index of output, the numbering of outputs starts with 0. Additionally, for each input, there is an attribute that makes it possible to verify that resources are transferred by their rightful owner [2].

Outputs of transaction specify number of BTC that is being transferred to that output, and a script that defines the way of accessing those resources.

As have been stated, outputs define the way of accessing resources, and inputs provide values that unlock those resources. This is done with scripting system.

3.5 Scripts

In Bitcoin providing access verification is done with scripting system. Scripts in Bitcoin are programs, written in not Turing complete language, that when processed determine whether access to resources protected by the script is granted or not [3].

Because this system is very versatile, granting access to resources can require e.g. providing multiple keys, providing a password, can be granted to everyone etc.

Although script system allows for many methods of authentication, one of them is used in most of the transactions. In this method is often called "pay-to-pubkey-hash". Script field of transaction output specifies the hash of public key and requires anyone wishing to spend this output to provide a signature and public key. Hash of public key must be equal to hash provided in output. Signature, which proves that spender owns the private key, must be correct [3].

3.6 Blockchain

Blockchain in Bitcoin is a data structure that holds every transaction that had taken place since the beginning of Bitcoin network. It consists of blocks of transactions, with each block holding in itself hash of the previous block. Every Bitcoin node, after it has synchronized with the network, holds entire blockchain in its memory [1].

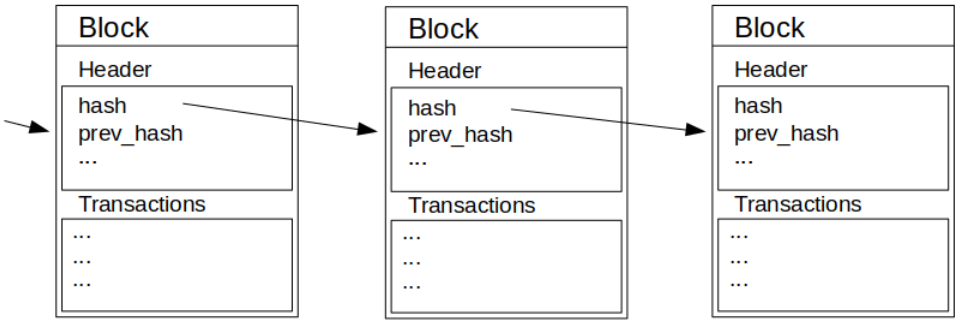

Figure 3.1 Visualization of blockchain

One notable property of this data structure is that modification of single block would require modification of every block that followed. This property of blockchain comes from the fact that modification of block changes its hash, and as mentioned before, blocks contain the hash of the previous block. This, of course, would require modification of previous block hash field in the following block.

Because of restrictions imposed on block creation, creating blocks is highly time-consuming, thus modification of block that had been followed by 6 other blocks is considered impossible without the consent of large portion of miners [7].

3.7 Mining - block creation

Mining in Bitcoin is the process of block creation. One of the most interesting properties of valid block is that its hash starts with a certain amount of "0". The number of "0"

needed at beginning of the correct block is being adjusted every 2016 blocks, so that rate at which blocks are being mined is approximately 1 block per 10 minutes. To achieve needed hash, attribute named "nonce" is set in trial and error manner. Everyone who wishes to create new block will need to:

- listen on the network for new transactions
- gather transactions into the block
- set all attributes of the block like previous block hash, transaction count etc.
- set value of "nonce" attribute
- calculate the hash of the block
- if hash starts with a needed number of "0" than propagate block through the network, otherwise set "nonce" to the different value and continue process till finding correct value or till some other miner finished

Naturally, the process described above is simplified. There are some nuances, e.g. "nonce" attribute is a 32-bit integer, and nowadays more often than not there is no such value of that attribute that results in the correct hash. Miners in such situation apply other techniques to find correct hash. Nonetheless, this is level of detail this work does not operate on.

Incentives for mining blocks are transaction fees and block rewards. A transaction fee is any unspent output in a transaction. For instance, when 0.9 BTC is sent from the output that holds 1 BTC, unspent output equals 0.1 BTC. This unspent output can be transferred to address controlled by the miner. Another incentive, block reward, is the last transaction in the block which creates new coins and transfers them to address specified by the miner. Amount of BTC in block reward started at 50 BTC, and halves every 210000 blocks [8].

Chapter 4

Alternative software providing similar features

4.1 Overview

This chapter describes software that is similar to what will be designed and implemented in this paper. These descriptions go through features, advantages, and disadvantages of those applications.

4.2 blockchain.info

Web application blockchain.info [20] provides free access to Bitcoin blockchain data by either website, JSON API or APIs dedicated to specific languages including Python. Blockchain.info does not have dedicated support for R. Number of requests is limited.

Relevant APIs provided by blockchain.info:

- getting a single block, by block hash
- getting a single block, by hight
- getting multiple block headers
- getting a single transaction, by transaction hash
- getting all transactions of single or multiple addresses

Most common usage scenario in analytic context is getting a range of blocks, for example, all blocks from 10.04.2017 to 20.04.2017. Nonetheless, blockchain.info does not provide a simple way of getting such data. To achieve this, thousands of requests to API providing us with single block data is needed, and this is not fast enough.

Another and the biggest problem is the API call limit that would make working with this application impossible for larger queries.

4.3 blockexplorer.com

This web application [21] is very similar to blockchain.info in almost every aspect, although there are some differences. Data is accessible either by the website or by JSON API, but there is no Python or R API provided. Set of APIs is almost identical to blockchain.info and does not provide an easy way to get multiple blocks. The main difference is that there is no official API call limit, but because it is an external tool, it means that such limit can appear every moment.

4.4 libBitcoin-database

This piece of software [15], after installation builds an in-memory database of Bitcoin blockchain, its description promises high performance. Because of almost nonexisting documentation, it is hard to describe its features in depth.

Although this software is actively developed, and at first sight, it might seem like the solution that fits needs of users, some problems can be found. One drawback is coming from its biggest selling point - being in-memory database makes it requires large amounts of RAM. This problem makes it not viable for hardware that is accessible to the average user. Next problem lies in its API that allows connecting to the database only via C/C++ library, which would make usage in R and Python at least problematic. Another problem is already mentioned lack of useful documentation, which would make the usage largely troublesome.

4.5 BitcoinDatabaseGenerator

BitcoinDatabaseGenerator [13] is a data transfer tool that can feed SQL database with blockchain data. It was written in C# and as its author states, it is only meant to be run on Windows machines. From the documentation, we know that this software should be run every time we want to update our database. The database schema that is created after an update operation, consists of separate tables for blocks, transactions, transaction inputs and so forth. Such database schema would require joining tables in a multitude of usage scenarios, which is known to be a slow operation.

Although connecting to SQL database from both R and Python is easily achievable, working with SQL databases in such case might be cumbersome. Abstracting away the relational structure of data to objects might be needed when working with SQL database.

Chapter 5

Design

5.1 Overview

This chapter goes through the design process in detail. Design starts with describing users and their needs. Then the system is described as a black box, with its inputs and outputs. After that design flows as questions and issues arise.

5.2 User requirements

5.2.1 Users description

The requester of this software is Ph.D. working at a university. He is interested in networks that emerge between cryptocurrencies users. The cryptocurrency that he researches the most is Bitcoin, because of its wide adoption which leads to his research having a bigger impact. He believes that science is a group activity, and thus he wants knowledge to be shared in an efficient way. He has created a group of people working on this subject at his university, and actively pursues better communication and collaboration with similar groups at other universities. The problem he and his peer's faces is lack of a standard way to share common analytic solutions, which hinders accumulation of knowledge. With such place, as he believes, common analytic tasks would not have to be implemented over and over by separate researchers. One of the common tasks that he notices to be implemented over and over is acquiring Bitcoin data.

5.2.2 User stories

The following sentences sum up the discussions with people that consider using this system.

- As a user, I want to access blockchain data from my Python/R script so that data analysis and data accessing can be made in the same code.
- As a user, I want the data to be in a format idiomatic to Python/R so that I don't have to convert it myself.
- As a user, I want easy access to block data by specifying the range of time or block height so that I can spend my time working with the data, and not with accessing it.
- As a user, I want the installation not to require complicated operations so that I can do it without specialized knowledge.

- As a user, I want to be able to run this software on my Linux server so that I don't have to learn a new operating system to be able to use it.
- As a user, I want fast access to the data so that my analytic scripts will be pleasurable to work with.
- As a user, I want to have an obvious place to create my own APIs on the server so that my new data-hungry features can be placed on the same server as data for better performance.

5.3 System inputs and outputs

In this section inputs and outputs of the system will be specified. This will then help to discover what transformation the input data will undergo, and what components are needed to provide outputs efficiently. Here will be considered only part of the project that is responsible for serving data, not the part that will be responsible for hosting future APIs.

5.3.1 Inputs

Bitcoind BLK files

The only source of Bitcoin block data will be BLK files, stored by full Bitcoin node. Daemon process bitcoind gets blocks from neighboring nodes and stores them in the data directory. The BLK files in default configuration store up to 128MB of raw network format block data. Blocks are stored in order in which they come from the network. Library's that handle parsing BLK files exist, so accessing this data should not be a problem.

Request for blocks

This request will be the way user communicates with the system. In this request, the user will specify which blocks he wants to receive. Usually, those will be blocks from a specified range of time or range of height.

5.3.2 Outputs

Response on request for blocks

This response will contain block data in format either native to Python/R or JSON.

5.4 Persistence

The fact that transaction sender address is not stored in Bitcoin blocks directly, but by reference to other transaction, creates responsibility for the system. This responsibility is to discover addresses by finding the referred transaction and getting the address of the recipient. This, of course, requires a substantial amount of time which should not be added to the time of user waiting for his response.
An additional thing to consider is the time needed to transform raw binary Bitcoin block into widely used data format such as JSON. Knowing this we will get to the conclusion that we need some type of persistence. Although some custom way to store this data

might be better suited to our needs, a database will be used, because of a constraint of time available to build this solution.

5.4.1 Database operations

Let's consider database operations that will need to be fast, and those that are not so important in this regard.

The least important operation will be data addition. The end user will not be performing these operations, and its performance will be of least priority.

The most important operation will be querying blocks, by their hash, time, and other attributes, especially querying for a range of consecutive blocks described by time frame or height range.

Another important operation is getting transactions by hash. This operation will be needed for discovering addresses of transaction senders. Although this operation will not be in usage scenarios started by the end user, the number of such operations needed to discover sender address of every Bitcoin transaction makes it crucial for it to be fast.

5.4.2 Database choice

At the beginning let's simplify the choice between RDBMS (Relational Database Management System) and NoSQL databases. Out of many NoSQL possibilities, MongoDB [22] have been chosen, based on initial research of different NoSQL systems strengths and weaknesses.

Let's lay out some facts that will help to decide whether to use a relational database or MongoDB.

- Storing blocks in RDBMS in multiple tables, for instance, table for blocks and table for transactions would require joining those tables which could be a slow operation.
- To achieve fast querying, the data should be strongly denormalized.
- Storing blocks in RDBMS in one table can be impossible in many systems, due to attribute count limit.
- Comparable performance can be achieved with MongoDB and RDBMS, but MongoDB makes storing denormalized data idiomatic.
- MongoDB has APIs for Python and R that makes data available as objects.

Based on these facts, the database that will be used is MongoDB. This choice will provide easy access to the data from Python and R code, and achieving needed performance will come without problems.

5.4.3 Database structure

Collections in MongoDB don't have structure defined upfront. To have mandatory attributes, actor responsible for data insertion needs to enforce it. Because of this, writing about database structure is not the most accurate term. It is easier to think about it as a structure of data, and in my system, the structure of data will not change in big degree

compared to form it will be obtained from blocks. The only difference is, that by default, Bitcoin transactions don't specify an address of the sender. Additionally, the attribute has been added, that is the time that had passed between receiving resources which were the input of transaction, and that transaction.

One document in MongoDB collection will be one Bitcoin block, its attributes are:

- bits - current target in compact format [4]
- difficulty
- hash - hash of this block
- height - number of blocks before this block in blockchain
- merkle root - hash based on all transactions of block
- n tx - number of transactions
- nonce - value set by miner to achieve needed block hash
- prev hash - hash of previous block
- size
- timestamp - time at which block had been mined
- transactions - list of transactions with each transaction consisting of:
  - hash - transaction hash
  - inputs - list of data regarding addresses from which resources had been transferred, consisting of:
    - transaction hash - hash of transaction in which sender of this transaction got resources spent in this transaction
    - transaction index - points to specific output in transaction described by previous attribute
    - output timestamp - additional value described before this list of attributes
    - addresses - list of addresses, each containing:
      - address
      - hash
      - public key
      - type
  - outputs - list of data regarding addresses to which resources had been transferred, consisting of:
    - value
    - addresses - list of addresses, each containing just like in inputs:
      - address
      - hash
      - public key
      - type

5.4.4 Database indexes

Database indexes serve a purpose of making queries faster and more efficient on the cost of space and data insertion time. Because the performance of queries is one of the key user needs, and space and data insertion time are very low on priority list, the indexes will be added to all of the fields that data will be queried by.

Indexed attributes:

- block hash
- block timestamp
- transaction hash - needed internally for discovering input addresses

5.5 High level architecture

This system will consist of separate entities, that will have their own, well-defined responsibilities. This will make those components and overall system easy to understand and maintain.

The system will consist of:

- Bitcoin node with its data directory
- database for storing processed blocks
- process constantly updating the database with new blocks that are gathered by Bitcoin node, which will be referred to as database updater
- web application allowing for analytic functionalities implementation

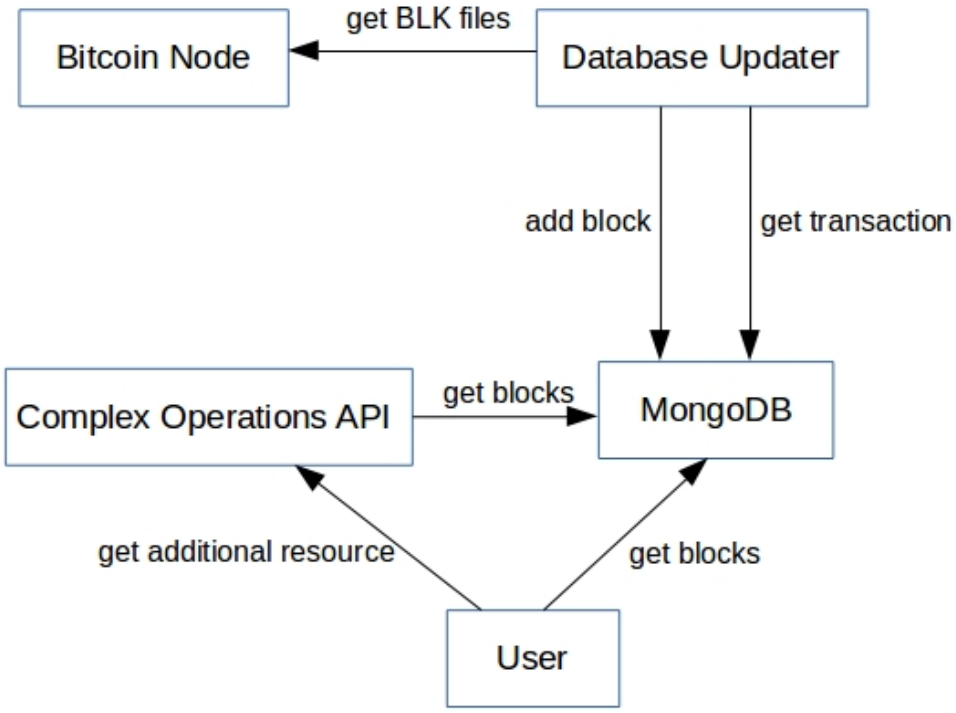

Figure 5.1 System components and their communication

5.5.1 Database Updater

This component will be responsible for:

- parsing raw Bitcoin blocks collected from BLK files
- discovering transaction sender address by transaction output referred in its input
- adding blocks to the database
- administering the database, namely adding indexes in the best moment

Although the performance of this component does not influence user wait time, it influences time needed to get to the point at which most of the recent blocks are in the database. Because of a large amount of data that needs to be processed and inserted into the database, optimizations are necessary to finish initial data load in reasonable time.

The main bottleneck is discovering sender address. Nowadays, one block holds about 2000 transactions, so to process one block with the simplest algorithm, database updater would have to ask the database about 2000 times for transaction for every processed block. This would lead to very poor performance. Because it would require too much memory for database updater to remember every transaction it has processed, it will remember some constant number of last processed transactions.

5.5.2 Complex Operations API

Reason for this component to be requested is the fact that when there is a group of people researching common field, there is a big overlap in needed functionalities. When researcher gets to the point where he needs functionality that he doesn't have, he will implement it

himself. The problem is that very often the same functionality was implemented by one of his peers, and this component is meant to solve this problem.

In order to incentivize users to use this API, in regard of developing new functionalities, it needs to be at the similar level of complexity as developing those functionalities locally, and it is crucial to consider it during design.

Complex Operations API will be web service, which in response to its user's request will return JSON data. At the main page will be located descriptions of implemented functionalities with URL's needed to access them. There will be one functionality implemented in this service which purpose will be to handle to the users a well working example of something similar to what they will be implementing.

5.6 Deployment

The main property of software, that can be accomplished with well-chosen deployment and packaging, is the simplicity of use and cross-platform capabilities. Those can positively influence adoption rate of this solution in perspective of both usage and further open source development.

For this project docker [23] containers will be used, which will help in keeping components decoupled, and will make installation process easy to follow.

The installation process should consist only of installing full Bitcoin node, docker, running one custom script and setting up simple configuration (database credentials, etc.).

All the containers will be connected to the same private network so that communications between them will lack any complexity.

Chapter 6

Implementation

6.1 Overview

This chapter shows most notable implementation details, selected by their importance to the system. Implementation starts with deployment which usually is the last concern, but in this case, it has guided whole implementation process. Later, implementation of system components is described.

6.2 Install.sh

This tour of implementation starts with the installation script because knowing its content will provide knowledge of containers set up, data flow between containers, an overview of the whole system.

Install.sh starts with reading configuration from file "config.conf" in which following things are being set up:

- database credentials for administrator and user with read-only privileges
- the port at which the database will be reachable
- data directory of Bitcoin node
- the directory at which database data will be stored
- transaction cache size

After configuration is loaded, the network is being created, and building and starting of containers begin.

6.2.1 Database container setup

For database container, the configuration will be shown, so that readers without experience with docker will have a chance to see how such configuration looks.

Listing 6.1 Database container run command

```
docker run -d \
    --name btc-blockchain-db \
    -e AUTH=yes \
    -e MONGODB_ADMIN_USER=$db_root_username \
    -e MONGODB_ADMIN_PASS=$db_root_password \
    -e MONGODB_APPLICATION_DATABASE=bitcoin \
```

```
-p 0.0.0.0:$db_port:27017 \
-v $database_dir:/data/db \
--network=btcnet \
aashreys/mongo-auth:latest
```

Database container is build on "aashreys/mongo-auth:latest" [24] docker image, which lets us create MongoDB database with enabled authentication (by default in MongoDB authentication is not enabled). Admin user credentials are being set, but credentials of the user with read-only privilege can't be set here, because the before-mentioned image does not support such action.

There are couple other things in this configuration worth noting, namely "-v" flag creates volume for container and its purpose comes from the fact that docker containers should be possible to be killed without any loss. With the volume set up, when we kill our database container and run it again, the database data will persist.

Another important thing is setting up a common network for all the containers, here it is accomplished with the flag "–network". With network configured, DNS is set up automatically and it is possible to refer other containers by their names. For instance, running

```
ping btc-blockchain-db
```

from within other container connected to the same network will send messages correctly

6.2.2 Database updater container setup

Before database updater docker image can run, it needs to be created first. To create docker image, configuration needs to be written in the file named "Dockerfile", then command "docker build" is used to build the image. With image ready, it can be run in the same way the previously described image was.

Listing 6.2 The dockerfile for database updater

```
FROM python:3

COPY requirements.txt ./
COPY ./src/ /src/
RUN pip install --no-cache-dir -r requirements.txt

# forked version from 22.03.2018
RUN pip install https://github.com/MarcinPieczka\
/python-bitcoin-blockchain-parser/archive/master.zip

RUN mkdir /btc-blocks-data/

EXPOSE 27017

CMD [ "python", "-u", "/src/update_database.py" ]
```

Dockerfile starts with the base image that is being used, here it is "python:3" [25] image that contains as its name suggests python3 installed. Next two lines copy needed files to the container, then libraries are installed, containers port is exposed for communication with the database, and in the last line, my application is being started.

Using docker makes it easy to create installation process repeatable and independent from the environment so that running software will have the same effect on every machine it is being run (to some extent naturally).

6.2.3 Complex Operation API container setup

This container is based on "tiangolo/uwsgi-nginx-flask:python3.6" [26] which have all the basic elements needed to run flask application already configured. The only non-standard configuration that needed to be done is setting the timeout in Nginx [28] web server to a larger value. By default, timeout is 60 seconds, but because of the type of operations that this application is made for, this is not enough.

6.3 Database updater

This container as stated before is responsible for the database, and in more detail, it is responsible for both putting data to it and for managing its configuration. In regards to managing configuration of the database, this module takes responsibility for everything that could not be achieved in docker command, and that is:

- Creating a user with privilege to read the database, but not to modify it. This account will be used by end user.
- Creating database indexes for block height, block timestamp, and transaction hash.

All this is being done in separate module responsible only for the database - "mongo.py". Other than caring for database configuration, it abstracts away connecting and loading data into the database. This module uses pymongo[27] library in version 3.6 to connect to the database.

Data insertion took the most effort to develop and is the most complex part of the system.

All of Bitcoin BLK files processing is done with blockchain-parser[14] library, but not the official release, but version straight from their git repository [14]. The reason for this is that changes in Bitcoin are far more frequent than releases of this software, and some features of Bitcoin were not supported in the official version. The version that is being used does not support BIP-0173 [6] (BIP stands for Bitcoin improvement proposal) so discovering addresses from transactions using techniques described in it cannot be done.

Processing and loading data can be broken into these steps:

- Initial parsing of BLK files contained in Bitcoin data directory. Only headers of blocks are parsed, to get hash of the block, and hash of the previous block.
- Representation of blockchain is created, to get to know the order of blocks, and to know which blocks are in the main chain.
- One by one, starting from the block next after last inserted, blocks are fully parsed. Their transactions are cached, and input addresses are being discovered. Those addresses are being discovered first by looking into the cache, then if the cache does not contain needed transaction, the database is being asked.

- The last step is inserting the processed block into the database.

Of course, there is more complexity hidden inside those steps, like handling exceptions from the blockchain-parser library. The reason there are exceptions while parsing Bitcoin blocks is that Bitcoin transactions do not have to be fully valid. Many transactions exist with errors, and although they are of no interest to the system, they need to be recognized and handled properly.

6.4 Complex Operation API

This component was implemented in Flask framework [29], which was chosen for its simplicity. Simplicity is well needed in the application that users, who very often don't have experience with web applications, will extend with their code.

The main page is made purely in HTML and CSS, for the sake of before mentioned simplicity.

Two exemplary functionalities have been implemented, one of which, because of its lack of complexity, will be a good way to show advantages of Flask.

Listing 6.3 API returning last block in blockchain

```
@app.route('/api/last_block')
def _last_block():
    return dumps(mongo.db.blocks.find().sort([('height', -1)])[0])
```

In this code, the first line specifies the address at which service will be available, the second one is the declaration of the function, and the last one is its implementation. This function connects to the database with previously created "mongo" object, finds the last block in the database, converts it to JSON with "dumps" function and returns.

The second functionality is surely more complex. First of all, let's notice that we can look at Bitcoin transactions as the graph. Every transaction has its input and output, the addresses of inputs and outputs are vertices of the graph, and transactions are the edges between them. My second functionality as input takes two parameters which specify block range by hight than creates graph out of transactions found in specified blocks using iGraph [30] library. After creating graph it separates unconnected graphs and returns an array of the count of vertices in each graph.

The choice of this functionality was not random. Operations on graphs are very common when analyzing Bitcoin transactions.

At the end it needs to be said that although that application by itself is easy to understand and in this regard it will be easy for users to add their code to it, the deployment with docker makes it not so easy to use as a developer. The biggest drawback of docker is a relatively slow startup. When some portion of code is written into this container, and it needs to run for testing, the process of building image will take about 15 s (time based on personal observations).

Chapter 7

Functionality testing

7.1 Getting blocks by height

Getting Bitcoin blocks by their height has been identified to be one of the most needed ways to query blockchain, and its performance is of high priority.

7.1.1 Getting blocks by height on the same machine as database

Tests were performed by running a simple script that accesses the database and fetches blocks that fit the specified range of block height. The query that was used in this test is as follows:

```
db.blocks.find({'height': {'$gte': lower_bound, '$lte': upper_bound}})
```

The query is run on the previously instantiated object of the database connection. Because pymongo library used in those tests fetches documents lazily, the result of the query was iterated over to truly fetch the data. Time stated in test results is time between running the query and retrieving all documents. In case of this tests, the testing script was run on the server hosting the database.

Consecutive blocks test

Requested blocks are taken just as they are in blockchain.

Test results:

```
blocks between height 0 - 99:                0.0035 s
blocks between height 100000 - 100099:       0.0065 s
blocks between height 200000 - 200099:       0.48 s
blocks between height 300000 - 300099:       1.7 s
blocks between height 400000 - 400099:       7.9 s
```

Spread blocks test

Requested blocks are taken from larger hight range, and are equally distributed in that range. For each range, 100 blocks are fetched.

Test results:

```
100 blocks spread between height 0 - 100000:            0.49 s
100 blocks spread between height 100000 - 200000:       1.2 s
```

```
100 blocks spread between height 200000 - 300000:       1.6 s
100 blocks spread between height 300000 - 400000:       3.5 s
```

7.1.2 Getting blocks by height with accessing the database by network

These tests were identical to previous ones, with exception of running them on machine separated from database server by the Internet.

Consecutive blocks test

Requested blocks are taken just as they are in blockchain.

Test results:

```
blocks between height 0 - 99:                 0.097 s
blocks between height 100000 - 100099:        0.15 s
blocks between height 200000 - 200099:        5.1 s
blocks between height 300000 - 300099:        13 s
blocks between height 400000 - 400099:        51 s
```

Spread blocks test

Requested blocks are taken from larger hight range, and are equally distributed in that range. For each range, 100 blocks are fetched.

Test results:

```
100 blocks spread between height 0 - 100000:            2.9 s
100 blocks spread between height 100000 - 200000:       5.7 s
100 blocks spread between height 200000 - 300000:       15 s
100 blocks spread between height 300000 - 400000:       42 s
```

7.1.3 Result interpretation

When comparing results for tests ran locally and over the network, about ten fold difference in time can be noticed. This means that performance of the database is not a bottleneck and there is no need for further optimizations.

Another noticeable thing is that query time highly depends on the height of blocks requested. The reason for this is the fact that, because of gradual adoption of Bitcoin, the average number of transactions in blocks had grown over the years.

7.2 Getting blocks by timestamp

Querying by hight and by timestamp should not give different results. Both of those attributes are indexed, and both of those attributes have an equal amount of unique values. Because of that only one test was performed to test those assumptions.

Test result of run over network:

```
blocks between timestamp 2016-02-25 16:24 - 2016-02-26 07:23:    58 s
```

7.2.1 Result interpretation

Because the range of time that was tested fully corresponds to getting blocks by hight with the range of height being from 400000 to 400099, those results can be compared. There seems to be no significant difference (51 and 58 seconds) in performance between getting by height and by timestamp.

7.3 Complex operations API

In functionalities prepared in this component, performance is not the main goal. All available services will be requested to see if results are as expected.

7.3.1 Getting last block

Service is located at http://base-address/api/last_block and calling it returns JSON data:

Listing 7.1 Response on last block API call

```
{
    "_id": {
        "$oid": "5ae122209043650001213f12"
    },
    "hash": "000000000000000000c0af3be64d9a335968225b35
                f9fb102d4962c0ff2558c2",
    "version": 536870912,
    "height": 496509,
    "prev_hash": "000000000000000000384595a7dc898a505d5
                c131dffa1c55431dc73cffe2355",
    "merkle_root": "9df11de2a46b352591f5980d63929c68d9a
                f01bc2e494f4b350ae697809773ac",
    "timestamp": {
        "$date": 1511869304000
    },
    "n_tx": 2425,
    "size": 1079456,
    "bits": 402706678,
    .
    .
    .
}
```

Data that was returned has form and content as expected.

7.3.2 Getting count of maximal connected graphs by height

Service is requested with the address

```
http://base-address/api/count_separate_graphs?
height_from=100000&height_to=100500
```

In this case, it returns `[1701,74,42,39,21,20,20,19,17,...]` which means that biggest connected graph that can be made from transactions from blocks with height from 100000 to 100150 has 1701 vertices, second biggest has 74 vertices, and so forth.

Chapter 8

Beyond implementation

8.1 Usage

After implementation finished, the described software was used by a fellow student in research he conducted for his master thesis [10]. In his research, he performed cluster analysis of Bitcoin transactions. His opinion of this software, as he stated, had fully met his needs, but he expressed his concern for the security of current authentication system. In his opinion, if someone unauthorized would acquire credentials, denial of service attack could be performed.

8.2 Important addresses

Code repository can be found at:

`https://github.com/MarcinPieczka/Mongo-BTC-Blocks-Database`

Database (port 27357) and Complex operations API (port 80) can be found at:

`156.17.248.236`

8.3 Problems

In this section, several issues this software have will be addressed.

Lack of support for newest blocks is the problem with the biggest impact on the usability of this software. Without sorting it out some users will find this system incapable of finding their needs.

Authentication by one set of credentials can lead to security issues. At current version, every end user is provided with credentials to access functionalities. Those credentials are identical for every user and provide read-only access. One problem with that solution is that when one user uses too much resources, it is hard to find who this user is. Another problem arises when the administrator of the system wants to restrict access for some users. In this case, he would need to create new credentials and distribute them to all users that should have access to the system.

Maintenance can become problematic if the first problem will not be addressed. At this point library that is used to parse Bitcoin, data is used in version being fork of a master branch of its repository. Periodically new fork needs to be created, to include newest features of this library. Because this is not a release version, some bugs might appear, which would lead to failure in database updater. All of this leads to need for regular updates by someone who knows this system to some extent.

Complex operations API to be extended with new functionalities, need to be administered. Currently, there is no way, other than by pull request, to include new functionalities by users from outside of the project. This in return will lead to users not sharing their work so often, and maintenance of these new functionalities could not be easily done by original creators.

8.4 Future directions

As for the database updater, full support of Bitcoin protocol is needed, so that most recent data can be accessed. One way to achieve full Bitcoin support is by implementing lacking features in libraries that had been used. Doing so would add constant maintenance burden which could lead to the eventual abandonment of this project. Another solution might be researching different libraries to a bigger extent, with the hope of finding more complete one than the one that had been used.

The direction in which the complex operations API should be heading is depending on feedback from its users. Adding new functionalities might need to become simpler. In the current situation, to add new functionality and to share it with others, the code needs to be added to the main repository. This, of course, creates a need for reviewing proposed functionalities by the administrator of the repository. A possible solution might be creating a plug-in system that can run arbitrary modules, which would be maintained by their creators.

Chapter 9

Summary

In this work system providing easy access to Bitcoin blockchain data have been designed and implemented. There is a big problem of unnecessary work being done among Bitcoin researchers, many of them would develop their own way of accessing blockchain data, others would use web services, which usually have API call limits. To combat this problem there needed to be free, easy to use and maintain solution that provides access to Bitcoin data. Nonetheless accessing data is not the only field on which researchers have a problem which mutual code reuse, that's why platform, that is ready to implement functionalities on, needed to exist as well.

Success in development of this system if partial, because libraries that make parsing binary Bitcoin data have a problem with keeping up with changes in Bitcoin, the data of newest Bitcoin blocks is not fully available through the system.

Bibliography

[1] Andreas Antonopoulos, Mastering Bitcoin: Unlocking Digital Cryptocurrencies, 2014

[2] Bitcoin transaction, https://en.bitcoin.it/wiki/Transaction last accessed 27.05.2018

[3] Bitcoin script, https://en.bitcoin.it/wiki/Script last accessed 27.05.2018

[4] Block hashing algorithm, https://en.bitcoin.it/wiki/Block_hashing_algorithm last accessed 27.05.2018

[5] Definition of cryptocurrency, https://en.oxforddictionaries.com/definition/cryptocurrency last accessed 27.05.2018

[6] Description of BIP-0173, https://en.bitcoin.it/wiki/Bech32 last accessed 16.05.2018

[7] Description of Bitcoin confirmation process, https://en.bitcoin.it/wiki/Confirmation last accessed 27.05.2018

[8] Description of supply on Bitcoin, https://en.bitcoin.it/wiki/Controlled_supply last accessed 27.05.2018

[9] History of Bitcoin, https://en.wikipedia.org/wiki/History_of_bitcoin last accessed 27.05.2018

[10] Marcin Twardak, Finding clusters in Bitcoin cryptocurrency blockchain, master thesis to be defended in July 2018, supervisor Radosław Michalski

[11] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. Handbook of Applied Cryptography. ISBN 0-8493-8523-7. Archived from the original on 7 March 2005.

[12] Michael J. Casey Paul Vigna. Cryptocurrency. Random House UK Ltd, 2016

[13] Repository of BitcoinDatabaseGenerator, https://github.com/ladimolnar/BitcoinDatabaseGenerator last accessed 25.04.2018

[14] Repository of the blockchain-parser library, https://github.com/alecalve/python-bitcoin-blockchain-parser last accessed 16.05.2018

[15] Repository of libbitcoin-database, https://github.com/libbitcoin/libbitcoin-database, last accessed 25.04.2018

[16] Satoshi Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008

[17] Starry Peng, BITCOIN: Cryptography, Economics, and the Future, 2013

[18] Stinson Douglas, Cryptography theory and practice, 2006

[19] Thomsen, Søren Steffen; Knudsen, Lars Ramkilde, Cryptographic Hash Functions, 2009

[20] Web application allowing blockchain exploration, https://blockchain.info/, last accessed 16.05.2018

[21] Web application allowing blockchain exploration, https://blockexplorer.com/, last accessed 16.05.2018

[22] Website of MongoDB, https://www.mongodb.com/, last accessed 16.05.2018

[23] Website of Docker, https://www.docker.com/, last accessed 16.05.2018

[24] Website of docker-mongo-auth docker image, https://github.com/aashreys/docker-mongo-auth, last accessed 16.05.2018

[25] Website of python docker image, https://hub.docker.com/_/python/, last accessed 16.05.2018

[26] Website of uwsgi-nginx-flask docker image https://hub.docker.com/r/tiangolo/uwsgi-nginx-flask/, last accessed 16.05.2018

[27] Website of Python MongoDB API, https://api.mongodb.com/python/current/, last accessed 16.05.2018

[28] Website of Nginx, https://www.nginx.com/, last accessed 16.05.2018

[29] Website of Flask framework, http://flask.pocoo.org/, last accessed 16.05.2018

[30] Website of Python iGraph package, http://igraph.org/python/, last accessed 16.05.2018

[31] Zychal Bartosz, Analiza łańcucha tranzakcji w sieci Bitcoin, 2018, master thesis defended in February 2018, supervisor Radosław Michalski

Listings

List of Figures