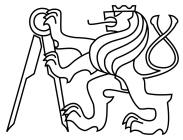
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vizualizace a vyhledávání v distribuované databázi
Student:	Bc. Martin Barus
Vedoucí:	Ing. Tomáš Zahradnický, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Navrhněte po dohodě s vedoucím práce kombinaci datového úložiště, schématu a vizualizačního prostředí vhodnou pro zadaná data grafového charakteru (platforma). Navrhněte vhodnou vizualizaci vztahů mezi uzly a skupinami uzlů v grafu. Při návrhu uvažujte situaci, ve které jeden či více uzlů v grafu mohou být sloučeny do jednoho shluku uzlů, a dále fakt, že hrany v grafu jsou orientované a mají svoji váhu a časovou značku. Nad zvolenou platformou navrhněte a implementujte vyhledávání podle vedoucím zadaných kritérií, minimálně však vyhledávání cest ze shluku A do shluku B a vyhledávání sousedních shluků ke shluku A. Všechna vyhledávání mohou mít omezení podle časové značky a/nebo váhy. Navrhněte a implementujte uživatelské rozhraní pro vyhledávání a zobrazování dat uživatelem. Implementaci otestujte s vedoucím zadanými daty a vyhodnoťte její časové a paměťové nároky.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Diplomová práca

Vizualizace a vyhledávání v distribuované databázi

Bc. Martin Barus

Vedúci práce: Ing. Tomáš Zahradnický, EUR ING, Ph.D.

9. mája 2017

Poďakovanie

Na tomto mieste sa chcem poďakovať vedúcemu práce Tomášovi Zahradnickému za cenné rady, svojím rodičom za ich srdečnosť, trpezlivosť a podporu. Rovnako ďakujem svojim blízkym.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. Ďalej prehlasujem, že som s ČVUT uzavrel dohodu, na základe ktorej sa ČVUT vzdalo práva na uzavrenie licenčnej zmluvy o používaní tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona. Táto skutočnosť nemá vplyv na ust. § 47b zákona č. 111/1998 Sb. o vysokých školách.

V Prahe 9. mája 2017 .....

Odkaz na túto prácu

Barus, Martin. *Vizualizace a vyhledávání v distribuované databázi.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Diplomová práca za zaoberá problematikou vyhľadávania, filtrovania a vizualizácie vzťahov. Cieľom práce je vytvoriť nástroj, schopný prehľadne zobraziť vzťahy medzi entitami pomocou grafovej vizualizácie podľa kritérií zadaných užívateľom. Práca navrhuje riešenie problémov uloženia dát, vyhľadávania v grafe a tvorbe užívateľského rozhrania. Práca ďalej skúma časové a pamäťové nároky výsledného implementovaného nástroja.

Klíčová slova vizualizácia vzťahov, grafové databázy, vyhľadávanie v grafoch, Neo4j

Abstract

The core topic of the thesis is searching, filtering and visualization of relationship data. Thesis aims to create a tool for convenient visualization of relationships between entities through graph visualizations, based on user specified criteria. Thesis proposes solutions of data storage, graph search and user interface. Runtime and memory requirements of the implemented tool are then evaluated.

Keywords relationship visualization, graph databases, graph search, Neo4j

Obsah

Zoznam obrázkov

Úvod

Pochopenie súvislostí medzi informáciami, udalosťami, ľudmi a ich správaním vedie k informačnej výhode, ktorú je možné zúročiť mnohými spôsobmi. Napríklad sociálne grafy sociálnych sietí umožňujú vytváranie rôznych analýz podobností užívateľov a cielených marketingových kampaní. Najprehľadnejšou a najpochopiteľnejšou interpretáciou vzťahov je pre človeka vizualizácia vzťahov pomocou grafov. Uzly grafov zvyčajne reprezentujú entity problému a orientované hrany predstavujú vzťahy medzi entitami.

Cieľom diplomovej práce je navrhnúť aplikáciu, schopnú vizuálne zobraziť vzťahy obsiahnuté vo veľkom objeme dát. Aplikácia bude umožňovať v dátach vyhľadávať a výsledky vyhľadávania následne filtrovať podľa zvolených kritérií. Dáta vyhovujúce kritériám zadaných užívateľom budú následne vizualizované formou grafu. Aplikácia bude schopná združovať uzly grafu do zhlukov a vyhľadávať cesty medzi zhlukmi a k zvolenému zhluku jeho susedné zhluky. Užívateľ bude môcť výsledné cesty grafu filtrovať podľa časových značiek a váh hrán ciest. Výstupom práce bude okrem aplikácie tiež zhodnotenie jej pamäťových a časových nárokov. K naplneniu cieľa bude nutné vyriešiť problémy uloženia dát, návrhu dátovej schémy a vytvorenia užívateľského rozhrania.

Z rôznych voľne dostupných zdrojov dát, zachytávajúcich vzťahy medzi uzlami, ponúkajú dáta kryptomeny Bitcoin cez dvesto miliónov uzlov transakcií a minimálne rovnaký počet uzlov výstupov transakcií, nehovoriac o ďalších uzloch a entitách dát. Aplikácia schopná vizualizovať vzťahy takéhoto množstva dát môže byť použitá na vizualizáciu mnohých iných zdrojov dát.

Prvá kapitola práce predstaví dáta kryptomeny Bitcoin a popíše systém kolektívneho schvaľovania transakcií. Druhá kapitola popíše rôzne spôsoby uloženia dát, vyhľadávania, filtrovania a vizualizácie dát. Tretia kapitola predstaví návrh riešenia, nad technológiami zvolenými na základe analýzy v predchádzajúcej kapitole. Štvrtá kapitola popíše implementáciu aplikácie a piata kapitola zhodnotí funkčnosť aplikácie a jej pamäťové a časové nároky.

Kapitola 1

Bitcoin infraštruktúra

Bitcoin [39] je celosvetovo známou kryptomenou, operujúcou bez centrálnej autority a iných tretích strán. Správa transakcií, objemu meny a celkového chodu systému je zabezpečená automatickou kolektívnou činnosťou účastníkov peer-to-peer siete, za čo sú účastníci odmeňovaní virtuálnou menou či už v podobe ťažby nových zdrojov alebo ziskom z transakčných poplatkov. Transakcie a všetky ich informácie sú voľne viditeľné každému účastníkovi, napriek tomu kryptomena v prípade správneho používania ponúka vysokú mieru anonymity.

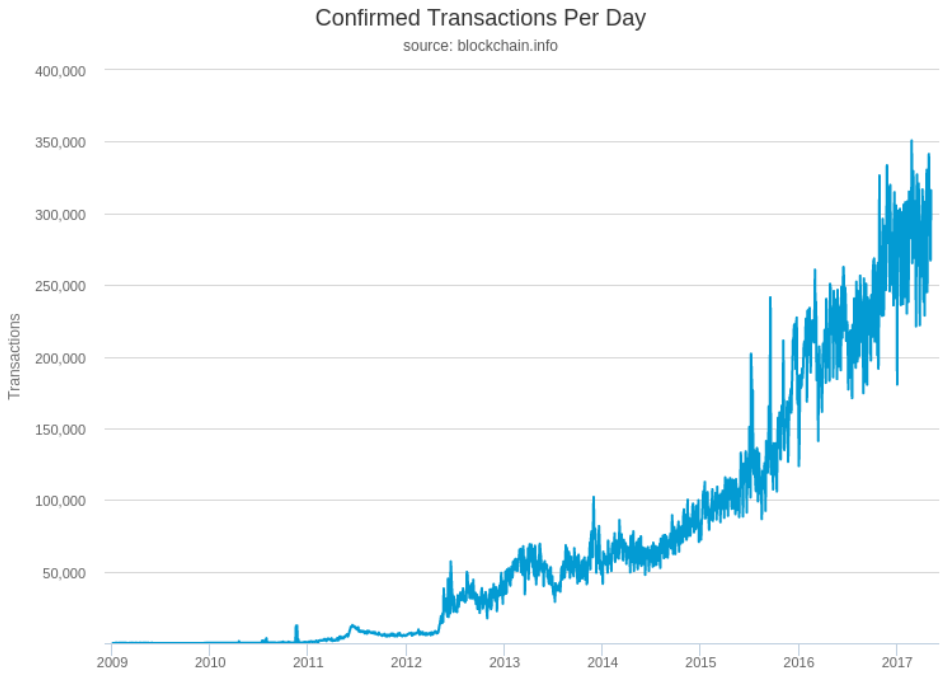
Okrem peer-to-peer systému mena naviac využíva oddelený software, peňaženky, uchovávajúce dodatočné dáta, schopné identifikovať príslušnosť transakcií užívateľom a preukázať vlastníctvo objemu meny. Dáta peňaženiek nie sú verejne viditeľné a v peer-to-peer sieti sa nenachádzajú. Každý účastník môže mať svoju vlastnú peňaženku, na svojom vlastnom hardware, ktorý má plne pod kontrolou.

1.1 Bitcoin

Myšlienka kryptomeny Bitcoin s potrebnými náležitosťami na automatické peer-to-peer overovanie a schvaľovanie transakcií bola publikovaná v článku Bitcoin: A Peer-to-Peer Electronic Cash System [39] v novembri roku 2008 autorom či skupinou autorov vystupujúcich pod pseudonymom Satoshi Nakamoto. V januári nasledujúceho roku bol publikovaný prvý Bitcoin klient s otvoreným zdrojovým kódom [40] a vyťažený prvý blok s hodnotou 50 bitcoinov.

1.1.1 Vývoj meny

Bitcoin sa od svojho fyzického vzniku v roku 2009, kedy bol menou s takmer nulovou hodnotou, pod hranicou 0.01 USD za 1 BTC a denným počtom transakcií okolo sto transakcií denne stal pomerne využívanou menou s aktuálnou



Obr. 1.1: Historický denný počet potvrdených Bitcoin transakcií. Graf je verejne dostupný v rámci online analytických nástrojov spoločnosti Blockchain Ltd. [21].

hodnotou 1 536 USD za 1 BTC ku dňu 8. 5. 2017 a denne sa v apríli roku 2017 uskutočnilo v priemere okolo 300 000 transakcií [22] [21]. Z dôvodu správnej funkčnosti meny musí byť každá historicky schválená transakcia uložená v lokálnej databáze každého uzlu peer-to-peer siete, k dátumu 8. 5. 2017 to činí 220 256 650 transakcií [23].

1.1.2 Incidenty

Tak ako sa zvýšila cena meny Bitcoin, zvýšila sa aj snaha o obohatenie sa na tejto mene prostredníctvom podvodov, podobne ako v prípade bežných mien. Útokov a incidentov v infraštruktúre meny Bitcoin bolo mnoho, podľa spôsobu napadnutia je možné kategorizovať útoky do dvoch druhov:

- útoky vyplývajúce z nedokonalosti peer-to-peer systému schvaľovania a overovania transakcií, teda útoky na verejne publikovanú a jednotnú zložku infraštruktúry
- útoky na samotné peňaženky a snaha zmocniť sa súkromných kľúčov a teda vlastníctva užívateľov

Obr. 1.2: Vývoj hodnoty menového páru BTC / USD. Graf je verejne dostupný v rámci online nástrojov spoločnosti Blockchain Ltd. [22].

1.1.2.1 Útoky na peer-to-peer systém

Útoky na samotný peer-to-peer systém meny Bitcoin sú principiálne nebezpečnejšie, pretože znamenajú zraniteľnosť meny a v prípade prelomenia systému mena stratí hodnotu, keďže stratí dôveryhodnosť.

Najvýraznejší útok na peer-to-peer systém sa stal 15. augusta 2010, kedy transakcia vytvorila 184 467 440 737.09551616 bitcoinov odoslaných na tri rôzne adresy. Tento útok bol úspešný, keďže kód overovania nepočítal s posielanými sumami tak veľkými, že dôjde ku pretečeniu v prípade ich sčítania [20].

Nová verzia Bitcoin klienta bola vydaná päť hodín po objavení incidentu, tá obsahovala nové spoločné pravidlo zákazu všetkých transakcií, ktoré použili výstup v hodnote prevyšujúcej 21 miliónov bitcoinov. Nasadením nových verzií Bitcoin klientov v dostatočnom počte uzlov siete bolo dosiahnuté postupné vymazanie tejto transakcie v peer-to-peer sieti 9 hodín od jej vzniku.

1.1.2.2 Útoky na peňaženky

Druh útokov smerovaný na odcudzenie privátnych kľúčov adries, ktoré prijali bitcoiny a majú ich k dispozícii priamo neohrozuje všetkých užívateľov siete, riziko útoku neznamená slabinu siete ale slabinu procesu spravovania kľúčov,

čo je úlohou peňaženiek. Zodpovednosť za správu kľúčov nesú užívatelia.

Historicky najväčšia škoda, spôsobená útokom na peňaženky, vznikla v roku 2014, útokom na najväčšiu Bitcoin burzu Mt.Gox Co., Ltd., ktorá spravovala 70% všetkých Bitcoin transakcií. Spoločnosť v apríli 2014 oznámila, že 850 000 bitcoinov, patriacich zákazníkom bolo pravdepodobne ukradnutých a nezvestných. Škoda mala v tom čase hodnotu cez 450 miliónov dolárov [16]. Dôkazy predložené v apríli 2015 vedú k záveru, že väčšina, možno všetka škoda, bola spôsobená ukradnutím bitcoinov priamo z peňaženiek spoločnosti.

Podľa štúdie [53] z Februára roku 2014 v tom čase existovalo 146 rôznych druhov počítačových Bitcoin vírusov. Víry boli v tom čase schopné ukradnúť obsah zašifrovaných súborov peňaženiek, obísť dvojfázovú autentifikáciu, odolať antivírovým programom, predstierať totožnosť užívateľa a tiež predstierať totožnosť webovej stránky internetových peňaženiek.

1.2 Databáza Blockchain

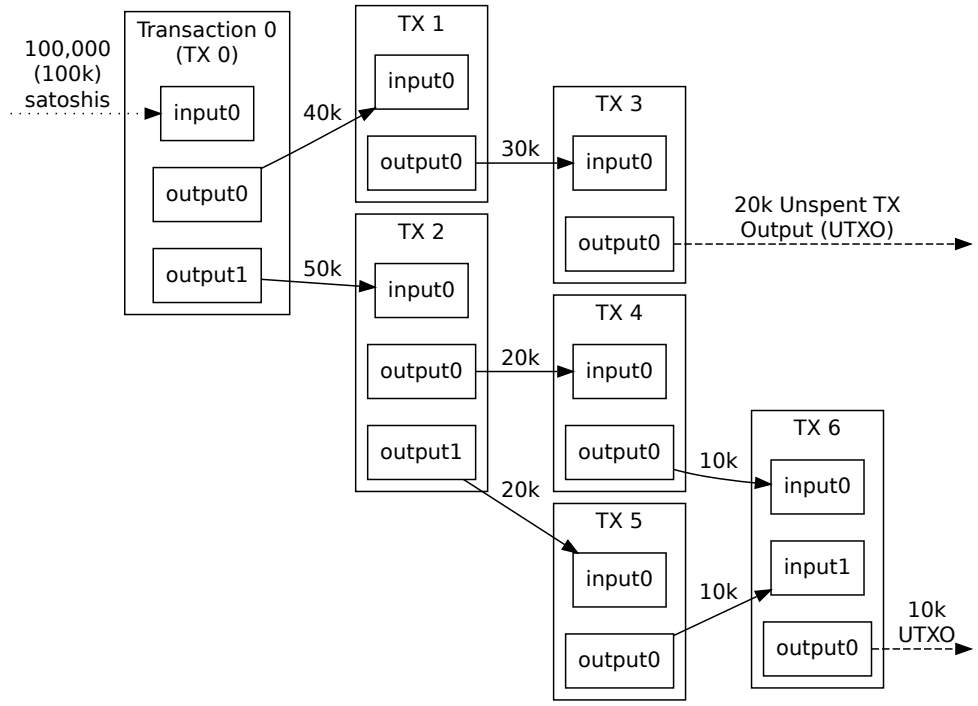
Blockchain je distribuovaná databáza, ktorá svojím návrhom znemožňuje pozmeniť uložené dáta bez vplyvu na nasledujúce dáta uložené v databáze. Zmena dát v istom momente zneplatní všetky dáta uložené chronologicky neskôr. Táto vlastnosť je kľúčová pre rôzne systémy, jasným príkladom je finančný sektor.

Bezhotovostné transakcie sú v dnešnej dobe bežnou vecou a samozrejmosťou. Pri transakcii medzi dvoma subjektami je štandardne potreba tretej strany, banky, ktorá dokáže a ručí za to, že odosielateľ posiela transakciu v objeme, ktorý má dostupný. Banka je teda prostredník, ktorý musí mať prístup k informáciám o subjektoch a vedieť odfiltrovať neplatné transakcie. Každá snaha vynechať finančnú inštitúciu z procesu odosielania a prijímania transakcií teda musí nutne riešiť problém overenia dostupných zdrojov odosielateľa a samotný proces prevedenia transakcie.

Vďaka svojím vlastnostiam tvorí blockchain vhodný základ pre implementáciu kryptomien, ako napríklad Bitcoin, ktoré umožňujú realizáciu a overovanie transakčných procesov, bez možnosti manipulácie s historickými transakciami. V nasledujúcich odstavcoch bude blockchain popísaný v kontexte kryptomeny Bitcoin, ktorá je čoraz viditeľnejšia a stáva sa paralelným spôsobom platby ku klasickým štátnym menám.

Databáza Blockchain uchováva všetky informácie potrebné na uloženie a overenie každej vykonanej transakcie. Transakcia samotná má n vstupov a m výstupov, pričom každý výstup je hodnota bitcoinov odoslaná na istú adresu a každý vstup je čerpanie plnej hodnoty predchádzajúceho výstupu inej transakcie. Transakcie sú usporiadané v blokoch, ktoré umožňujú definovať chronologické poradie transakcií.

O odosielanie transakcií a udržovanie informácií o dostupných zdrojoch a adresách konkrétneho užívateľa sa stará špecializovaný software úplne odde-



Obr. 1.3: Diagram transakcií, pochádzajúci z popisu kryptomeny Bitcoin [13] znázorňuje propagáciu výstupov transakcií a ich použitie ako vstupov do nových transakcií. Transakcie môžu mať jeden či viac vstupov, ako napríklad transakcia TX6, tiež môžu mať rôzny počet výstupov. Každý vstup transakcie míňa práve jeden výstup predchádzajúcej transakcie, na ktorý sa odkazuje, a to v jeho plnej výške.

lený od Blockchain databázy, takzvané peňaženky. Informácie o príslušnosti adries užívateľom a ich príslušným údajom, privátnych kľúčoch, nie sú v databáze uložené. Doporučeným postupom je pre každú prijatú transakciu použiť novú adresu, pričom generovanie nových adries má za úlohu peňaženka. Adresa sa do databázy zapíše až v prípade odoslania transakcie na danú adresu, jej príslušnosť konkrétnemu užívateľovi však nie je známa.

1.2.1 Transakcia

Základným prvkom Blockchain databázy je transakcia. Každá transakcia má minimálne jeden vstup a minimálne jeden výstup. Počet vstupov a výstupov transakcie sa môže líšiť. Jeden vstup či výstup transakcie je vlastne jeden konkrétny objem bitcoinov. Transakcia je jediný spôsob zlučovania skupiny malých objemov do jedného väčšieho objemu a tiež rozdelenia väčšieho ob-

jemu do viac menších objemov. V rámci transakcie sa teda všetky vstupy sčítajú, označia sa za minuté a celková suma sa prerozdelí do nových objemov v hodnote výstupov transakcie a transakčných poplatkov.

Dôležité je, že vstupy transakcie, teda jednotlivé vstupné objemy, sú v rámci transakcie minuté v plnej celkovej sume. To má často za následok nutnosť dvoch výstupov v prípade jednoduchej transakcie. Ak má totiž užívateľ na svojom konte sumu bitcoinov v istej hodnote a v rámci novej transakcie si želá poslať zlomok tejto časti, je pravdepodobné, že na svojom účte nemá konkrétny objem v hodnote zlomku, ale viac rôznych objemov.

V prípade, že chce subjekt A poslať transakciu subjektu B v hodnote H, musí byť suma vstupných objemov transakcie, teda suma vstupov, väčšia alebo rovná hodnote H. Ak suma vstupných objemov presahuje hodnotu H, v rámci jednej transakcie sa tieto jednotlivé objemy zlúčia a následne rozdelia na objem v hodnote H, poslaný subjektu B a objem v hodnote zvyšku sumy, ktorý je poslaný naspať subjektu A. V prípade, že je suma vstupných objemov presne rovná hodnote H, vznikne iba jeden výstup poslaný subjektu B.

Transakcia [19] explicitne obsahuje následovné údaje binárne uložené za sebou:

1. **version no** — verzia transakcie, aktuálne hodnota 1, v budúcnosti sa môže zmeniť, 4 byty;
2. **in counter** — počet vstupov transakcie, 1-9 bytov;
3. **list of inputs** — vstupy transakcie uložené ako zoznam, veľkosť vyplýva z počtu vstupov;
4. **out counter** — počet výstupov transakcie, 1-9 bytov;
5. **list of outputs** — výstupy transakcie uložené ako zoznam, veľkosť vyplýva z počtu výstupov;
6. **lock time** — indikuje najskorší možný čas prijatia transakcie, umožňuje zmazať; transakciu ak do časového limitu použije nová transakcia nejaký zo vstupov danej transakcie, 4 byty.

Každá transakcia je jednoznačne identifikovateľná pomocou identifikátoru **txid**. Hodnota identifikátora však nie je uložená, je vypočítaná ako výsledok dvojnásobného aplikovania kryptografickej hash funkcie SHA256 [52] na uložených binárnych dátach transakcie.

1.2.1.1 Vstup transakcie

Vstup transakcie je odkaz na výstup minulej transakcie, celý objem výstupu je v danej transakcii minutý a po potvrdení transakcie sa už nedá znova použiť. Obsahuje následovné dáta v binárnej podobe [19]:

1. **outpoint** — 32-bytový hash transakcie, z ktorej odkazovaný výstup pochádza a 4-bytový index, určujúci poradie výstupu v rámci zoznamu výstupov danej transakcie;
2. **script length** — veľkosť poľa podpisu 1-9 bytov;
3. **signature script** — podpis, dáta vygenerované odosielateľom transakcie, preukazujúce jeho vlastníctvo odkazovaného výstupu, veľkosť je uložená v predošlej položke;
4. **sequence number** — číslo vytvorené za zámerom zmeny nepotvrdených časovo obmedzených transakcií, aktuálne sa používa iba pre zrušenie časového obmedzenia transakcie, 4 byty.

1.2.1.2 Výstup transakcie

Výstup transakcie určuje čiastku a cieľovú destináciu, adresu, transakcie. Výstup transakcie obsahuje následovné položky [19]:

1. **value** — počet odoslaných satoshi, 1 satoshi je 10^{-8} BTC, 8 bytov;
2. **script length** — veľkosť poľa popisujúceho verejný kľuč, 1-9 bytov;
3. **pubkey script** — definuje podmienky preukázania vlastníctva v momente budúceho čerpania, veľkosť je uložená v predošlej položke.

Výstupy jednej transakcie sú uložené vrámci zoznamu výstupov danej transakcie, preto samotný výstup nemá uloženú žiadnu informáciu o transakcii, ktorej patrí, navyše, jeho poradové číslo, na ktoré sa vstupy transakcií odvolávajú, je tiež implicitné a v databáze uložené nie je.

Naplnenie definovaných podmienok preukázania vlastníctva pomocou hodnoty *pubkey script* sa dokáže príslušnou hodnotou *signature script* uloženou v momente minutia výstupu novou transakciou. Pole *pubkey script* obsahuje adresu, na ktorú je daná čiastka poslaná.

Následovný príklad demonštruje formu uloženej transakcie [19]. Položky *OP_DUP*, *OP_HASH160*, *OP_EQUALVERIFY* a *OP_CHECKSIG* reprezentujú operácie pre overenie odpovedajúcich položiek. Prekladajú sa do 1-bytového kódu. Výsledná transakcia vniká uložením hodnôt položiek v binárnej podobe za sebou bez popisov položiek.

```
version: 1
in counter: 1
list of inputs:
    input:
        outpoint:
            previous tx: f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9
                cea205b009ca73dd04470b9a6
```

```
                index: 0
        script length: 80
        signature script: 304502206e21798a42fae0e854281abd38bacd
            1aeed3ee3738d9e1446618c4571d10 90db022100e2ac980643b
            0b82c0e88ffdfec6b64e3e6ba35e7ba5fdd7d5d6cc8d25c6b241
            501
        sequence number: 4294967295
out counter: 1
list of outputs:
    output:
        value: 5000000000
        script length: 16
        pubkey script: OP_DUP OP_HASH160 404371705fa9bd789a2fcd5
            2d2c580b65d35549d OP_EQUALVERIFY OP_CHECKSIG
lock time: 0
```

1.2.2 Blok

Blok je zoradená postupnosť transakcií, pričom pre každú transakciu v rámci bloku platí, že všetky výstupy, na ktoré sa jej vstupy odvolávajú, patria transakciám uloženým buď v predošlých blokoch alebo v aktuálnom bloku pozične pred aktuálnou transakciou. Toto pravidlo definuje chronologické poradie transakcií a zabezpečuje, že transakcie môžu minúť iba výstupy už schválených transakcií a nemôžu používať fiktívne vstupy.

Blok [18] obsahuje:

1. **magic no** — hodnota 0xD9B4BEF9, umožňuje ľahkú identifikáciu začiatku bloku, 4 byty;
2. **blocksize** — veľkosť bloku v bytoch, zaberá 4 byty;
3. **blockheader** — dodatočné informácie popísané v ďalšej podkapitole, 80 bytov;
4. **transaction counter** — počet transakcií v danom bloku, 1-9 bytov;
5. **transactions** — transakcie bloku v binárnej podobe uložené za sebou, veľkosť je možné odvodiť z veľkosti bloku.

1.2.2.1 Blockheader

Hlavička bloku [15] obsahuje dodatočné informácie bloku a to:

1. **version** — verzia bloku, informácia pre možnosti budúcich zmien uchovávania a spracovávania blokov, 4 byty;
2. **hashPrevBlock** — hash predchádzajúceho bloku, 32 bytov;

3. **hashMerkleRoot** — hash všetkých transakcií v bloku, 32 bytov;
4. **time** — časová značka zmeny bloku, 4 byty;
5. **bits** — 32-bytové číslo, určujúce obtiažnosť ťažby bloku, blok s hashom nižším rovným danej hodnote bude prijatý a tým pádom vyťažený;
6. **nonce** — náhodné 4-bytové číslo, zmenou čísla sa mení aj časová značka zmeny bloku a odpovedajúci hash bloku, ten, kto nájde kombináciu čísla a zmeny času s dostatočne nízkou hodnotou hashu, vyťaží blok.

V hlavičke sú uložené dva hashe, hash predchádzajúceho bloku a hash transakcií uložených v aktuálnom bloku usporiadaných do merklovského stromu. Vďaka tomu nie je možné zmeniť dáta uložené v predošlom bloku, keďže každý blok v sebe ukrýva kumulatívny hash všetkých predošlých blokov. Hash aktuálneho bloku uložený nie je a je možné získať ho vypočítaním hashu zdrojových dát.

1.2.3 Adresa

Adresa je 26-35 znakov dlhý alfanumerický reťazec, ktorý sa používa za účelom špecifikácie adresáta, príjemcu, transakcie. Príkladom adresy je reťazec 36DDDcKSb8a8c5iKCndw3hstNn7qjtG33r. Adresa je vypočítaná ako hash verejného kľúča peňaženky, popísanej nižšie. Adresa je fyzicky v Blockchain databáze uložená nepriamo ako súčasť položky *pubkey script* výstupu transakcie [8]. Adresy sú generované peňaženkami, vygenerovanie novej adresy sa v databáze nijak neprejaví do momentu, kedy je táto adresa použitá ako cieľ výstupu transakcie.

Podľa doporučení [3] by mal užívateľ prijímať každú transakciu na novú adresu, keďže informácie o príslušnosti rôznych adries jednému účtu nie sú nikde verejne uložené. V prípade, že užívateľ odosiela transakciu a minimálna suma zozbieraných vstupov, potrebných pre zlúčenie požadovaného objemu bitcoinov presahuje požadovaný objem, je užívateľovi peňaženkou vygenerovaná nová adresa, kam putuje zvyšok sumy.

1.2.4 Fyzická podoba Blockchain databázy

Blockchain databáza je postupnosť blokov rozdelená do súborov na disku, pričom bloky sú postupnosti transakcií a transakcie obsahujú postupnosti vstupov a výstupov. Okrem samotných dát sa pre rýchlejšie overovanie blokov používajú indexy, uložené v LevelDB databáze.

1.2.4.1 Transakčné dáta

Fyzická podoba uloženia dát je veľmi priamočiara. Bloky sú binárne uložené za sebou s poradím bytov Litte-endian, tvoria reťaz, tá začína prvým blokom a

postupuje ďalšími blokmi, blok má uložené informácie o svojej fyzickej veľkosti a počte transakcií. Transakcie v bloku sú uložené opäť zreťazené za sebou a každá transakcia obsahuje informácie o počte vstupov a výstupov. Vstupy a výstupy sú znova uložené v súvislej postupnosti. Výsledná reťaz blokov je rozdelená do menších podreťazí a tie sú uložené na disku ako samostatné súbory s formátom *blk*.dat* [17].

Okrem schválených transakčných dát sa na disku nachádzajú aj pomocné dáta, potrebné pre prípadné odvolanie aktuálnych zmien lokálnej kópie Blockchain databázy z dôvodu reorganizácie blokov vo formáte *rev*.dat* [17].

1.2.4.2 Indexy

Pre rýchle použitie Blockchain databázy sa používajú dva druhy indexov [17]. Prvý je index blokov, ktorý ako kľúč používa index bloku a umožňuje lokalizovať blok v konkrétnom mieste Blockchain databázy, teda v konkrétnom súbore na konkrétnom mieste. Bez tohto indexu by trvalo overovanie blokov veľmi dlho.

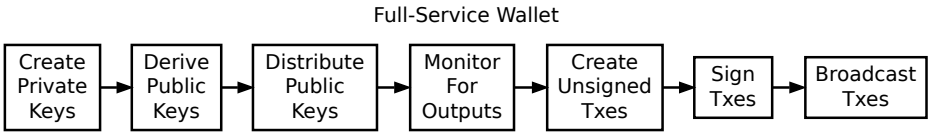
Druhým indexom je index uchovávajúci informácie o doteraz neminutých výstupoch všetkých predchádzajúcich transakcií, tento index umožňuje rýchlo overiť platnosť vstupov a zamedziť problému dvojitého minutia zdrojov. Bez tohto indexu by bolo nutné pre overenie každej novej transakcie prejsť celú databázu Blockchain.

1.3 Peňaženky

Kryptomena Bitcoin je často označovaná za anonymnú menu, v skutočnosti je to však mena pseudonymná, každá transakcia je uložená so všetkými údajmi, ktoré sú verejne dostupné. Keďže výstupy transakcií obsahujú informácie o cieľových adresách, anonymita spočíva v predpoklade, že príslušnosť skupiny adries konkrétnemu užívateľovi je neznáma. Na splnenie tohoto predpokladu je nutné, aby užívateľ používal čo najčastejšie rôzne adresy, v prípade používania jednej adresy sú všetky transakcie užívateľa identifikovateľné danou adresou.

Peňaženka je software, spravujúci súkromné a verejné kľúče užívateľa, ktorá mu umožňuje prijímať a odosielať bitcoiny. Funkcie peňaženky sú následovné [3].

- generovanie súkromných a im prislúchajúcich verejných kľúčov, hashom verejného kľúča vznikne adresa
- monitorovanie prijatých výstupov transakcií, odoslaných na adresy peňaženky
- vytváranie a podpisovanie transakcií, míňajúcich prijaté výstupy



Obr. 1.4: Diagram odoslania transakcie použitím plnohodnotnej peňaženky, pochádzajúci z online popisu kryptomeny [6]. Peňaženka vytvára súkromné a im odpovedajúce verejné kľúče, verejné kľúče rozdistribuuje, monitoruje výstupy transakcií, vytvára, podpisuje a vysiela transakcie do peer-to-peer siete.

Odosielanie nových transakcií a správa adries je úlohou peňaženky. Peňaženka ku každej svojej adrese, verejnému kľúču, uchováva privátny kľúč. V prípade, že chce užívateľ prijať novú transakciu, vygeneruje si peňaženkou nový verejný kľúč a príslušný súkromný kľúč, z verejného kľúča peňaženka hashom spočíta adresu a užívateľ adresu zverejní odosielateľovi transakcie.

V prípade odosielania transakcie peňaženka zozbiera sumu väčšiu alebo rovnú požadovanému objemu transakcie z dostupných zdrojov, teda predchádzajúcich výstupov iných transakcií odoslaných na adresy patriace peňaženke, a pre každý taký výstup použije privátny kľúč jeho cieľovej adresy na preukázanie vlastníctva a tým vygeneruje hodnotu *signature script* a vytvorí tak vstup do aktuálnej transakcie. Podľa cieľových adries aktuálnej transakcie, adries na ktoré užívateľ posiela bitcoiny, vygeneruje peňaženka hodnoty *pubkey script* a výstupy aktuálnej transakcie.

1.3.1 Funkčné delenie

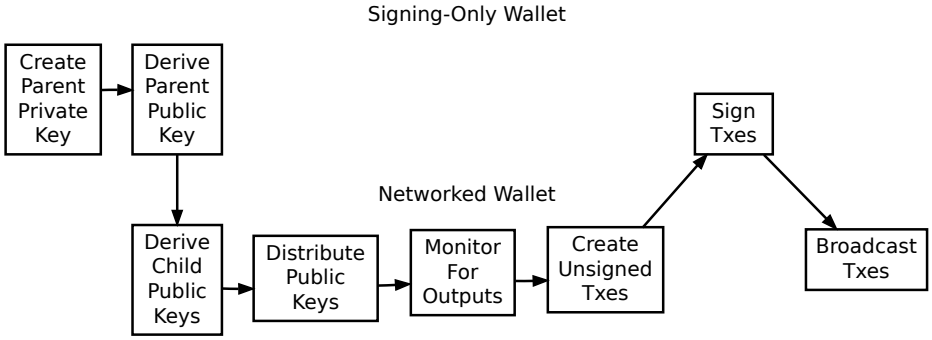
Podľa funkčného postupu peňaženiek je možné peňaženky rozdeliť na plnohodnotné peňaženky, podpisovacie peňaženky a distribuované peňaženky [3].

1.3.1.1 Plnohodnotná peňaženka

Plnohodnotná peňaženka spĺňa všetky funkcie sama, teda vygeneruje súkromné kľúče, *Create Private Keys*, z nich odvodí verejné kľúče, *Derive Public Keys*, tie rozdistribuuje, *Distribute Public Keys*, monitoruje prijaté výstupy, *Monitor For Outputs*, vytvorí transakcie, *Create Unsigned Txes*, podpíše transakcie, *Sign Txes* a nakoniec ich vyšle broadcastom do peer-to-peer siete [6].

1.3.1.2 Podpisovacia peňaženka

Pre zvýšenie bezpečnosti existujú čisto podpisovacie peňaženky, tie fungujú v spolupráci s inými peňaženkami v sieti. Podpisovacia peňaženka vytvorí jeden rodičovský súkromný a verejný kľúč, verejný kľúč pošle peňaženke v sieti,



Obr. 1.5: Diagram odoslania transakcie použitím podpisovacej peňaženky v kombinácii s inou peňaženkou v sieti, pochádzajúci z online popisu kryptomeny [11]. Podpisovacia peňaženka vytvorí súkromné a verejné kľúče, iná peňaženka z verejných kľúčov odvodí nové verejné kľúče, rozdistribuuje ich, získa informácie o transakčných výstupoch kľúčov, vytvorí transakcie, tie následne podpíše pôvodná podpisovacia peňaženka a nakoniec podpísané transakcie peňaženka v sieti odošle.

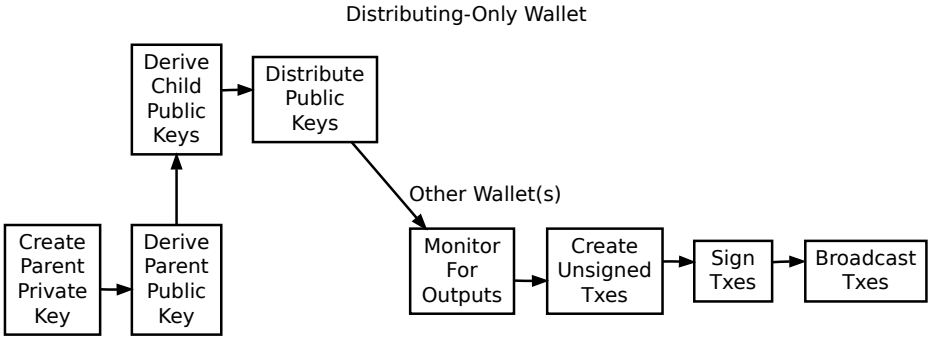
peňaženka v sieti potom robí všetky kroky plnohodnotnej peňaženky, až na podpisovanie transakcie, teda vytvorí množinu verejných a súkromných kľúčov, monitoruje výstupy, vytvára transakcie, transakcie musia byť následne podpísane podpisovacou peňaženkou a podpísané transakcie peňaženka v sieti broadcastom odošle [11].

1.3.1.3 Distribuovaná peňaženka

V prípade, že je peňaženka prevádzkovaná v ťažko zabezpečiteľnom prostredí, je možné použiť prístup distribuovaných peňaženiek. Užívateľ má pôvodnú peňaženku, ktorá obsahuje jeden rodičovský súkromný kľúč a z neho odvodený verejný kľúč. Rodičovský verejný kľúč je poslaný distribuovanej peňaženke, tá z neho odvodí potomka, nový verejný kľúč a ten rozošle iným peňaženkám [5]. Tieto peňaženky potom monitorujú výstupy, vytvoria nové transakcie, podpíšu ich odvodeným verejným kľúčom a následne ich rozošlú do peer-to-peer siete.

1.3.2 Delenie podľa média

Podľa média a spôsobu uloženia kľúčov môžme peňaženky rozdeliť na fyzické, hardvérové a softwarové.



Obr. 1.6: Diagram odoslania transakcie použitím distribuovanej peňaženky v kombinácii s inými peňaženkami v sieti, pochádzajúci z online popisu kryptomeny [5]. Distribuované peňaženky slúžia na vytváranie a distribúciu verejných kľúčov v ťažko zabezpečiteľnom prostredí. Ostatné potrebné úkony spojené so správou kľúčov a transakcií sú vykonávané v rámci iných peňaženiek. Typický príklad je generovanie a zverejnenie nových adries na webovom serveri, pričom správa privátnych kľúčov a ďalšia správa prostriedkov bude vykonávaná mimo webového servera.

1.3.2.1 Fyzická peňaženka

Niekedy sa pojmom peňaženka označujú samotné páry verejných a súkromných kľúčov. Jeden takýto pár je v prípade papierovej peňaženky vytlačený v podobe QR kódov na papieri alebo kartičke. Problémom fyzických peňaženiek je ich neschopnosť generovať nové súkromné a z nich odvodené verejné kľúče, rovnako nie sú schopné vytvoriť, podpísať a odoslať transakcie, používajú sa vždy s pomocou iných peňaženiek. Takéto peňaženky sa odporúča použiť iba raz na prenesenie ich zdrojov do iného druhu peňaženky. Na odoslanie transakcie z takejto peňaženky je nutné naskenovať okrem verejného aj privátny kľúč, keďže sa ním podpisuje časť transakcie.

Fyzické peňaženky je možné použiť viac krát, závisí to však od správneho použitia pomocnej peňaženky, ktorá transakciu vytvorí, podpíše a odošle, opakované použitie fyzickej peňaženky sa ale nedoporučuje. Riziko straty bitcoinov vzniká v prípade potreby odoslania časti čerpanej hodnoty naspäť odosielateľovi, v prípade, že transakcia nečerpá plnú sumu vstupov, ale iba jej časť. Tradične by v takom prípade peňaženka odosielateľa vygenerovala nový pár kľúčov a zvyšok transakcie odoslala na novo vzniknutú adresu. Ak by to však pomocná peňaženka urobila, majiteľ papierovej peňaženky sa nedozvie nový pár kľúčov a o zbytkovú čiastku transakcie by prišiel. Pomocná peňaženka preto musí vrátiť zvyšok vstupu transakcie na rovnakú adresu, z ktorej obnos čerpala. Takýto postup pomocnej peňaženky nie je garantovaný. Závisí od implementácie použitej peňaženky. V momente opakovaného použitia adresy je

naviac možné spárovať transakcie k jednému užívateľovi, čím sa znižuje miera anonymity.

1.3.2.2 Hardvérová peňaženka

Hardvérové peňaženky sú podpisovacie peňaženky. Transakcie sú vytvorené inými peňaženkami a hardvérové peňaženky ich podpíšu svojimi súkromnými kľúčmi. Výhodou oproti papierovým peňaženkám je fakt, že hardvérové peňaženky môžu generovať a uchovávať nové páry kľúčov, a teda ich opakované použitie neohrozuje anonymitu užívateľa [33].

1.3.2.3 Softvérová peňaženka

Softwarové peňaženky sú najrozšírenejšími peňaženkami, líšia sa v mnohých ohľadoch, najvýraznejší rozdiel je v potrebnom množstve uložených dát. Softvérové peňaženky, ktoré overujú prijaté transakcie svojou lokálnou kópiou Blockchain databázy vyžadujú toľko miesta, koľko aktuálne zaberá celá globálna Blockchain databáza, rádovo cez 100 GiB úložného priestoru. Ich výhodou je, že nepríjmu falošné transakcie odporujúce lokálnej kópii blockchain ani v prípade, kedy viac ako polovica uzlov celej Bitcoin siete vykazuje rovnakú falošnú históriu transakcií, falošný blockchain [2]. Naproti tomu sú tenké klienty ako mobilné a webové peňaženky, ktoré spravujú kľúče a transakcie vysielajú do peer-to-peer siete, akceptujú však väčšinovo schválenú históriu Blockchain databázy. Nevýhodou softvérových peňaženiek sú časté softvérové útoky.

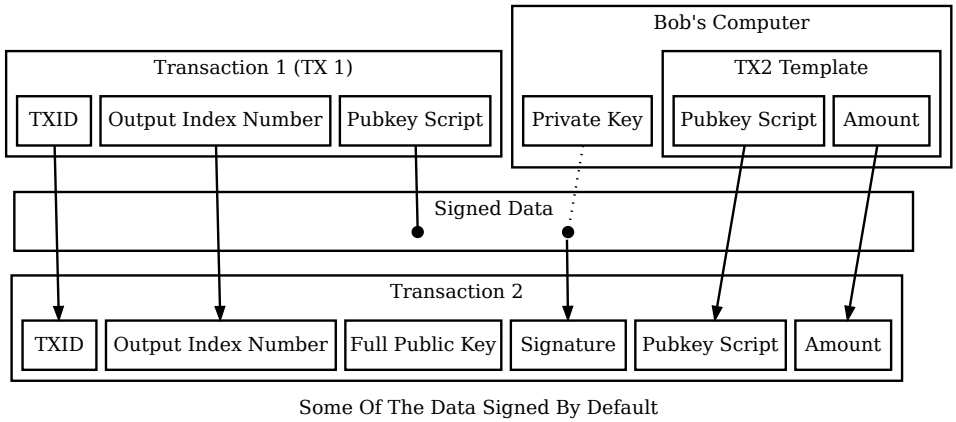
1.4 Funkcie infraštruktúry

Pre správny chod transakčného systému musí systém zabezpečiť jednoznačné preukázanie vlastníctva finančných prostriedkov, bitcoinov, a taktiež zamedziť problému dvojitého minutia prostriedkov, prípadu, kedy sa užívateľ snaží platiť prostriedkami, ktoré vlastnil, no už ich v minulosti vyčerpal.

1.4.1 Dôkaz vlastníctva prostriedkov

Dôkaz vlastníctva prebieha na princípe páru verejného a privátneho kľúča. Nech užívateľ Bob obdržal platbu od užívateľky Alice, ktorej pred platbou zverejnil svoju adresu. Alica vytvorila transakciu, obsahujúcu výstup, ktorý umožňuje komukoľvek so znalosťou Bobovho privátneho kľúča odpovedajúceho adrese, ktorú Bob zverejnil, tento výstup minúť, použiť v ďalšej transakcii.

V momente, keď sa Bob rozhodne prijatú platbu od Alice použiť, vytvorí novú transakciu so vstupom, odkazujúcim sa na transakciu Alice použitím *txid* a ďalej odkazujúcim sa na poradové číslo *output index* výstupu v poli všetkých výstupov Alicinej transakcie. Bob vytvorí podpis, *signature script*,



Obr. 1.7: Obrázok prevzatý z popisu kryptomeny [12] demonštruje preukázanie vlastníctva v momente vytvorenia novej transakcie. Užívateľ Bob použije položku *pubkey script* obdržaného výstupu transakcie a podpíše ho svojím súkromným kľúčom. Takto vzniknutý podpis je súčasťou novej transakcie, ktorá prijatý výstup míňa. Podpis dokazuje Bobovo vlastníctvo použitých zdrojov.

ktorý dokazuje vlastníctvo a naplnenie podmienok uložených v *pubkey script* položke výstupu Alicinej transakcie. Kombinácia poľa *Pubkey script* výstupu minulej transakcie a poľa *signature script* vstupu aktuálnej transakcie teda jednoznačne preukáže vlastníctvo Boba hodnoty výstupu transakcie Alice [3].

1.4.2 Kolektívne schvaľovanie transakcií

Najväčší prínos meny Bitcoin spočíva v automatickom kolektívnom procese overovania a schvaľovania nových transakcií bez nutnosti dôvery jednotlivým užívateľom siete za predpokladu, že sa väčšina, viac ako polovica, uzlov správa poctivo a nesnaží sa prijať identické falošné transakcie. Schvaľovanie transakcií funguje na princípe kolektívneho hlasovania, pričom schválený je majoritný názor a zvyšovania náročnosti prijatia nových blokov transakcií v závislosti na priemernom čase prijatia nového bloku. Uzol peer-to-peer siete obsahuje všetky transakcie od prvotnej transakcie uložené v reťazci blokov databázy Blockchain.

Každá nová transakcia je vyslaná broadcastom do peer-to-peer siete. Každý uzol v sieti má uložený doterajší schválený blockchain, teda reťazec blokov schválených transakcií a pracuje na schvaľovaní nových transakcií. Nové transakcie ukladá uzol do aktuálneho bloku, keď je veľkosť transakcií v aktuálnom bloku dostatočná, uzol sa snaží schváliť blok.

Blok má vo svojej hlavičke uloženú obtiažnosť schválenia v položke *bits*. Toto číslo určuje koľkými nulovými bitmi musí začínať hash aktuálneho bloku aby bol schválený sieťou. Spôsob, akým uzol môže ovplyvniť výsledný hash ce-

lého bloku je náhodnou zmenou položky *nonce* v hlavičke bloku, spolu s ktorou sa zmení tiež položka *time*, čas zmeny bloku. Uzol sa teda snaží nájsť náhodné číslo *nonce* vo vhodnom čase *time* tak, aby výsledný hash spĺňoval požadovanú vlastnosť počtu *bits* počiatočných nulových bitov [15]. Čas uhádnutia čísla a času spĺňujúcich podmienku rastie s požadovaným počtom nulových bitov začiatku hashu.

Obtiažnosť schválenia aktuálneho bloku je určená tak, aby priemerný čas medzi schválením dvoch blokov v sieti bol desať minút [3]. Táto obtiažnosť umožňuje regulovať rýchlosť schvaľovania nových blokov a zamedzuje útočníkom s veľkou výpočetnou kapacitou rýchlo generovať nové falošné bloky.

V prípade úspešne nájdeného čísla *nonce* uzol broadcastom pošle sieti aktuálny blok. Ostatné uzly v sieti tento blok prijmú, za predpokladu, že všetky transakcie v ňom sú validné a neobsahujú transakcie s už minutými zdrojmi, teda nemajú problém dvojitého minutia prostriedkov. Prijatie bloku uzlom sa prejaví začlenením bloku do Blockchain databázy daného uzla, pričom uzol začne zbierať transakcie do nového bloku, odkazujúc sa na prijatý blok ako na predchádzajúci blok.

Celý postup je následovný:

1. Nové transakcie sú poslané broadcastom všetkým uzlom.
2. Každý uzol zbiera transakcie do aktuálneho bloku.
3. Každý uzol sa snaží splniť požadovanú vlastnosť hashu bloku.
4. Uzol ktorý splní požadovanú vlastnosť hashu bloku pošle blok broadcastom všetkým uzlom.
5. Uzly prijmú blok, ak sú všetky transakcie validné a nemajú problém s dvojitým míňaním zdrojov.
6. Uzly prijmú validný blok jeho začlenením do lokálnej Blockchain databázy a rozpracovaním nového uzla, odkazujúc sa na prijatý uzol ako na prechádzajúci uzol.

1.4.3 Ťažba bitcoinov

Člen siete, vlastník uzla či skupiny uzlov môže svoju kolektívnu výpočetnú kapacitu využiť na overovanie a schvaľovanie nových skutočných transakcií, no rovnako sa môže pokúsiť úmyselne schváliť neexistujúce a falošné transakcie. Motiváciou ku čestnému správaniu sa uzlov a overovanie skutočných transakcií siete je odmena uzla za každý nový nájdený blok, spĺňujúci požadovanú vlastnosť hashu bloku. Uzol, ktorý nájde takýto blok, je odmenený istým objemom meny Bitcoin.

Táto odmena môže pochádzať z dvoch zdrojov. Prvým je nové množstvo Bitcoinov, uvoľnených do systému. Peer-to-peer sieť sa riadi pravidlami, ktoré

určujú celkový počet Bitcoinov v sieti, tento počet v čase rastie, má však svoju hornú hranicu a rýchlosť rastu časom klesá. Zatiaľ čo na začiatku fungovania siete Bitcoin bol tento spôsob prijímania zdrojov takmer výlučný, zmenšovaním počtu novo uvoľnených Bitcoinov do obehu sa zvyšuje pomer odmeny získanej z druhého zdroja, poplatkov za transakcie. Výška poplatku za transakciu sa odvíja od fyzickej veľkosti transakcie a od garantovaného času prijatia transakcie sieťou [3].

Fyzická veľkosť transakcie závisí od počtu použitých vstupov a výstupov transakcie, typicky najmenšie poplatky sú za transakcie, ktoré čerpajú zdroje len z jedného veľkého predchádzajúceho výstupu inej transakcie a obsahujú len jeden či dva výstupy, prvý výstup smerovaný na cieľovú adresu a druhý výstup smerovaný do svojej peňaženky ako vrátenie zvyšku veľkej sumy čerpanej vstupom. Naopak najvyššie poplatky sú za transakcie čerpajúce zdroje z mnohých malých minulých výstupov, alebo transakcie odoslané na mnoho iných adries.

Garantovaný čas schválenia transakcie sieťou sa meria v maximálnom počte schválených blokov, pričom posledný blok musí obsahovať danú transakciu a priemerná doba schválenia nového bloku je desať minút. V prípade nedostatočne veľkých poplatkov za transakciu sa môže stať, že transakcia nebude sieťou schválená, pretože všetky ostatné schvaľované transakcie majú vyššiu prioritu, v takom prípade sa odoslané prostriedky neminuli a odosielateľ ich má stále k dispozícii. Existujú dva spôsoby ťažby bitcoinov a to

- nezávislá ťažba;
- ťažba v rámci tzv. poolu

1.4.3.1 Nezávislá ťažba

V prípade nezávislej ťažby ide o scenár, kedy uzol v sieti funguje samostatne, ak vyťaží blok, teda nájde blok s požadovanou vlastnosťou jeho hashu, ponechá si všetku odmenu získanú nájdením bloku. Keďže obtiažnosť ťažby je nastavená tak, aby nájdenie nového bloku celej sieti všetkých uzlov trvalo v priemere desať minút, je šanca získania odmeny pomerne malá.

1.4.3.2 Ťažba v rámci poolu

Pool je množina uzlov, pracujúcich spoločne. V sieti samozrejme pracujú spoločne všetky uzly, no uzly v rámci poolu si v prípade, že jeden z uzlov vyťaží nový blok, odmenu rozdelia. Pool, teda podmnožina siete, si sám zvolí obtiažnosť ťažby bloku, ktorá je rádovo ľahšia než náročnosť celej siete Bitcoin. Bloky splňujúce ľahšiu náročnosť, no nesplňujúce náročnosť siete bitcoin slúžia na identifikáciu pomeru vykonanej práce uzlov v rámci poolu. V prípade, že jeden uzol z poolu úspešne vyťaží blok siete splňujúci náročnosť siete Bitcoin, sa

odmena uzlom v rámci poolu rozdelí pomerom nájdených blokov splňujúcich ľahšiu náročnosť hashu, nastavenú pravidlami poolu [3].

1.5 Bitcoin klient BitcoinCore

BitcoinCore je oficiálny klient pre Bitcoin. Je to software, ktorý vytvorí uzol siete Bitcoin s vlastnou kópiou databázy Blockchain, s vlastnou kontrolou všetkých prijatých transakcií, s vlastnou oddelenou peňaženkou a správou kľúčov. Tento uzol však neťaží nové bloky, no schvaľuje ich a schválené bloky ďalej vysiela do peer-to-peer siete. BitcoinCore je software s otvoreným zdrojovým kódom dostupný na populárnej stránke github [2]. Vďaka klientovi je možné udržovať správu peer-to-peer siete Bitcoin decentralizovanú a taktiež udržovať pravidlá siete a stanoviť spoločný konsenzus hlasovaním.

1.5.1 Úplná validácia transakcií

V prípade použitia klasických webových peňaženiek sa peňaženky spoliehajú na spoločný konsenzus siete, teda veria nadpolovičnej väčšine siete. V prípade použitia plnohodnotného klienta klient neprijme žiadnu transakciu odporujúcu ktorémukoľvek pravidlu, napríklad pravidlu o vstupe presahujúcom 21 miliónov, či iné dôležité, prípadne novo zavedené pravidlá. Každú transakciu od historicky prvej Bitcoin transakcie overí klient sám. Ďalšou výhodou je, že klient prispieva k decentralizácii systému a stáva sa jedným z overovateľov [14].

1.5.2 Súkromie

BitcoinCore klient ponúka vysokú mieru súkromia v niekoľkých rovinách. Prvá nesporná výhoda klienta je, že o užívateľovi nevyžaduje žiadne informácie, internetové peňaženky vždy vyžadujú emailovú adresu užívateľa, v niektorých prípadoch aj fotku dokumentu identifikujúcu osobu, napríklad fotku pasu či občianskeho preukazu.

Druhá výhoda klienta je, že užívateľ nemusí dôverovať tretej strane v otázke správy kľúčov, žiadny poskytovateľ služby nemôže zneužiť informácie o privátnych kľúčoch a použiť ich vo svoj prospech.

Poslednou výraznou výhodou klienta je možnosť anonymne vysielať transakcie do peer-to-peer siete, klient prekladá vlastné transakcie vysielaním iných transakcií ostatných užívateľov peer-to-peer siete, čím môže sťažiť hľadanie pôvodu vlastnej transakcie, naviac je kompatibilný s anonymnou sieťou Tor [9].

1.5.3 Užívateľské rozhranie

Klient ponúka dva druhy rozhrania, grafické rozhranie a rozhranie použiteľné z príkazového riadku [7]. Grafické rozhranie umožňuje vytvárať nové adresy a im príslušné QR kódy, zobrazovať informácie o stave peňaženky, teda aktuálny

stav a odoslané a prijaté transakcie, vytvárať nové transakcie a manuálne zvoliť vstupy transakcií či poplatok za transakciu, z ktorého sa odvíja rýchlosť schválenia transakcie peer-to-peer sieťou.

Z príkazového riadku je možné vytvoriť nové adresy, zistiť aktuálny stav peňaženky, vypísať neminuté prijaté transakcie, vytvoriť, podpísať a odoslať nové transakcie a vytvoriť notifikáciu, spustiť požadovaný skript, v prípade prijatia nových transakcií a blokov.

1.5.4 Réžia

Možnosť overenia všetkých transakcií vyžaduje ich fyzické uloženie na disku, preto jednou z najväčších požiadavok klienta na systém je dostatočne veľký dostupný úložný priestor, aktuálne 125 GiB. Druhá najväčšia požiadavka na systém pre správny chod klienta je potreba dostatočnej sieťovej kapacity systému, Podľa odporúčaní oficiálnej stránky klienta musí mať klient k dispozícii kapacitu odoslania 5 GiB dát denne a prijatia 500 MiB dát denne. Stránka tiež odporúča operačnú pamäť s kapacitou minimálne 1 GiB [10].

Kapitola 2

Analýza

Cieľom práce je vyhľadať vzťahy medzi entitami, nájsť cesty medzi uzlami grafovej databázy a filtrovať ich podľa časových značiek a váh. Nájdené vzťahy je ďalej potrebné zobraziť užívateľovi. Úlohu je možné rozdeliť na štyri problémy a to získanie dát, uloženie dát, vyhľadávanie a filtrovanie ciest podľa zadaných kritérií a nakoniec zobrazenia výsledkov užívateľovi. Spôsob uloženia dát priamo ovplyvňuje možné riešenie vyhľadávania a filtrovania dát. Použitie možných vizualizačných prostredí tiež závisí od výberu predchádzajúcich krokov.

2.1 Dáta kryptomeny Bitcoin

Prvým krokom k naplneniu cieľa je získanie požadovaných dát. Dáta bude nutné stiahnuť a ďalej udržovať. Prvotné dáta je možné stiahnuť použitím oficiálneho odkazu [4] alebo nainštalovaním BitcoinCore klienta a jeho spustením. Spustený BitcoinCore klient vždy zosynchronizuje svoju lokálnu kópiu blockchain s aktuálne najdlhšou schválenou kópiou v peer-to-peer sieti a tým pridá nové dáta do svojej lokálnej kópie.

2.2 Databáza

Výber databázy ovplyvní výber nástrojov pre vyhľadávanie, filtrovanie a vizualizáciu dát, rýchlosť týchto úkonov a požiadavky na potrebnú veľkosť volného miesta na disku. Použitie akejkoľvek databázy inej ako databázy Blockchain bude znamenať duplikáciu dát na disku, keďže dáta bude nutné uchovávať v Blockchain databáze z dôvodu integrovania nových dát a druhá databáza bude použitá na vyhľadávanie a filtrovanie dát. Prítomnosť druhej databázy však môže urýchliť vyhľadávanie a filtrovanie, rovnako môžu niektoré databázy ponúknuť jednoduchý a prehľadný dotazovací jazyk použiteľný za týmito účelmi. Keďže účelom práce je vizualizovať vzťahy medzi adresami a transak-

cie, potenciálne viac zreťazených transakcií, medzi nimi, v prípade vlastného dotazovacieho jazyka databázy záleží na vyjadrovacích schopnostiach a možnostiach jazyka. Niektoré databázy naviac ponúkajú vlastné vizualizačné prostredia, ktoré sú kompatibilné iba s danou databázou, prípadne daným typom databázy.

2.2.1 Blockchain

Blockchain je zaujímavým adeptom na použitú databázu riešenia, keďže získané dáta budú uložené v databáze Blockchain v každom prípade. Použitím blockchain ako databázy pre následnú vizualizáciu by nevznikla zbytočná potreba duplikácie dát, čo je veľkou výhodou, keďže veľkosť Blockchain databázy sa aktuálne pohybuje okolo 125 GiB a počet transakcií v sieti Bitcoin rastie exponenciálne, viď 1.1.

Naopak veľkou nevýhodou použitia databázy Blockchain na ďalšiu vizualizáciu je jej nízko úrovňový charakter a potreba implementácie všetkých ďalších krokov vyhľadávania a filtrovania dát. Databáza síce obsahuje index blokov a v prípade manuálneho zapnutia tiež index transakcií, indexy sa však odvolávajú na miesta na disku v ktorom sú dáta uložené binárne za sebou, dáta preto treba správne načítať a odfiltrovať podľa požiadavok.

2.2.2 Relačné databázy

Použitie relačnej databázy prináša zvýšené nároky na voľné miesto na disku, rovnako ako v prípade každej inej databázy okrem databázy Blockchain. Použitie relačnej databázy však umožňuje implementáciu vyhľadávania a filtrovania dát na úrovni SQL príkazov, čo by značne uľahčilo následnú implementáciu úkonov prostredníctvom SQL dotazov, prípadne spojením SQL a procedurálnych možností jazyka.

Základnou dátovou štruktúrou relačných databáz sú tabuľky. Tabuľka najčastejšie predstavuje jednu triedu či vzťah medzi triedami. Riadky tabuľky reprezentujú konkrétne objekty či vzťahy medzi objektami a stĺpce potom hodnotu atribúty triedy, hodnota stĺpca na istom riadku je hodnota atribútu istého objektu, inštancie triedy tabuľky.

Nevýhodou relačných databáz je potreba opakovaného spájania tabuliek v prípade vyhľadávania dlhšieho tranzitívneho vzťahu medzi objektami. V prípade vyhľadávania väčšieho počtu zreťazených transakcií je nutné zakaždým prechádzať index tabuľky vzťahov medzi transakciami.

2.2.2.1 Oracle relačná databáza

Jednou zo zaujímavých relačných databáz je databáza Oracle database 12c, keďže okrem normy SQL92 implementuje aj hierarchické dotazy, vhodné pre reťazenie transakcií či imperatívny jazyk PL/SQL umožňujúci definovanie funkcií a procedúr, využívajúcich okrem klasického SQL aj premenné či cykly [47].

2.2.2.2 PostgreSQL

PostgreSQL je open source alternatívou Oracle databázy, tiež ponúkajúcou hierarchické dotazy či svoj imperatívny PL/pgSQL jazyk, podobný PL/SQL jazyku databázy Oracle. Umožňuje taktiež spustenie procedúr napísaných v iných jazykoch ako napríklad C/C++, Java alebo Python [62].

2.2.2.3 Iné relačné databázy

Databázy Oracle a PostgreSQL ponúkajú okrem klasického SQL aj rozšírenú funkcionalitu. Úlohu by bolo možné riešiť viacerými relačnými databázami, no pre ich vzájomnú podobnosť som sa rozhodol ostatné relačné databázy ďalej neskúmať.

2.2.3 Grafové databázy

Povaha riešených úloh, hľadanie a filtrovanie ciest medzi adresami tvorenými jednou či viacerými transakciami má grafový charakter. Úlohy sa dajú zobecniť na úlohy nájdenia susedných uzlov, či vyhľadávania ciest medzi uzlami v grafe. Výhodou grafových databáz je ten, že sú konštruované za účelom rýchleho vyhľadávania v grafových štruktúrach a naviac obsahujú dotazovací jazyk, schopný jednoducho formulovať grafové problémy. Návrh týchto databáz je optimalizovaný pre grafové účely, čím získavajú výhodu v rýchlosti vyhľadávania.

Základnou dátovou štruktúrou grafových databáz sú uzly a hrany. Uzly reprezentujú objekty, atribúty objektu sú uložené ako atribúty uzlov. Hrany reprezentujú vzťahy medzi objektami a tiež môžu mať svoje atribúty. Výhoda takejto reprezentácie je rýchle vyhľadanie susedných uzlov k aktuálnemu uzlu, keďže návrh databáz umožňuje rýchle vyhľadanie odpovedajúcich hrán k uzlom a naopak. Vyhľadanie tranzitívnych vzťahov je teda jednoduchšie ako v prípade relačných databáz. Problém grafových databáz je naopak vytváranie vzťahov, k vytvoreniu vzťahu je nutné lokalizovať oba uzly uložené v databáze.

Pre porovnanie s relačnými databázami je teda nutné odviesť rovnakú prácu, spárovať objekty, ktoré majú medzi sebou vzťah, no zatiaľ čo relačné databázy riešia párovanie opakovane pri vyhodnocovaní dotazu, relačné databázy vykonávajú párovanie raz pri ukladaní dát.

2.2.3.1 Neo4j

Neo4j [44] je v súčastnosti jednou z najpopulárnejších grafových databáz. Medzi komerčné firmy, využívajúce databázu patrí napríklad Cisco Systems, Inc., Wal-Mart Stores, Inc., eBay Inc., Hewlett-Packard Company, LinkedIn Corporation či CGI Group Inc. [45]. Databáza je dostupná v dvoch edíciách, *Community* edícia, dostupná pod licenciou GPL v.3 a potom spoplatnená *Enterprise* edícia.

Štandardná *Community* edícia ponúka vytvorenie lokálnej grafovej databázy, *Enterprise* edícia naviac zahŕňa možnosti vytvorenia viacerých databázových clustrov, zdieľanie a pokročilý caching medzi clustrami, monitorovacie nástroje a nástroje pre vytváranie záloh databázy. Obe edície ponúkajú grafový dotazovací jazyk Cypher Query Language. Okrem tohto jazyku je možné vytvárať procedúry a funkcie napísané v jazyku Java, využívajúce natívne triedy databázy. Neo4j naviac v oboch edíciách ponúka vizualizačné rozhranie Browser [43].

2.2.3.2 OrientDB

Databáza OrientDB [50] je grafová a zároveň dokumentová databáza. Uzly a vrcholy sú uložené ako dokumenty. OrientDB takisto vychádza v komunitnej verzii s otvoreným zdrojovým kódom, tá je dostupná na github pod licenciou Apache 2 a v komerčnej verzii. Medzi zákazníkov patrí opäť Cisco Systems, Inc. ďalšími zákazníkmi sú napríklad Accenture PLC, KPMG a Dell Inc. [48].

Štandardná komunitná verzia databázy OrientDB na rozdiel od Neo4j poskytuje škálovateľnosť na úrovni clustrov už v základnej verzii. Podobne ako v prípade Neo4j, komerčná verzia oproti štandardnej verzii obsahuje naviac nástroje na monitorovanie výkonu a prevádzky databázy či vytváranie záloh.

Databáza umožňuje použitie rozšíreného SQL o vyhľadávanie vzorov, taktiež umožňuje použitie grafového dotazovacieho jazyka Gremlin [58] a SparQL. Podobne ako Neo4j obsahuje základný vizualizačný nástroj Graph Editor [49].

2.2.4 Databázy typu kľúč-hodnota

Databázy typu kľúč-hodnota sú NoSql databázy, obsahujúce páry kľúč a hodnota, kde kľúčom je identifikátor objektu a hodnota je samotný objekt. Tieto databázy sú svojou funkčnosťou pomerne obmedzené, fungujú podobne ako hash tabuľky, je z nich možné čítať a zapisovať do nich celé objekty, umožňujú však priamočiaru distribúciu dát a procesov zápisu a čítania.

Častým rozšírením databáz typu kľúč-hodnota sú stĺpcovo orientované databázy typu kľúč-hodnota, tieto databázy neukladajú informácie o objektoch riadkovo, jeden záznam netvorí celý objekt, ale stĺpcovo, teda každý atribút je uložený zvlášť a pre každý objekt, ktorý má daný atribút existuje záznam v tabuľke atribútu, indexovaný identifikátorom objektu. Tento spôsob uloženia umožňuje flexibilné modelovanie objektov, ktoré môžu mať ľubovolné atribúty a naviac uľahčuje agregácie na úrovni atribútov objektov. Toto rozšírenie zachováva jednoduchosť škálovania uloženia, IO operácii a naviac aj agregácií.

Stĺpcovo orientované databázy typu kľúč-hodnota síce neponúkajú dotazovací jazyk s veľkými vyjadrovacími schopnosťami, existuje však mnoho externých knižníc postavených nad týmto typom databáz, ktoré ponúkajú rozšírenú funkcionalitu.

2.2.4.1 HBase

HBase [60] je open source databáza vyvíjaná pod záštitou Apache Software Foundation ako súčasť Hadoop projektu a beží nad distribuovaným súborovým systémom HDFS. Je implementáciou databázy Bigtable popísanou spoločnosťou Google, Inc. Medzi jej výhody patrí škálovateľnosť, schopnosť ukladať veľké množstvo riedkych dát, vysoká miera kompresie a cashovania dát do operačnej pamäte. Najčastejšie sa používa v spojení s MapReduce procesmi a technológiou Hadoop.

2.2.4.2 Cassandra

Ďalšou škálovateľnou a distribuovanou databázou s otvoreným zdrojovým kódom spadajúcou pod Apache Software Foundation je Cassandra [55]. Opäť postavená nad projektom Hadoop, najväčším rozdielom oproti HBase databáze je orientácia na vysokú dostupnosť dát a nízku latenciu, na prípadný úkor konzistencie. V prípade distribuovaného behu databázy je možné zvoliť level konzistencie, ktorý určuje politiku zápisov a čítania dát na jednotlivých výpočetných uzloch [26].

Najvoľnejší prístup umožní každej operácii zápisu zapísať zmenu na jeden z uzlov a označiť transakciu za vykonanú, pričom zmena sa do ostatných uzlov, replík databázy, dostane až po istom čase čím vzniká nekonzistencia databázy, zápis je v tomto prípade rýchly a latencia nízka. Naopak najopatrnejší prístup označí transakciu za zapísanú až v momente, kedy sa zmena dopropaguje do každej repliky, čím sa značne spomalí rýchlosť zápisu a zvýši latencia.

Cassandra ponúka vlastný dotazovací jazyk Cassandra Query Language, spoločne s databázou HBase podporuje ďalšie knižnice umožňujúce ďalšie spracovanie uložených dát.

2.2.5 Dokumentové databázy

V prípade, že objekty uložené pod daným kľúčom sú štrukturované dokumenty, napríklad vo formáte JSON alebo XML, hovoríme o tzv. dokumentových databázach. Výhoda dokumentových databáz je, že v sebe ukladajú informácie o štruktúre každého dokumentu, umožňujú ukladať vnorené štruktúry a polia a zároveň ponúkajú vysokú mieru flexibility, keďže štruktúra dokumentov sa môže líšiť. Dokumentové databázy ponúkajú rozšírenú funkcionalitu, schopnosť vytvárať agregácie nad kľúčmi dokumentov.

2.2.5.1 MongoDB

Jednou z najznámejších dokumentových databáz je MongoDB [38]. MongoDB je škálovateľnou databázou s otvoreným zdrojovým kódom dostupnou pod GNU AGPL licenciou, umožňuje ad-hoc vyhodnocovanie dotazov, indexáciu

a agregácie dát v reálnom čase. Databáza obsahuje natívnu podporu pre spracovanie grafových dát pomocou agregácie *graphLookup*.

2.2.5.2 Elasticsearch

Elasticsearch [28] je vyhľadávací a analytický nástroj, predovšetkým využívaný k fulltextovému vyhľadávaniu. Opäť sa jedná o škálovateľné riešenie. Úložisko tvorí dokumentová databáza, nástroj naviac umožňuje realizáciu komplexných dotazov, agregácií a ich reťazení. Komunikácia s databázou prebieha pomocou RESTful služieb. Elasticsearch podporuje indexáciu vnorených objektov.

Medzi oficiálne rozšírenia nástroja Elasticsearch patrí rozšírenie *Graph*, to v kombinácii s nástrojom umožňuje distribuované vyhodnocovanie grafových dotazov, dostupnosť dát v reálnom čase a indexáciu na ľubovolnej škále. Rozšírenie je súčasťou balíčka *X-pack* [31], ktorý poskytuje pokročilé možnosti monitorovania a zabezpečenia nástroja Elasticsearch, balíček však nie je open source, je dostupný pod komerčnou licenciou s možnosťou 30 dňovej skúšobnej verzie [29].

2.2.6 Titan

Titan [25] je škálovateľná distribuovaná databáza optimalizovaná pre ukladanie a analýzu veľkých grafov distribuovaných vrámci clustru. Implementované úložisko je v operačnej pamäti, databáza však umožňuje pre backend storage použiť databázy HBase, Cassandra a Oracle BerkeleyDB, čím tvorí vhodnú vrstvu pre implementáciu grafových dotazov nad databázami typu kľúč-hodnota. Dotazy je možné formulovať v jazyku Gremlin [58].

2.2.7 Porovnanie databáz

Všetky spomínané databázy umožňujú uložiť požadované dáta. Výhodou použitia databázy Blockchain by boli menšie pamäťové nároky, keďže zdrojové dáta sú uložené v Blockchain databáze. Relačné databázy ponúkajú štandardný osvedčený spôsob uchovávania dát. Grafové databázy sú navrhnuté pre uchovávanie grafových dát, čomu je prispôsobená organizácia indexov. Databázy typu kľúč-hodnota sú ľahko distribuovateľné, no ich funkcionalita je obmedzená. Dokumentové databázy rozširujú funkcionalitu databáz typu kľúč-hodnota.

2.3 Vyhľadávanie a filtrovanie

Nad uloženými dátami budú prebiehať grafové vyhľadávania, filtrovanie výsledkov a prípadne ďalšia analýza. Spôsoby implementácie týchto krokov je možné rozdeliť podľa prístupu na tri skupiny:

i) implementácia nad jazykom či prostriedkami databáz;

ii) implementácia pomocou knižnice, schopnej operovať nad databázami;

iii) vlastná implementácia vyhľadávania a filtrovania nad dátami.

2.3.1 Dotazovací jazyk databázy

Pre potenciálnu implementáciu grafových dotazov musí jazyk umožniť prinajmenšom formulovanie vyhľadávania tranzitívnych relácií. Ak existuje spojenie medzi Alicou a Bobom a tiež spojenie medzi Bobom a Cyrilom, je potrebné vedieť vyhľadať tranzitívny vzťah medzi Alicou a Cyrilom, s Bobom ako ich prostredníkom.

Všetky nasledujúce ukážky pochádzajú z oficiálnych dokumentácií použitých jazykov. Každá ukážka rieši iné zadanie, ukážky demonštrujú syntax jazykov.

2.3.1.1 Oracle SQL

SQL jazyk databázy Oracle umožňuje formulovať tranzitívne dotazy pomocou klauzule CONNECT BY. Príklad prebratý z dokumentácie [46] zobrazuje pracovnú hierarchiu.

```
SELECT last_name, employee_id, manager_id, LEVEL
      FROM employees
      START WITH employee_id = 100
      CONNECT BY PRIOR employee_id = manager_id
      ORDER SIBLINGS BY last_name;

Result:

LAST_NAME                 EMPLOYEE_ID MANAGER_ID      LEVEL
------------------------- ----------- ---------- ----------
King                              100                     1
Cambrault                         148        100          2
```

2.3.1.2 PostgreSQL

Nasledujúca ukážka pochádza z dokumentácie rekurzívnych dotazov v PostgreSQL [61]. Príklad demonštruje vyhľadanie ciest v grafe s detekciou cyklov.

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle)
AS (
        SELECT g.id, g.link, g.data, 1,
          ARRAY[g.id],
          false
        FROM graph g
```

```
      UNION ALL
        SELECT g.id, g.link, g.data, sg.depth + 1,
          path || g.id,
          g.id = ANY(path)
        FROM graph g, search_graph sg
        WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;

Result:

ID      LINK    DATA        DEPTH   PATH        CYCLE
---     ----    -------     -----   ----        -----
1       2       "DATA1"     1       [1]         false
2       3       "DATA2"     2       [1,2]       false
3       0       "DATA3"     3       [1,2,3]     false
```

2.3.1.3 Neo4j Cypher

Cypher je dotazovacím jazykom grafovej databázy neo4j, príklad pochádzajúci z dokumentácie jazyka [42] ukazuje tranzitívne vyhľadanie vzťahu medzi dvoma entitami obmedzené počtom hrán na ceste. Forma dotazov pripomína regulárne výrazy. Výsledok volania je dlhý JSON dokument, ukážka demonštruje textovú reprezentáciu výstupu.

```
MATCH (you {name:"You"})
MATCH (expert)-[:WORKED_WITH]->(db:Database {name:"Neo4j"})
MATCH path = shortestPath( (you)-[:FRIEND*..5]-(expert) )
RETURN db,expert,path

Result:

-------------------------------------------------------------
|"db"            |"expert"         |"path"                  |
-------------------------------------------------------------
|{"name":"Ne|{"name":"Expert"}|[{"name":"You"},{},{"name":"Pe|
|o4j"}      |                 |ter"},{"name":"Peter"},{},{"na|
|           |                 |me":"Expert"}]                |
-------------------------------------------------------------
```

2.3.1.4 OrientDB SQL - Match

OrientDB taktiež ponúka dotazovací jazyk umožňujúci definovanie grafových vyhľadávaní, tento jazyk označuje za rozšírený SQL jazyk. Opäť príklad z oficiálnej dokumentácie jazyka [51], tento príklad demonštruje vyhľadanie priateľov užívateľa John do hĺbky zanorenia grafu šesť. Podobne ako v predchádzajúcom prípade je výstup zobrazený v textovej forme.

```
MATCH {class: Person, as: person, where: (name = 'John' AND
          surname = 'Doe')}.both('Friend'){as: friend,
          where: ($matched.person != $currentMatch) while:
                                                ($depth < 6)}
          RETURN person, friend

Result:

--------+--------
 person | friend
--------+--------
 #12:0  | #12:1
 #12:0  | #12:2
--------+--------
```

2.3.1.5 MongoDB Agregácie

Príklad z dokumentácie agregácie graphLookup databázy MongoDB [37] vyhľadávania letov s dvoma prestupmi.

```
db.travelers.aggregate( [
   {
      $graphLookup: {
         from: "airports",
         startWith: "$nearestAirport",
         connectFromField: "connects",
         connectToField: "airport",
         maxDepth: 2,
         depthField: "numConnections",
         as: "destinations"
      }
   }
] )

Result:

{
```

```
   "_id" : 1,
   "name" : "Dev",
   "nearestAirport" : "JFK",
   "destinations" : [
      { "_id" : 3,
        "airport" : "PWM",
        "connects" : [ "BOS", "LHR" ],
        "numConnections" : NumberLong(2) },
      { "_id" : 2,
        "airport" : "ORD",
        "connects" : [ "JFK" ],
        "numConnections" : NumberLong(1) },
      { "_id" : 1,
        "airport" : "BOS",
        "connects" : [ "JFK", "PWM" ],
        "numConnections" : NumberLong(1) },
      { "_id" : 0,
        "airport" : "JFK",
        "connects" : [ "BOS", "ORD" ],
        "numConnections" : NumberLong(0) }
   ]
}
```

2.3.1.6 Gremlin pre Titan a OrientDB

Nasledujúca ukážka z dokumentácie jazyka Gremlin [58] demonštruje vyhľadanie vedúceho na pozícii CEO v hierarchickej štruktúre reprezentovanej hranami grafu s názvom manages.

```
g.V().has("name","gremlin").
  repeat(in("manages")).
    until(has("title","ceo")).
  path().by("name")

Result:

[gremlin, josh, marko]
```

2.3.2 Grafové knižnice

Z dôvodu častej potreby špecifických analýz nad dátami vznikli rôzne riešenia a knižnice, ktoré si kladú za cieľ zjednodušiť implementáciu nových funkcií a procedúr. Takéto riešenia uľahčujú implementáciu zložitých analýz, samy obsahujú funkcie, ktoré je možné použiť v kombinácii s vlastnými definovanými

funkciami a tým docieliť požadovaný výsledný efekt. Riešenia poskytujúce grafovú funkcionalitu môžu tvoriť základ pre potrebné vyhľadávanie a filtrovanie.

2.3.2.1 Spark GraphX

Apache Spark [56] je univerzálny škálovateľný nástroj s otvoreným zdrojovým kódom pre spracovávanie dát. Podporuje vykonávanie výpočetných úloh ako lokálne tak na úrovni clustrov, podporuje jazyky Java, Scala, Python a R. Systém pracuje s dátami v operačnej pamäti, čím rastie rýchlosť vykonávania úloh. Podporuje dátové úložiská Cassandra a HBase a umožňuje nad nimi vykonávať analytické operácie. Súčasťou systému je GraphX, programové rozhranie pre grafové a grafovo paralelné výpočty.

Príklad implementácie vyhľadania najkratšej cesty medzi uzlami grafu použitím knižnice GraphX pochádza z dokumentácie programového rozhrania [57], výsledná cesta je uložená v premennej *sssp*.

```
import org.apache.spark.graphx.{Graph, VertexId}
import org.apache.spark.graphx.util.GraphGenerators

// A graph with edge attributes containing distances
val graph: Graph[Long, Double] =
  GraphGenerators.logNormalGraph(sc, numVertices = 100)
                 .mapEdges(e => e.attr.toDouble)
val sourceId: VertexId = 42 // The ultimate source
// Initialize the graph such that all vertices except the
// root have distance infinity.
val initialGraph = graph.mapVertices((id, _) =>
    if (id == sourceId) 0.0 else Double.PositiveInfinity)
val sssp = initialGraph.pregel(Double.PositiveInfinity)(
  (id, dist, newDist) => math.min(dist, newDist),
  triplet => {  // Send Message
    if (triplet.srcAttr + triplet.attr < triplet.dstAttr) {
      Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
    } else {
      Iterator.empty
    }
  },
  (a, b) => math.min(a, b) // Merge Message
)
println(sssp.vertices.collect.mkString("\n"))
```

2.3.2.2 Giraph

Ďalšou knižnicou s otvoreným zdrojovým kódom pre grafovú analytiku je Apache Giraph [54], používaný spoločnosťou Facebook pre analýzu sociál-

neho grafu užívateľov [24]. Použitie nástroja pre jednoduchý dotaz je zložitejšie, vyžaduje implementáciu dotazu v jazyku Java. Knižnica využíva Hadoop MapReduce implementácie grafových funkcií, proces tak dáta číta a zapisuje na disk a jedná sa teda o dávkové procesy. Ukážka nájdenia najkratšej cesty pochádza z dokumentácie knižnice [59].

```
@Override
public int run(String[] argArray) throws Exception {
    if (argArray.length != 4) {
        throw new IllegalArgumentException(
            "run: Must have 4 arguments <input path>" +
                  "<output path> <source vertex id> " +
                  " <# of workers>");
    }
    GiraphJob job = new GiraphJob(getConf(),
                        getClass().getName());
    job.setVertexClass(getClass());
    job.setVertexInputFormatClass(
        SimpleShortestPathsVertexInputFormat.class);
    job.setVertexOutputFormatClass(
        SimpleShortestPathsVertexOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(argArray[0]));
    FileOutputFormat.setOutputPath(job, new Path(argArray[1]));
    job.getConfiguration().setLong(
        SimpleShortestPathsVertex.SOURCE_ID,
                 Long.parseLong(argArray[2]));
    job.setWorkerConfiguration(Integer.parseInt(argArray[3]),
                               Integer.parseInt(argArray[3]),
                               100.0f);
    if (job.run(true) == true) {
        return 0;
    } else {
        return -1;
    }
}

public static void main(String[] args) throws Exception {
    System.exit(ToolRunner.run(new SimpleShortestPathsVertex(),
                                                        args));
}
```

2.3.3 Vlastná implementácia

Vyhľadávanie a filtrovanie v grafe je tiež možné realizovať vlastnou implementáciou v ľubovolnom jazyku, pre ktorý existuje konektor pre zvolenú databázu. V takom prípade by sa jednalo o kombináciu možností databázy a doimplementovania potrebnej funkcionality. Nevýhoda takéhoto riešenia je náväzná potreba vlastnej implementácie vizualizácie dát, podobne ako v prípade použitia niektorej z grafových knižníc.

2.3.4 Porovnanie možností vyhľadávania a filtrovania

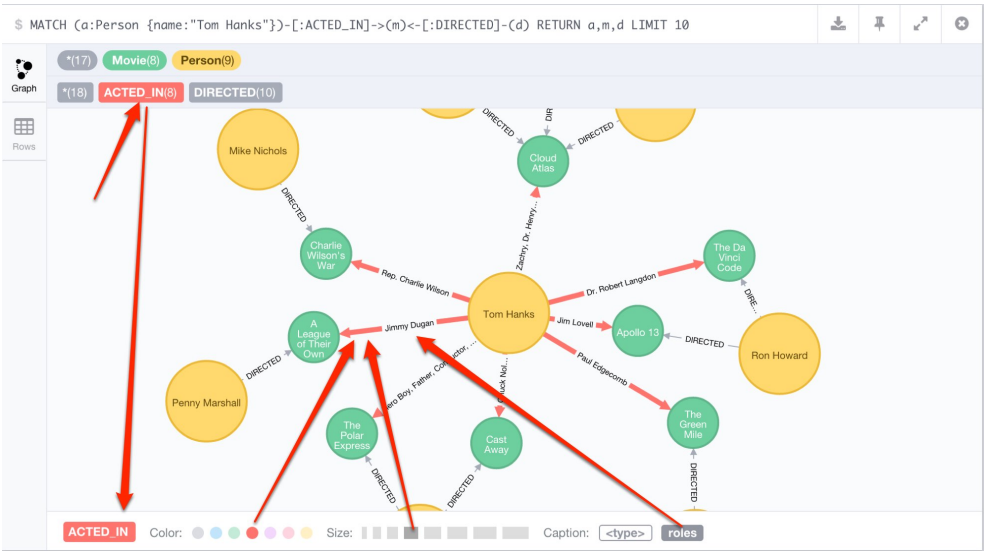
Vyhľadávanie a filtrovanie implementované pomocou jazyka databázy znižuje počet použitých nástrojov a umožňuje využiť návrh databázy pre efektívnejšie vyhodnocovanie dotazov. Z ukážok dokumentácií jazykov ma najviac zaujali jazyky grafových databáz, SQL jazyk databázy Oracle rozšírený o klauzulu CONNECT BY a grafová agregácia databázy MongoDB. Tieto jazyky umožnili kompaktne formulovať dotazy na tranzitívne vzťahy medzi entitami. Popísané grafové knižnice tvoria analytickú vrstvu nad databázami. Formulácie problémov jazykom knižníc pôsobia dlhšie a menej prehľadne. Vlastná implementácia vyhľadávania je z porovnávaných prístupov časovo najnáročnejšia.

2.4 Vizualizácia

Práca si kladie za cieľ vyhľadávať vzťahy medzi uzlami grafovej databázy. Jednoduchosť a rýchlosť interpretácie výsledkov vyhľadávania závisí od ich vizualizácie. Na zobrazenie vzťahov medzi entitami sa často používajú vizualizácie v podobe grafu, kde uzly grafu predstavujú entity a hrany predstavujú vzťahy medzi entitami.

Vizualizáciu je možné v prípade použitia niektorých databáz docieliť v štandardných nástrojoch, ktoré databázy ponúkajú. Výhoda takýchto riešení je možnosť priameho interaktívneho použitia nástrojov, teda užívateľ môže zadať a zobraziť výsledok vyhľadávania v už existujúcom nástroji, bez nutnosti akejkoľvek modifikácie. Štandardné vizualizačné databázové nástroje však nie je možné ďalej upravovať.

Rovnako existujú univerzálne nástroje, umožňujúce vizualizácie grafov, či vizualizačné knižnice. V prípade použitia univerzálnych nástrojov či knižníc vzniká potreba implementácie prepojenia vyhľadávania, zadávania dotazov a importu dát do vizualizačného prostredia, v prípade použitia knižníc tiež potreba implementácie vizualizácie.



Obr. 2.1: Ukážka grafického prostredia Browser databázy Neo4j prebratá z oficiálnej online príručky [43]. Na vrchu sa nachádza príkazový riadok, do ktorého je možné zadávať príkazy jazyka Cypher databázy. Pod ním sa nachádza vizualizácia výsledku v podobe grafu. Farba a veľkosť uzlov a hrán grafu závisí od ich typov, pre každý typ je možné definovať vlastný vizuálny štýl.

2.4.1 Vizualizačné nástroje viazané na databázy

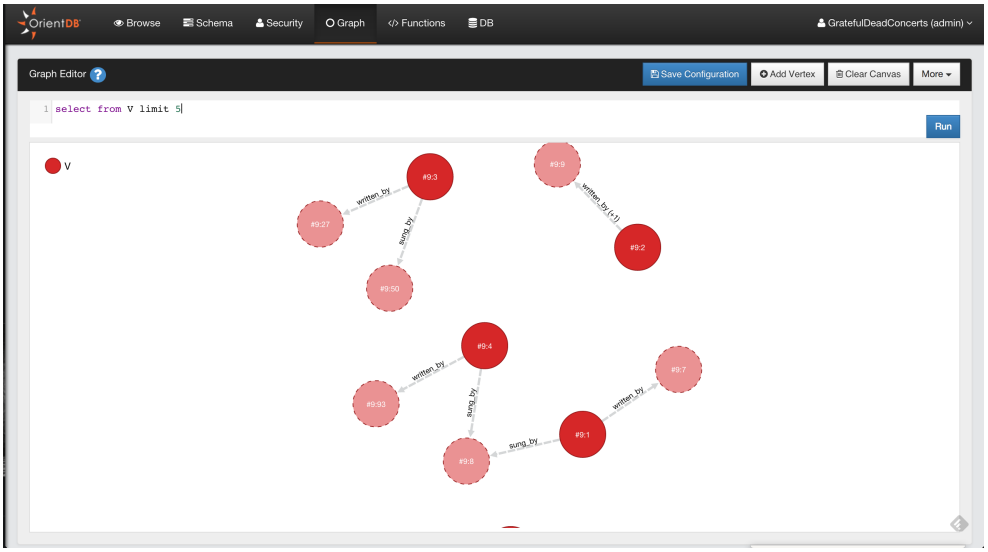
Najväčšou výhodou nástrojov viazaných na databázy je možnosť implementácie vyhľadávania a filtrovania v jazyku databázy a priama interaktívna vizualizácia výsledkov bez nutnosti akejkoľvek vlastnej implementácie vizualizácie.

2.4.1.1 Neo4j Browser

Neo4j Browser [43] je vizualizačný nástroj databázy Neo4j, ktorý beží vo webovom prehliadači, umožňuje zadávanie dotazov v jazyku Cypher, výsledky dotazov vizualizuje ako graf, tabuľku, či JSON dokument, v závislosti od nastavení. Nástroj naviac zobrazuje základné informácie o databáze a umožňuje jej správu. Toto prostredie obsahuje sadu tutoriálov, umožňuje ukladanie obľúbených dotazov a ponúka možnosť zmeny štýlu vizualizácie v závislosti od typu uzlu a hrany. Každý uzol je možné rozkliknúť, čím sa zobrazia všetky ďalšie uzly, ktoré majú s daným uzlom spoločnú hranu. Kliknutím na uzol či hranu sa zobrazí detail entity a jej atribúty.

2.4.1.2 OrientDB Graph Editor

OrientDB je grafovou databázou priamo konkurujúcou Neo4j, preto sa snaží ponúknuť všetky nástroje s funkčnosťou konkurenčných nástrojov. Vizuali-



Obr. 2.2: Ukážka grafického prostredia Graph Editor databázy OrientDB prebratá z dokumentácie nástroja [49]. Na vrchu sa opäť nachádza príkazový riadok a pod ním vizualizácia výsledku v podobe grafu.

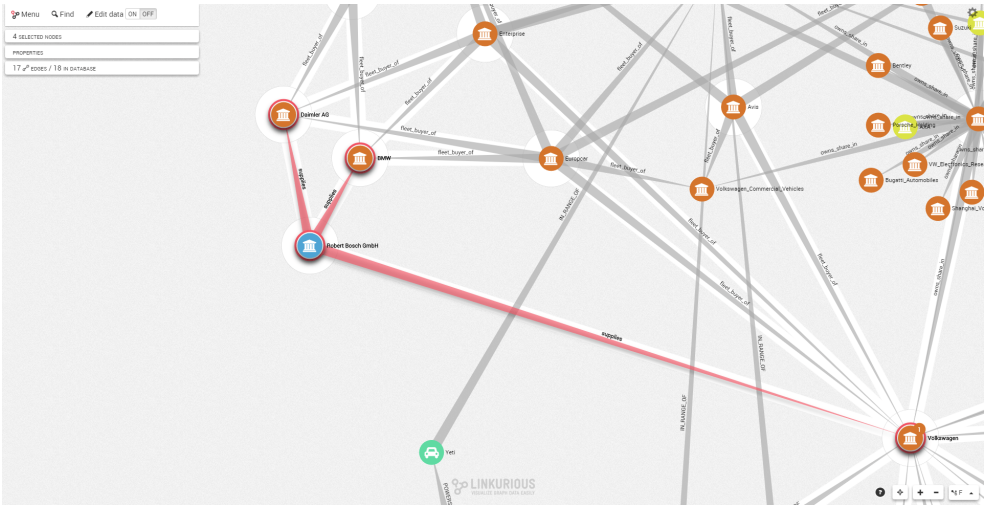
začné prostredie Graph Editor [49] je podobné konkurenčnému nástroju Browser, umožňuje zadávať dotazy v jazyku databázy, výsledky vizualizuje v podobe grafu, tabuľky či JSON dokumentu, umožňuje zobrazenie detailu uzlov a hrán, susedných uzlov aktuálneho uzlu rozkliknutím a vlastnú štylizáciu uzlov a hrán.

2.4.1.3 Linkurious

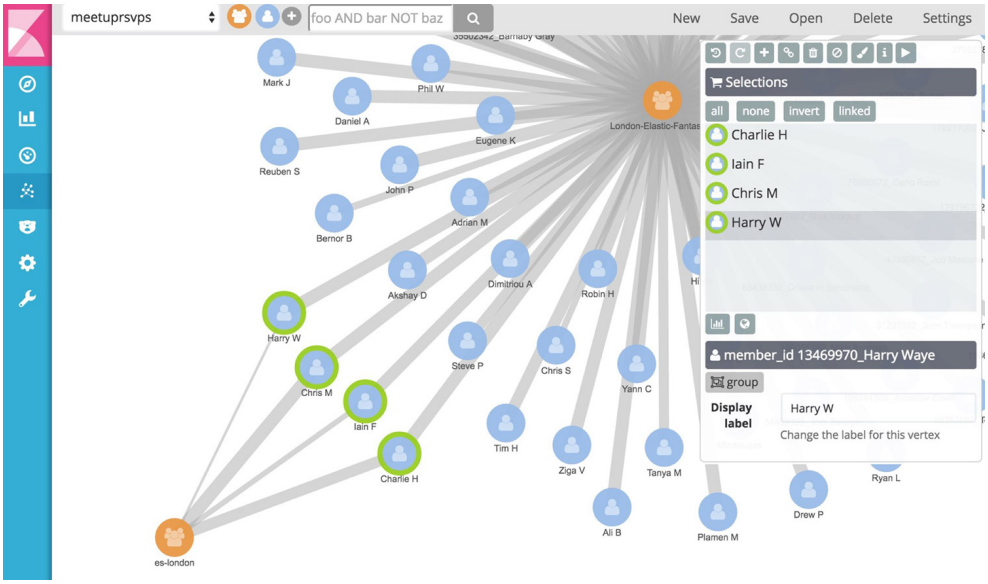
Linkurious [36] je komerčný vizualizačný nástroj, ktorý podporuje viacero grafových databáz, konkrétne Neo4j, Titan, DataStax Enterprise Graph a AllegroGraph. Nástroj ponúka pestré vizualizácie s možnosťami zadávania dotazov v jazyku použitej grafovej databázy, oproti predchádzajúcim nástrojom ponúka možnosti vytvárania upozornení, použitie geografického rozloženia, full text vyhľadávanie s možnosťami automatického doplňovania a fuzzy vyhľadávania uzlov, väčšie zabezpečenie a podporu využívania viacerých grafových databáz naraz.

2.4.1.4 Kibana Graph

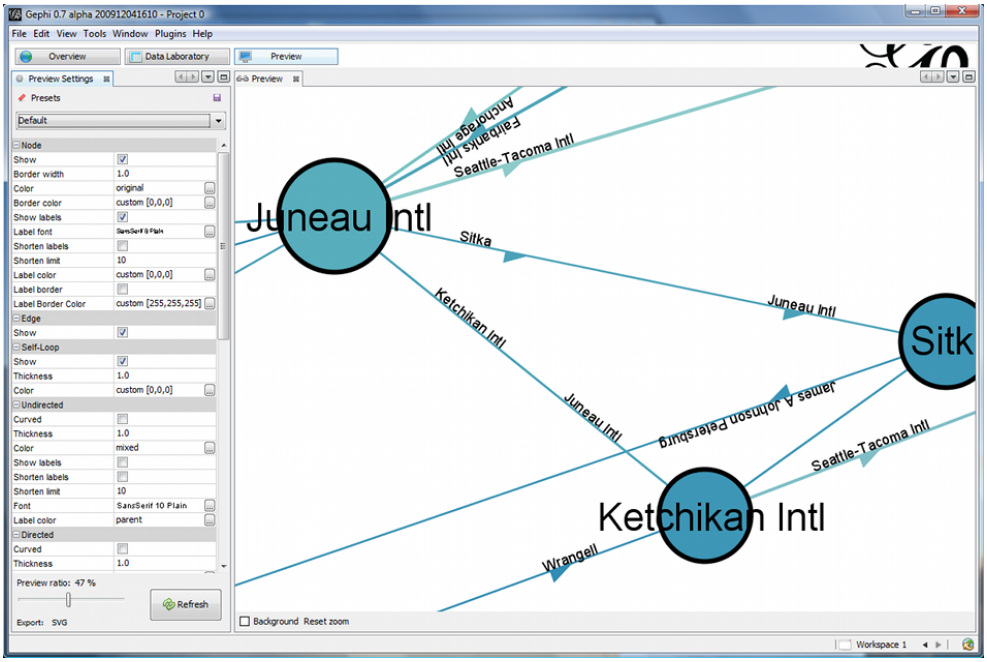
Komerčný zásuvný modul pre elasticsearch X-Pack obsahuje rozšírenie Graph [29], ktoré umožňuje vytvárať grafové dotazy a následne ich vizualizovať vo vizualizačnom nástroji Kibana [30]. Použitím rozšírenia v nástroji Kibana je možné definovať grafy a vyhľadávať v nich použitím agregácií. Nástroj tiež podporuje pridanie susedných uzlov aktuálne zobrazených uzlov.



Obr. 2.3: Ukážka vizualizácie docielenej nástrojom Linkurious. obrázok je prevzatý z blogu produktu [34]. V ľavom hornom rohu sa nachádzajú nastavenia vyhľadávania a zvyšok tvorí výsledná vizualizácia.



Obr. 2.4: Obrázok demonštruje grafovú vizualizáciu dosiahnutú pomocou nástroja Kibana databázy Elasticsearch s použitím rozšírenia Graph. Ukážka je prevzatá zo stránok rozšírenia Graph[29].



Obr. 2.5: Ukážka vizualizačného nástroja Gephi prebratá z popisu nástroja [32]. Vľavo sa nachádza lišta s nastaveniami vizualizácie grafu, v pravo samotná vizualizácia.

2.4.2 Nezávislé vizualizáčné nástroje

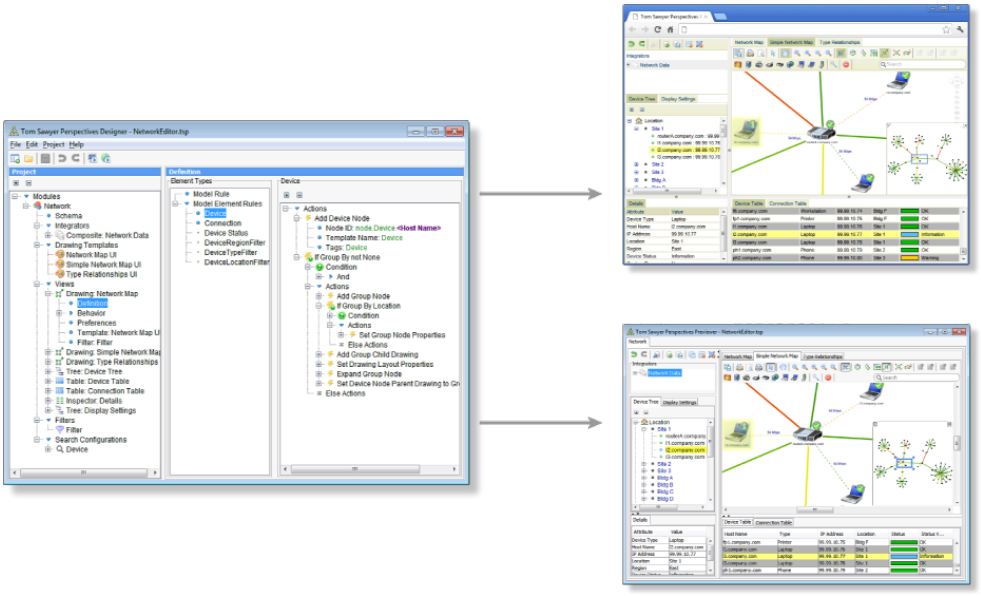
Nástroje, ktoré umožňujú vizualizáciu grafov a nie sú úzko viazané na konkrétnu databázu, vyžadujú v porovnaní s riešeniami viazanými na databázy importovanie zobrazovaných grafov. Pre každý dotaz vyhľadávania a filtrovania v grafe je nutné nahrať výsledok dotazu do vizualizačného nástroja, tým vzniká potreba implementácie prostredia pre interaktívne zadávanie dotazov.

2.4.2.1 Gephi

Gephi [32] je desktopová aplikácia s otvoreným zdrojovým kódom, ktorá slúži na vizualizáciu grafov. Dáta je možné importovať zo zošitov a niektorých databáz, napríklad Neo4j, pomocou prídavných modulov. Umožňuje vyhľadávanie a filtrovanie v grafoch a poskytuje viacero možných rozložení grafov, ktoré prispievajú k prehľadnosti vizualizácie. Tiež poskytuje rôzne štýly vizualizácie a exportovanie obrázkov z grafov.

2.4.2.2 Tom Saweyer Perspectives

Tom Saweyer Perspectives [63] je komerčný vývojový kit pre vývoj grafových vizualizácií a analýz. Vývojový kit je dostupný vo verzii pre jazyk Java a



Obr. 2.6: Ukážka nástroja Tom Saweyer Perspectives prebratá z domovskej stránky produktu [63]. Nástroj umožňuje vytvárať vizualizácie v podobe webových a desktopových aplikácii. Obrázok znázorňuje vytvorenie webovej aplikácie v pravo hore a desktopovej aplikácie vpravo dole, použitím nástroja Tom Saweyer Perspectives vľavo.

.NET. Umožňuje vývoj desktopových a webových aplikácií. Riešenie umožňuje integrovať dáta uložené v rôznych databázach ako Neo4j, OrientDB, AllegroGraph, či dáta uložené v spreadsheetoch, JSON dokumentoch alebo napríklad dostupné z REST služieb.

Riešenie poskytuje celú škálu vizualizácií, okrem rôznych typov grafov aj mapy, tabuľky, a stromové zoznamy. Pre grafy poskytuje mnoho rozložení a nad grafmi viacero funkcií pre vyhľadávanie a filtrovanie.

2.4.2.3 linkurious.js

Knižnica linkurious.js je javascriptovou vizualizačnou knižnicou s otvoreným zdrojovým kódom, pôvodne využívanou pre software Linkurious, ten momentálne používa proprietárnu knižnicu a túto knižnicu označuje za zastaralú. Knižnica je navrhnutá pre interaktívne vykresľovanie grafov, existuje k nej mnoho rozšírení a tvorí vhodný základ pre vlastnú implementáciu vizualizačnej vrstvy [35].

2.4.3 Porovnanie možností vizualizácie

Výhoda popísaných vizualizačných nástrojov databáz je možnosť vytvárania ad-hoc dotazov a zobrazenie výsledkov v podobe grafov bez nutnosti implementácie vizualizačnej vrstvy. Všetky popísané vizualizačné nástroje databáz znázorňujú grafy prehľadnou formou a umožňujú vytvoriť požadované vizualizácie. Nezávislé vizualizačné nástroje neumožňujú zadávanie dotazov, zobrazujú importované grafy. Pri použití nezávislých nástrojov vzniká potreba implementácie užívateľského rozhrania, do ktorého by užívateľ zadal dotaz. Výsledok dotazu by následne bolo nutné importovať do nástroja a použiť vizualizáciu nástroja.

2.5 Zvolené nástroje

Pre naplnenie zadania práce je nutné stiahnuť a synchronizovať dáta, uložiť ich, vyhľadávať a filtrovať v nich a následne zobraziť výsledky odpovedajúce kritériám. Stiahnutie a synchronizácia dát budú zabezpečené klientom Bitcoin-Core. Technológie zvolené pre uloženie, vyhľadávanie, filtrovanie a vizualizáciu boli vybraté tak, aby minimalizovali počet použitých nástrojov.

Medzi nástroje ktoré umožňujú implementáciu všetkých krokov patria grafové databázy Neo4j, OrientDB a dokumentová databáza Elasticsearch s rozšírením Graph. Rozšírenie Graph nástroja Elasticsearch je však dostupné len pod skúšobnou 30-dňovou verziou a platenou komerčnou verziou.

Z grafových databáz som nakoniec zvolil databázu Neo4j pre jazyku Cypher a väčšie osobné sympatie k vizualizačnému nástroju Browser v porovnaní s nástrojom Graph Editor databázy OrientDB. Dáta budú uložené v databáze Neo4j, vyhľadávanie a filtrovanie bude implementované pomocou dotazov jazyka databázy Cypher a vizualizácia bude docielená využitím nástroja Browser.

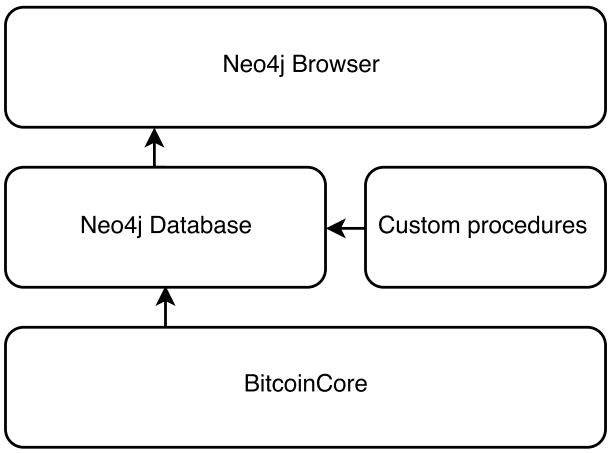
Kapitola 3

Návrh

Na základe zvolených technológií databázy Neo4j, dotazovacieho jazyka Cypher a vizualizačného nástroja Browser bude navrhnuté riešenie aplikácie pre vyhľadávanie a filtrovanie dát grafovej databázy užívateľom a následnú vizualizáciu výsledkov v podobe grafu.

3.1 Synchronizácia dát kryptomeny

BitcoinCore slúži v návrhu na jednoduchú synchronizáciu dát kryptomeny Bitcoin na lokálnom PC. Spustený klient ukladá nové bloky transakcií vysielané



Obr. 3.1: Architektúra riešenia. Dáta kryptomeny bude synchronizovať klient BitcoinCore. Synchronizované dáta budú nahraté do databázy Neo4j, rozšírenej o vlastné procedúry. Vizualizácia analýzy dát bude docielená nástrojom Browser.

peer-to-peer sieťou a ukladá ich na disk. V prípade, že klient nebol nikdy či dlhšiu dobu spustený, zosynchronizuje historické dáta a následne čaká na nové bloky. Synchronizácia dát je teda dosiahnutá zapnutím klienta, bez ďalších potrebných krokov.

3.2 Databáza

Druhú vrstvu návrhu tvorí grafová databáza, ktorá poskytuje možnosti efektívneho uloženia grafových dát a dotazovania nad nimi. Nevýhodou takéhoto návrh je potreba duplikácie dát, uložených raz v Blockchain databáze klienta BitcoinCore a druhý krát v databáze Neo4j pre ďalšiu analýzu.

3.2.1 Dátová schéma

Základné stavebné prvky databázy Neo4j sú uzly a hrany. Uzly aj hrany majú svoje typy, ktoré umožňujú identifikovať skupinu uzlov či hrán istého typu. Uzly môžu mať viacero typov. Rovnako majú uzly a hrany svoje atribúty, ktoré môžu byť číselné alebo reťazce. Pokročilé dátové typy ako napríklad dátumy aktuálne podporované nie sú, no v rámci dotazov a rozšírení databázy je možné atribúty skonvertovať pomocou funkcií.

Navrhnutá schéma 3.2 obsahuje päť typov uzlov a päť typov hrán. Uzly vychádzajú z entít Blockchain databázy kryptomeny Bitcoin popísaných v 1.2, pričom entity sú zjednodušené, obsahujú informácie potrebné na ďalšiu analýzu a niektoré pôvodné entity sú zlúčené do jedného typu uzla.

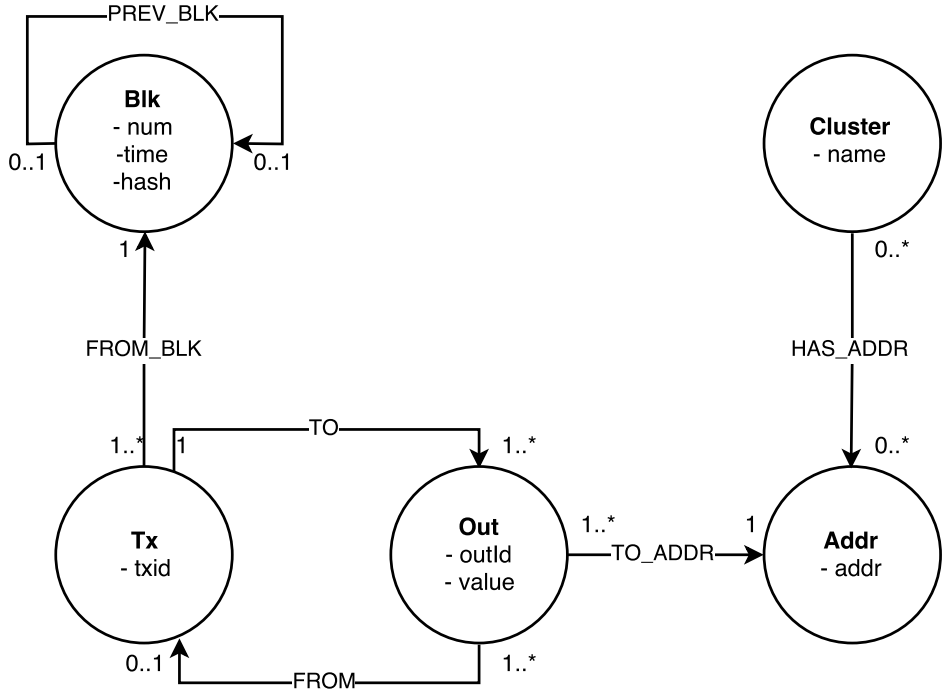
V návrhu sú zlúčené bloky so svojimi odpovedajúcimi hlavičkami, rovnako sú zlúčené vstupy a výstupy transakcií, keďže každý výstup transakcie je možné použiť ako vstup do novej transakcie iba raz a v plnej výške. Uzly reprezentujú následovné entity:

- **Blk** — bloky transakcií;
- **Tx** — transakcie;
- **Out** — vstupy a výstupy transakcií;
- **Addr** — adresy;
- **Cluster** — zhluky adries.

3.2.1.1 Blok

Blok je množina transakcií, ktoré boli peer-to-peer sieťou schválené a prijaté naraz v danom čase. Atribúty bloku sú:

- **time** — čas prijatia a schválenia bloku, uložený ako unixová časová značka, typ long;



Obr. 3.2: Navrhnutá dátová schéma obsahuje uzly blokov *Blk*, transakcií *Tx*, výstupov transakcií *Out*, adries *Addr* a zhlukov adries *Cluster*. Hrany grafu reprezentujú vzťahy medzi uzlami.

- **num** — hĺbka bloku v Blockchain databáze, typ long;
- **hash** — identifikátor bloku, jeho hash, typ reťazec.

Uzol typu blok má typ hrany *FROM_BLK* s uzlom transakcie a typ hrany *PREV_BLK* s iným uzlom rovnakého typu blok. Hrana typu *FROM_BLK* je orientovaná v smere od transakcie k bloku, blok môže obsahovať jednu a viac transakcií, zatiaľ čo každá transakcia môže byť obsiahnutá iba v jednom bloku.

Hrana typu *PREV_BLK* definuje poradie blokov. Každý blok, s výnimkou úplne prvého bloku, sa hranou odkazuje na blok schválený pred ním. Jediný blok, na ktorý sa žiaden iný blok neodkazuje, je posledný blok. Takto vzniká lineárna história schválených blokov.

3.2.1.2 Transakcia

Transakcia je typ uzla, ktorý spája vstupy transakcií a výstupy transakcií a tým vytvára cesty v grafe. Má jediný atribút typu reťazec, je ním hash transakcie a zároveň jej identifikátor *txid*.

Okrem hrany typu *FROM_BLK* musí mať každá transakcia aspoň jednu hranu typu *FROM* a jednu hranu typu *TO*. Hrany typu *FROM* spájajú transakciu so všetkými jej vstupmi. Tento typ hrany je orientovaný od výstupu, uzlu typu *Out*, minulej transakcie do aktuálnej transakcie. Každý výstup minulej transakcie môže byť obsiahnutý najviac v jednej hrane typu *FROM*, keďže výstup z definície pravidiel kryptomeny Bitcoin buď minutý nie je, alebo je minutý v plnej výške.

Hrana typu *TO* je orientovaná v smere od transakcie do nového výstupu. Transakcia môže mať viac výstupov, každý výstup však pochádza práve z jednej transakcie.

3.2.1.3 Transakčný vstup a výstup

Keďže každý výstup transakcie môže byť použitý ako vstup novej transakcie práve raz, sú tieto dve entity databázy Blockchain v návrhu zlúčené do jedného uzla. Vstup transakcie je teda každý výstup inej, predchádzajúcej transakcie, z ktorého vedie hrana typu *FROM* do inej, novej transakcie. Typy hrán medzi vstupmi/výstupmi transakcií a transakciami sú popísané v predchádzajúcej stati o transakciách.

Atribúty výstupov transakcií sú:

- **outId** — identifikátor výstupu, skladajúci sa z identifikátoru transakcie, podtržítka a poradového čísla výstupu vrámci pôvodnej transakcie, uložený ako string;
- **value** — hodnota výstupu transakcie v satoshi, v prípade, že výstup je zároveň vstupom, celá hodnota je minutá, teda hodnota atribútu je aj hodnotou vstupu transakcie, hodnota je uložená ako long, keďže satoshi je aktuálne najmenšia nedeliteľná jednotka meny Bitcoin.

Novým typom hrany pre výstup transakcie je typ *TO_ADDR*, ktorý reprezentuje odoslanie výstupu na konkrétnu adresu a je orientovaný od výstupu k adrese. Každý výstup je odoslaný na práve jednu adresu, ak je takýto výstup použitý ako vstup novej transakcie, musel byť odoslaný z adresy prijímateľa. Adresa môže byť cieľom jedného či viacerých výstupov, podľa odporúčaní autorov kryptomeny Bitcoin by mala byť každá adresa použitá na príjem práve jedného výstupu, ide však len o doporučenie z dôvodu anonymity užívateľov. Adresa by teoreticky nemusela prijať žiaden výstup, v takom prípade sa však o existencii adresy nevie, takže v databáze uložená nebude.

3.2.1.4 Adresa

Uzol typu adresa slúži k identifikácii cieľových adries výstupov transakcií. Adresa nereprezentuje jedného užívateľa, ten môže mať veľa adries. Uzol typu adresa má jediný atribút *addr* typu reťazec, v ktorom je uložená cieľová adresa

výstupu transakcie. Hrana typu *TO_ADDR* medzi výstupom transakcie a adresou je popísaná v predchádzajúcom odstavci.

Hrana typu *HAS_ADDR* je orientovaná v smere od uzla zhluku do adresy, adresa môže patriť ľubovolnému počtu zhlukov, a rovnako zhluky môžu mať ľubovolný počet adries.

3.2.1.5 Zhluk Adries

Zhluk adries je typ uzla ktorý sa v Blockchain databáze nenachádza. Zhluk je umelo navrhnutý pre zlučovanie adries do zhlukov, čím vznikne možnosť vytvárania dotazov, operujúcich nad preddefinovanými množinami adries bez potreby opakovanej manuálnej enumerácie zoznamu adries.

Uzol zhluku adries má jediný atribút *name* typu reťazec, je to meno zhluku, ktoré umožní jeho identifikáciu v dotazoch, operujúcich nad zhlukmi adries. Hrana medzi uzlom zhluku a uzlom adresy je opäť popísaná vyššie.

3.2.1.6 Problém časovej značky

V zadaní práce sa píše o časových značkách hrán. Pri použití databázy Blockchain sa jedná o časové značky prevedenia jednotlivých transakcií. Jediný časový údaj dát kryptomeny Bitcoin je údaj o schválení a prijatí bloku peer-to-peer sieťou. Problém časovej značky som sa v návrhu rozhodol riešiť uložením značky do uzla reprezentujúceho blok, alternatívou by bolo použiť túto časovú značku ako atribút uzlu transakcie, alebo hrany *FROM_BLK* smerujúcej od transakcie k bloku. V takom prípade by sa však informácia zbytočne duplikovala, keďže čas individuálnych transakcií nie je známy a všetky transakcie jedného bloku by mali časovú značku rovnakú.

3.2.1.7 Problém váh hrán

Váhy hrán spomenuté v zadaní súvisia s objemom vstupov a výstupov transakcií. Objem vstupov transakcií by bolo možné uložiť ako atribút hrany *FROM* od vstupu transakcie do transakcie a objem výstupov ako atribút hrany *TO* z transakcie do výstupu. Objem výstupu je rovnaký v prípade jeho použitia ako vstupu a tak stačí informáciu uložiť iba raz, buď ako atribút uzlu výstupu, alebo atribút hrany *TO* z transakcie do výstupu. Túto informáciu som sa v návrhu rozhodol uložiť vrámci uzlu výstupu v atribúte *value*.

3.2.2 Naplnenie databázy

Informácie potrebné pre požadovanú analýzu transakčných dát kryptomeny Bitcoin budú pochádzať z dvoch zdrojov, prvým budú dáta kryptomeny uložené v lokálnej Blockchain databáze. Tie bude nutné uložiť do databázy Neo4j a budú tvoriť väčšinu dát databázy. Spôsob ich nahrania do databázy sa bude líšiť, v prípade prvotného nahrania totiž databáza umožňuje použiť nástroj,

vytvárajúci offline databázu, schopný rýchlejšie nahrať dáta. Naopak v prípade, že databáza už bude existovať a úlohou bude nahrať nové dáta, ktorých relatívny objem ku objemu existujúcej databázy bude malý, bude postup nahrávania dát iný. Druhým zdrojom dát databázy budú umelo vytvorené informácie o zhlukoch adries, zadané užívateľom, pre možnosť analýz vzťahov medzi množinami adries.

3.2.2.1 Naplnenie prázdnej databázy

Oficiálna stránka databázy, popisujúca efektívne vkladanie veľkých objemov nových dát do databázy Neo4j uvádza nástroj neo4j-import [41] ako najrýchlejšiu alternatívu. Nástroj vytvorí novú databázu, ktorá bude v čase plnenia offline a rovno vytvára databázové súbory. Nástroj nie je možné použiť na vkladanie dát do už existujúcej databázy a ako vstup vie použiť iba dáta z CSV súborov. Použitie tohoto nástroja bude znamenať trojnásobnú duplikáciu dát na disku počas vytvárania novej databázy, dáta budú uložené v lokálnej Blockchain databáze, v CSV súboroch a v novej databáze. Po úspešnom vytvorení dát bude možné CSV súbory zmazať, disková kapacita potrebná na vytvorenie databázy však bude pomerne vysoká.

3.2.2.2 Doplnenie existujúcej databázy

V prípade, že databáza bude existovať a obsahovať väčšinu dát lokálnej Blockchain databázy, cieľom bude nahrať nové dáta Blockchain databázy do databázy Neo4j. Pre tento účel je možné použiť Cypher jazyk databázy Neo4j a jeho príkaz LOADCSV, ktorý umožní, podobne ako v prípade použitia nástroja neo4j-import, nahrať dáta z CSV súborov do databázy. Vkladanie dát do už existujúcej databázy môže byť značne pomalšie [41]. V tomto prípade medzikrok uloženia dát z Blockchain databázy do CSV súborov nepredstavuje veľký problém, keďže objem nových dát bude relatívne malý k celkovému objemu dát Blockchain databázy.

3.2.2.3 Pridanie zhlukov

Pridanie informácií o zhlukoch adries bude prevedené užívateľom v užívateľskom rozhraní prostredníctvom procedúr, implementovaných v rámci rozšírenia databázy. Užívateľ bude môcť pridať zoznam adries konkrétnemu zhluku, zadaním názvu zhluku a zoznamu adries. V prípade, že zhluk nebude existovať, vytvorí sa nový zhluk s danými adresami, v opačnom prípade sa zoznam adries už existujúceho zhluku rozšíri o zadané adresy. Pridanie zhluku a priradenie adries zhluku bude možné pomocou funkcie *mergeCluster*, ktorej vstup bude názov zhluku a zoznam adries. Užívateľ bude tiež môcť mazať zhluky funkciou *deleteCluster*, ktorá bude mať jediný vstup názov zhluku. Návrh deklarácie funkcií je následovný:

```
void mergeCluster(String cluster, List<String> addresses);
void deleteCluster(String cluster);
```

3.3 Vyhľadávanie

Jazyk Cypher databázy Neo4j poskytuje dostatočné vyjadrovacie schopnosti na implementáciu požadovaných dotazov. Databáza však neumožňuje jednoducho uložiť dotaz v jazyku Cypher a opakovane ho vyhodnocovať s inými parametrami. Databáza podporuje rozšírenia implementované v jazyku Java preložené do jar súborov, v rámci ktorých je možné používať jazyk Cypher, preto návrh rozširuje databázu o vlastné procedúry. Tie umožnia jednoduché opakované dotazovanie nad grafovými dátami s inými vstupnými parametrami dotazov.

3.3.1 Vyhľadanie susedných zhlukov a adries

Vyhľadávanie susedov zhlukov a adries bude možné pomocou dvoch funkcií. Funkcia *neighboursSimple* bude mať jediný vstup typu reťazec a jeho hodnota môže byť:

- reťazec identifikátora adresy, hodnota *addr* uzla typu *Addr*;
- reťazec identifikátora zhluku adries, hodnota *name* uzla typu *Cluster*.

Výsledkom funkcie bude graf, ktorý obsahuje všetky susedné uzly typu adresa a zhluk k počiatočnému uzlu a všetky uzly typu transakcia, výstup transakcie a hrany na ceste k nim.

Druhou funkciou na vyhľadávanie bude funkcia *neighbours*. Tá bude mať naviac parametre filtrujúce všetkých susedov podľa času prevedenia a objemu transakcií medzi susednými uzlami typu adresa alebo zhluk adries. Obmedzenie času aj objemu bude dosiahnuté vyžadovaným intervalom, pre otvorený interval z ľubovolnej strany budú slúžiť záporné hodnoty príslušných ohraničení. Keďže databáza nepodporuje priamo dátový typ Date, dátumy budu načítané z reťazca. Podporované budú dva formáty, prvým je formát 'd.M.yyyy', ktorého príkladom je '1.4.2017', v prípade, že reťazec nebude spĺňať tento formát, načíta sa reťazec ako číslo typu long a použije sa ako unixová časová značka. Ohraničenie objemu transakcie bude docielené spodnou a hornou hranicou číselným typom long, jednotka bude predstavovať hodnotu jeden Satoshi. Návrh deklarácie funkcií je následovný:

```
Graph neighboursSimple(String startAddrOrCluster);
Graph neighbours(String startAddrOrCluster, String minDate,
             String maxDate, Long minValue, Long maxValue);
```

3.3.2 Vyhľadanie ciest medzi zhlukmi a adresami

Vyhľadanie ciest medzi zhlukmi a adresami bude opäť dosiahnuté dvojicou funkcií. Prvou jednoduchšou funkciou bude funkcia *pathsSimple*, ktorá bude mať tri vstupy, identifikátor počiatočného uzlu typu adresa alebo zhluk adries, identifikátor cieľového uzlu typu adresa alebo zhluk adries a maximálnu povolenú dĺžku cesty. Identifikátory uzlov sú rovnaké ako v prípade vyhľadávania susedných uzlov. Novým atribútom je dĺžka cesty, tá určuje, cez koľko transakcií môže cesta najviac viesť, pričom jedna transakcia spôsobí minutie celého objemu vstupov a presunutie zdrojov na novú množinu adries.

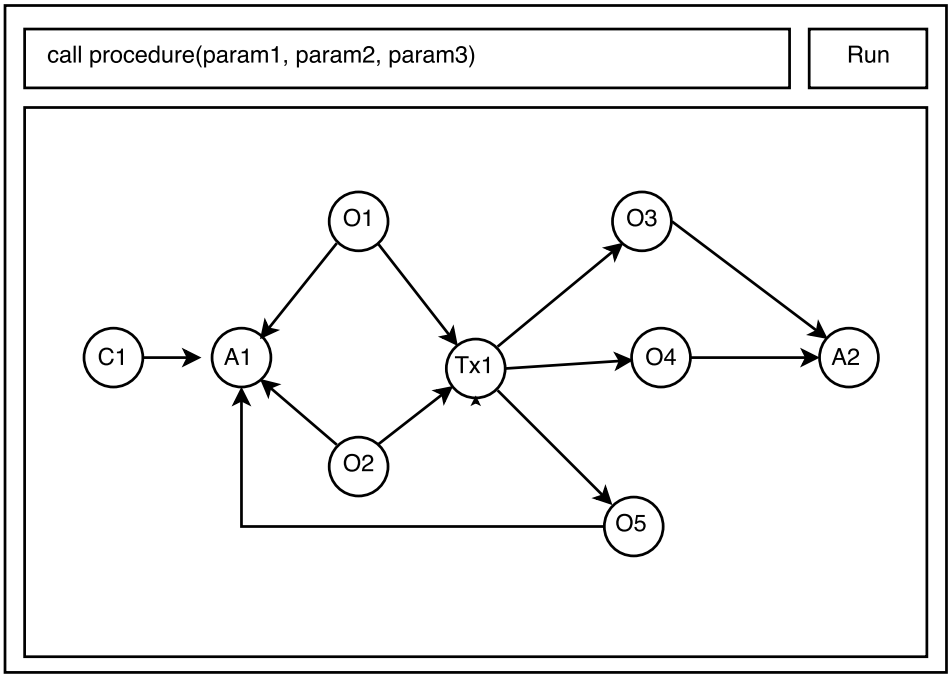
Druhou funkciou bude funkcia *paths*. Tá bude naviac obsahovať parametre pre filtrovanie výsledkov podľa času a objemu, rovnako ako v prípade vyhľadávania susedných uzlov. Výsledkom oboch funkcií bude opäť graf, ktorý zachytí všetky cesty maximálnej zadanej dĺžky medzi počiatočným a cieľovým uzlom spĺňujúce prípadné časové a objemové požiadavky. Toto je návrh deklarácie funkcií:

```
Graph pathsSimple( String startAddrOrCluster,
                   String endAddrOrCluster,
                   Long depth);

Graph paths(String startAddrOrCluster,
            String endAddrOrCluster,
            String minDate, String maxDate,
            Long minValue, Long maxValue,
            Long depth);
```

3.4 Užívateľské rozhranie

Užívateľské rozhranie bude tvoriť vizualizačný nástroj databázy Neo4j Browser, ten umožňuje zadávať a zobrazovať výsledky dotazov. Do hornej lišty nástroja sa budú vkladať dotazy v podobe preddefinovaných procedúr. Parametre procedúr zvolí užívateľ v závislosti od konkrétneho dotazu. Výsledok dotazu sa zobrazí ako graf, s ktorým bude ďalej možné manipulovať. Bude možné meniť farby a veľkosti pre všetky typy uzlov a hrán, posúvať uzly a rozkliknutím uzlu sa graf rozšíri o priame susedné uzly rozkliknutého uzlu. Prostredie naviac umožní užívateľovi nad databázou vytvárať ľubovolné adhoc dotazy v jazyku Cypher.



Obr. 3.3: Návrh užívateľského rozhrania. V hornej lište sa nachádza príkazový riadok, do ktorého bude možné zadávať implementované procedúry so zvolenými parametrami. Stlačením tlačidla Run sa zobrazí výsledok dotazu ako graf. Graf reprezentuje vyhľadanie ciest medzi zhlukom *C1*, ktorému patrí adresa *A1*, s dostupnými výstupmi *O1* a *O2*, vstupujúcich do transakcie *Tx1*. Transakcia zlúčené objemy rozdeluje na tri nové výstupy, dva výstupy *O3* a *O4* putujú na cieľovú adresu *A2*, zatiaľčo výstup *O5* putuje naspať na adresu *A1* zhluku *C1*.

Kapitola 4

Realizácia

Na základe návrhu boli implementované jednotlivé úkony stiahnutia a synchronizácie dát, uloženia dát do databázy a synchronizácia databázy, rozširujúce procedúry databázy a výsledné vizualizácie v užívateľskom rozhraní, ktoré umožňuje volať implementované procedúry podľa parametrov zadaných užívateľom.

4.1 Synchronizácia dát

Synchronizácia dát bola dosiahnutá nainštalovaním a spustením klienta BitcoinCore, ktorý stiahol a zosynchronizoval dáta kryptomeny a uložil ich do lokálnej Blockchain databázy, ktorú vytvoril. Dáta o všetkých transakciách do dátumu 7.3.2017 majú celkovú veľkosť 123 GiB a sú uložené v 796 súboroch typu blk. Pred naplnením databázy je nutné spustiť klienta a počkať, kým sa nové dáta zosynchronizujú.

4.2 Naplnenie databázy

Návrh riešenia popisuje dva spôsoby plnenia databázy. Na vytvorenia a naplnenie novej databázy je možné použiť nástroje, navrhnuté pre tento účel, ktoré celý proces urýchľujú, nie je však možné použiť ich pre doplnenie už existujúcej databázy. Pre naplnenie nových dát do existujúcej databázy je proces nahrávania odlišný. V oboch prípadoch je postup nahrávania podobný, v prvom kroku sa vytvoria potrebné CSV súbory, v druhom kroku sa obsah súborov nahrá do databázy.

4.2.1 Vytvorenie potrebných CSV súborov

Pre načítanie a export dát z lokálnej Blockchain databázy do CSV súborov som porovnal dve knižnice. Viac vláknovú knižnicu Rusty Blockparser [27] s otvoreným zdrojovým kódom a licenciou GPL v3.0 a knižnicu Bitcoingraph [1]

s otvoreným zdrojovým kódom, dostupnú pod MIT licenciou. Výsledná implementácia využíva knižnicu Bitcoingraph z licenčných dôvodov.

Formát výsledných CSV súborov kopíruje navrhnutú dátovú schému, pre každú orientovanú hranu existuje osobitný CSV súbor. Formát všetkých potrebných súborov je následovný:

blk.csv

- hash — identifikátor bloku, reťazec;
- num — hĺbka bloku v Blockchain databáze, long;
- time — unixová časová značka zmeny bloku, long.

tx.csv

- txid — identifikátor transakcie, reťazec.

out.csv

- outId — identifikátor výstupu transakcie, tvorí ho id transakcie, podtržítko a jeho poradové číslo v zozname výstupov transakcie, z ktorej pochádza, reťazec;
- value — hodnota v satoshi.

addr.csv

- addr — adresa, string.

blk_rel.csv

- start — id aktuálneho bloku, reťazec;
- end — id predchádzajúceho bloku, reťazec.

tx_blk_rel.csv

- start — id transakcie, reťazec;
- end — id bloku, do ktorého transakcia patrí, reťazec.

tin_rel.csv

- start — id transakčného výstupu, reťazec;
- end — id transakcie, pre ktorú je daný výstup vstupom, reťazec.

tou_rel.csv

- start — id transakcie, reťazec;
- end — id transakčného výstupu, ktorý vznikol danou transakciou, reťazec.

addr_rel.csv

- start — id transakčného výstupu, reťazec;
- end — cieľová adresa, na ktorú bol výstup odoslaný, reťazec.

4.2.2 Naplnenie novej databázy

Pre vytvorenie a naplnenie novej Neo4j databázy je nutné vytvoriť CSV súbory obsahujúce informácie o všetkých historických transakciách, uložených v Blockchain databáze. Následne sa nástrojom neo4j-import vytvorí Neo4j databáza, obsahujúca informácie z CSV súborov.

Nová Neo4j databáza vznikla použitím nástroja neo4j-import, ktorý umožňuje vytvoriť uzly a hrany z CSV súborov, každý CSV súbor musí buď obsahovať hlavičku, alebo musí pre CSV súbor existovať osobitný hlavičkový súbor. V rámci argumentov nástroja sa pomocou dvojbodky špecifikujú typy uzlov a hrán. Databázu je možné vytvoriť následovne:

```
$NEO4J_HOME/bin/neo4j-import --into  $DATABASE_PATH \
--nodes:Blk "$HEADER_PATH/blk_h.csv", blk.csv \
--nodes:Tx "$HEADER_PATH/tx_h.csv", tx.csv \
--nodes:Out "$HEADER_PATH/out_h.csv", tout.csv \
--nodes:Addr "$HEADER_PATH/addr_h.csv", addr.csv \
--relationships:PREV_BLK "$HEADER_PATH/rl_h.csv", blk_rel.csv \
--relationships:FROM_BLK "$HEADER_PATH/rl_h.csv",tx_blk_rel.csv\
--relationships:FROM "$HEADER_PATH/rl_h.csv", tin_rel.csv \
--relationships:TO "$HEADER_PATH/rl_h.csv", tout_rel.csv \
--relationships:TO_ADDR "$HEADER_PATH/rl_h.csv", addr_rel.csv \
```

4.2.3 Naplnenie existujúcej databázy

Postup pre naplnenie nových dát do existujúcej databázy je podobný, najprv sa vytvoria potrebné CSV súbory, tento krát iba z niektorých súborov Blockchain databázy a následne sa dáta nahrajú do databázy. V tomto prípade sa dáta nahrajú do spustenej databázy pomocou príkazov v jazyku Cypher.

Dáta sú do databázy nahrané klauzulou LOAD CSV jazyka Cypher. Nasledujúca ukážka demonštruje nahranie uzlov transakcií, uzlov výstupov transakcií a nahranie vzťahov medzi týmito typmi uzlov:

```
// Tx
USING PERIODIC COMMIT 100000
LOAD CSV FROM 'tx.csv' AS line FIELDTERMINATOR ','
MERGE (f:Tx { txid: line[0]})

// Out
USING PERIODIC COMMIT 100000
LOAD CSV FROM 'out.csv' AS line FIELDTERMINATOR ','
MERGE (o:Out { outId: line[0] })
ON CREATE SET o.value = toInt(line[1]);

// Tx -> Out
USING PERIODIC COMMIT 100000
LOAD CSV FROM 'tout_rel.csv' AS line FIELDTERMINATOR ','
MATCH (tx:Tx { txid: line[0]})
MATCH (o:Out { outId: line[1]})
MERGE (tx)-[:TO]->(o);

// Out -> Tx
USING PERIODIC COMMIT 100000
LOAD CSV FROM 'tin_rel.csv' AS line FIELDTERMINATOR ','
MATCH (o:Out { outId: line[0] })
MATCH (tx:Tx { txid: line[1]})
MERGE (o)-[:FROM]->(tx);
```

4.2.4 Nahranie dát o zhlukoch adries

Vytvorenie zhlukov a priradenie adries do zhlukov je implementované dotazom v jazyku Cypher, zaobalenom spolu s dotazmi vyhľadávania do balíčka jazyka Java, ktorý po preložení rozširuje funkčnosť databázy Neo4j. Dotaz v jazyku Cypher je následovný:

```
MERGE (c:Cluster{name:{clusterName}})
WITH c
MATCH (a:Addr) WHERE a.addr IN {addrs}
CREATE (c)-[:HAS_ADDR]->(a)
```

V rámci dotazu sa najprv vyhodnotí klauzula MERGE, ktorá v prípade, že uzol typu *Cluster* s menom *clusterName* neexistuje, taký uzol vytvorí alebo ho v opačnom prípade len vráti. Vzniknutý či nájdený uzol je uložený do premennej *c*. Následne sa vyhľadajú adresy, uzly typu *Addr*, ktorých adresa je v požadovanom zozname *addr* a pre každú takú adresu, dočasne uloženú do premennej *a* sa vytvorí hrana zo zhluku *c* do adresy *a*.

Následná ukážka demonštruje zaobalenie dotazu jazyku Cypher do funkcie procedúry Neo4j databázy v jazyku Java.

```
@Description("Merge addresses to a cluster")
  @Procedure(value = "bitcoin.mergeCluster", mode = Mode.WRITE)
    public Stream<MapResult> mergeCluster(@Name("name") String
              name, @Name("Addresses") List<String> addresses){
      String query ="MERGE (c:Cluster{name:{clusterName}})\n"+
                    "WITH c \n"+
                    "MATCH (a:Addr) WHERE a.addr IN {addrs}\n"+
                    "CREATE (c)-[:HAS_ADDR]->(a)";
      Map<String,Object> map = new HashMap<>();
      map.put("addrs",addresses);
      map.put("clusterName",name);
      return db.execute(query,map).stream().map(MapResult::new);
    }
```

4.3 Procedúry vyhľadávania

Procedúry vyhľadávania susedov uzlov a ciest medzi uzlami som tiež implementoval pomocou dotazov v jazyku Cypher. Tie som následne zaobalil do funkcií jazyku Java, ktoré tvoria procedúry databázy Neo4j. Spôsob zaobalenia Cypher dotazu do funkcie jazyku Java je demonštrovaný vyššie, preto sa bude nasledujúci text venovať iba dotazom jazyka Cypher. Oba typy dotazov vyhľadávania začínajú rovnako, vyhľadaním počiatočného uzlu, ktorý môže byť typu *Cluster* alebo typu *Adresa*.

Následujúci príkaz vyhľadá zhluk s menom *startNode* a všetky cesty zo zhluku do adries zhluku, ak zhluk s takým menom existuje. Ďalej príkaz vyhľadá adresu s menom *startNode* a všetky cesty zo zhlukov, ktorým adresa patrí, opäť len v prípade, že daná adresa existuje. Následne sa všetky nájdené cesty zo zhlukov do adries zlúčia do množiny výsledkov, pričom premenná *addrs_paths* obsahuje všetky cesty zo zhlukov do adries, ak nejaké existujú a premenná *a* obsahuje odpovedajúce počiatočné adresy.

```
OPTIONAL MATCH addrs_paths = (:Cluster{name:{startNode}})-\
    [:HAS_ADDR]->(a)

WITH collect({addrs_paths:addrs_paths,a:a}) as rows
OPTIONAL MATCH (a2:Addr{addr:{startNode}})

WITH a2, rows
OPTIONAL MATCH addrs_paths2=(:Cluster)-[:HAS_ADDR]->(a2)

WITH rows + collect({addrs_paths:addrs_paths2,a:a2}) as allrows
UNWIND allrows as row

WITH row.addrs_paths as addrs_paths ,row.a as a ...
```

4.3.1 Vyhľadanie susedov uzlov

Vyhľadanie susedov znamená vyhľadať všetky adresy, ktoré vstupovali do transakcií smerujúcich na počiatočnú adresu, uloženú v premennej *a* a tiež vyhľadať všetky adresy, do ktorých smerovali transakcie z počiatočnej adresy. Ku takto nájdeným adresám je navyše vhodné nájsť zhluky, do ktorých adresy patria. Ukážka demonštruje vyhľadanie susedných adries a ich zhlukov, z ktorých bola odoslaná transakcia na počiatočnú adresu. Vyhľadanie adries a zhlukov, na ktoré boli transakcie odoslané z počiatočnej adresy je analogické. Premenné *addrs_paths* a *a* pochádzajú z poddotazu, popísanom vyššie.

Obmedzenie časovej značky bloku a objemu transakcie je docielené klauzulou WHERE a podmienkami hodnôt atribútu *time* uzla bloku *b* a atribútu *value* uzla výstupu transakcie *o*. V prípade otvoreného intervalu, docieleného dosadením zápornej hodnoty premenných *minVal*, *maxVal*, *minDate* alebo *maxVal* sa daná okrajová podmienka do dotazu nepridá. Adaptívne pridávanie podmienok je docielené generovaním klauzuly WHERE v jazyku java v závislosti od hodnôt parametrov funkcie.

```
...
MATCH out_path = (a) <-[:TO_ADDR] - (out:Out)

WITH addrs_paths, a, out_path, out
OPTIONAL MATCH path_to_blk = (out)<-[:TO]-(tx)-[]-(b:Blk),
path_to_final_a = (tx)<-[:FROM]-(o:Out)-[:TO_ADDR]->(final_a)
WHERE o.value >= {minVal} AND o.value <= {maxVal}
 AND b.time >= {minDate} AND b <= {maxDate}

WITH addrs_paths, a, path_to_blk,path_to_final_a,
  final_a, out_path
OPTIONAL MATCH cluster_path=(final_a)<-[:HAS_ADDR]-(c)

RETURN addrs_paths, a, path_to_blk, path_to_final_a,
  cluster_path, out_path
```

4.3.2 Vyhľadanie všetkých ciest medzi uzlami

Vyhľadanie ciest medzi uzlami je mierne zložitejšie, figurujú v ňom dva nové atribúty a to maximálny povolený počet transakcií na ceste *depth*, a cieľový uzol typu zhluk adries alebo cieľová adresa *destinationNode*. Premenná *depth* je v dotaze vynásobená dvoma, keďže na prechod cez jednu transakciu je nutné použiť dve hrany, hranu vstupu do transakcie a hranu jej výstupu. V skutočnosti je hodnota premennej vynásobená dvoma v jazyku Java pred vykonaním dotazu, a dotaz jazyku Cypher násobenie dvoma neobsahuje.

Klauzula WHERE sa vo výraze objavuje dvakrát. Prvý raz zadáva podmienky pre veľkosť objemu cieľového výstupu a čas vykonania prvej transak-

cie, podobne ako v prípade vyhľadania susedných uzlov. Druhý raz už filtruje čas prevedenia cieľovej transakcie. Oba časy musia patriť zvolenému intervalu. Časy transakcií prevedených medzi prvou a poslednou transakciou musia byť z definície dát Blockchain databázy tiež v požadovanom intervale, keďže žiadna transakcia nemôže čerpať vstup z transakcie vykonanej neskôr. Premenné *addrs_paths* a *a* opäť pochádzajú z vyššie popísaného poddotazu.

```
...
MATCH path_to_start_blk = (a)<-[:TO_ADDR]-(out)-[:FROM]->(tx)-\
    [:FROM_BLK]->(b),
path_to_final_a = (tx)-[:TO|:FROM*1..{{depth}*2}]->(o)-\
    [:TO_ADDR]->(final_a)
WHERE o.value >= {minVal} AND o.value <= {maxVal}
 AND b.time >= {minDate} AND b <= {maxDate}

WITH addrs_paths,path_to_start_blk,path_to_final_a,o,\
    final_a
MATCH path_to_end_blk = (o)<-[:TO]-(tx)-[]-(b:Blk)
WHERE b.time >= {minDate} AND b <= {maxDate}

WITH addrs_paths,path_to_start_blk,path_to_final_a,final_a,\
    path_to_end_blk
OPTIONAL MATCH cluster_path=(final_a)<-[:HAS_ADDR]-(c)

WITH addrs_paths,path_to_start_blk,path_to_final_a,
   path_to_end_blk,cluster_path,final_a,c
WHERE final_a.addr = {destinationNode} OR \
    c.name = {destinationNode}
RETURN addrs_paths, path_to_blk, path_to_final_a,
    path_to_end_blk, cluster_path
```

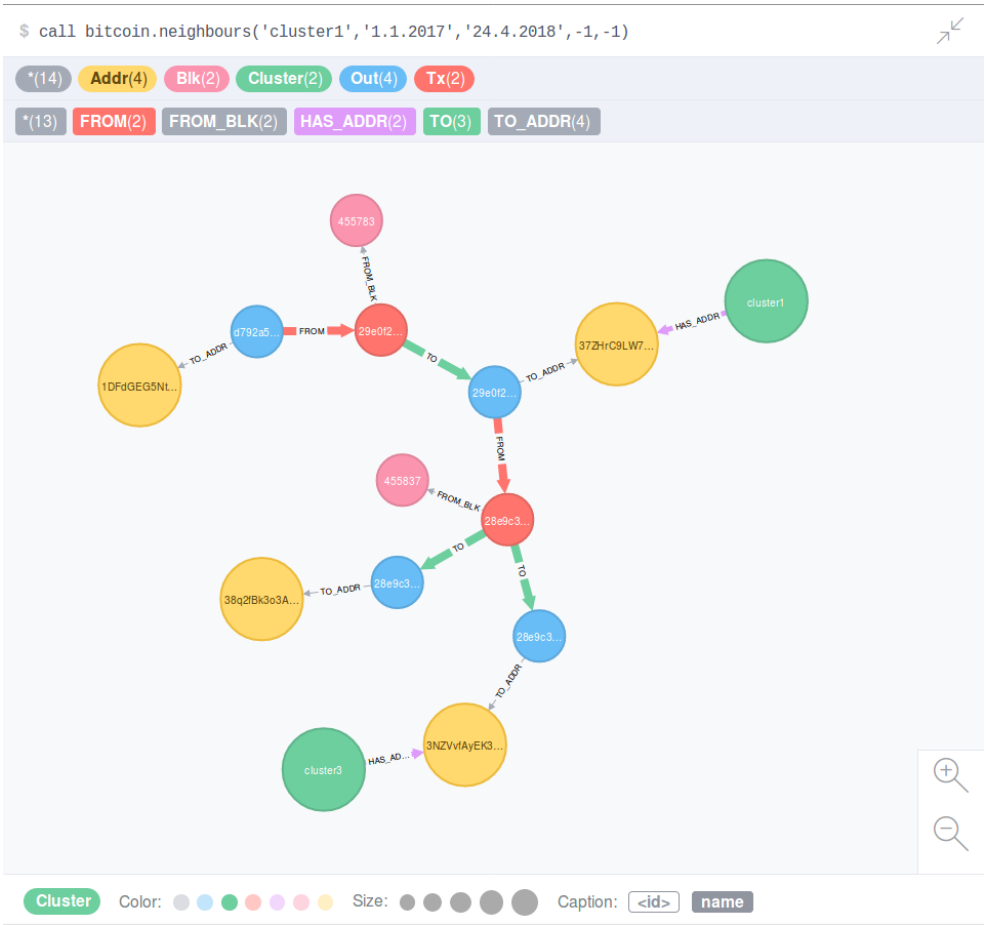
4.4 Užívateľské rozhranie

Rozhranie medzi užívateľom a databázou s rozšírenou funkcionalitou je dosiahnuté použitím nástroja Neo4j Browser. V rámci nástroja je možné meniť nastavenia vizualizácií a databázy. Nástroj tiež ponúka príkazový riadok, do ktorého je možné zadávať dotazy v jazyku Cypher, ten umožňuje volať vlastné procedúry pomocou príkazu CALL. Implementované rozširujúce procedúry je možné volať následovne:

- **call mergeCluster('name', ['address1', 'address2'])** — Ak zhluk s menom *name* neexistuje, vytvorí sa. Adresy *address1* a *address2* sa následne pridajú do zhluku *name*;

- **call deleteCluster('name')** — Zmaže zhluk adries *name* a všetky jeho hrany k adresám, adresy ostanú uložené v databáze;
- **neighboursSimple('startNode')** — Nájde všetky susedné adresy a zhluky uzlu *startNode* a cesty k nim;
- **neighbours('startNode', 'minDate', 'maxDate', minVal, maxVal)** — Funkcia rozširuje funkciu *neighboursSimple* o možnosť filtrovania ciest podľa času a objemu vykonaných transakcií;
- **call pathsSimple('startNode', 'endNode', depth)** — Nájde všetky cesty medzi uzlom *startNode* a *endNode* cez maximálne *depth* transakcií;
- **call paths('startNode', 'endNode', 'minDate', 'maxDate', minVal, maxVal, depth)** — Rozšírenie funkciu *pathsSimple* o možnosť filtrovania.

Po zadaní príkazu do príkazového riadku sa zobrazí výsledný graf, s ktorým je ďalej možné manipulovať a pridávať susedné uzly aktuálnych uzlov grafu. Pre rýchle vizuálne rozlíšenie som použil následné štýly: Väčšie uzly sú žlté uzly adries typu *Addr* a zelené uzly zhlukov adries typu *Cluster*. Menšie uzly sú modré uzly výstupov transakcií typu *Out*, červené uzly transakcií typu *Tx* a ružové uzly blokov typu *Blk*. Hrubé výrazné hrany sú medzi transakčnými výstupmi a transakciami, červené hrany sú hrany vstupov transakcií, ktoré míňajú výstupy a zelené sú hrany výstupov transakcií, ktoré tvoria nové neminuté výstupy. Medzi uzlami zhlukov adries a adresami sú stredne hrubé fialové hrany, ostatné hrany sú šedé a tenké.



Obr. 4.1: Ukážka vizualizácie výsledku jednej z implementovaných procedúr v nástroji Neo4j Browser.

Kapitola 5

Experimenty

Experimenty overujú funkčnosť riešenia a popisujú časové a pamäťové nároky na vytvorenie a naplnenie novej databázy a na doplnenie nových dát do už existujúcej databázy. V rámci overenia funkčnosti boli vytvorené Bitcoin adresy, medzi ktorými boli vykonané transakcie analyzované v implementovanom nástroji.

5.1 Overenie funkčnosti riešenia

Postup overenia funkčnosti systému bol následovný. Prvým krokom bolo založenie Bitcoin peňaženky, spravujúcej páry súkromných kľúčov a im prislúchajúcich verejných kľúčov a adries. Po vygenerovaní adries a prijatí zakúpených bitcoinov boli následne vytvorené nové adresy a transakcie medzi nimi v reálnom prostredí kryptomeny. Dáta kryptomeny boli zosynchronizované klientom BitcoinCore a nahraté do databázy Neo4j. Transakcie boli následne vyhľadané pomocou implementovaných rozšírení databázy a vizualizované v nástroji Browser.

5.1.1 Zobrazenie prijatých prostriedkov

Adresa 36DDDcKSb8a8c5iKCndw3hstNn7qjtG33r vygenerovaná založenou peňaženkou prijala celkovo 0.0028 BTC z troch rôznych transakcií. Celkovú prijatú sumu je možné vidieť v ukážke prehľadu peňaženky 5.1, jednotlivé sumy v ukážke 5.2.

Prijaté vstupy zobrazené v databáze 5.3 majú iné poradie ako vstupy zobrazené v peňaženke 5.2, je to spôsobené tým, že dva vstupy pochádzajú z rovnakého bloku a zatiaľčo databáza zobrazuje čas schválenia bloku prijatej transakcie, peňaženka ukazuje čas, kedy prvý krát zachytila vyslanú transakciu, aj keď v tom čase ešte nebola transakcia schválená.

Z vizualizácie 5.4 je vidno, ktoré výstupy pochádzajú z rovnakého bloku, rovnako je vidno, že každý prijatý výstup bol už minutý, keďže z neho vedie

LABEL	ADDRESS	RECEIVED
Receive Address 3	34TTNtLCJxBndt6eb6PZojY2KLxm3fctWf	0
addr2	36DDDcKSb8a8c5iKCndw3hstNn7qjtG33r	0.0028
addr1	3D8yJ6ArkZuZ5PybCcCXzkzKtRpdRJxzfP	0

Obr. 5.1: Celková prijatá suma bitcoinov adresy *addr2* činí 0.0028 BTC.

DATE	DESCRIPTION	AMOUNT
Mar 5th 2017, 4:57 PM	Received at addr2	+0.0016 $ 2.01
Mar 4th 2017, 11:37 PM	Received at addr2	+0.0004 $ 0.50
Mar 4th 2017, 11:32 PM	Received at addr2	+0.0008 $ 1.00

Obr. 5.2: Zoznam jednotlivých prijatých objemov adresy *addr2* zobrazený pomocou peňaženky.

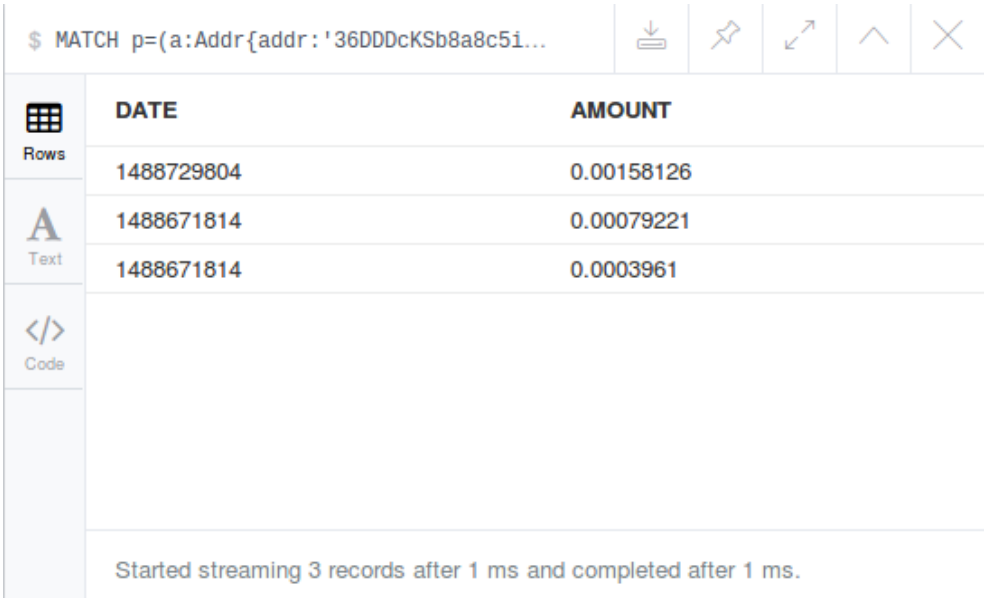
hrubá červená hrana do novej transakcie, ktorá výstup minula. Význam farby uzlov a hrán je vysvetlený v legende na vrchu vizualizácie.

5.1.2 Vizualizácia transakčného scenára

V prevedenom scenári posielania transakcií figurujú tri zhluky adries:

- **cluster1** — zhluk má jednu adresu;
- **cluster2** — zhluk má dve adresy;
- **cluster3** — zhluk má dve adresy.

Zhluk *cluster1* mal k dispozícii jeden transakčný výstup, ten v plnom objeme poslal na prvú adresu zhluku *cluster3*. Zhluk *cluster2* mal pôvodne k dispozícii tri transakčné výstupy, všetky viazané ku jednej adrese. Zhluk *cluster2*



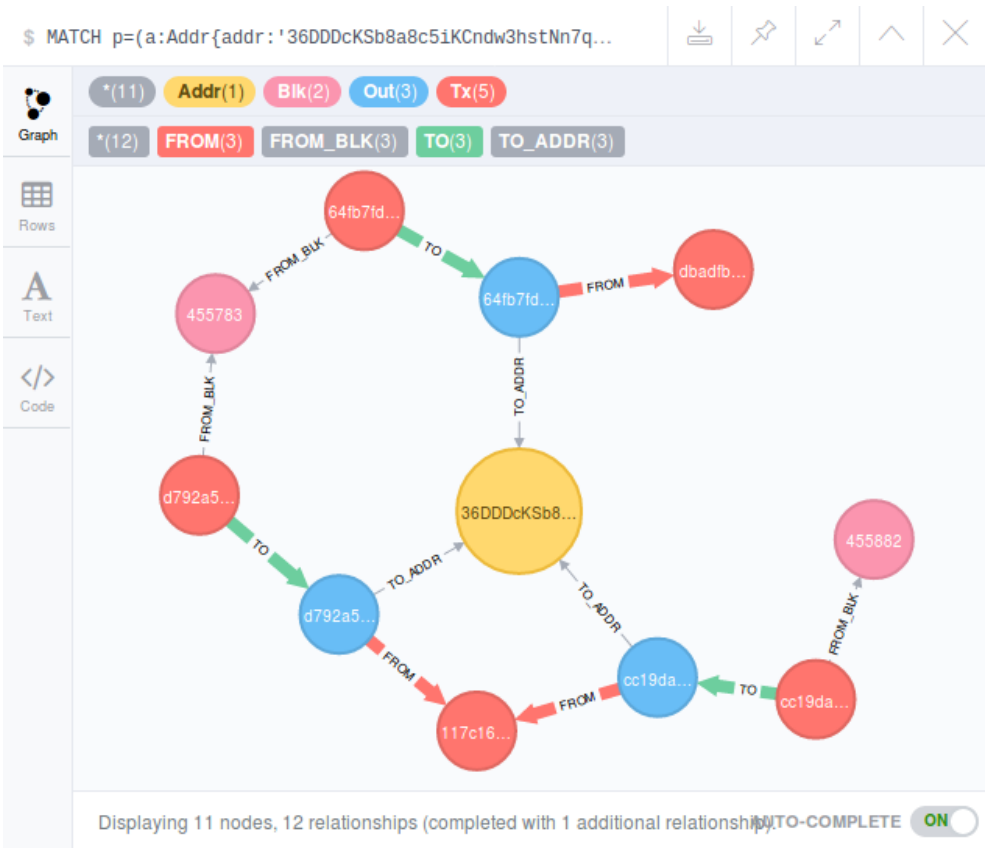
Obr. 5.3: Prijaté výstupy adresy *addr2* uložené v databáze, zoradené podľa času prijatia. Poradie prijatých výtupov je odlišné ako v prípade zobrazenia zoznamu v peňaženke, keďže dve transakcie pochádzajú z rovnakého bloku.

poslal časť objemu jedného z dostupných výstupov na prvú adresu zhluku *cluster3*, zvyšok výstupu si previedol na novú adresu, pôvodný výstup tak vyčerpal. Následne poslal zhluk *cluster2* transakciu v objeme všetkých svojich dostupných výstupov oboch adries na druhú adresu zhluku *cluster3*.

Scenár je možné vidieť vo vizualizačnom nástroji, pomocou implementovanej funkcie *neighbours*, ako je možné vidieť z 5.5. Od zhluku *cluster2* existujú dve transakčné cesty ku zhluku *cluster3*, prvá posiela objem v hodnote 20000 Satoshi a druhá v hodnote 78133 Satoshi. Pomocou implementovanej funkcie *paths* je možné nájsť transakčné cesty s obmedzením času a objemu transakcie, viď 5.6.

5.2 Overenie pamäťových a časových nárokov

Pamäťové a časové nároky riešenia sa líšia v závislosti od spôsobu vkladania dát do databázy a od objemu nahrávaných dát. Po uložení dát do databázy Neo4j sú časové aj pamäťové nároky na dotazy a ich vizualizáciu minimálne. Pamäťovo najnáročnejším úkonom je vytvorenie novej databázy, obsahujúcej všetky relevantné aktuálne dáta kryptomeny Bitcoin. Časové nároky pre vytvorenie novej databázy so všetkými dátami sú prekvapivo porovnateľné s nárokmi na nahranie malého množstva nových dát do už existujúcej databázy.



Obr. 5.4: Vizualizácia prijatých výstupov adresy *addr2* v nástroji Browser. Dve transakcie pochádzajú z rovnakého bloku, uzol bloku má ružovú farbu.

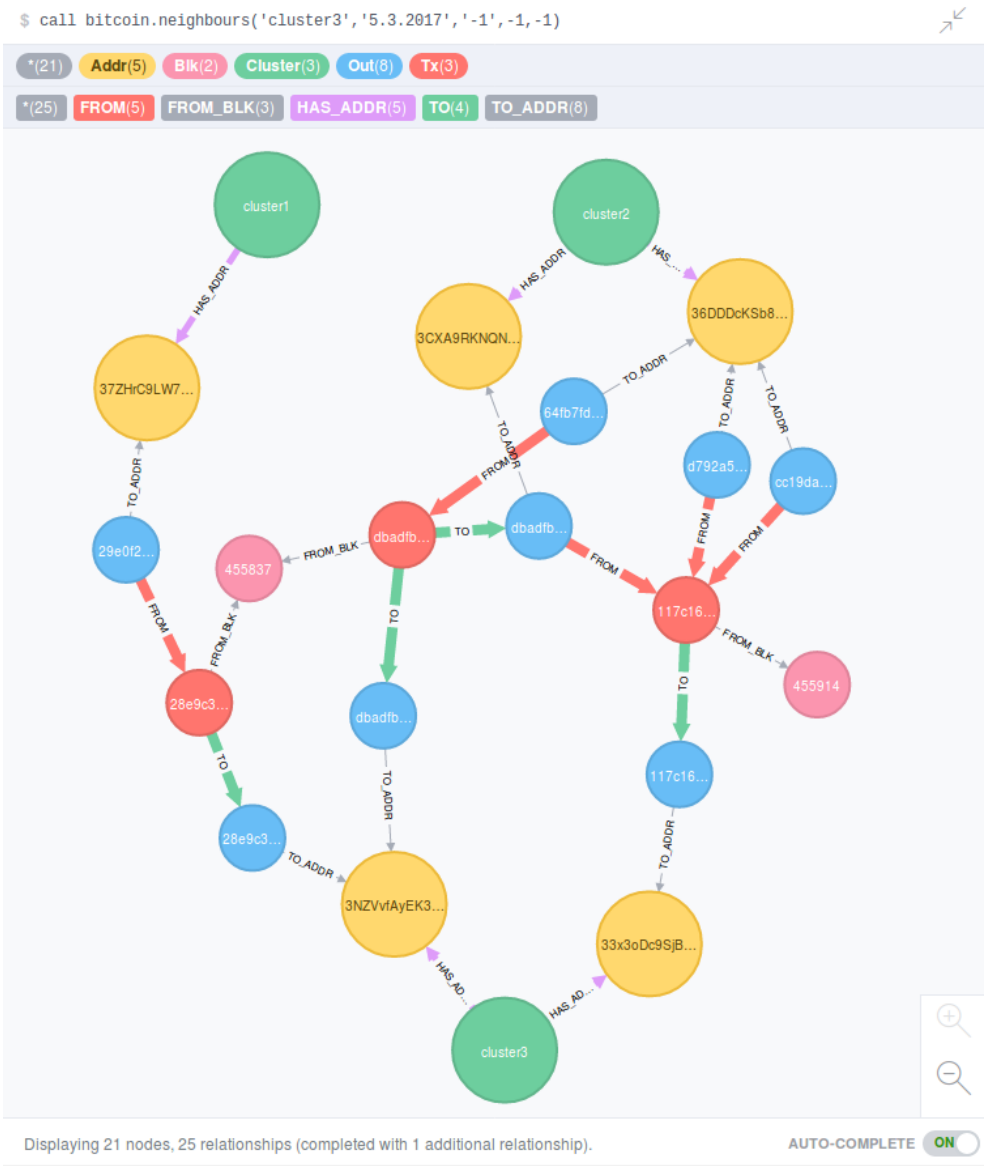
5.2.1 Popis použitého PC

Nasledujúce merania boli prevedené na PC s procesorom i7-6700K, operačnou pamäťou 64 GiB a SSD diskom s kapacitou 1TiB.

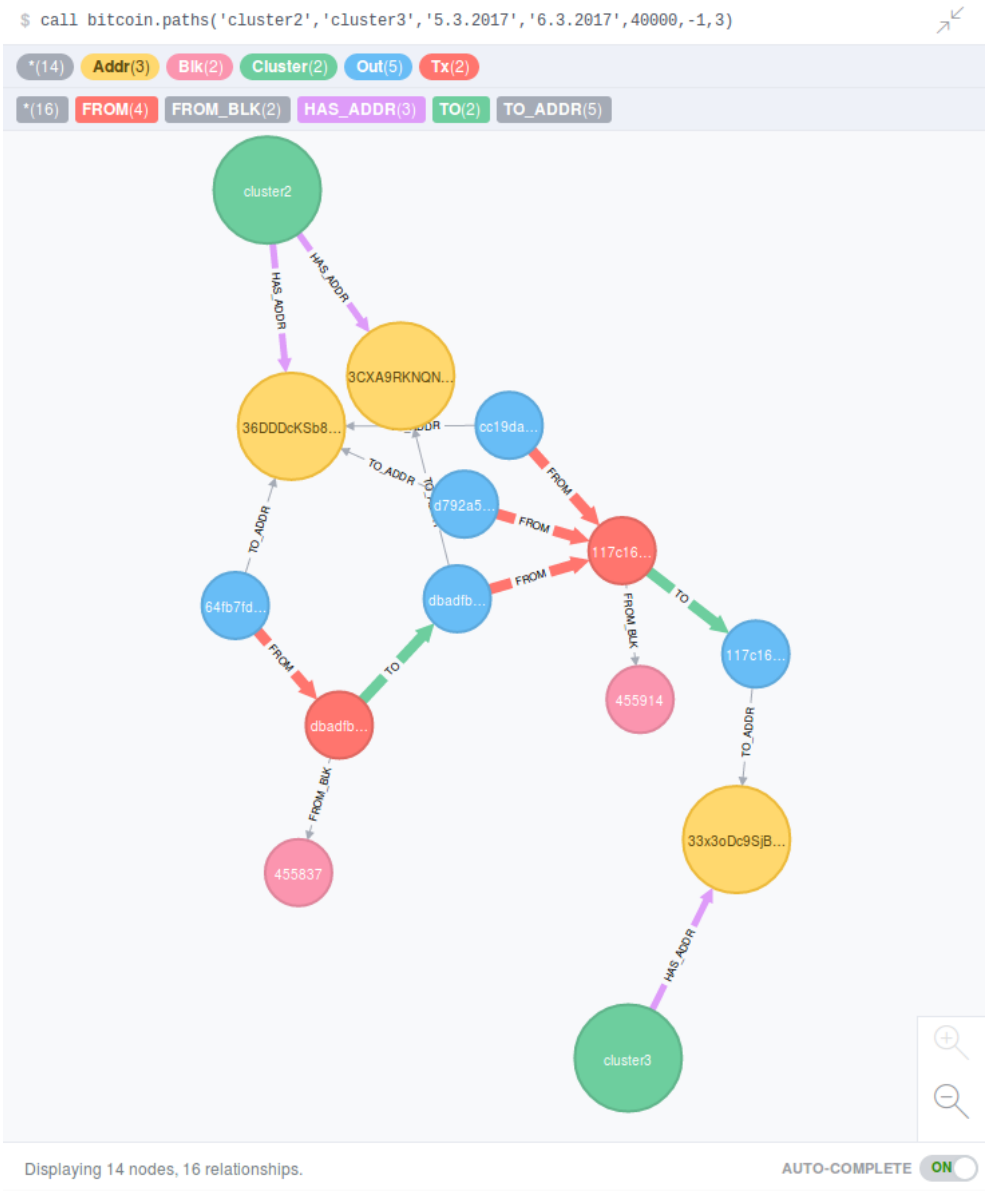
5.2.2 Porovnanie knižníc pre tvorbu CSV súborov

Rýchlosť a pamäťové nároky na tvorbu CSV súborov závisia od použitej knižnice. Viacvláknová knižnica Rusty blockparser je schopná vytvoriť prvotné CSV súbory, vo formáte odlišnom od požadovaného formátu. Vytvorené CSV súbory je nutné ďalej upraviť do požadovaného formátu pomocou bash skriptov. Upravená knižnica Bitcoingraph generuje CSV súbory priamo v požadovanom formáte, je však značne pomalšia.

Časové a pamäťové nároky knižníc sú porovnané v tabuľke 5.1. Zatiaľčo knižnica Rusty blockparser má oproti knižnici Bitcoingraph výrazne menšie časové nároky a nároky na operačnú pamäť, jej nároky na diskovú kapacitu sú



Obr. 5.5: Vizualizácia vykonaného scenára posielania transakcií pomocou implementovanej funkcie *neighbours*, so zadaným obmedzením na minimálnu hodnotu časovej známky prevedenej transakcie. Zľava hore hore smerom doprava dole je poradie zhlukov *cluster1*, *cluster2* a *cluster3*.



Obr. 5.6: Vyhľadanie všetkých transakčných ciest z uzla *cluster2* do uzla *cluster3* obmedzené časom schválenia transakcie a minimálny objemom 40000 Satoshi. Výsledkom sú dve cesty smerujúce cez jednu transakciu a jedna cesta smerujúca cez dve transakcie.

Tabuľka 5.1: Nároky na vytvorenie CSV súborov obsahujúcich dáta databázy Blockchain ku dátumu 7. 3. 2017. Výsledky odpovedajú priemeru desiatich meraní.

Použité nástroje	Čas výkonu	Kapacita disku	Operačná pamäť
Rusty blockparser	2h 27m	517 GiB	5.2 GiB
Bitcoingraph	13h 16m	202 GiB	32.9 GiB

Tabuľka 5.2: Časové a pamäťové nároky synchronizácie dát a vytvorenia novej databázy z CSV súborov. Výsledky odpovedajú priemeru desiatich meraní.

Činnosť	Čas výkonu	Kapacita disku	Operačná pamäť
Blockchain synch.	-	120 GiB	1 GiB
Naplnenie databázy	3h 33m	287 GiB	34.2 GiB

Tabuľka 5.3: Celkové nároky všetkých úkonov nutných k vytvoreniu databázy obsahujúcej dáta databázy Blockchain ku dátumu 7. 3. 2017.

Použitá knižnica	Časové nároky	Pamäťové nároky	Operačná pamäť
Rusty blockparser	6h 0m	637 GiB	34.2 GiB
Bitcoingraph	16h 49m	609 GiB	34.2 GiB

výrazne vyššie. Je to spôsobené iným formátom výstupu knižnice, ktorý vyžaduje tvorbu nových CSV súborov. V čase tvorby nových CSV súborov sú dáta v CSV súboroch duplikované. Výsledná implementácia využíva z licenčných dôvodov knižnicu Bitcoingraph.

5.2.3 Vytvorenie novej databázy

V prípade vytvorenia novej databázy je nutné stiahnuť a zosynchronizovať dáta pomocou klienta BitcoinCore. Doba synchronizácie závisí od posledného dátumu synchronizácie. Prvotné stiahnutie dát môže byť zdĺhavé. Následne treba vytvoriť popísané CSV súbory a obsah súborov nahrať nástrojom neo4j-import do novej databázy. Nároky na synchronizáciu dát a nahranie obsahu CSV súborov do novej databázy sú popísané v tabuľke 5.2.

Blockchain databáza musí byť na disku uložená stále, pre rýchlu synchronizáciu nových dát. Pri použití knižnice Rusty blockparser musia byť naviac na disku naraz uložené prvotné a finálne CSV súbory, čo spolu s dátami Blockchain databázy tvorí 637 GiB. Pred Vytvorením novej databázy je možné prvotné CSV súbory zmazať. Tesne pred úspešným vytvorení databázy zaberajú dáta 609 GiB, na disku sa totiž nachádza Blockchain databáza, výsledné CSV súbory a potrebné súbory databázy. Po úspešnom vytvorení databázy je možné vymazať všetky CSV súbory a pamäťové nároky pre blockchain a databázu tvoria 407 GiB. Výsledné nároky na vytvorenie novej databázy sú popísané v tabuľke 5.3.

5.2.4 Nahranie dát do existujúcej databázy

V prípade doplnenia dát do existujúcej databázy je postup jej naplnenia iný. V tomto prípade sú pamäťové nároky zanedbateľné, časové nároky však nie. Výsledky merania odpovedajú nahratiu transakčných dát za obdobie dvoch dní, pričom zdrojové dáta v databáze Blockchain zaberajú 33.6 MiB. Čas tvorby CSV súborov bol 4 minúty, čas nahratia dát do spustenej databázy je však 4 hodiny a 58 minút, čo dokopy činí 5 hodín a 2 minút.

Problém nahrávania nových dát je ten, že sa každá nová transakcia odkazuje na n predošlých výstupov Tie je v databáze nutné individuálne vyhľadať, vytvoriť uzly novej transakcie a pridať hrany nájdeným výstupom, výstupy môžu byť naviac chronologicky úplne odlišné a fyzicky sa nachádzať na rôznych miestach disku. Pri pridávaní nových dát do existujúcej databázy sa realizuje veľké množstvo vyhľadávaní a zmien v indexácii.

5.2.5 Porovnanie prístupov nahrávania do databázy

Výsledky meraní jednoznačne ukazujú na problematické vkladanie nových dát do už existujúcej databázy. Zatiaľčo vytvorenie novej databázy zo všetkých 120 GiB dát databázy Blockchain trvá 6 hodín použitím knižnice Rusty blockparser a necelých 17 hodín použitím knižnice Bitcoingraph, pridanie 33.6 MiB dát z Blockchain databázy trvá 5 hodín 2 minúty. Preto je v prípade, že dáta neboli synchronizované dlhšie ako dva dni pri použití knižnice Rusty blockparser, alebo šesť dní použitím knižnice Bitcoingraph, výhodné aktuálnu databázu vymazať a znova vytvoriť. Tvorba novej databázy je rýchlejšia z toho dôvodu, že sa vytvárajú rovno cieľové databázové súbory a databáza je v tom čase vypnutá. Indexy uzlov a hrán je teda možné vytvoriť raz na konci, zatiaľ čo v prípade nahrania dát do existujúcej databázy je nutné index udržovať stále.

Záver

Cieľom diplomovej práce bolo navrhnúť nástroj, schopný vizualizovať vzťahy medzi dátami grafového charakteru a overiť jeho časové a pamäťové nároky. Jednotlivé problémy, ktorými sa práca zaoberala sú uloženie dát, návrh dátovej schémy, zlučovanie uzlov grafu do zhlukov, vyhľadávanie ciest medzi zhlukmi a vyhľadávanie susedných zhlukov, filtrovanie ciest grafu podľa obmedzení váh a časových značiek hrán ciest a nakoniec tvorba užívateľského rozhrania a výslednej vizualizácie.

V rámci práce som analyzoval databázu Blockchain v kontexte dát kryptomeny Bitcoin. Popísal som základné entity databázy, dátovú štruktúry, princíp kolektívneho overovania databázy a schvaľovania nových dát. Dáta Blockchain databázy som namapoval na riešený problém.

Následne som analyzoval dostupné nástroje pre ukladanie dát, vyhľadávanie a filtrovanie ciest v grafoch a vizualizáciu vzťahov grafov. Medzi sebou som porovnal relačné, grafové a dokumentové databázy spolu s databázami typu kľúč-hodnota. Pre ďalší návrh som porovnal možnosti vyhľadávania ciest v grafoch použitím jazyka databáz, grafových knižníc a vlastnou implementáciou. Taktiež som porovnal vizualizačné nástroje databáz a nezávislé grafové vizualizačné nástroje. Z analyzovaných nástrojov som vybral vhodné nástroje pre ďalší postup.

Vybrané technológie som použil pre návrh nástroja, ktorý bude vykonávať vyhľadávanie, filtrovanie a vizualizácie podľa zadania. Navrhol som dátovú schému pre zvolenú grafovú databázu, spôsoby naplnenia databázy, funkcie pre vyhľadávanie a filtrovanie ciest grafu a užívateľské rozhranie.

Podľa návrhu som vytvoril grafovú databázu Neo4j a implementoval vyhľadávanie ciest a ich filtrovanie v jazyku Java a Cypher. Implementované procedúry som preložil do rozšírenia databázy a to použil v kombinácii s vizualizačným nástrojom Browser, čím vzniklo užívateľské rozhranie.

Funkčnosť výsledného nástroja som otestoval prevedením reálneho scenára v kontexte kryptomeny a jeho vizualizáciou vo vytvorenom nástroji. Následne som zmeral časové a pamäťové nároky riešenia. Prácu som ukončil porovnaním

nárokov vytvorenia novej databázy a doplnenia dát do existujúcej databázy. Oba spôsoby naplnenia dát som porovnal s použitím dvoch rôznych knižníc, výsledná implementácia nakoniec z licenčných dôvodov používa pomalšiu knižnicu.

Literatúra

[1] Bernhard Haslhofer: A Python library for exploring the Bitcoin transaction graph. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://github.com/behas/bitcoingraph

[2] Bitcoin: Bitcoin Core. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://github.com/bitcoin/bitcoin

[3] Bitcoin Project: Bitcoin Developer Guide. [online], 2014, [cit. 2014-05-01]. Dostupné z: https://bitcoin.org/en/developer-guide

[4] Bitcoin Project: Blockchain data. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://BitcoinProject/bin/block-chain

[5] Bitcoin Project: Distributing-Only wallet. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/img/dev/en-wallets-distributing-only.svg

[6] Bitcoin Project: Full service wallet. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/img/dev/en-wallets-full-service.svg

[7] Bitcoin Project: Inteface. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/en/bitcoin-core/features/user-interface

[8] Bitcoin Project: P2PKH Address, Pay To PubKey Hash. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/en/glossary/p2pkh-address

[9] Bitcoin Project: Privacy. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/en/bitcoin-core/features/privacy

[10] Bitcoin Project: Requirements. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/en/bitcoin-core/features/requirements

[11] Bitcoin Project: Signing-Only wallet. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/img/dev/en-wallets-signing-only.svg

[12] Bitcoin Project: Signing output to spend. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/img/dev/en-signing-output-to-spend.svg

[13] Bitcoin Project: Transaction Propagation. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/img/dev/en-transaction-propagation.svg

[14] Bitcoin Project: Validation. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://bitcoin.org/en/bitcoin-core/features/validation

[15] Bitcoin Wiki: Block hashing algorithm. [online], 2015, [cit. 2017-05-03]. Dostupné z: https://en.bitcoin.it/wiki/Block_hashing_algorithm

[16] Bitcoin Wiki: Collapse of Mt. Gox. [online], 2015, [cit. 2017-05-01]. Dostupné z: https://en.bitcoin.it/wiki/Collapse_of_Mt._Gox

[17] Bitcoin Wiki: Bitcoin Core 0.11 (ch 2): Data Storage. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_2):_Data_Storage

[18] Bitcoin Wiki: Block. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://en.bitcoin.it/wiki/Block

[19] Bitcoin Wiki: Transaction. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://en.bitcoin.it/wiki/Transaction

[20] Bitcoin Wiki: Value overflow incident. [online], 2016, [cit. 2017-05-01]. Dostupné z: https://en.bitcoin.it/wiki/Value_overflow_incident

[21] Blockchain Ltd.: Confirmed Transactions Per Day. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://blockchain.info/charts/n-transactions?timespan=all

[22] Blockchain Ltd.: Market Price (USD). [online], 2017, [cit. 2017-05-03]. Dostupné z: https://blockchain.info/charts/market-price?timespan=all

[23] Blockchain Ltd.: Total Number of Transactions. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://blockchain.info/charts/n-transactions-total?timespan=all

[24] Ching, A.: Scaling Apache Giraph to a trillion edges. [online], 2013, [cit. 2017-05-03]. Dostupné z: https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920/

[25] DataStax, Inc.: TITAN Distributed Graph Database. [online], 2015, [cit. 2017-05-03]. Dostupné z: http://titan.thinkaurelius.com/

[26] DataStax, Inc.: Configuring data consistency. [online], 2017, [cit. 2017-05-03]. Dostupné z: http://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html

[27] Egger, M.: Rusty blockparser. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://github.com/gcarq/rusty-blockparser

[28] Elasticsearch BV: Elasticsearch Reference. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://www.elastic.co/products/x-pack

[29] Elasticsearch BV: Graph. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://www.elastic.co/products/x-pack/graph

[30] Elasticsearch BV: Kibana. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://www.elastic.co/products/kibana

[31] Elasticsearch BV: X-Pack. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

[32] Gephi.org: Features. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://gephi.org/features/

[33] Ledger SAS : Cryptocurrency hardware wallet. [online], 2009, [cit. 2017-05-01]. Dostupné z: https://www.ledgerwallet.com/

[34] Linkurious SAS: Exploring the VW scandal with graph analysis. [online], 2015, [cit. 2017-05-03]. Dostupné z: https://linkurio.us/blog/exploring-the-vw-scandal-with-graph-analysis/

[35] Linkurious SAS: A JavaScript library to visualize and interact with graphs. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://github.com/Linkurious/linkurious.js

[36] Linkurious SAS: Visualize and explore your data in minutes. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://linkurio.us/product/

[37] MongoDB, Inc.: graphLookup (aggregation). [online], 2017, [cit. 2017-05-03]. Dostupné z: https://docs.mongodb.com/manual/reference/operator/aggregation/graphLookup/

[38] MongoDB, Inc.: MongoDB 3.4 Your Database Evolved. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://www.mongodb.com/mongodb-3.4

[39] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. Listopad 2008. Dostupné z: https://bitcoin.org/bitcoin.pdf

[40] Nakamoto, S.: Bitcoin v0.1 released. [online], 2009, [cit. 2017-05-01]. Dostupné z: http://www.mail-archive.com/cryptography@metzdowd.com/msg10142.html

[41] Neo Technology, Inc.: Effective Bulk Data Import into Neo4j. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://neo4j.com/blog/bulk-data-import-neo4j-3-0/

[42] Neo Technology, Inc.: Intro to Cypher. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://neo4j.com/developer/cypher-query-language/

[43] Neo Technology, Inc.: Neo4j Browser User Interface Guide. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://neo4j.com/developer/guide-neo4j-browser/

[44] Neo Technology, Inc.: Neo4j Editions. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://neo4j.com/editions/

[45] Neo Technology, Inc.: Our Customers. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://neo4j.com/customers/

[46] Oracle: Hierarchical Queries. [online], 2016, [cit. 2017-05-03]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm

[47] Oracle: Introduction to Oracle Database. [online], 2017, [cit. 2017-05-03]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/intro.htm

[48] OrientDB Ltd: Clients. [online], 2017, [cit. 2017-05-03]. Dostupné z: http://orientdb.com/customers/

[49] OrientDB Ltd: Graph Editor. [online], 2017, [cit. 2017-05-03]. Dostupné z: http://orientdb.com/docs/2.0/orientdb-studio.wiki/Graph-Editor.html

[50] OrientDB Ltd: OrientDB - The World's First Distributed Multi-Model NoSQL Database with a Graph Database Engine. [online], 2017, [cit. 2017-05-03]. Dostupné z: http://orientdb.com/orientdb/

[51] OrientDB Ltd: SQL - MATCH. [online], 2017, [cit. 2017-05-03]. Dostupné z: http://orientdb.com/docs/last/SQL-Match.html

[52] Schneier, B.; Ferguson, N.: *Practical Cryptography*. 2003.

[53] Stewart, J.: Cryptocurrency-Stealing Malware Landscape. [online], 2014, [cit. 2017-05-03]. Dostupné z: `https://github.com/alecalve/python-bitcoin-blockchain-parser`

[54] The Apache Software Foundation: Welcome to Apache Giraph! [online], 2016, [cit. 2017-05-03]. Dostupné z: `http://giraph.apache.org/`

[55] The Apache Software Foundation: Apache Cassandra. [online], 2017, [cit. 2017-05-03]. Dostupné z: `http://cassandra.apache.org/`

[56] The Apache Software Foundation: Apache Spark™ is a fast and general engine for large-scale data processing. [online], 2017, [cit. 2017-05-03]. Dostupné z: `http://spark.apache.org/`

[57] The Apache Software Foundation: GraphX Programming Guide. [online], 2017, [cit. 2017-05-03]. Dostupné z: `http://spark.apache.org/docs/latest/graphx-programming-guide.html`

[58] The Apache Software Foundation: The Gremlin Graph Traversal Machine and Language. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://tinkerpop.apache.org/gremlin.html`

[59] The Apache Software Foundation: Shortest Paths Example. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://cwiki.apache.org/confluence/display/GIRAPH/Shortest+Paths+Example`

[60] The Apache Software Foundation: Welcome to Apache HBase™. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://hbase.apache.org/`

[61] The PostgreSQL Global Development Group: PostgreSQL 9.4.11 Documentation. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://www.postgresql.org/docs/9.4/static/queries-with.html`

[62] The PostgreSQL Global Development Group: PostgreSQL: About. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://www.postgresql.org/about/`

[63] Tom Sawyer Software: Tom Sawyer Perspectives. [online], 2017, [cit. 2017-05-03]. Dostupné z: `https://www.tomsawyer.com/products/perspectives/`

Dodatok **A**

Zoznam použitých skratiek

AGPL Affero General Public License

CSV Comma Separated Values

GNU GNU's Not Unix

GPL General Public License

HDFS Hadoop Distributed File System

IO Input Output

JSON JavaScript Object Notation

MIT Massachusetts Institute of Technology

PC Personal Computer

PL/SQL Procedural Language/Structured Query Language

QR Quick Response Code

REST Representational State Transfer

SHA Secure Hash Algorithm

SQL Structured Query Language

XML Extensible Markup Language

Dodatok **B**

Obsah priloženého CD

- `DP_Barus_Martin_2017.pdf` ..... text práce vo formáte PDF
- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `neo4j-procedures` . zdrojové kódy procedurálneho rozšírenia databázy
  - `neo4j-scripts` ..... zdrojové kódy pre vytvorenie databázy
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
  - `bitcoingraph` ..... zdrojové kódy upravenej knižnice Bitcoingraph pre tvorbu CSV súborov z Blockchain databázy