Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5212-8279

Matúš Cimerman

Analýza prúdu údajov

Bakalárska práca

Vedúci práce: Ing. Jakub Ševcech

máj, 2015

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5212-8279

Matúš Cimerman

Analýza prúdu údajov

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 9.2.1 Informatika
Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU Bratislava
Vedúci práce: Ing. Jakub Ševcech

máj, 2015

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

ZADANIE BAKALÁRSKEHO PROJEKTU

Meno študenta:	**Cimerman Matúš**
Študijný odbor:	Informatika
Študijný program:	Informatika
Názov projektu:	**Analýza prúdu údajov**

Zadanie:
V ostatnom čase sa štandardom pre spracovanie veľkého objemu údajov stalo distribuované spracovanie pomocou map-reduce paradigmy. Rôzne nástroje založené na tejto paradigme sú vhodné pre dávkové spracovanie veľkého objemu údajov, ktoré však nedokážu poskytnúť presný pohľad na údaje meniace sa v reálnom čase. Pre spracovanie takýchto prúdov údajov vzniklo viacero prístupov (napr. Lambda Architektúra) a nástrojov (napr. Storm), ktoré sa nepozerajú na spracovávané údaje ako na statickú kolekciu, ale ako na potenciálne neobmedzený prúd nových údajov.

Analyzujte súčasný stav spracovania prúdov údajov a rôzne nástroje a techniky používané pri spracovaní takýchto prúdov. Zvoľte si jeden z existujúcich prístupov alebo nástrojov a overte jeho vlastnosti na rôznych úlohách spracovania prúdu údajov ako je napríklad analýza trendov v prúde správ zo služby Twitter alebo v prúde údajov z populárneho skracovača URL adries Bitly.

Práca musí obsahovať:

Anotáciu v slovenskom a anglickom jazyku
Analýzu problému
Opis riešenia
Zhodnotenie
Technickú dokumentáciu
Zoznam použitej literatúry
Elektronické médium obsahujúce vytvorený produkt spolu s dokumentáciou

Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU, Bratislava
Vedúci projektu: Ing. Jakub Ševcech

Termín odovzdania práce v zimnom semestri: 10. 12. 2014
Termín odovzdania práce v letnom semestri: 12. 5. 2015

Bratislava 22. 9. 2014

prof. Ing. Pavol Návrat, PhD.
riaditeľ ÚISI

Anotácia

Fakulta Informatiky a Informačných Technológií
Slovenská Technická Univerzita

Meno:	Matúš Cimerman
Vedúci bakalárskej práce:	Ing. Jakub Ševcech
Bakalárska práca:	Analýza prúdu údajov
Študijný program:	Informatika
Máj 2015	

V súčastnosti pozorujeme spracovanie a analýzu veľkých objemov dát (angl. Big Data) v mnohých oblastiach. Rastúci objem, rôznorodosť a rýchlosť dát zvyšuje záujem o Big Data. Najviac oblasti, v ktorých je veľmi dôležité spracovať veľký objem prúdiacich údajov sú napríklad dáta zo senzorov, počítačové siete, či sociálne médiá ako napríklad Twitter. Tieto potencionálne nekonečné zdroje dát, ktoré sú generované prudko meniacou rýchlosťou a objemom, sú jednoducho nazývané *prúdy údajov*. Spracovanie a analýza prúdu údajov je komplexná úloha. Riešenie musí poskytnúť nízku odozvu, odolnosť voči chybám a horizontálnu škálovateľnosť. V práci budujeme softvérovú súčiastku na spracovanie prúdov v takmer reálnom čase. Poskytuje hodnotné výstupy na základe používateľských dopytov do prúdu, ktorý je citlivý na okamžité spracovanie. Bežne používaný spôsob spracovania Big Data je dávkové spracovanie s použitím MapReduce modelu. Kvôli povahe prúdu údajov je tento prístup v podstate nepoužiteľný na to, aby sme zabezpečili požiadavku získať filtrované alebo analyzované výstupy v takmer reálnom čase. Na dosiahnutie týchto požiadaviek musíme použiť iný prístup nazývaný jednoducho *prúdové spracovanie*. Paradigmy na prúdové spracovanie predstavujú napr. dátovody a filtre. V našom projekte sme analyzovali existujúce riešenia a programovacie rámce na spracovanie dátových tokov. Poskytujeme overenie výkonnosti nášho riešenia. Navrhované riešenie je určené na spracovanie dátových tokov veľkého objemu, je škálovateľné a odolné voči chybám. Splnili sme požiadavku na spracovanie v takmer-reálnom čase použitím nášho riešenia a inkrementálnych algoritmov.

Annotation

Faculty of Informatics and Information Technology
Slovak University of Technology

Name:	Matúš Cimerman
Supervisor:	Ing. Jakub Ševcech
Bachelor thesis:	Data stream analysis
Course:	Informatics
2015, May	

Nowadays we can see Big Data processing and analysis in many domains. Increasing volume, variety and velocity of data has initiated growing interest in Big Data. There are most affected domains where it is essential to process large amounts of streaming data, for example data produced from sensors in general, computer networks or social media such as Twitter. These potentially infinite sources of data, which generating data in rapidly changing velocity and volume, are simply called *data streams*. Processing and analysis of data streams is a complex issue, stream needs to be processed with low-latency, a solution must be fault-tolerant and horizontally scalable. We are building software component to process data streams with near real-time latency. It is providing valuable outputs based on user queries over data stream, which are sensitive to near real-time processing. Routinely used method of Big Data processing is batch processing using MapReduce model. This approach is basically not applicable to process data streams because of the characteristics and requirements to gain filtered or analyzed outputs in near real-time. To achieve these requirements, we need use different approach essentialy called data *stream processing*. One of paradigms to process data streams is for example, pipelines and filters. In our project, we analyzed existing solutions and frameworks for processing data streams. We provide performance verification of the our solution. Proposed solution is designed to process high-volume data streams, its highly scalable and fault-tolerant. We achieved requirement to process data streams in near real-time using proposed topology and incremental algorithms.

Poďakovanie

Na prvom mieste vyslovujem poďakovanie vedúcemu mojej bakalárskej práce, Ing. Jakubovi Ševcechovy, za všetky jeho odborné rady, odovzdané skúsenosti a usmernenie pri tvorení práce.
Touto cestou taktiež vyslovujem poďakovanie všetkým výskumníkom zo skupiny PeWe, za prínosné diskusie a ich spätnú väzbu týkajúcu sa mojej práce.
V neposlednom rade ďakujem celej mojej rodine a priateľom.

Matúš Cimerman

Obsah

1 Úvod **1**

2 Princípy spracovania údajov **3**

2.1 Dávkové spracovanie dát 5

2.1.1 Metódy dávkového spracovania 8

2.2 Prúdové spracovanie dát 10

2.2.1 Vlastnosti prúdu dát 12

2.2.2 Model prúdového spracovania 14

2.2.3 Dopytovanie sa do prúdu dát 15

2.2.4 Problémy a výzvy dolovania znalostí v prúdoch 17

3 Existujúce riešenia modelov spracovania údajov **19**

3.1 Dávkové spracovanie 19

3.1.1 Hadoop 19

3.1.2 EasyBatch 21

3.1.3 Misco 21

3.1.4 Disco 22

3.2 Prúdové spracovanie 22

3.2.1 S4 22

3.2.2 Spark 23

3.2.3 Apache Samza 24
3.2.4 Apache Storm 25
3.3 Zhodnotenie existujúcich riešení 28

4 Návrh riešenia **29**
4.1 Požiadavky na riešenie 29
4.2 Všeobecný návrh 30
4.2.1 Predspracovanie vstupných údajov 31
4.2.2 Spracovanie v reálnom čase 32
4.2.3 Postpracovanie výsledku 33
4.3 Implementácia a použité technológie 34

5 Experimenty **37**
5.1 Metodológia overovania 37
5.2 Filtrovanie prednastaveným filtrom 40
5.3 Filtrovanie viacerými dopytmi 43
5.4 Smerovanie správ na základe dopytu 46
5.5 Zhodnotenie 48

6 Zhodnotenie a budúca práca **49**

Literatúra **51**

Prílohy **55**
A Technická dokumentácia 55
B Inštalačná a používateľská príručka 58

C Grafy 59

C.1 Filtrovanie viacerými dopytmi 59

C.2 Smerovanie správ na základe dopytu 61

D Obsah elektronického média 63

1. Úvod

V ostatnom čase si téma spracovania prúdu údajov v kontexte veľkého objemu dát (angl. Big Data) vyžaduje stále väčšiu pozornosť. Čím ďalej, sa vo viacerých doménach stretávame s problémom spracovania narastajúceho objemu údajov, ktoré sú rozmanité. Tradičné prístupy obchodných informácií (angl. business intelligence) nie sú postačujúce pri riešení týchto problémov. (Liu et al., 2014).

Spracovanie Big Data sa stáva súčasťou stále väčšieho množstva odvetví. Či už ide o veľkých telekomunikačných operátorov, komerčné podniky, vyhľadávače alebo sociálne média. Značným zdrojom dát v súčasnosti sú senzory nachádzajúce sa v zariadeniach, ktoré sú súčasťou bežného života dnes. Inteligentné telefóny a hodinky, športové náramky, či vo všeobecnosti internet vecí (angl. Internet of Things[1]) zmenšuje priepasť medzi svetom fyzických zariadení a prepojení s internetom. Ako príklad masívnosti Big Data môžeme uviesť šesť hodinový medzištátny let Boeingu 737 z New Yorku do Los Angeles, ktorý vygeneruje počas letu približne 240 terabajtov dát (Higginbotham, 2010). Takéto rýchlo vznikajúce dáta, ktoré prúdia vo veľkých objemoch nazývame *prúd údajov* (angl. data stream).

Preto stúpa motivácia a význam vybudovať infraštruktúru, ktorá bude schopná spracovať takýto masívny objem prúdiadich dáť v reálnom čase. Existuje veľa aplikácií, ktorých správne fungovanie môže byť kritické vzhľadom na správnosť výsledku zo súvislého a nekonečného prúdu dát. Ako príklad môžme uviesť letové údaje zo senzorov lietadla Boeing 737 alebo analýzu trendov na sociálnej sieti Twitter[2]. Keby sme takýto prúd spracovali tradičným prístupom spracovania Big Data, dávkovým spracovaním, mohlo by to mať kritický dopad na výsledky aplikácie a súvisiace následky. V prípade Boeingu 737 je jasné, že systém musí poskytnúť výsledky v takmer reálnom čase, inak to môže mať katastrofické následky. Pri analýze trendov na sociálnej sieti Twitter je veľmi pravdepodobné(Mathioudakis and Koudas, 2010), že sa v spracovanej dávke dát stratí trend, ktorý je aktuálny iba pre krátky časový interval. Preto je dáta

[1]IoT council: http://www.theinternetofthings.eu/

[2]http://www.twitter.com/

potrebné spracovávať okamžite po vstupe do aplikácie.

Na úvod je potrebné správne vysvetliť a pochopiť pojem spracovanie v *reálnom čase*. V mnohých systémoch je kritické spracovanie a odozva systému v reálnom čase, avšak chápanie tohto pojmu sa výrazne líši v závislosti od systému, či aplikácie. Aplikácie spĺňajúce kritérium spracovania v reálnom čase považujeme vtedy, ak tieto kritéria spĺňajú pre požiadavky reálneho sveta. Napríklad vysvietenie a vychýlenie fotónu na zobrazovej jednotke osciloskopu určite spĺňa kritériá reálneho sveta. V našej práci zameranej na analýzu prúdu dát chápeme spracovanie v reálnom čase ako metódu kontinuálneho spracovania prúdiacich informácií. Presné časové ohraničenie na spracovanie nie je stanovené, zásadné kritrérium je garancia spracovania všetkých správ, ktoré vstupujú do systému. V nami rozoberanom probléme bude tiež kritické časové ohraničenie v takmer reálnom čase, rádovo desiatky milisekúnd.

V našej práci analyzujeme existujúce metódy a programovacie rámce na spracovanie prúdiacich údajov. Metodologicky a inkrementálne navrhujeme softvérovú architektúru na spracovanie prúdov v takmer-reálnom čase. Pomocou tejto architektúry extrahujeme informácie z prúdu údajov a poskytujeme hodnotné výstupy na základe používateľských dopytov aplikovaním filtrov, agregácií, či zoskupení.

Práca je štrukturovaná do štyroch logických celkov, ktoré predstavujú jadro práce. Prvé dve časti hovoria o analýze problému a existujúcich riešeniach tohto problému. V ďalšej časti hovoríme o návrhu nášeho riešenia. Posledná časť do hĺbky pojednáva o overovaní a experimentoch nad našim návrhom riešenia. Na konci práce je uvedené zhodnotenie a pohľad do budúcich smerov, zoznam literatúry a prílohy práce.

2. Princípy spracovania údajov

V tejto kapitole sa venujeme metódam a prístupom na spracovanie veľkého objemu informácií/dát (angl. Big Data).

Teorém 2.1 (HACE) *Big Data súbor vzniká z veľko-objemných, heterogénnych, distribuovaných a decentralizovaných autonomných zdrojov, pričom hľadá a skúma komplexné (angl. complex) a vyvyjajúce (angl. evolving) sa vzťahy medzi dátami.*

Na základe uvedených charakteristík Big Data predstavujú extrémnu výzvu v rámci objavovania užitočných znalostí v údajoch (Wu et al., 2014). Dávkové spracovanie je jednou z najčastejšie používaných metód spracovania. Použitie dávkových metód zlyháva v prípadoch, kedy môže byť čas odozvy spracovania kritický a podstatný pre správne fungovanie aplikácie. Môžu to byť napríklad systémy, či aplikácie spracujúce údaje zo senzorov lietadla alebo aplikácia pre analýzu trendov na sociálnej sieti. Z tohto dôvodu v tejto kapitole porovnávame dva prístupy spracovania Big Data, *dávkové spracovanie* (angl. batch processing) a *prúdové spracovanie* (angl. stream processing), pričom druhá metóda hovorí, ako to vyplýva z názvu, o spracovaní prúdu dát.

Je potrebné všeobecne zadefinovať, čo je to prúd údajov. Prúd údajov môže naberať rôzny význam v závilosti od jeho zdroja. Všeobecne je prúd dát potenciálne nekonečný a neohraničený tok udalostí v pohybe. Takýto prúd má tri význačné vlastnosti, veľký *objem* (angl. volume), vysokú *frekvenciu/rýchlosť* (angl. velocity) a *pestrosť* (angl. variety) prúdiacich správ za jednotku času (Kaisler et al., 2013). Pre priblíženie, objem sa pohybuje rádovo v stovkách tisícoch a viac správ za sekundu. Ako príklad môžme uviesť aktivitu na sociálnej sieti Twitter, ktorá presahuje 50 milionóv nových správ (tweetov) za jeden deň (Mathioudakis and Koudas, 2010).

Najčastejšie zdroje takýchto dát, ktoré vyžadujú prúdové spracovanie dát v reálnom čase pre dosiahnutie hodnotných výstupov (Higginbotham, 2010; Kaisler et al., 2013; Toshniwal et al., 2014) sú:

- *sociálne média* ako napr. Twitter, či Facebook produkujú masívny objem dát v reálnom čase, ktoré strácajú svoju informačnú hodnotu veľmi rýchlo.
- *údaje z logovacích súborov*, t.j. webové servery, databázy a rôzne iné zariadenia produkujúce masívne súbory, do ktorých sú ukladané logy.
- *sieťové zariadenia alebo prvky* kontinuálne generujú hodnotné dáta v obrovských objemoch, tiež v závislosti na veľkosti siete.
- *senzory a snímače*, ktoré môžu merať fyzikálne veličiny.

Takéto zdroje dát má zmysel spracovať v reálnom čase, aby sme získali hodnotné výstupy. Spracovanie v reálnom čase sa môže opäť meniť v kontexte problému. V našom prípade to bude najčastejšie znamenať, že správnosť výsledku nebude závislá iba na jeho správnosti, ale aj na čase za aký bude poskytnutý (Stankovic et al., 1999).

Pre dávkové spracovanie sú všetky dáta ukladané tak, ako vznikajú (čisté neštrukturované a neupravené dáta) do dátového úložiska Big Data. Pre získanie hodnotného výstupu z takejto kolekcie dát na základe dopytu používateľa, je nutné vykonať operáciu nad všetkými dostupnými dátami (Marz and Warren, 2013, s. 8-10).

$$dopyt = fn(všetky\ dáta,\ ktoré\ sú\ k\ dispozícii)$$

Pri všeobecnom navrhovaní architektúry softvérového systému na spracovanie Big Data, ktorej úlohou je spracovať veľký objem dát, je nevyhnutné, aby spĺňala aspoň nasledujúce vlastnosti:

1. *robustná a odolná voči chybám*, architektúra je robustná, ak bez chyby odolá nečakaným stavom, ako napríklad výpadok elektrickej energie. Je odolná voči chybám, ak chybné dáta alebo strata dát nemá za následok poškodený výstup.
2. *nízka odozva*, aplikácie postavené na tejto architektúre sú schopné poskytnúť odozvu v požadovanom čase. Typicky rádovo desiatky, maximálne stovky milisekúnd.

3. *horizontálne škálovateľná*, je možné zvýšiť výkon celej architektúry pridaním fyzického uzla bez dopadu na funkcionalitu.

4. *všeobecná*, architektúru je možné použiť na rôzne aplikácie.

5. *rozšíriteľná*, do fungujúcej architektúry je možné pridávať novú funkcionalitu za prijateľnej námahy.

2.1 Dávkové spracovanie dát

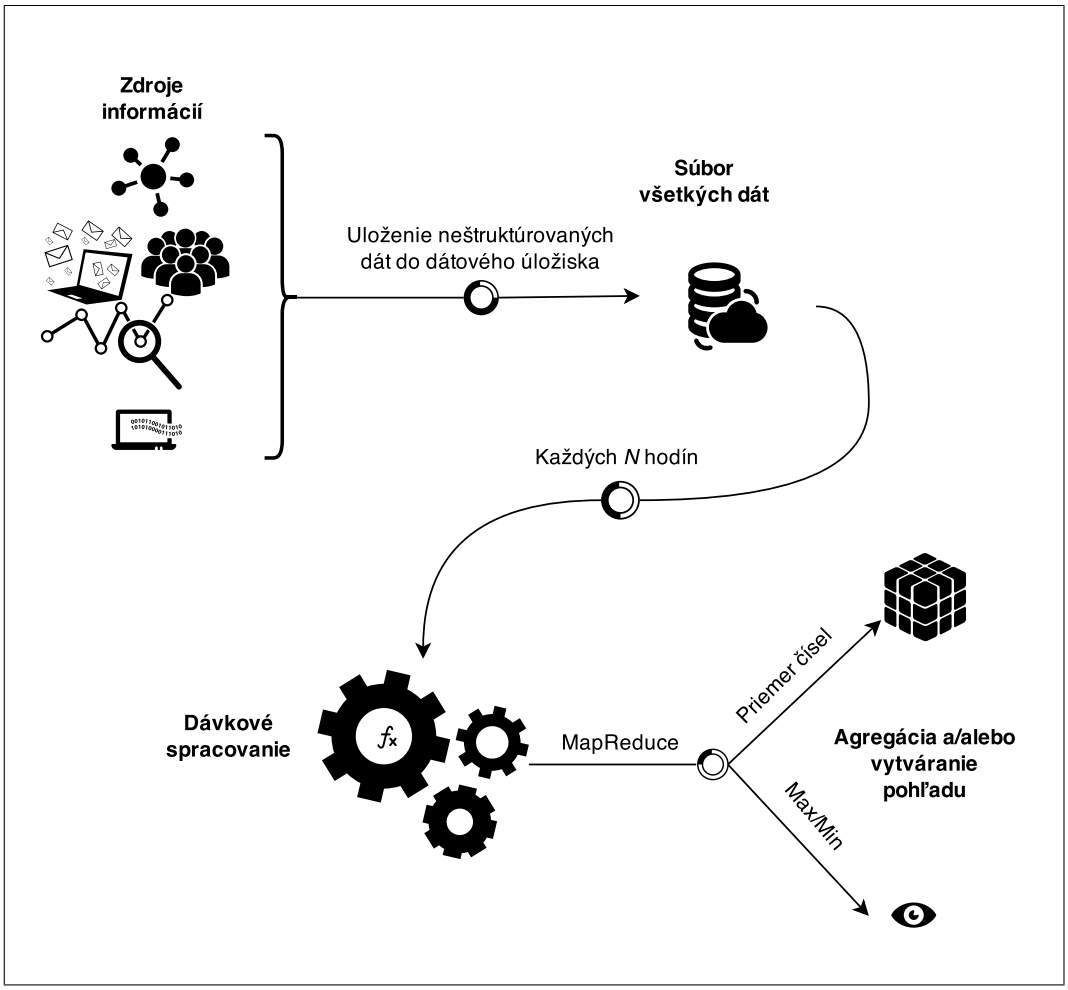
Informácie sú uložené do dátových úložísk neštrukturované alebo len čiastočne štruktúrované a bez úprav, či predspracovania. Dávkové spracovanie je vykonávané nad celou statickou kolekciou dát. Dávky, v ktorých sú dáta spracované, nazývame *práce* (angl. jobs, vychádza z terminológie Hadoop[1] a MapReduce paradigmy). Tieto práce môžu byť ohraničené časovo alebo veľkosťou. Najčastejší prístup predstavuje vytvorenie dávky na spracovanie v časových intervaloch, napr. každých sedem hodín. Znamená to, že počas sedem hodín sú nové dáta akumulované do úložiska. Na takéto uloženie sa najčastejšie používajú NoSQL[2] databázy, pretože tradičné extrahuj-transformuj-načítaj (ETL)[3] relačné databázy sú priveľmi štrukturované a pomalé (Liu et al., 2014). Po uplynutí tohto časového intervalu je nad všetkými dátami, ktoré boli doteraz zaznamenané, vykonaná operácia (napríklad priemer všetkých čísel), a jej výsledok je uložený vo forme *pohľadu*. Pohľady sú informácie alebo znalosti odvodené z celej kolekcie dát, ich vytvorenie asistuje pri vytváraní odpovede na používateľské dopyty do aplikácie (Marz and Warren, 2013, s. 29). Pohľad je často reprezentovaný v agregovanej forme. Agregované dáta nám už často dávajú hodnotné poznanie (priemer všetkých čísel), narozdiel od čistých a neštrukturovaných informácií (Chaudhuri and Dayal, 1997). O niektorých možnostiach agregácie hovoríme ďalej v tejto kapitole.

S dávkovým prístupom bude len veľmi ťažké dosiahnuť nízku odozvu na dopyt používateľa, pretože minimálne v jednom momente budú *agregované dáta* sedem hodín neaktuálne. Tomuto by sa teoreticky dalo zabrániť rapídnym

[1] Apache™ Hadoop®: https://hadoop.apache.org/

[2] NoSQL definícia: Databázy ďalšej generácie adresujúce nasledovné body: nerelačné, distribuované, open-source, horizontálne škálovateľné

[3] extract-transformation-load



Obrázok 2.1: Dávkové spracovanie dát, pričom v pravom dolnom rohu sú vytvárané rôzne pohľady na dáta.

zmenšením časového intervalu dávkového spracovania. Pri tomto prístupe sú všetky dáta ukladané do databázy, tak ako prišli. Nie sú vôbec modifikované, takže sa pri dávkovom spracovaní pracuje s čistými dátami. Súbor dát, ktorý uchovávame môže byť preto veľmi objemný (rádovo terabajty a viac). To znamená, že každé spracovanie dávky je časovo a výpočtovo náročná operácia, ktorá trvá niekoľko minút, dokonca niekoľko hodín. Trvanie tiež závisí od výpočtovej sily hardvéru a veľkosti súboru dát. Je nutné poznamenať, že *práca* dávkového systému je vždy vykonaná nad celým súborom dát, ktorého veľkosť rastie s časom. Napríklad, pri spätnej analýze trendov na sociálnych sietiach alebo systémov na odporúčanie reklám nieje možné vykonávať *práce* z kombinácie agregovaných výsledkov a nových dát, pretože by výsledky mohli byť nepresné a skresľujúce.

Nami načrtnuté riešenie na to, ako sa vyhnúť vysokej odozve dávkového spra-

covania, t.j. zvýšením frekvencie *prác* dávkové systému zlyháva hneď z niekoľkých dôvodov. Keďže sa *práce* vždy vykonávajú nad celou kolekciou dát, čas výpočtu rastie exponenciálne s narastajúcim objemom uložených dát. Tento prístup bude určite vďaka výkonnosti dávkových modelov z počiatku akceptovateľný. Nezabúdajme, že sa bavíme o Big Data kde súbor dát rýchlo narastie do veľkosti terabajtov až petabajtov a viac. S nárastom objemu uložených informácií, ako sme spomenuli, rastie aj odozva dávkového systému, ktorý dáta spracuje. Čo je v rozpore s nami zamýšľaným riešením zníženia odozvy dávkového spracovania.

Dávkové spracovanie dát nám poskytuje distribuované a spoľahlivé (t.j. odolné voči chybám) riešenie, pretože sú všetky údaje ukladané resp. vždy zapisované bez zmeny (pozn.: mazanie je tiež len zápis do databázy). Tento prístup tiež poskytuje škálovateľnosť, je dostatočne všeobecný a rozšíriteľný. Ukladanie všetkých dát tak, ako vznikajú sa môže zdať ako nevýhoda kvôli objemu, avšak dnes je disková kapacita veľmi lacná. Potom sa z ukladania všetkých dát stáva veľká výhoda, pretože dosahujeme odolnosť voči chybám (aj spôsobené človekom) a pri vytváraní *pohľadu* zohľadňujeme všetky údaje z minulosti, pretože všetky udalosti sú zapisované ako nový záznam. Odolnosť voči chybám je zabezpečená aj tým, že výpočet (práca) môže byť spustený od začiatku znova kedykoľvek, pretože ukladáme všetky dáta a zanesenie malej odozvy do dávkového systému za cenu správneho výsledku nemá kritické následky. V prípade dávkového spracovania je dôležitejšie poskytnúť bezchybný výsledok za cenu odozvy. Nasledujúci zoznam vydeľuje najdôležitejšie vlastnosti dávkového spracovania (Marz and Warren, 2013):

1. *neohraničené počítanie* značí že máme teoreticky neobmedzenú výpočtovú silu a neohraničený čas výpočtu. Teoreticky môžeme vykonať dávkové spracovanie nad akýmkoľvek dátovým súborom.

2. *ukladanie normalizovaných dát*, neexistuje potreba dáta denormalizovať, pretože pohľady sú generované často z pohľadu aplikácie.

3. *architektúra je horizontálne škálovateľná.*

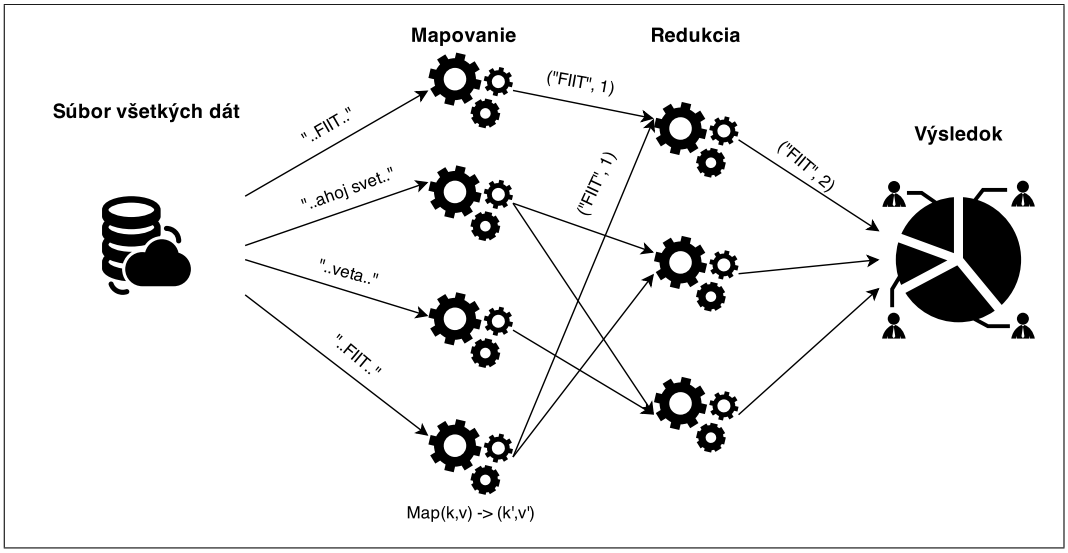
4. *vykonáva len zápis*, žiadne dáta sa nie sú *nikdy* zmazané.

5. *dáta sú konzistentné*, za predpokladu, že vstupné dáta nie sú poškodené.

2.1.1 Metódy dávkového spracovania

MapReduce je programovací model a príslušná implementácia na spracovanie a generovanie masívnych údajových korpusov (Dean and Ghemawat, 2008). Tento model bol vyvinutý na riešenie výpočtovo náročných problémov, ktorých výpočet je potrebné distribuovať a paralelizovať. Model je založený na primitívach prezentovaných vo funkcionálnom programovacom jazyku Lisp, *mapovať* a *redukovať*. Kľúčový prínos MapReduce modelu, alebo tiež programovacieho rámca, nie vo funkciách *mapovať* a *redukovať*, ale v možnosti použitia rôznymi aplikáciami. Zároveň poskytuje škálovateľnosť riešenia a odolnosť voči chybám. Zdrojový kód je jednoduché distribuovať a paralelizovať tak, aby výpočet bežal v *strapci* resp. *klastri* (angl. cluster) výpočtových uzlov zapojených vo fyzickej sieti (napr. internet).

Výpočet je vykonávaný len v týchto dvoch fázach. Funkcia *mapovanie* má ako vstupný argument dvojicu *kľúč/hodnota* s kľúčom K, nad ktorou spraví používateľom definovanú operáciu, a vytvorí dočasnú dvojicu s kľúčom K. Túto hodnotu s kľúčom K ďalej konzumuje funkcia *redukcia*, ktorá najčastejšie spojí hodnoty do jednej (napr. operácia sčítania). Na obrázku 2.2 ilustrujeme jednoduchý problém spočítania výskytov slova vo veľkej kolekcii dokumentov (Dean and Ghemawat, 2008). Ukážka implementácie takéhoto MapReduce modelu je pomocou algoritmu 1.



Obrázok 2.2: Zobrazuje dávkové spracovanie dát s aplikovaním MapReduce modelu pre počítanie slov vo vetách z korpusu dokumentov.

Efektívne implementovaný model, ktorý je vykonávaný na dostatočne veľkej farme serverov dokáže za niekoľko hodín usporiadať rádovo petabajty dát. Pochopiteľne, problém musí byť možné paralelizovať. Okrem efektívnej implementácie je dôležité zabezpečiť optimalizované distribuovanie dát v *klastri* serverov (výpočtových uzlov). Optimalizácia komunikačných nákladov je základom pre dobrú implementáciu MapReduce modelu (Dean and Ghemawat, 2008).

```
function mapovanie(veta)
    for each slovo in veta do
        emituj(slovo, 1) ;
    end
function redukcia(slovo,pocty)
    //pocty predstavujú agregovaný zoznam neúplných počtov slov;
    suma = 0 ;
    for each pocet in pocty do
        suma += pocet ;
    end
    emituj(slovo, suma);
    //emituje početnosť slova ;
```

Algoritmus 1: Jednoduchá ukážka implementácie MapReduce modelu na počítanie výskytu slov vo vete.

Dátové kocky (angl. Data cubes), tiež známe pod pojmom OLAP[4] kocka, je dátová štruktúra, ktorá umožňuje rýchlu analýzu dát. Pužitie OLAP v dátových skladoch (angl. Data warehouse) je dnes časté v korporátnom biznis svete ako agregačná platforma, analógia pohľadov na dáta v Obrázku 2.1 na strane 5. Hlavná motivácia prečo vytvárať dátove kocky je materializácia všeobecných údajov. Takáto kocka má dva hlavné atribúty - *dimenzia* a *hodnota*. Pričom dimenzie predstavujú nezávislé hodnoty. Závislou hodnotou je napr. vek, pretože bude viazaná k veku v konkrétnom zázname. Viacrozmerný priestor je vymedzený dimenziami kocky (Chaudhuri and Dayal, 1997). Nad dátovými kockami poznáme niekoľko základných operácií, *krájanie* (angl. slicing), *delenie* (angl. dicing), *pivotovanie* (angl. pivoting), *zrolovanie* (angl. roll up), *prechádzanie-nadol* (angl. drill-down) (Franek, 2013).

[4]Online Analytical Processing

Hviezdicová schéma (angl. Star schema) sa používa na modelovanie viacdimenzionálnych dát a vychádza priamo z princípov relačných databáz. V tabuľke je uložených len niekoľko základných faktov a kľúčov, túto tabuľku nazývame *tabuľka faktov*. Kľúče ďalej odkazujú na ďalšie tabuľky (dimenzie), ktoré môžu ešte odkazovať na ďalšie tabuľky (Chaudhuri and Dayal, 1997) a tak ďalej. Dané odkazované tabuľky sú zdrojom pre fakty. Dopytovanie sa do takejto schémy je pomerne efektívne, zvlášť pri agregovaných dopytoch (napr. priemer čísel alebo operácia group-by), keďže nie je vždy potrebné dotazovanie vo všetkých dimenziách databázy. Dopyt je vždy smerovaný len do *tabuľky faktov*. Nevýhoda hviezdicovej schémy je integrita dát, pretože sa tabuľka faktov aktualizuje iba v istých časoch. Integrita dát nie je vynútená, narozdiel od normalizovaných databáz. Rafinovanejšia verzia hviezdicovej schémy je schéma *snehovej vločky* (angl. Snowflake schema), kde je viac dimenzionálna hierarchia explicitne reprezentovaná normalizáciou dimenzií (Chaudhuri and Dayal, 1997).

2.2 Prúdové spracovanie dát

Definícia 2.2.1 *Prúd je neohraničená sekvencia n-tíc (A1, A2, ..., An, t) generovaná kontinuálne v čase. Ai znamená atribút a t čas.*

Správne fungovanie niektorých aplikácií je priamo závislé na takmer okamžitom výsledku extrahovaného z nekonečného spojitého prúdu dát. Nové dáta prichádzajú kontinuálne, a teda aj výsledky a zmeny z nich vyplývajúce prichádzajú rovnakým spôsobom. Dáta musia byť spracované tak ako prichádzajú. Spracovanie takýchto nekonečných prúdov dát aplikovaním tradičných metód môže predstavovať problém, keďže výpočtové prostriedky sú limitované (Babcock et al., 2002). Množstvo (za rádovo sekundy) vygenerovaných dát je zvyčajne veľké, a preto sú často po spracovaní zahadzované, prípadne môžu byť zrkadlené do hlavného úložiska pre neskoršie dávkové spracovanie historických dát. V prípade keby sme aj chceli pristupovať k dátam z minulosti (užitočné pri detekcii trendov) zvýšenú výpočtovú náročnosť, pretože dopyt byť vykonaný nad všetkými doteraz uloženými dátami (dávkové spracovanie) pre získanie jedného výstupu (Silvestri, 2006).

Systémy spracujúce prúdiace dáta sú vytvorené na podporu vznikajúcich aplikácií, pre ktoré je kritické dopytovanie sa prúdov dát často vznikajúcich v *oblačnom* (angl. cloud) prostredí. Takto pracujú systémy spracujúce prúdiace dáta na rozdiel od tradičných prístupov k spracovaniu a ukladaniu dát, kde je kritické, aby boli všetky dáta doručené a uložené bezchybne, aj za cenu výkonnosti aplikácie. Niektoré aplikácie, ktoré pracujú s veľkým prúdom tolerujú malú stratu, či chybovosť dát v prospech výkonnosti spracovania. Tieto aplikácie môžu byť systémy sledujúce stav siete, analýza webovej prevádzky alebo monitorovanie senzorov. Pre špecifické problémové oblasti (napr. spracovanie dát zo senzorov) môže byť tiež kritická garancia bezchybnosti a doručenia dát. V tejto časti práce sa venujeme modelu prúdového spracovania údajov jeho špecifikám, prístupom a problémom. Aplikácie, ktoré sú prúdovo orientované sa nepozerajú na prichádzajúce dáta ako na statickú kolekciu dát, ale ako na neohraničený, potenciálne nekonečný prúd alebo sekvenciu udalostí a dát. Preto sa zameriavame na prúdové spracovanie, ktoré spĺňa nasledujúce požiadavky (Stonebraker et al., 2005):

1. *nechať dáta prúdiť*, na dosiahnutie nízkej odozvy resp. spracovania v reálnom čase, systém musí byť schopný spracovať správu bez nákladnej operácie zápisu na disk, takže výpočet prebieha v hlavnej pamäti počítača.

2. *dopytovanie sa do prúdu*, pre prúdové aplikácie musí byť prístupný nejaký dopytovací mechanizmus na získanie výstupov z prúdu udalostí.

3. *spoľahlivosť*, to znamená, že model musí byť odolný voči chybám a garantovať doručenie *všetkých* správ, tiež musí byť schopný spracovať oneskorené, chybné, chýbajúce údaje, či údaje, ktoré prišli v zlom poradí.

4. *integrácia ukladaných a prúdiacich údajov*, pre mnohé aplikácie je potrebné "pozerať"sa na dáta z minuloti pre spätnú analýzu v kontexte nových informácií, preto je potrebné, aby systém poskytoval možnosť zrkadlenia prúdiacich údajov do dátového úložiska (pozn.: často používaný Hadoop opísaný v kap. 3.1.1).

5. *rýchlosť*, spracovanie neohraničeného prúdu *udalostí* v takmer-reálnom čase, výsledky na dopyty musia byť poskytnuté okamžite.

6. *dostupnosť*, model musí byť horizontálne škálovateľný a odolný voči výpadku výpočtového uzla.

2.2.1 Vlastnosti prúdu dát

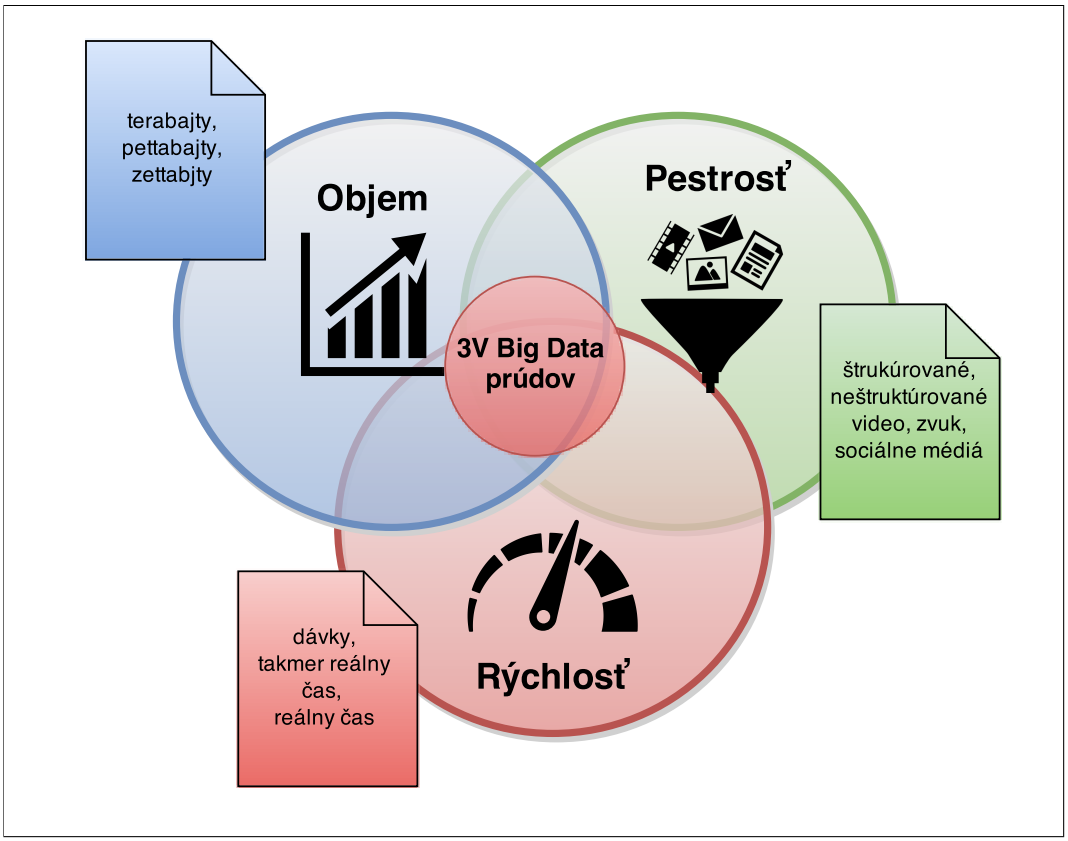
Prúd alebo tiež sekvencia udalostí a údajov je potenciálne nekonečný prúd udalostí. Hlavné aspekty prúdu dát sú často opisované ako 3V (Zikopoulos et al., 2011; Zaslavsky et al., 2013):

1. *Objem* (angl. Volume) dát vznikajúcich každý deň sa v súčasnosti blíži rádovo k terabajtom nových dát (napr. Twitter vyprodukuje každý deň až 7 terabajtov nových informácií (Mathioudakis and Koudas, 2010)). O ďalších konkrétnych číslach nemá veľmi zmysel hovoriť, pretože už v čase písania tejto práce sa spomenutý objem mení, pričom s veľkou pravdepodobnosťou narastá. Existuje predpoklad, že v roku 2020 budú dnešné internetové služby, produkovať denne rádovo zettabajty[5] (sextillion) dát.

2. *Pestrosť* (angl. Variety) prúdiacich dát môže byť zásadná. Dáta môžu prichádzať zo zdrojov často neúplné, či poškodené, dokonca nemusia prísť v tom poradí v akom vznikli. Dáta môžu mať rôznu štruktúru (napr. JSON, video, zvuk, atď.) a taktiež prúdiť z odlišných zdrojov. Zdrojom sme sa podrobnejšie venovali v úvode kapitoly 2.

3. *Rýchlosť* (angl. Velocity) alebo tiež frekvencia znamená ako rýchlo a často udalosti vznikajú (napr. každú milisekundu, sekundu alebo deň, či mesiac). Väčšinou predstavuje frekvencia rádovo milióny udalostí generovaných za každú sekundu. Pre lepšie pochopenie tohto aspektu je dobré uviesť pojem dát v pohybe, čo znamená rýchlosť akou prúdia dáta. Frekvencia spracovania sa môže líšiť v závislosti od požiadaviek na aplikáciu. Najčastejšie je však vyžadované spracovanie v takmer reálnom čase. Všeobecne vymedzujeme tri hlavné kategórie frekvencie spracovania: *občasné* (hodí sa dávkový prístup), *časté* a spracovanie v *reálnom čase*.

[5] Zettabyte: http://en.wikipedia.org/wiki/Zettabyte



Obrázok 2.3: Vzťah medzi jednotlivými vlastnosťami Big Data

Niektorí výskumníci považujú *význam/hodnotu* (angl. Value) za ďalší a dokonca hlavný aspekt Big Data vo všeobecnosti. Často to znamená, že vo veľkej statickej kolekcii dát alebo širokom prúde existuje významný kus poznania, nazývaný tiež *zlatý úsek* dát (Zaslavsky et al., 2013), ktorý je najdôležitejší pre poznanie. Ostatné iformácie sa môžu javiť ako bezcenné voči tejto informácii. S integráciou rozličných dátových systémov vzniká piata vlastnosť - *pravdivosť* (angl. Veracity) alebo tiež *správnosť* a *presnosť* informácie. Rôzne zdroje dát často poskytujú dáta v odlišnej kvalite, presnosti a aktuálnosti. Toto tvrdenie súhlasí s pozorovaním, že *"jeden z troch biznis lídrov neverí informáciam, na základe ktorých robí rozhodnutie"* (Dong and Srivastava, 2013). Je potrebné odlíšiť pravdivé informácie, ako dobrý príklad môžeme uviesť sociálne siete, kde sa často šíria nepravdivé, tzv. hoax správy. Opísané vlastnosti sú vo všeobecnosti pripisované Big Data, my ich spomíname pri prúdovom spracovaní a to práve preto, že najviac odzrkadľujú vlastnosti prúdiacich dát. Ide najmä o štvrtú vlastnosť t. j. *hodnotu*, ktorá môže byť vo veľkej statickej kolekcii ľahko prehliadnutá, ak zmena voči ostatným dátam nie je signifikantná.

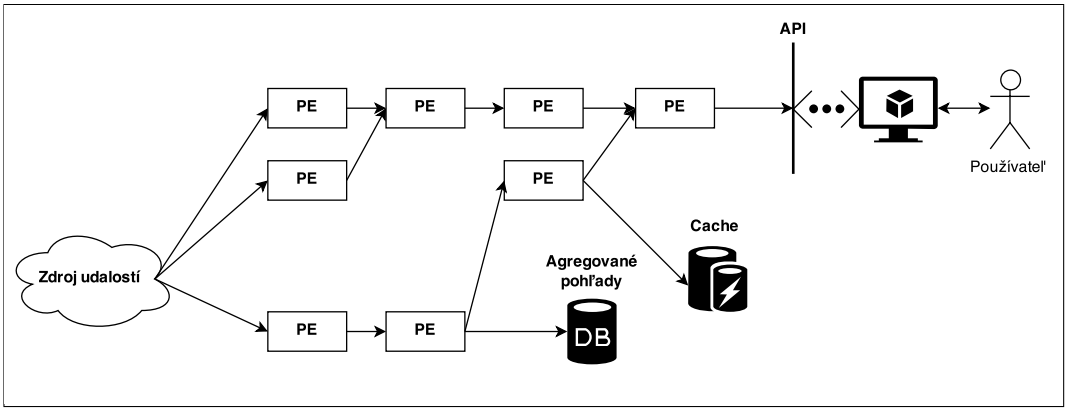
2.2.2 Model prúdového spracovania

Prúdove spracovanie je vykonávané "za behu", to znamená predtým, než sú dáta kamkoľvek uložené. Toto viditeľný rozdiel oproti dávkovému, či tradičnému (napr. relačné databázy) spracovaniu.

Spracovanie prebieha najmä v hlavnej pamäti výpočtového uzla, a kvôli požiadavke na rýchlosť systému nie sú vykonávané zápisy na disk. To však nevylučuje možnosť vykonávať dopyty do tradičných relačných databáz počas spracovania prúdov. Často sú takto spájané (pozn.: paralela známej operácie *join* z relačných SQL databáz) relačné dáta a prúdiace informácie pre získanie lepšieho výsledku (Babcock et al., 2002). Výpočtové prvky prúdového systému sú obvykle nazývané elementy (angl. processing element, ďalej len *PE*), pričom každý element má vstupnú a výstupnú frontu správ. Keď je na vstupe nejaká n-tica (napr. správa, informácia alebo udalosť), PE vykoná definovanú operáciu nad n-ticou a výsledok posiela do výstupnej fronty. N-tice tvoriace prúd najčastejšie predstavujú prostú operáciu generujúcej aplikácie, napríklad stlačenie tlačidla *odoslať* alebo *návšteva* stránky.

Výpočtové elementy tvoria logickú sieť uzlov zapojených do orientovaného acyklického grafu (skratka *DAG* z angl. directed acyclic graph), ktorý vytvára prúdový systém. Výpočtové elementy predstavujú uzly grafu, v ktorom vstup a výstup uzlov sú hrany grafu. Tento graf ilustrujeme na obrázku 2.4. Prúd tečie cez uzly grafu, ktoré vykonajú špecifickú operáciu nad n-ticami a ďalej emitujú výsledok vo forme n-tíc. Nie je podmienka, že s každou n-ticou musí uzol emitovať n-ticu do výstupnej fronty. Niektoré uzly môžu emitovať výsledok každých N sekúnd v zhluku n-tíc alebo v agregovanej forme. Celé to prebieha paralelne a na viacerých fyzických strojoch v klastri serverovej farmy.

Tento model sa vyvinul z architektonického štýlu *dátovody a filtre*. Aplikuje inkrementálne (aproximačné) algoritmy. Problém tohto modelu je, že neberie do úvahy údaje z minulosti, čize spracúva len novovznikajúce dáta. Samozrejme, niekedy to môže byť nevýhodou, závisí to od povahy aplikácie.



Obrázok 2.4: Ukážka DAG, kde PE sú procesné uzly grafy a hrany predstavujú vstup a výstup PE. Na pravej strane sú zobrazené možnosti ďalšieho spracovania čiastočných výsledkov alebo celkových výsledkov, napríklad agregácia do relačnej databázy vo forme pohľadov, ukladanie do vyrovnávacej pamäte alebo priamo komunikácia cez API s klientskou aplikáciou.

2.2.3 Dopytovanie sa do prúdu dát

Dopytovanie sa do spojitého prúdu dát má veľa spoločných čŕt s dopytovaním sa do tradičných relačných databáz. Existujú tu ale dva hlavné rozdiely v type dopytov (Terry et al., 1992):

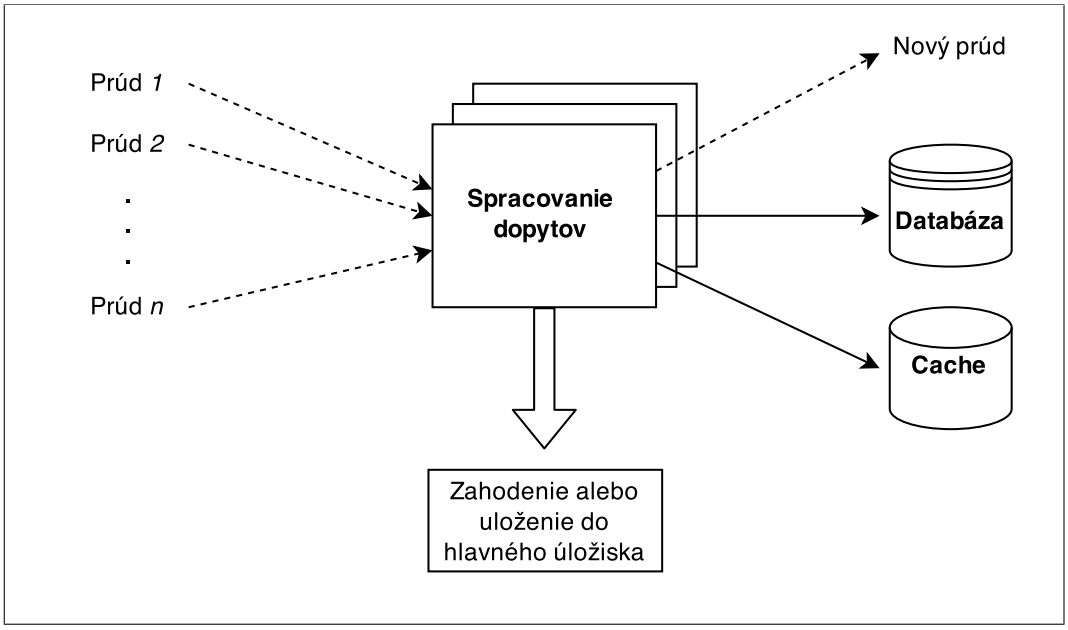
1. *jednorázove* dopyty a *kontinuálne* dopyty,
2. *vopred definované* dopyty a dopyty *pripravené pre špecifický účel* (angl. ad-hoc queries).

Jednorázový dopyt je vykonaný nad istou množinou dát z prúdu a následne je vrátený výsledok. Táto množina je najčastejšie ohraničená časovým intervalom. Kontinuálne dopyty sú vyhodnotené v takom poradí, ako prúdia dáta, a preto majú väčší zmysel kontinuálne dopyty pri dopytovaní sa do prúdu údajov. Výsledok je nepretržite produkovaný v čase a reflektuje prúdiace údaje, taktiež môže predstavovať jednoduchú hodnotu (napr. priemer čísel), alebo môže vznikať nový prúd údajov. Tieto výsledky môžu byť ďalej spracované, alebo ukladané a agregované do databázy. Nové prúdy vznikajú obvykle pri spájaní dvoch a viacerých prúdov (pozn. analógia operácie SQL join), pretože výsledky sú produkované vysokou rýchlosťou.

Vopred definované dopyty sú vo všeobecnosti kontinuálne dopyty, ktoré sú de-

finované pred vznikom prúdu údajov. Aj jednorázové dopyty môžu byť vopred definované. Dopyty pripravené pre špecifický účel vznikajú až potom keď začnú dáta tiecť. Môžu byť jednorázové alebo kontinuálne. Znamená to, že tieto dopyty vytvárajú používatelia interakciou so systémom alebo klientskou aplikáciou. Podpora ad hoc dopytov komplikuje návrh prúdového modelu, pretože dopyty nie sú vopred známe je ťažké ich optimalizovať. Zjednodušená a všebecná architektúra na spracovanie dopytov je zobrazená na obrázku 2.5.

Je potrebné efektívne spracovať rádovo viac ako tisíce dopytov do prúdu úda-



Obrázok 2.5: Zjednodušená všeobecná architektúra pre spracovanie kontinuálnych dopytov do prúdu údajov. Správy, ktoré nespĺňajú dopyt sú zahodené alebo môžu byť materializované v hlavnom úložisku, kde sú ukladané všetky čisté dáta pre neskoršie dávkové spracovanie (Babu and Widom, 2001).

jov. Aby sme túto požiadavku zabezpečili je potrebné čeliť viacerým výzvam. Jednou z nich je spracovanie dát v hlavnej pamäti výpočtového uzla, pretože kvôli zabezpečeniu rýchlej odozvy systému nie je efektívne zapisovať dáta na disk počas ich spracovania.

2.2.4 Problémy a výzvy dolovania znalostí v prúdoch

Medzi časté výzvy a problémy spracovania prúdiacich údajov patria nasledujúce témy (Aggarwal, 2007; Hwang et al., 2005):

- *klastrovanie* v kontexte dátových prúdov môže byť žiadúce pre špecificky definované časti súboru dát namiesto celého súboru.
- *klasifikácia* prúdu údajov a požitie príslušných algoritmov.
- *detekcia zmien* alebo *vzorov* je často žiadúca a tiež stým spojená analýza povahy zmien a vzorov v čase.
- operácie počas definovaného *časového okna* nad prúdiacimi informáciami.
- *spájanie* si vyžaduje veľkú pozornosť, pretože môže spôsobovať úzke hrdlo systémov spracujúcich prúdy dát.
- *indexovanie*,
- *distribuované dolovanie* v prúdiacich údajoch.
- *filtrovanie* prúdiacich dát.

3. Existujúce riešenia modelov spracovania údajov

V tejto kapitole bližšie rozoberieme existujúce riešenia, ktoré sme analyzovali. Zamerali sme sa najmä na riešenia, ktoré nie sú proprietárne. Existujúce riešenia je možné rozdeliť do dvoch skupín a to, dávkové a prúdove spracovanie údajov.

3.1 Dávkové spracovanie

Analyzovali sme riešenia dávkového spracovania veľkého objemu údajov (angl. Big Data). Nástroje aplikujúce princípy dávkového spracovania pracujú väčšinou so statickou kolekciou dát. Tieto nástroje poskytujú možnosti spracovania a analýzy masívnych objemov dát, niektoré nástroje dovoľujú spracovať chybné a neúplné údaje. Spracovanie nie je v reálnom čase, výpočet je vykonávaný periodicky a trvá v závislosti na objeme korpusu rádovo sekundy, minúty, či hodiny a viac.

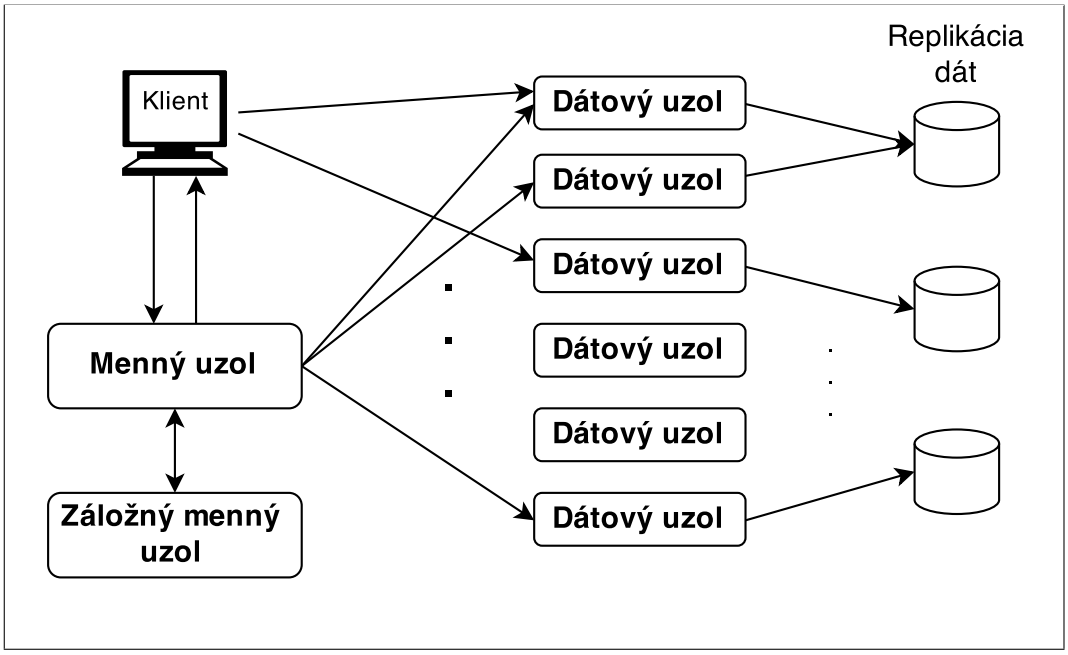
3.1.1 Hadoop

Apache Hadoop[1] je populárna open source implementácia MapReduce programovacieho modelu. Hadoop pozostáva z dvoch základných komponentov: Distribuovaný Súborový Systém Hadoop *HDFS* (angl. Hadoop Distributed File System) a *MapReduce* programovacieho rámca (Liu et al., 2014). Obidva komponenty sú postavené na architektúre *otrokár-otrok* (angl. master-slave). Hadoop program, alebo klient, odosiela *prácu* (angl. job) cez *stopovač práce* (angl. jobtracker), ktorý beží na serveri, do MapReduce rámca. Stopovač práce

[1] Apache Hadoop: https://hadoop.apache.org/

priraďuje úlohy *stopovačovi úloh* (angl. tasktracker), ktorý beží na mnohých slave uzloch. Stopovač úloh posiela pravidelné *správy*[2] informujúce stopovač práce o statuse jeho úloh, napríklad *čakajúca*, *činná*, *zaneprázdnená*, atď. Ak vykonávaná úlohá zlyhá, vyprší časový limit alebo uzol, na ktorom je úloha vykonávaná, bude nejakých príčin bude vypnutý, stopovač úloh môže automaticky preplánovať úlohu na iné živé uzly v klastri.

Ďalší komponent Hadoop HDFS (obr. 3.1) pozostáva z jedného menného uzla (angl. namenode) a viacerých *dátových uzlov* (angl. datanode) Namenode udržuje metadáta dát udržovaných v dátových uzloch. Ak chce klientská aplikácia zapisovať alebo čítať z HDFS musí najprv komunikovať s namenode odkiaľ získa pozíciu dátového bloku, z ktorého môže čítať alebo zapisovať. Metadáta sú načítané do hlavnej pamäti pri štarte Hadoop-u. Dátové uzly, podobne ako stopovač úloh, aktualizujú stav menného uzla.



Obrázok 3.1: Hadoop distribuovaný súborový systém (HDFS) s replikáciou dát a klientom, ktorý číta niektoré dátové uzly.

Hadoop MapReduce rámec je navrhnutý s cieľom dosiahnuť horizontálnu škálovateľnosť a odolnosť voči chybám, ale nie je optimalizovaný pre rýchle vstupno/výstupné operácie. Odolnosť voči chybám je zabezpečená správnou implementáciou modelu MapReduce, o čom sme viac hovorili v podkapitole 2.1.1.

[2]Často nazývané údery srdca (angl. heartbeats) sú správy, ktoré informujú master uzol a stave slave uzlov a úloh.

Problém MapReduce rámca je chýbajúce plánovanie vykonávania kvôli optimalizácií výpočtu a komunikácie v strapci. Aj v prípade efektívneho plánovania úloh a komunikácie naprieč strapcom je stále Hadoop systémom dávkového spracovania údajov kvôli jeho povahe spracovania dát.

3.1.2 EasyBatch

EasyBatch[3] je Java programovací rámec zameraný na zjednodušenie dávkového spracovania dát. Hlavný cieľ je zjednodušiť často používané operácie nad dátami ako napríklad čítanie, filtrovanie, parsovanie a validácia vstupných dát. EasyBatch môže bežať samostane alebo spustený na aplikačnom serveri. Rámec má niekoľko hlavných vlastností: *jednoduchosť* (pozn.: jadro nemá žiadne závislosti, a preto zaberá málo miesta na disku 80kB), *POJO*[4] orientovaný vývoj, *deklaratívna validácia dát*, *generovanie reportov*, *monitoring* dávkového spracovania, *paralelné spracovanie*. Tento rámec je síce vytvorený na dávkové spracovanie dát, ale nie je horizontálne škálovateľný, aspoň nie v jeho podstate bez pridania externých programov a ďalšieho úsilia, narozdiel od Hadoop-u.

3.1.3 Misco

Distribuovaný programovací rámec Misco[5] (Dou et al., 2010) je určený pre mobilné zariadenia. Rámec je implementovaný v programovacom jazyku Python vďaka čomu by ho malo byť jednoduché portovať na všetky platformy, ktoré podporujú tento jazyk a jeho knižnice na sieťovú komunikáciu. Je založený na MapReduce modeli, je potrebné programovať len funkcie Map a Reduce, o ostatné sa postará rámec, t. j. distribúcia kódu a dát, paralelné spracovanie, plánovanie a zlyhania uzlov. Misco systém pozostáva z hlavného serveru a viacerých klientských uzlov, na ktorých prebieha výpočet. Hlavný server zodpovedá za sledovanie stavu klientských uzlov, distribúciu dát, plánovanie a priraďovanie úloh. Klientské uzly sú, naopak, zodpovedné za to, aby si dopytovali úlohy na spracovanie od serveru vykonaním operácie nad dátami a vratením výsledku serveru.

[3]EasyBatch: http://www.easybatch.org/
[4]Plain old Java Object je nijak obmedzený obyčajný Java objekt
[5]Misco: http://alumni.cs.ucr.edu/ jdou/misco/

3.1.4 Disco

Disco[6] je implementácia MapReduce modelu pre distribuované dávkové spracovanie. Disco podporuje paralelné spracovanie veľkých dátových korpusov. Jadro je napísané vo funkcionálnom jazyku Erlang, ktorý bol vytvorený na budovanie robustných distribuovaných aplikácií odolných voči chybám. Disco je postavený na otrokár-otrok (angl. master-slave) architektúre. *Otrokár* (master) prijíma *práce* (angl. jobs), ktoré distribuuje na vykonanie naprieč klastrom uzlov. *Klienti* odosielajú *práce* do hlavného uzla (otrokár). Podradené uzly (otrok) hlavnému uzlu sú ním tiež riadené a monitorované. V klastri Disco môže existovať viac hlavných uzlov.

3.2 Prúdové spracovanie

Prúdové spracovanie v dnešnej dobe získava stále vyššiu pozornosť a záujem. Keďže sa v našej práci venujeme najmä prúdovému spracovaniu, v tejto časti analyzujeme viacero nástrojov vytvorených na spracovanie prúdu údajov.

3.2.1 S4

Skratka S4 znamená *Jednoduchý Škálovateľný Prúdový Systém* (angl. Simple Scalable Streaming System, S4). Tento systém je založený sčasti na modeli MapReduce. S4 je všeobecná, distribuovaná a škálovateľná platforma, ktorá je *čiastočne* odolná voči chybám. Napríklad, ak spracujúci uzol v topológií zlyhá kvôli chybe, spracovanie je automaticky presunuté na iný uzol, ale stav pôvodného uzla je stratený, a nemôže byť obnovený (Neumeyer et al., 2010).

Tento systém nepoužíva zdieľanú pamäť a je založený na modeli Hráčov (angl. Actor model) v kombinácií so spomenutým modelom MapReduce. S4 má decentralizovanú symetrickú architektúru, v ktorej sú všetky uzly na rovnakej úrovni (rozdiel oproti otrokár-otrok architektúre). Tieto uzly sú nazývané *spracujúce elementy* (angl. processing elements, ďalej len PE). S4 si vymieňa informácie medzi PEs vo forme dátových udalostí, čo je aj jediná možnosť interakcie

[6]Disco: http://discoproject.org/

a výmeny informácií medzi PEs. Klaster (angl. cluster) S4 pozostáva z PEs na spracovanie týchto udalostí. Vzhľadom na to, že dáta prúdia medzi PEs a výpočet beží len v hlavnej pamäti PE, nie je potrebné ukladanie na disk. Z toho vyplýva už spomenutá, čiastočná odolnosť voči chybám.

Pretože v systéme neexistuje žiadny hlavný uzol (master), ktorý by sa staral o uzly, kde prebieha výpočet (slave), ale sú všetky uzly na rovnakej úrovni je potrebné zabezpečiť manažment klastra uzlov. Toto je mnohokrát vyriešené platformou ZooKeeper[7]. ZooKeeper je centralizovaná platforma na manažment distribuovaných aplikácií.

3.2.2 Spark

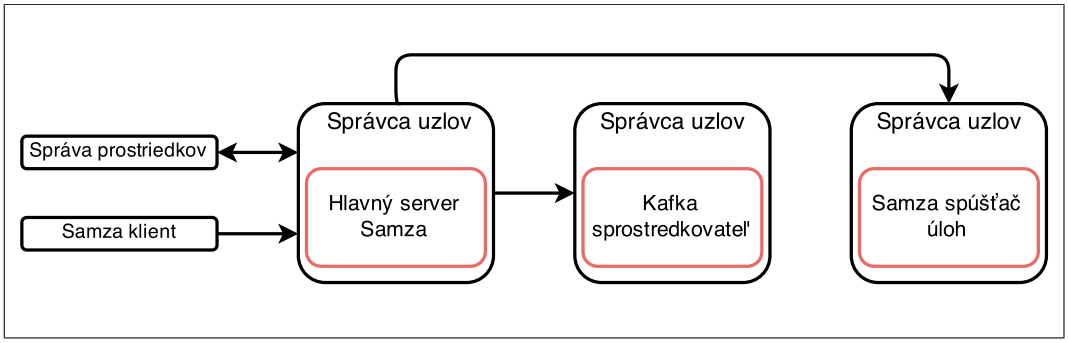
Spark (Liu et al., 2014) je klastrový výpočtový systém. Kombinuje spracovanie uložených dát v dávkovom móde so spracovaním prúdu údajov v reálnom čase. Dávková časť Spark-u je postavená na Hadoop HDFS opísaná podrobnejšie v kapitole 3.1.1 Hadoop. Cieľom Spark-u je poskytnúť rýchlu výpočtovú platformu pre analýzu dát. Spark poskytuje všeobecný model výkonávania ľubovolných dopytov, ktoré sú výkonávané v hlavnej pamäti (pokiaľ ide o prúdové spracovanie). Tento model je nazvaný *Pružný distribuovaný dataset*, skr. RDD (angl. Resilient Distributed Dataset), čo je dátova abstrakcia distribuovanej pamäti. Keďže výpočet beží v hlavnej pamäti (pri prúdovom spracovaní), nie je potrebné vykonávať zápisy na disk, vďaka čomu môže byť dosiahnuté spracovanie v reálnom čase. Výpočet prebieha vo veľkom klastri uzlov s dosiahnutím odolnosti voči chybám za použitia RDD. RDD sídli v hlavnej pamäti, ale môže byť periodicky ukladaný na disk. Vďaka distribuovanej povahe RDD môže byť stratená časť RDD obnovená z pamäti iného uzla. Samotné prúdové spracovanie nie je vykonávané správa po správe (angl. message by message), ale v mikro dávkach, ktoré môžu byť automaticky paralelne distrubované v strapci.

[7]ZooKeeper: https://zookeeper.apache.org/

3.2.3 Apache Samza

Apache Samza (Kamburugamuve et al., 2013) je programovací rámec pre prúdové spracovanie vyvinutý firmou LinkedIn[8]. Prúd správ je neohraničená nemenná kolekcia správ rovnakého typu. Prúd môže byť čítaný viacerými konzumetami v systéme, a správy môžu byť pridávané alebo odstraňované z prúdu. Prúdy sú vždy udržiavané na úrovni *dohadzovačov* (angl. broker). Práca (angl. job) je logická kolekcia výpočtových jednotiek alebo uzlov, ktoré spracujú prúd a vytvárajú výstupný prúd dát. Táto kolekcia vytvára sieťovú topológiu, ktorá spracuje správy. Prúdy su rozdeľované do pod prúdov, ktoré sú distribuované a spracované paralelne. Úloha predstavuje PE (processing element) v Samza rámci. Jedna úloha môže emitovať jeden výstupný prúd, ale na vstupe môže byť viacero vstupných prúdov.

Distribuované plánovanie a prideľovanie prostriedkov sa spolieha na Apache Yarn[9], o distribuované posielanie údajov sa stará Apache Kafka[10]. Architektúra Samza je zobrazená na obrázku 3.2. Klaster Samza uzlov je manažovaný Yarn-om. Systém je optimalizovaný pre spracovanie veľkého objemu správ a poskytuje možnosť ukladať správy na disk pre dávkové spracovanie. Samza využíva rozdeľovanie správ podľa tém (topic) systému Kafka na dosiahnutie distribuovaného rozdelenia prúdu v strapci. Uzol v Samza strapci môže konzumovať správy z Kafka komunikačného systému správ, ale tiež produkovať nový prúd správ do Kafka sprostredkovateľov.



Obrázok 3.2: Samza architektúra postavená na Apache Yarn a Apache Kafka, zdroj: (Kamburugamuve et al., 2013).

[8] LinkedIn: https://www.linkedin.com/

[9] Apache Hadoop NextGen MapReduce (YARN): http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[10] Apache Kafka je publish-subscribe distribuovaný systém správ: http://kafka.apache.org/

Samza poskytuje garanciu a odolnosť voči chybám a doručenia správ *aspoň raz* (angl. at least once). Ak nejaký uzol zlyhá, nastane preplánovanie úlohy do iného uzla, ktorý prevezme čítanie správ danej témy od sprostredkovateľa (sprostedkovateľ si udržuje zadefinovaný časový úsek správ). Keďže táto odolnosť voči chybám je dosiahnutá prostredníctvom Kafka sprostredkovateľov, ak nastane výpadok alebo chyba v sprostredkovateľoch, Samza stratí správy, ktoré nemôžu byť obnovené. Replikáciou súborového systému sprostredkovateľov je možné sa tomuto vyhnúť.

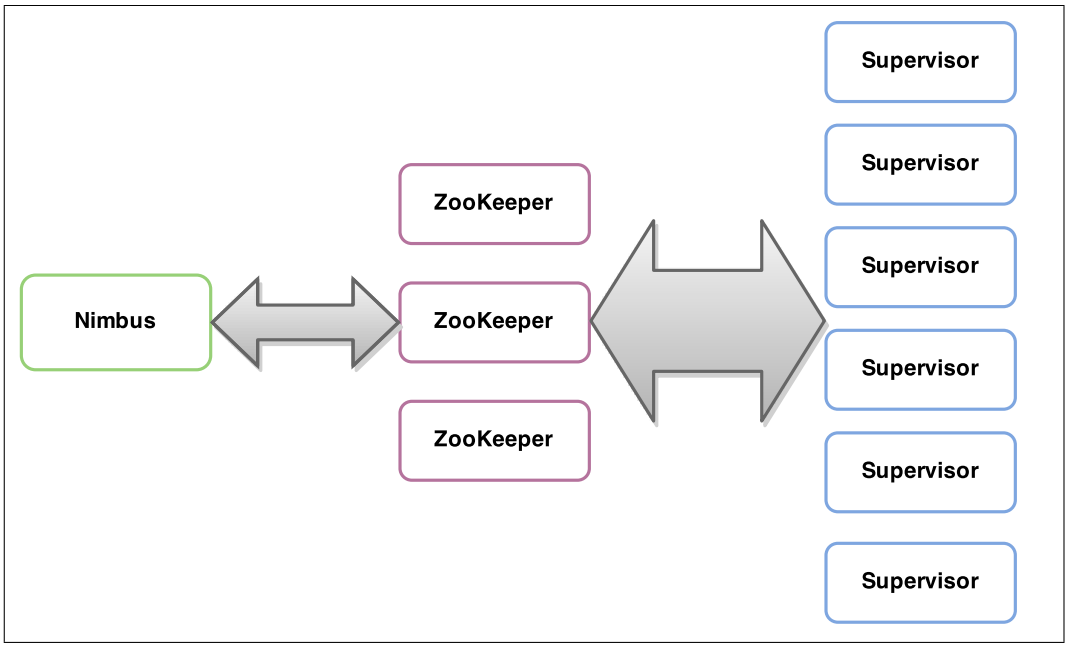
3.2.4 Apache Storm

Apache Strom[11] (Liu et al., 2014; Kamburugamuve et al., 2013) je programovací rámec vytvorený na spracovanie prúdu dát. Je to open-source riešenie, ktoré poskytuje spracovanie prúdu dát s nízkou odozvou. Storm je navrhnutý aby poskytol odolnosť voči chybám (garancia doručenia správ), robustnosť a škálovateľnosť.

Architektúra Storm-u zobrazená na obrázku 3.3 pozostáva z viacerých častí, koordinátora uzlov klastra ZooKeeper, manažérov stavov Nimbus[12] a spracovávajúcich uzlov Supervisor[13]. Nimbus je hlavný server, kde sa nahráva klientský kód na vykonanie. Tento kód je Nimbus-om distribuovaný v rámci *pracovníkov* (workers) v strapci. Nimbus si tiež udržuje status všetkých pracovníkov, takže sa stará o prípadné reštarty alebo presun *úloh* (task) v prípade zlyhania niektorého pracovníka. Množina pracovníkov v klastri Storm-u beží na pozadí ako démon Supervisor-a. ZooKeeper sa stará o koordináciu medzi uzlami Supervisor-a a hlavného servera Nimbus. Tok správ v systéme je zabezpečený komunikačným systémom správ ZeroMQ[14].

Storm nie je prispôsobený žiadnemu konkrétnemu programovaciemu modelu (napríklad MapReduce model). Programovací model Storm-u postkytuje distribuované rozdelenie prúdu medzi uzly. Tento model pozostáva zo *skrutiek* (angl. bolt) a *prameňov* (angl. spout), *topológií* a *prúdov*. Skrutky a pramene sú logicky spojené v orientovanom acyklickom grafe (skr. DAG, angl. directed acyclic graph). Vytvárajú tak určitú sieť uzlov nazývanú topológia. Používateľ

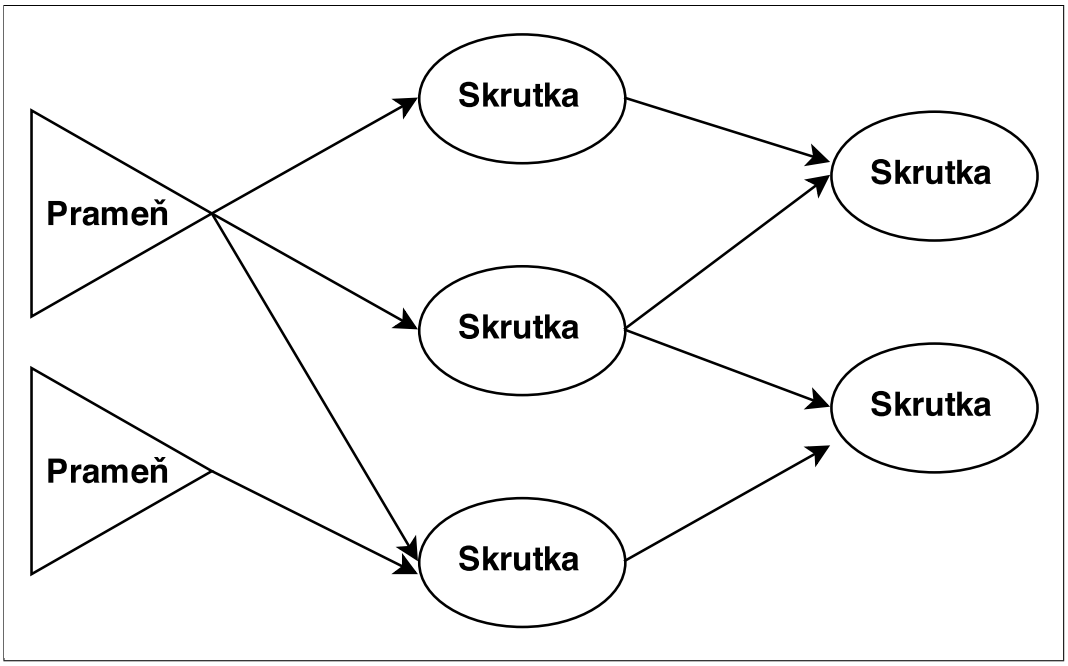
[11] Apache Storm: https://storm.apache.org/
[12] Nimbus: http://www.nimbusproject.org/
[13] Supervisor: http://supervisord.org/
[14] ZeroMQ: http://zeromq.org/



Obrázok 3.3: Zjednodušná architektúra Storm-u a jej komponenty, zdroj: https://storm.apache.org/documentation/Tutorial.html

odosiela vytvorené topológie hlavnému serveru klastra Storm, ktoré sa majú vykonať v klastri Storm. Abstrakcia nad prúdiacimi údajmi sa v skratke nazýva prúd, čo je neohraničená sekvencia n-tíc. Prúdy môžu byť definované používateľom (udalosti z Twittru) alebo systémové prúdy (hodinový signál). Skrutky konzumujú údaje a môžu ďalej emitovať nové prúdy po spracovaní n-tice na vstupe. Pramene len produkujú správy vo forme prúdu n-tíc. Topológia Storm-u väčšinou začína niekoľkými prameňmi a ďalej nasledujú len skrutky, ktoré spracujú prúd. Práca (job), ktorú poznáme z dávkových systémov, v Storm-e predstavuje Java objekt topológie, ktorý je odoslaný do hlavného servera Nimbus. Nimbus sa už postará o distribúciu a vykonanie topológie v rámci klastra. Na obrázku 3.4 je ukážka jednoduchej topológie postavenej na programovacom rámci Storm.

Kód skrutiek môže byť vykonávaný vo viacerých úlohách pracovníka paralelne. Skrutky navzájom prepojené sú tiež vykonávané paralelne. Takto logicky prepojené skrutky si medzi sebou posielajú správy. Tieto správy musia prísť vždy do správnej úlohy v správnom vlákne, preto Storm zavádza pravidlá pre smerovanie správ medzi boltami. Presnejšie ide o to, ako budú správy zoskupované naprieč hranou medzi dvoma uzlami v grafe topológie. Správy môžu byť napríklad *rovnomerne miešané* (angl. shuffle) medzi úlohami, vtedy väčšinou nezáleží na tom, ktorá skrutka spracuje ktorú správu. Niekedy je potrebné aby



Obrázok 3.4: Ukážka topológie Storm-u

len konkrétna úloha vždy spracovala všetky správy obsahujúce nejaký atribút, alebo je žiadané replikovať správy do všetkých úloh. O toto sa tiež stará Storm implementáciou niekoľkých možností zoskupovania správ (angl. grouping) naprieč medzi uzlami.

Odolnosť voči chybám je v Storm-e zabezpečená tak, že si každý uzol grafu drží výstupnú frontu správ. Správy vo fronte zostávajú, pokiaľ nie sú potvrdené (angl. acknowledged). Uzol ktorý správu spracuje ju potvrdzuje, po úspešnom spracovaní. V prípade, ak nepríde potvrdenie na správu, ktorá bola emitovaná do nejakého definovaného času, uzol preposiela správu znova. Tento mechanizmus je ohraničený len hlavnou pamäťou uzla, keďže Storm beží v hlavnej pamäti a posktyuje garanciu doručenia *aspoň raz*.

Zlyhania uzlov monitoruje hlavný server Nimbus, ktorý sa tiež stará o ich riešnie. Uzly supervisor-a posielajú pravidelné správy o svojom stave hlavnému serveru (heartbeats). Zlyhanie uzla a chyba správy sú dve ortogonálne udalosti, chyba alebo zlyhanie správy môže nastať chybou v sieti alebo chybou v softvéri. Manipulácia chybných správ a chybých uzlov preto prebieha úplne nezávisle odlišným spôsobom nezávisle od seba. Tento prístup robí systém ešte viac robustným.

3.3 Zhodnotenie existujúcich riešení

V súčasnosti existuje veľké množstvo nástrojov na spracovanie veľkých objemov údajov (angl. Big Data). Pre dávkové spracovanie Big Data sa dnes používa majoritne Hadoop, ostatné riešenia nie sú tak robustné a často používané ako práve Hadoop. Pre zjednodušenie práce s Hadoop existuje, naviac, mnoho rozšírení resp. úzko súvisiacich projektov (napr. Hive[15], HBase[16], Mahout[17], Apache Pig[18]). Z množstva rôznych dostupných nástrojov na prúdové spracovanie je očividné, že v poslednej dobe zvyšuje vysokú pozornosť práve prúdové spracovanie. Na analýzu všetkých týchto nástrojov v tejto práci nie je priestor.

Všetky nástroje pre prúdové spracovanie sú postavené na istej obdobe otrokár-otrok (angl. master-slave) architektúry. Niektoré sú odolné voči chybám a garantujú doručenie každej správy minimálne jedenkrát. Každé analyzované riešenie bolo distribuované a škálovateľné, čo je správnym predpokladom pre spracovanie veľkoobjemných a rýchlych nekonečných prúdov. Výpočtové uzly v klastri sú spájané do logických celkov a predstavujú orientovaný acyklický graf, niekedy nazývaný topológia (Toshniwal et al., 2014). Jeden nástroj, Apache Spark, je akýmsi hybridom, pretože je kombináciou možnosti pre dávkové aj prúdové spracovanie.

Zo všetkých analyzovaných možností sme sa pre našu prácu rozhodli zvoliť nástroj Storm. Cieľom práce je vytvoriť riešenie na analúzu prúdu údajov, konkrétne filtrovanie podľa používateľského dopytu, a overiť výkonnosť nástroja, identifikovať prípadné problémy a navrhnúť ich riešenie. Storm spĺňa všetky požiadavky na prúdové spracovanie, ktoré sme si na začiatku stanovili a to hlavne, že riešenie musí byť distribuované, odolné voči chybám a horizontálne škálovateľné.

[15]Hive, dátové sklady poskytujúce sumarizáciu dát: http://hive.apache.org/

[16]HBase, podpora štruktúrovaných údajov pre veľké dátové súbory: http://hbase.apache.org/

[17]Mahout, strojové učenia sa a dolovanie dát: http://mahout.apache.org/

[18]Apache Pig, vysoko úrovňový jazyk pre paralelné počítanie: http://pig.apache.org/

4. Návrh riešenia

V tejto práci navrhujeme metódu na analýzu prúdu dát. Presnejšie sa zameriavame na filtrovanie prichádzajúceho prúdu údajov na základe používateľského dopytu. Dáta prúdia zo zdroja v reálnom čase vysokou rýchlosťou a vo veľkom objeme. Naša metóda spracuje prúd údajov v reálnom čase a poskytytuje používateľovi hodnotný výstup na základe jeho dopytu do prúdu. Metódu navrhujeme pre doménu sociálnych sietí, konkrétne pre Twitter. Cieľom metódy je overiť vlastnosti spracovania a filtrovania prúdu správ pomocou zvoleného programovacieho rámca. Iteratívne a metodologicky zdokonaľujeme základnú myšlienku návrhu. Po každej iterácii overujeme niekoľkými experimentami vlastnosti riešenia. Navrhujeme len systém, ktorý spracuje prúd údajov a poskytne výsledok na ďalšie spracovanie. Naopak nenavrhujeme klientskú aplikáciu, ktorá ďalej s výsledkami pracuje. Návrhu a implementácii konkrétnej klientskej aplikácie sa v tejto práci nevenuje, tento krok bude predmetom ďalšej práce.

4.1 Požiadavky na riešenie

Cieľom je filtrovať prúdiace údaje zo sociálnej siete Twitter na základe používateľského dopytu v reálnom čase. Je potrebné paralelne spracovávať rádovo desaťtisíce dopytov. Udalosti prúdu údajov z Twittru môžu vznikať rádovo v desaťtisícoch až miliónoch udalostí za sekundu (Mathioudakis and Koudas, 2010). Je potrebné všetky tieto udalosti spracovať efektívne a následne poskytnúť odozvu v reálnom čase. Vymedzili sme nasledujúce požiadavky na navrhovaný systém:

- *spracovanie v reálnom čase*, systém musí poskytnúť používateľovi výsledok podľa dopytu v reálnom čase (pozn.: chápanie pojmu reálny čas sme definovali v úvode práce), rádovo maximálne desiatky milisekúnd.

- *horizontálna škálovateľnosť*, riešenie musí byť horizontálne škálovateľné takým spôsobom, aby bolo možné pridávať nové fyzické uzly do klastra za behu, bez akýchkoľvek výpadkov.
- *odolnosť voči chybám* je kritická, žiadne správy z prúdu nesmú byť stratené, pričom musí byť zabezpečené doručenie správy *práve raz* (angl. exactly once).
- spracovanie prebieha iba v *hlavnej pamäti*, žiadne dáta nie sú ukladané na disk a prúd bude spracovaný tak, ako údaje vznikajú.
- spracovanie *nedokonalých* alebo *chybných* správ. Poškodené, alebo správy, ktoré prišli v zlom poradí, musia byť spracované v čo najlepšej možnej miere (pozn.: poškodené dáta nemusí byť možné spracovať, potom sú zahodené).
- zaťaženie systému by malo byť *rovnomerné*, takže by sa nemalo stať, aby jeden uzol z desiatich spracovával 90% prúdu a zvyšných 10% prúdu bude distribuovaných medzi deväť uzlov.

Pre splnenie týchto požiadaviek bola najlepšia voľba programovacieho rámca Storm. Použitím tohto rámca sa môžeme sústrediť na implementáciu logiky a funkcionality systému, ladenie odozvy a rozloženie zaťaženia systému. Využitím vlastností Storm-u (distribuované a škálovateľné riešenie, ktoré je odolné voči chybám) sa nemusíme sústrediť na riešenie niektorých z vymenovaných problémov. Použitím Storm-u sa tiež vyhneme blokujúcim operátorom (napr. agregácia, alebo zoskupenie podľa hodnoty) opísaných v kapitole 2.2.

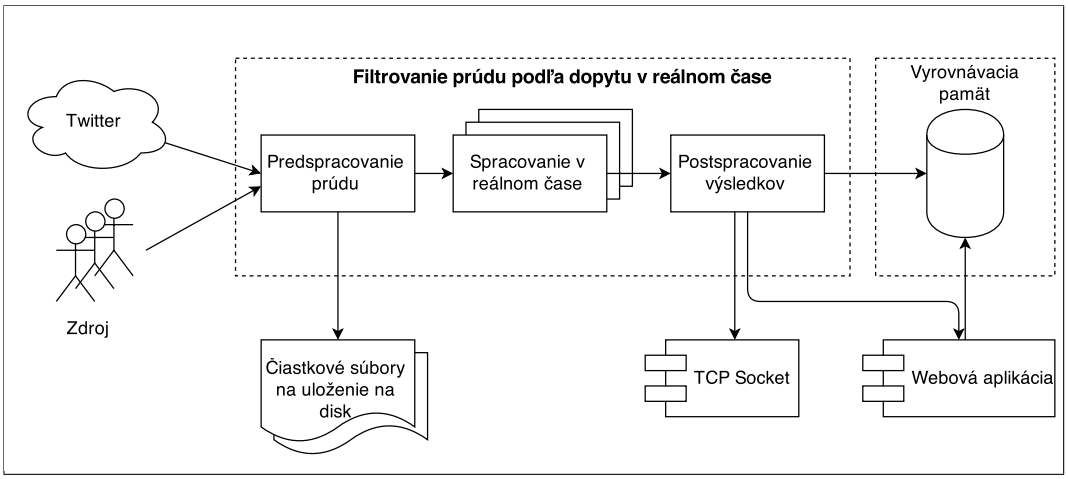
4.2 Všeobecný návrh

Pre zjednodušenie budeme poškodené alebo chybné dáta zahadzovať. V reálnom svete by malo zmysel ich špeciálne spracovávať, avšak v našom scenári sú najčastejšie chybné dáta identifikované ako správy *delete*, ktoré značia zmazanie nejakej správy z prúdu v minulosti.

Tradičný pristup k spracovania údajov, je použitie relačnej databázy. Informácie z prúdiacich udalostí by museli byť najprv extrahované a následne uložené v

relačnej databáze. Tento prístup zlyháva pri masívnom zapisovaní štrukturovaných údajov, a paralelnom dopytovaní sa do databázy. Dávkové spracovanie je veľmi podobné, ale dáta sú často zapisované neštruktúrované. Aj keď je vtedy zapisovanie rýchle, vzhľadom na objem ukladaných dát spracovanie trvá priliš dlho. Preto navrhujeme všeobecnú topológiu postavenú na rámci Storm.

Navrhujeme topológiu, ktorá vychádza z princípu dátovodov a filtrov (Aggarwal, 2007), hlavný rozdiel je v tom, že spracovanie prebieha distribuovane a paralelne. Vďaka tomu môžeme dosahovať odozvu v reálnom čase. Podrobnejšie je návrh rozobratý a načrtnutý pri každej iterácií opísanej v časti 5 Experimenty.



Obrázok 4.1: Všeobecný návrh softvérovej súčiastky na filtrovanie prúdu údajov na základe dopytu v reálnom čase. Venujeme sa len tejto časti, moduly za postspracovaním sú zobrazené len pre ilustráciu toho, ako je ďalej možné pracovať s výsledkami. Hlavným častiam návrhu sa venujeme v nasledujúdich troch podkapitolách samostnatne.

4.2.1 Predspracovanie vstupných údajov

Na vstupe sú dva prúdy: prúd udalostí z Twittru a prúd dopytov používateľa. Ich povaha si vyžaduje špeciálnu pozornosť. Údaje prúdiace vo veľkých objemoch a vysokou rýchlosťou je potrebné bezchybne spracovať. Keby sme udalosti len jednoducho čítali a spracovávali tak, ako vznikajú, mohlo by sa nesprávnym preplánovaním procesov stať, že niektoré údaje sa nenávratne stratia už na vstupe. Naviac chceme zabezpečiť, aby údaje prišli v takom poradí, v

akom vznikajú (pozn.: preusporiadanie zdrojom nevieme ovplyvniť). Aby sme tomuto predišli, budú všetky vstupné údaje zaraďované do špeciálnej *fronty*. Toto riešenie musí byť *distribuované*, pretože vlastnosti prúdiacich údajov sa v čase menia. Na tieto zmeny musíme byť schopní reagovať.

Potom má zmysel použiť samostatný nástroj na spracovanie údajov vstupujúcich do topológie, resp. softvérovej súčiastky na spracovanie prúdu. Oddelením biznis logiky od predspracovania má výhodu v tom, že nástroj predspracujúci vstupné údaje je zameraný len na jednu činnosť, ktorú robí najlepšie ako vie. Predspracovávané údaje musia byť naviac zapisované na disk tak, aby sme mohli zaručiť odolnosť voči chybám. Toto ukladnie musí byť vyriešené takým spôsobom, že to neovplyvní výkonnosť systému. V hlavnej pamäti sú vždy najnovšie správy a tie, ktoré už boli spracované (odoslané a potvrdené *ack*) sa môžu na pozadí ukladať *dočasne* (pre stanovený časový interval) na disk.

Keďže celé riešenie má distribuovaný charakter, je predpoklad, že bude existovať viac uzlov konzumujúcich rovnaký prúd. Potom musí byť prúd duplikovaný. Konzumenti si musia vedieť vybrať čo budú konzumovať (pozn.: v tomto prípade hovoríme o údajoch z Twittru alebo dopytoch). Niekedy môže byť užitočné konzumovať viac prúdov jedným konzumentom. Preto navrhujeme použiť nástroj aplikujúci vzor *publikovať-predplatiť* (angl. publish-subscribe) (Kreps et al., 2011).

4.2.2 Spracovanie v reálnom čase

V tejto časti filtrujeme prúdiace dáta na základe používateľského dopytu podľa extrahovaných *identifikačných slov* (hashtag) zo správ (tweet), ktoré prúdia zo sociálnej siete Twitter. Identifikačné slová sú vždy označené znakom #. Dopyty prúdia v rovnakej forme takže sa používateľ sa dopytuje pomocou identifikačných slov, ktoré by sa mohli nachádzať v správe. Identifikačné slová je potrebné pred aplikovaním filtra extrahovať zo správ. Je to jednoduchá operácia, pretože za identifikačné slová považujeme slová označené znakom #.

Pre efektívne filtrovanie musíme zohľadniť objem a rýchlosť prúdiacih údajov a dopytov. Niektoré dopyty môžu byť duplikáty, a preto nemá zmysel vytvárať nový filter. Navrhujeme, aby sa len isté uzly starali o filtrovanie určitých identifikačných slov. Tým rovnomerne rozložíme zaťaženie filtračnej časti systému.

Bude preto potrebné extrahovať identifikačné slová v skoršej fáze filtrovania, a následne správy smerovať do filtračných uzlov podľa extrahovaných identifikačných slov. Naviac vyžadujeme zabezpečiť doručenie každej správy *práve raz*. Pre splnenie tejto požiadavky navrhujeme aplikovať podobný potvrdzovací mechanizmus odoslaných správ, ktorý je známy zo sieťového protokolu TCP[1]. Daný mechanizmus funguje tak, že odosielateľ si udržuje frontu odoslaných a nepotvrdených správ. Odosielateľ čaká zadefinovaný čas po odoslaní správy a na jej *potvrdenie*, ak potvrdenie (pozn.: zaužívaná notácia je *ack*) nepríde do tohto časového intervalu od prijímateľa, správa je odosielateľom preposlaná.

4.2.3 Postpracovanie výsledku

V poslednej fáze spracovania máme k dispozícii výsledky, ktoré je potrebné nejakým spôsobom spracovať. Táto fáza nepredstavuje hlavnú funkcionalitou, ktorú navrhujeme. Najčastejšie sú výsledky emitované do výstupného frontu, odkiaľ ich môže konzumovať klientská aplikácia. Avšak, výsledky môžu byť posielané tiež priamo do otvoreného sieťového socketu alebo zapisované do vyrovnávacej pamäti. V zásade, to, čo sa má stať v poslednej fáze, nie je nutné vedieť dopredu. Môžeme aplikovať zapisovanie do všeobecného frontu (obdoba predspracovania) odkiaľ výsledky konzumuje kto potrebuje. To čo sa už ďalej stane s výsledkami (napr. čítanie a mazanie vyrovnávacej pamäte) pre nás nie je rozhodujúce, je to väčšinou aplikačný problém.

My budeme pre testovanie a overenie návrhu výsledky ukladať do vyrovnávacej pamäti typu kľúč-hodnota (angl. key-value) a tiež do súboru CSV[2].

[1] Transmission Control Protocol
[2] Hodnoty oddelené čiarkou (angl. Comma-separated values)

4.3 Implementácia a použité technológie

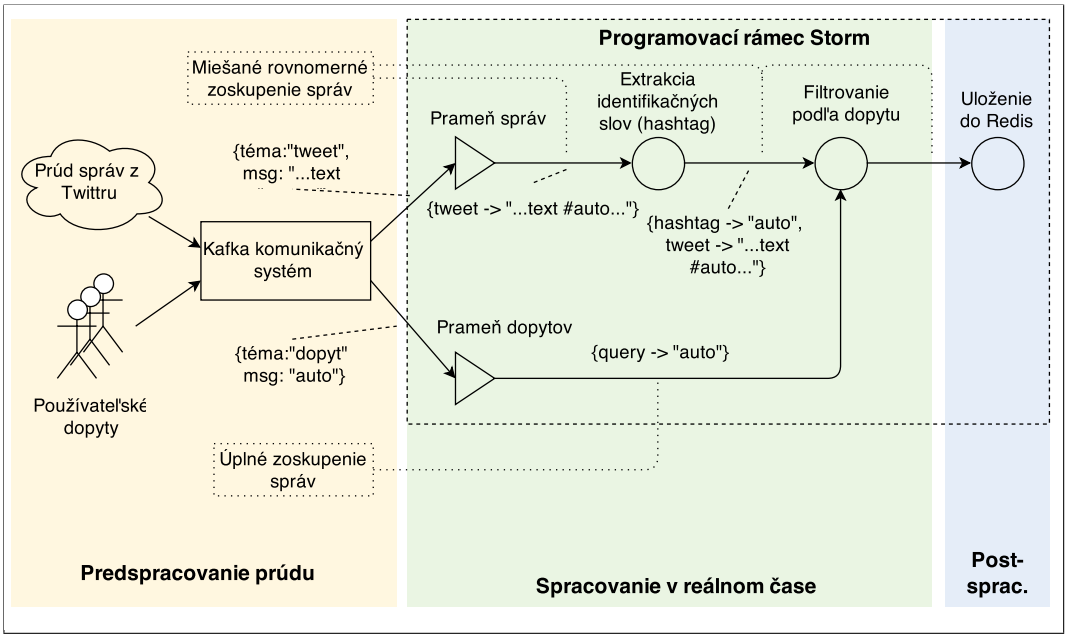
Implementácia návrhu riešenia vychádza z jeho opisu v predchádzajucej kapitole. Každá podstatná časť návrhu si vyžiadala špeciálnu pozornosť, a dôraz na výber a aplikovanie správnych nástrojov a postupov. Programovací rámec Storm[3] sme si zvolili na implementáciu hlavnej časti nášho návrhu spracovania prúdu v reálnom čase. Hlavnými dôvodmi voľby nástroja Storm je, že poskytuje distribuované riešenie odolné voči chybám. Z toho vyplýva, že spĺňa stanovené požiadavky, ktoré sme si na začiatku stanovili. Tento nástroj sme podrobnejšie analyzovali v časti 3.2.4 Storm.

Nástroj Kafka[4] sme vybrali pre fázu predspracovania. Zabezpečuje počiatočné zachytenie prúdu údajov, pričom vychádza zo vzoru publikovať-predplatiť (angl. publish-subscribe). Tento nástroj poskytuje oddeľovanie prúdov podľa tém (topic), distribuované riešenie odolné voči chybám, a zaručuje doručenie správ ich potvrdzovaním. Je možné ho využiť aj vo fáze postspracovania výsledkov ako rýchlu frontu výsledkov. Výsledky budú rozdelené do prúdov podľa tém na základe filtračných dopytov. Klientská aplikácia sa môže následne dopytovať na podmnožinu tém, či prúdov. My sme výsledky ukladali do rýchlej distribuovanej kľúč-hodnota databázy Redis[5], ktorú sme použili ako vyrovnávaciu pamäť výsledkov pre neskoršie vyhodnotenie experimentov a výkonnosti celej implementácie. Redis je primárne databáza, ktorá ukladá všetko v hlavnej pamäti (pozn.: vďaka tomu rýchla, a preto používaná ako vyrovnávacia pamäť), a v prípade prekročenia limitu pridelenej pamäte zapisuje na dáta disk.

Navrhujeme všeobecnú topológiu zobrazenú na obrázku 4.2 (pozn.: topológia predstavuje orientovaný acyklický graf[6] výpočtových uzlov), ktorá je postavená na programovacom rámci Storm. Využívame to, že náš problém je možné ľahko paralelizovať a teda všetky uzly topológie bežia vo viacerých vláknach. Topológia môže naviac bežať v klastri fyzických výpočtových uzlov, ktoré sú prepojené napríklad v sieti internet a úplne nezávislé od seba (napr. hardvér, lokácia, atď.).

[3] Apache Storm: https://storm.apache.org/
[4] Apache Kafka: http://kafka.apache.org/
[5] Redis: http://redis.io/
[6] DAG: directed acyclic graph

Smerovanie a distribúcia prúdu správ (tweet) medzi uzlami je rovnomerná. Znamená to, že správy budú v rámci topológie distribuované tak, aby bolo zaťaženie všetkých uzlov rovnomerné. Toto je implementované *zoskupením* (angl. grouping) správ, ktoré poskytuje a vnútorne implementuje rámec Storm vrátane sledovania stavu (Nimbus a ZooKeeper) uzlov. Toto zoskupenie, resp. distribúcia správ v rámci topológie je nazvané *miešané rovnomerné zoskupenie* (angl. shuffle grouping). Distribúcia dopytov sa mierne odlišuje. Každý dopyt musí byť doručený do všetkých filtračných uzlov (pozn.: nastáva duplikácia dopytov), pretože na začiatku nevieme, do ktorého uzla, resp. pracovníka (worker) a úlohy (task) bude správa nasmerovaná. Takúto distribúciu správ nazývame *úplné zoskupenie* (angl. all grouping).



Obrázok 4.2: Všeobecná topológia postavená na programovacom rámci Storm spolu s komunikačným systémom Kafka a kľúč-hodnota úložiskom Redis.

Je nutné pripomenúť, že všetky uzly v topológií bežia paralelne a často na viacerých fyzických strojoch v klastri, preto má napríklad distribúcia dopytov a správ veľký význam na optimálne fungovanie systému. Počiatočné nastavenie systému volíme také, že každý pracovník (worker) uzla grafu (pozn.: skrutky a pramene) beží z počiatku v štyroch vláknach (pozn.: v terminológií Storm vlákna predstavujú *spúšťače*, angl. executors), kde jedno vlákno má najviac šesťdesiatštyri úloh. Tieto parametre sa pravdepodobne budú meniť s ladením topológie vo fáze experimentovania a overovania výkonnosti. Celé riešenie je

implementované použitím inkrementálnych algoritmov. Algoritmus filtrovania správ vo filtračnom uzle topológie je zobrazený na tejto strane.

Správy z Twittra prúdia vo formáte *JSON*[7], preto sme použili knižnicu JSON (org.json) na uľahčenie práce s objektami JSON (správami). Pri spracovaní prúdu priamo z Twitter API (pozn.: API znamená aplikačné programovacie rozhranie, angl. Application Programming Interface) je potrebné aplikáciu voči API autorizovať a pripojiť sa na verejný prúd. Na tento účel sme použili knižnicu *Twitter4j*[8], ktorá poskytuje jednoduchú integráciu Twitter API do programovacieho jazyka Java. Na správu závislostí projektu Java sme použili široko používany nástroj Maven[9], a na správu verzií projektu známy nástroj z unixu Git[10].

Data: Prúd n-tíc obsahujúci dvojice dopyt - *používateľský dopyt*, hashtag - *hastag, ktorý sa nachádza v texte správy*, tweet - *správa resp. tweet*
Result: Vyfiltrované správy podľa dopytu
inicializuj;
while *nie je koniec prúdu* $n-tic$ **do**
    načítaj(n-ticu);
    if $n-tica$ *je dopyt používateľa* **then**
        $dopyt$ = získajDopyt($n-tica$);
        pridaj($vsetkyDopyty$, $dopyt$);
    else
        $hashtag$ = získajHashtag($n-tica$);
        $tweet$ = ziskajTweet($n-tica$);
        if $hashtag$ *je v množine* $vsetkyDopyty$ **then**
            časováPečiatka($tweet$,"filter-timestamp");
            emituj($hashtag$,$tweet$);
        end
    end
    potvrďAck($n-tica$);
end

Algoritmus 2: Filtračná časť topológie a príslušný algoritmus filtrovania podľa dopytu.

[7] JavaScript Object Notation: http://www.json.org/
[8] Twitter4j: http://twitter4j.org/en/index.html
[9] Apache Maven: https://maven.apache.org/
[10] Git: http://git-scm.com/

5. Experimenty

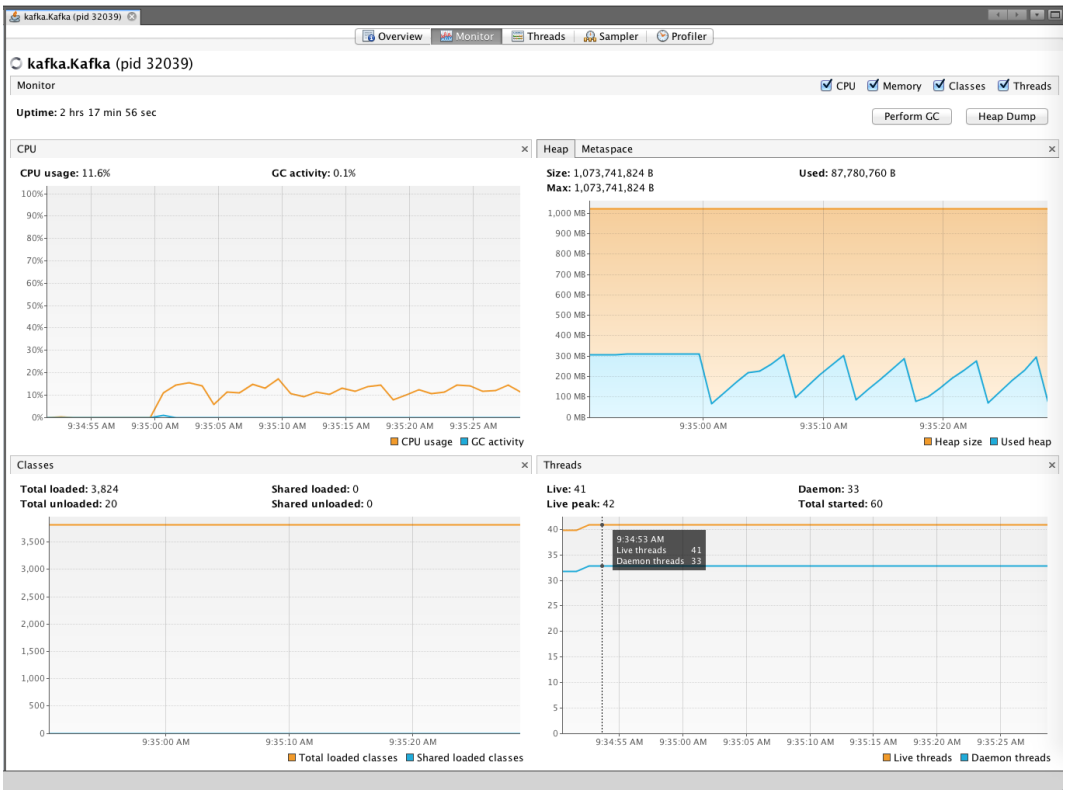
Experimentovanie a overenie riešenia prebiehalo vo viacerých iteráciách. Overovanie sme vykonávali iteratívne a metodologicky. Proces testovania je zobrazený na obrázku 1 v prílohe A. V každej iterácií sme sa snažili identifikovať prípadné problémy a okamžite navrhnúť ich riešenie, ktoré sme následne implementovali a opäť overili. Najčastejšie bolo potrebné správne identifikovať *úzke hrdlo* (angl. bottleneck) overovanej topológie, ktoré spôsobovalo zníženie výkonnosti systému v špecifických prípadoch. Úlohou experimentu bolo filtrovanie prúdu údajov zo sociálnej siete Twitter podľa dopytov.

5.1 Metodológia overovania

Aby sme boli schopní merať výkonnosť riešenia, museli sme zaviesť časové značkovanie každej správy. Správa je označovaná časovou pečiatkou v každom uzle tak, ako prechádza naprieč topológiou. Na označenie správy časovou pečiatkou používame metódy *nanoTime* a *currentTimeMillis* triedy *System*[1] programovacieho jazyka Java. Metóda *nanoTime* nám poskytuje čas bežiaceho virtuálneho stroja Java (angl. Java Virtual Machine, ďalej len JVM), nad ktorým je vykonávaný kód, v nanosekundách. Tento čas neodzrkadľuje systémový čas, ani čas reálneho sveta. Je vhodné ho využiť na meracie účely, a preto vyhovuje nášmu scenáru. Z týchto časových pečiatok odčítavame čas spracovania správy v každom bode topológie, a tiež celkový čas. Druhá metóda *currentTimeMillis* vracia systémový čas JVM v milisekundách od epochy. Tento čas používame na porovnanie s časom spustenia zberača odpadkov Java (angl. Java Garbage Collector, ďalej len Java GC). Na monitorovanie Java GC sme použili nástroj VisualVM[2], ktorý poskytuje komplexný monitorovací nástroj aplikácií vykonávaných nad JVM. Ukážka VisualVM sa nachádza v obrázku 5.1.

[1] Trieda System Javadoc: https://docs.oracle.com/javase/8/docs/api/java/lang/System.html
[2] VisualVM: https://visualvm.java.net/

Na to, aby sme boli schopní porovnať výsledky, sme nemohli overovať experimenty na dátach priamo zo sociálnej siete Twitter, pretože neprichádzajú v koštantnej rýchlosti. Naviac, verejné API prúdu správ, ktoré poskytuje Twitter, predstavuje len malé percento (pozn.: presné číslo nie je zverejnené) z celého prúdú. Práve preto nie tento prúd nieje možné použiť na správne meranie výkonnosti aplikácie. My sme si vytvorili vlastnú množinu dát, ktorú sme istý čas zbierali z daného prúdu, a vytvorili sme súbor dát o veľkosti 1GB správ (tweetov).



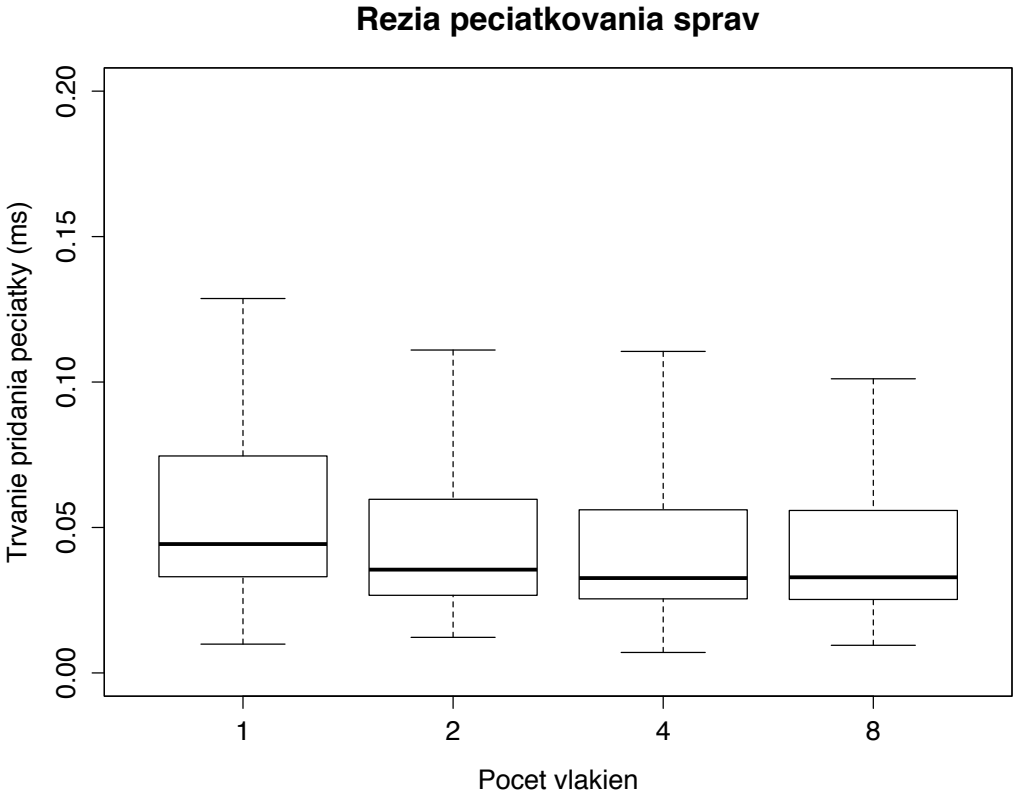
Obrázok 5.1: Príklad monitorovacieho panelu VisualVM pri monitorovaní bežiacej Java Aplikácie (konkrétne Apache Kafka).

Pri overovaní sme simulovali prúd údajov zo sociálnej siete množinou dát, ktorú sme si skôr pripravili. Topológie sme spúšťali v lokálnom klastri lebo sme nemali k dispozícií klaster fyzických strojov. Na overenie riešenia je to postačujúce. Dokonca tvorca Storm-u odporúča ladiť a overovať navrhnuté topológie lokálne pred odoslaním do vzdialeného klastra.

Pre korektnosť testovania sme museli zistiť, aká je réžia samotného pečiatkovania správ, aby nám operácia pridania časovej pečiatky neskresľovala konečný sledovaný čas. Toto sme overili postavením jednoduchej topológie, ktorá nefiltrovala žiadne správy. Všetky iba prešli cez topológiu s jednou skrutkou (bolt),

ktorá pridávala časovú pečiatku. Označený čas sme pri výstupe správy z topológie odčítali a a tak sme získali réžiu pečiatkovania. Zároveň týmto testom overujeme priepustnosť topológie (pozn.: nachádza sa tu iba jedna skrutka!). Výsledkok tejto réže je vo väčšine prípadov úplne zanedbateľný. Namerané výsledky zobrazuje grafe na obrázku 5.2.

Z grafu vyplýva, že réžia pridania časovej pečiatky je takmer zanedbateľná, a nebude nám skresľovať merania. Pri tomto meraní sme výsledky ukladali do súboru CSV, pričom vychyľujúce sa hodnoty sú spôsobé spustením Java GC.



Obrázok 5.2: Graf zobrazuje réžiu pridania časovej pečiatky do správy. Testovanie bolo realizovné štyrikrát s rôznym počtom vlákien. Na Y osi grafu je v poradí z dola zobrazený *10%*, *25%*, *50% (medián)*, *75%* a *90%* kvantil.

Ako sme spomenuli, testovanie prebiehalo na lokálnom klastri, resp. na jednom lokálnom počítači. Parametre počítača, na ktorom boli topológie testované, sú zobrazené v prílohe A tabuľka 2. Verzie všetkých použitých nástrojov, programovacích jazykov a rámcov je možné vidieť v prílohe A tabuľka 1. Navrhujeme štyri testovacie scenáre v tabuľke 5.1. Všetky testovacie scenáre sú overené na rovnakej konfigurácii, pričom počas každého testovania je

nastavený počet vlákien na štyri. V reálnom nasadení by sa tento počet určite navýšil a hlavne adaptoval v závislosti od zaťaženia systému. Pre náš účel overenia výkonnosti sme tieto parametre zvolili ako najlepšií kompromis, aby sme získali konzistentné výsledky odzrkadľujúce reálne nasadenie. Dané parametre sme získali z dlhodobého experimentovania a pozorovania správania sa všetkých systémov a lokálneho klastra. Detaily týchto pozorovaní sú mimo rozsah našej práce. Hlavnou nevýhodou testovania lokálne bola simulovácia prúdu dát. Na jednom procesore sa snažíme generovať obrovské objemy dát, následne ich spracovať, a ešte aj uložiť, bolo nutné nájsť a zvoliť vhodný kompromis.

5.2 Filtrovanie prednastaveným filtrom

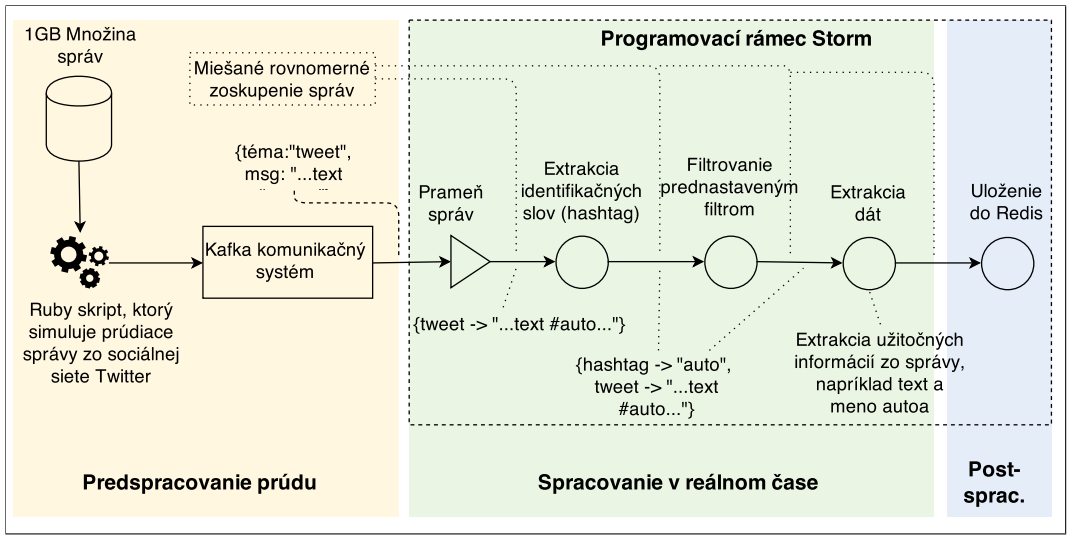
V tomto experimente nás zaujíma výkonnosť a priepustnosť samotného filtrovania a systému ako takého. To znamená, či je schopný spracovať a filtrovať veľké objemy správ (pozn.: v reálnom svete je pravdepodobné, že dopytov bude menej ako správ a výsledný prúd je potom podstatne zúžený) a produkovať vysoko objemový výstup. Preto filtrujeme približne polovicu objemu prúdu prednastaveným filtrom (pozn.: filtrujeme každú druhú správu), ktorý sme implementovali priamo do zdrojového kódu. V tomto experimente je teda iba jeden jednoduchý testovací scenár - filtrovanie podľa prednastaveného dopytu. Filtrujeme simulovaný prúd údajov zo sociálnej siete Twitter. Simuláciu zabezpečuje jednoduchý Ruby skript, ktorý cyklicky posiela správy do vstupného komunikačného systému Kafka. Keď je 1GB súbor správ prečítaný, skript pokračuje v čítaní od začiatku. Test trval 160 sekúnd. Testovaná topológia je zobrazená na obrázku 5.3.

Experiment prebiehal v štyroch vláknach, bolo spracovaných takmer 150 000 správ. Špičky zobrazené na grafe sú spôsobené spustením Java GC a výpadkom stránky databázy Redis, kam boli zapisované výsledky. Tento výpadok nastáva, keď veľkosť databázy prekročí stanovený limit, ktorý je nakonfigurovaný (pozn.: Redis je primárne databáza, ktorá ukladá všetko do hlavnej pamäti, v prípade prekročenia tohto limitu zapisuje čiastkový súbor na disk).

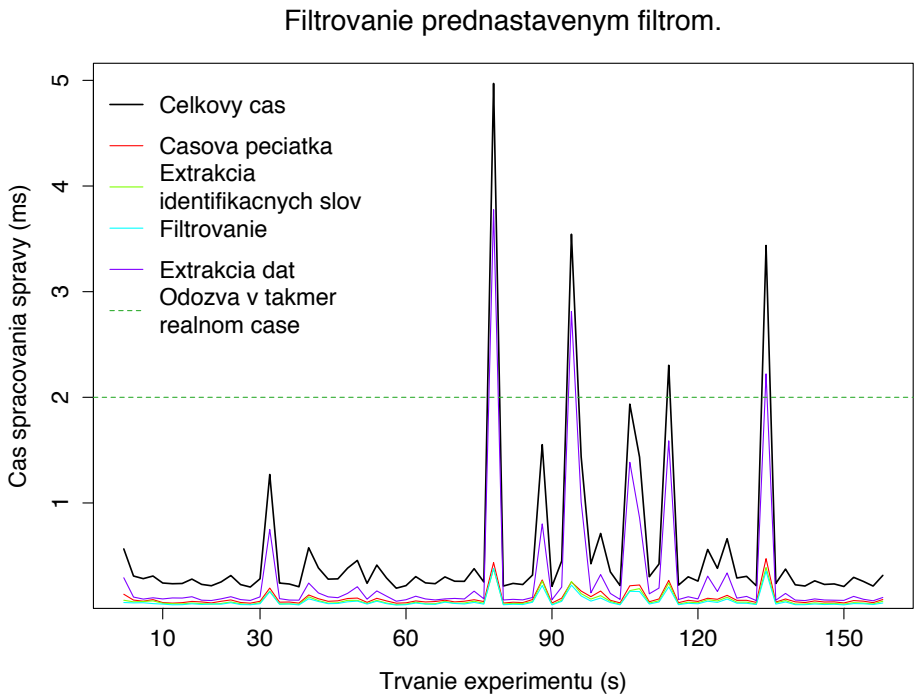
V grafe na obrázku 5.4 môžeme vidieť viac časových špičiek spracovania prúdu

počas experimentu. Tieto špičky sú spôsobené spustením Java GC (pozn.: spustenie Java GC sme pozorovali nástrojom VisualVM a ručne sme porovnali čas spustenia s časom špičiek počas experimentu), ale mohlo sa na tom podielať aj preplánovanie procesov operačným systémom. Je to spôsobné tiež tým, že na jednom počítači bežia všetky systémy (pozn.: Kafka, ZooKeeper, Storm, Redis, simulačné skripty) potrebné pre experiment. Prúd síce nie je veľmi objemný, ale ak ho máme na jednom fyzickom stroji generovať, predspracovávať (pozn.: Kafka), spracovávať (pozn.: naša topológia nad rámcom Storm) aj ukladať (pozn.: Redis), veľmi zaťažuje PC. Navyše, okrem databázy Redis a simulačných skriptov, všetko beží nad jedným JVM (pozn.: čo zvyšuje šance na spustenie Java GC). Aj napriek týmto špičkám hodnotíme priepustnosť a výkonnosť ako výbornú, pretože sa odozva systému rýchlo ustáli do prijateľných hodnôt.

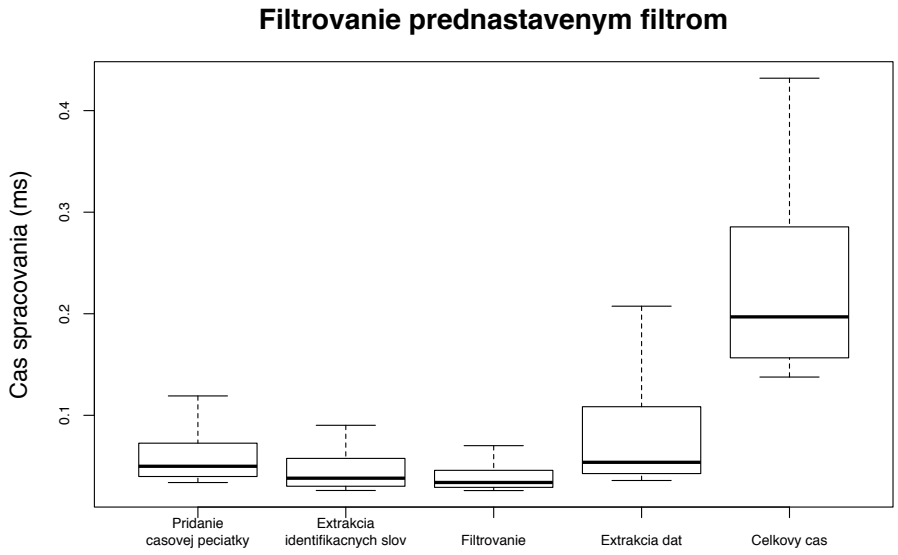
Miera rozdelenia nameraných časov je zobrazená na grafe v obrázku 5.5.



Obrázok 5.3: Topológia pre filtrovanie prúdu prednastaveným filtrom.



Obrázok 5.4: Filtrovanie prúdu údajov prednastaveným filtrom po dobu 160 sekúnd.



Obrázok 5.5: Filtrovanie prúdu údajov prednastaveným filtrom po dobu 160 sekúnd. Na Y osi grafu sú zobrazené kvantily v poradí *10%*, *25%*, *50% (medián)*, *75%* a *90%* kvantil.

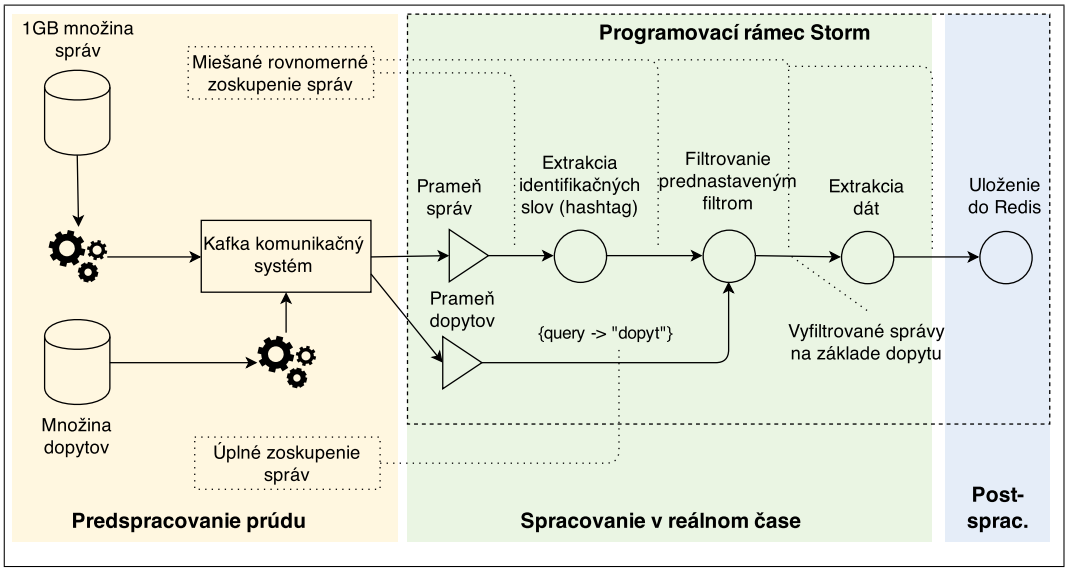
5.3 Filtrovanie viacerými dopytmi

V tomto experimente filtrujeme prúd údajov podľa dopytov. Na prichádzajúce dopyty sa tiež pozeráme ako na prúdiace údaje. Simulujeme prúdiace správy zo siete Twitter a dopyty od používateľov. Aplikujeme testovacie scenáre, ktoré sú zobrazené v tabuľke 5.1. Smerovanie dopytov je také, že všetky dopyty sú duplikované do každého filtračného uzla (skrutky) topológie. Toto duplikovanie dopytov môže byť problematické pri veľkom objeme prúdu správ a dopytov, pretože bude rásť časová náročnosť filtrovania bez ohľadu na škálovateľnosť riešenia. Tým sa zvyšuje šanca prúdenia duplikátov cez topológiu, pretože ich kontrola je možná len v postspracovaní prúdu kontrolou ID správy. Na obrázku 5.6 je zobrazená testovacia topológia.

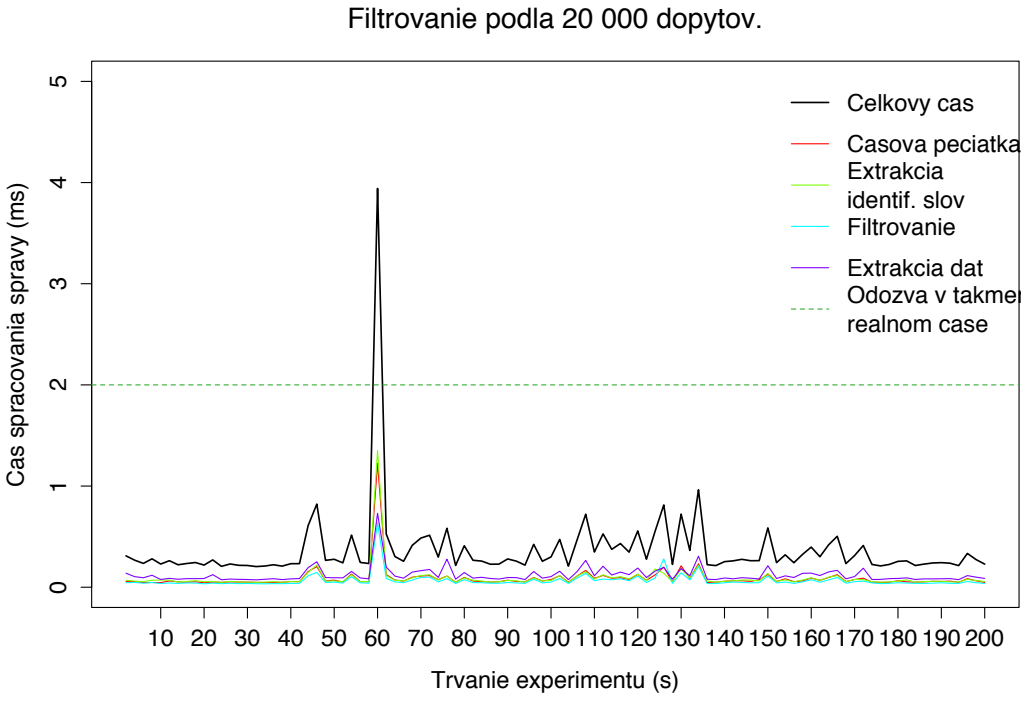
Na grafe v obrázku 5.7 je zobrazený priebeh času spracovania filtrovania viacerými dopytmi v priebehu experimentu. Približne po šesťdesiatich sekundách nastane náhle zvýšenie odozvy systému. Táto situácia nastáva opäť pri spustení Java GC (pozn.: odčítanie sme vykonali ručne z monitorovacieho nástroja VisualVM). Síce sa čas spracovania rýchlo ustáli do prijateľných hodnôt, spracovanie sa nejaví ako stabilné. Je možné pozorovať nestálosť a anomálie v čase spracovania správy. Tieto anomálie môžu byť spôsobené preplánovaním procesov operačným systémom lokálneho klastra. Je ale málo pravdepodobné, že toto by vnieslo do spracovania tak často pozorovateľné výchylky. Preto tvrdíme, že tieto anomálie predstavujú zvýšenú programovú réžiu pri spracovaní duplikovaných správ a dopytov (pozn.: keďže každý filtračný uzol obsahuje množinu všetkých dopytov, pri prechode jednej správy musí skontrolovať zhodu s viacerými dopytmi, či nespĺňajú filtračnú podmienku, toto tiež zvyšuje nároky na hlavnú pamäť, ktorá je veľmi obmedzená). Tento jav môžeme tiež pozorovať na grafe zobrazenom na obrázku 5.8, kde sledujeme širokú distribúciu časov filtračnej časi a tiež celkového času, pričom pozorujeme tendeciu zvyšovania šírky tejto distribúcie časov.

Toto vymedzujeme ako problém, ktorý môže pri spracovaní väčších, resp. prúdov reálneho sveta, spôsobovať problémy zanášaním *nestability* systému a *vysokej odozvy* v špecifických prípadoch. S ďalším zväčšením objemu prúdu a kadencie vzniku nových udalostí by toto mohlo predstavovať úzke hrdlo.

Ďalšie výsledky sú zobrazené tiež v tabuľke 5.1, podrobnejšie čiarové grafy sú dostupné v prílohe C.1.



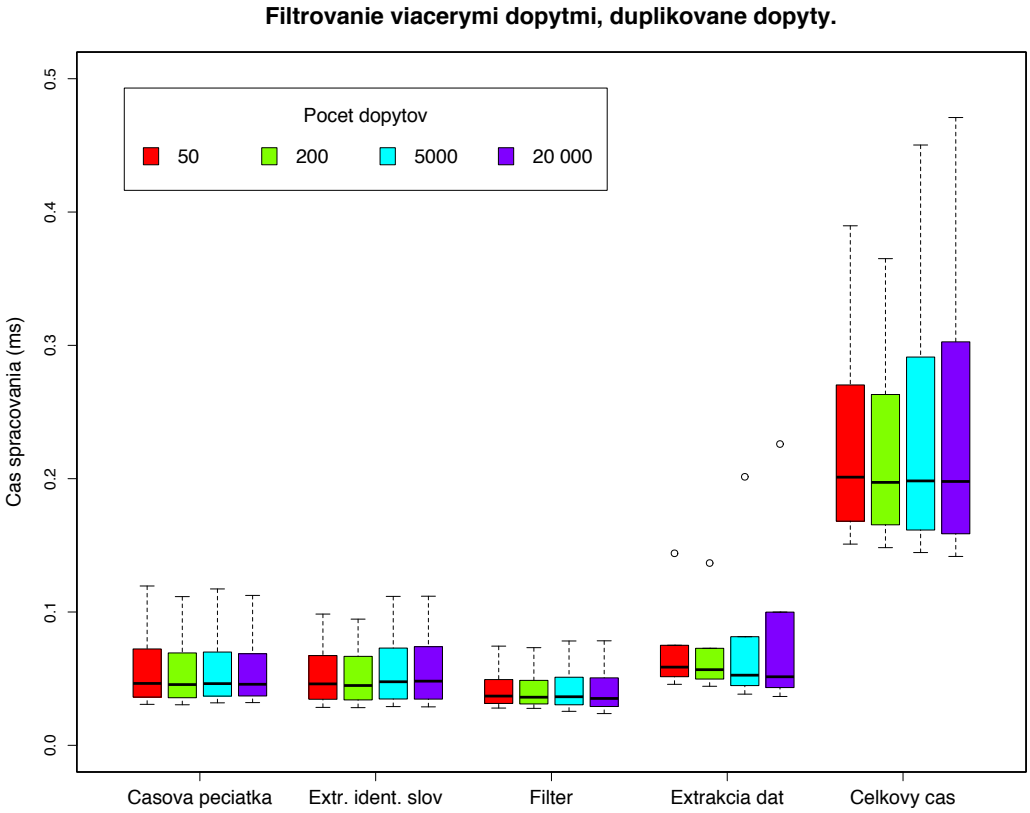
Obrázok 5.6: Topológia pre filtrovanie podľa dopytu pri duplikovaní (zoskupenie resp. smerovanie) dopytov do všetkých filtračných uzlov.



Obrázok 5.7: Filtrovanie viacerými dopytmi po dobu 200 sekúnd.

#	Dopyty	Trvanie (s)	Dáta	Výsledok(medián)
1	50	160	1GB dataset cyklicky	0.2011 ms
2	200	160	1GB dataset cyklicky	0.1972 ms
3	5000	160	1GB dataset cyklicky	0.19830 ms
4	20000	200	1GB dataset cyklicky	0.19790 ms

Tabuľka 5.1: Testovacie scenáre.



Obrázok 5.8: Zobrazuje čas spracovania správy v uzle topológie a celkový čas. Na Y osi sú zobrazené kvantily v tomto poradí *10%*, *25%*, *50% (medián)*, *75%* a *90%*.

5.4 Smerovanie správ na základe dopytu

V predošlom experimente sme identifikovali problematickú časť topológie, ktorá by s nárastom objemu prúdu (pozn.: v praxi také prúdy dát existujú, ale nevieme ich doma jednoducho simulovať) predstavovala úzke hrdlo topológie. Je potrebné navrhnúť také riešenie, kde budú dopyty rovnomerne rozdelené do filtračných uzlov v topológii. Prúdiace správy budú následne smerované do týchto filtračných uzlov podľa extrahovaných identifikačných slov. Tu môže vzniknúť duplikovanie správ, avšak šanca, že budú duplikáty pokračovať aj po filtrovaní závisí od obsahu identifikačných slov v správe. Mohlo by sa to stať napríklad vtedy, ak budú dve rôzne identifikačné slova v texte správy, potom je nutné správu duplikovať (pozn.: správa je emitovaná práve raz pre každé rôzne extrahované identifikačné slovo). Tymto garantujeme doručenie správy práve raz, ak správne odstránime duplikáty, ktoré prejdu filtračnou časťou. Filtračnou časťou prejdú vtedy a len vtedy, ak existuje správa s dvomi rôznymi identifikačnými slovami a zároveň existujú dva dopyty na tieto slová, potom je filtračná podmienka splnená dva krát. Takto vzniknuté duplikáty je potrebné odstrániť v neskorších fázach spracovania.

Pôvodný nápad bol všetky dopyty ukladať do rýchlej vyrovnávacej pamäte, kde by sa ukladali iba unikátne dopyty. Filtračné uzly by sa pri spracovávaní správy dopytovali do tejto pamäte, či existuje dopyt na identifikačné slovo, ktoré obsahuje správa. Toto riešenie by bolo určite funkčné, ale pri veľkom počte správ by sa táto pamäť stala úzkym hrdlom. Preto navrhujeme riešenie, ktoré zabezpečuje smerovanie správ v topológii podľa identifikačných slov resp. dopytov. Znamená to, že uzly spracujúce dopyt X vždy dostanú správy obsahujúce identifikačné slovo X. Rozdelenie teda nie je miešané (shuffle). Samozrejme, mohlo by sa zdať, že distribúcia správ a zaťaženie nie je rovnomerné, ak bude existovať veľa správ s identifikačným slovom X a málo s Y. Avšak toto smerovanie, ktoré implementuje ako zoskupenie rámec Storm, zabezpečuje rovnomerné rozdelenie zaťaženia naprieč topológiou. Týmto riešením znižujeme tiež šancu na vznik duplikátov, s ktorými je neskôr spojená réžia a odstraňujeme potenciálne úzke hrdlo topológie.

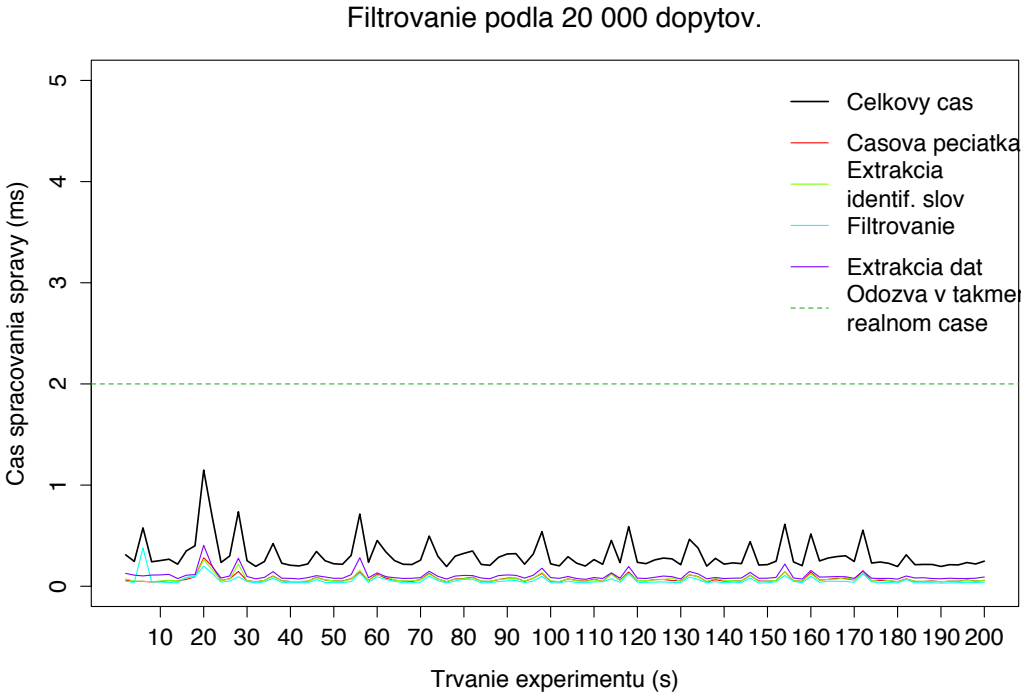
Topológia je identická ako v predošlom experimente, rozdielne je iba smerovanie, resp. zoskupenie správ (n-tíc prúdu). Z grafu zobrazenom na obrázku 5.9 môžeme pozorovať znížený výskyť anomálií počas spracovania. Toto zlepšie je

ešte lepšie viditeľné na grafe z obrázku 5.10 kde pozorujeme zúženie distribúcie času spracovania. Zlepšenie je navýraznejšie pri filtračnej časti, pričom ostatné časti vykazujú ustálené časy. Podstatné je, že sa tento čas nestáva počas priebehu experimentu nestabilný aj v prípade zvyšovania počtu dopytov. Takéto riešenie nám potom prinesie určite stabilnejší systém bez zanášania častej zvýšenej odozvy. Tvrdíme, že sme týmto odstránili vymedzené úzke hrdlo v predchádzajucej iterácii experimentovania.

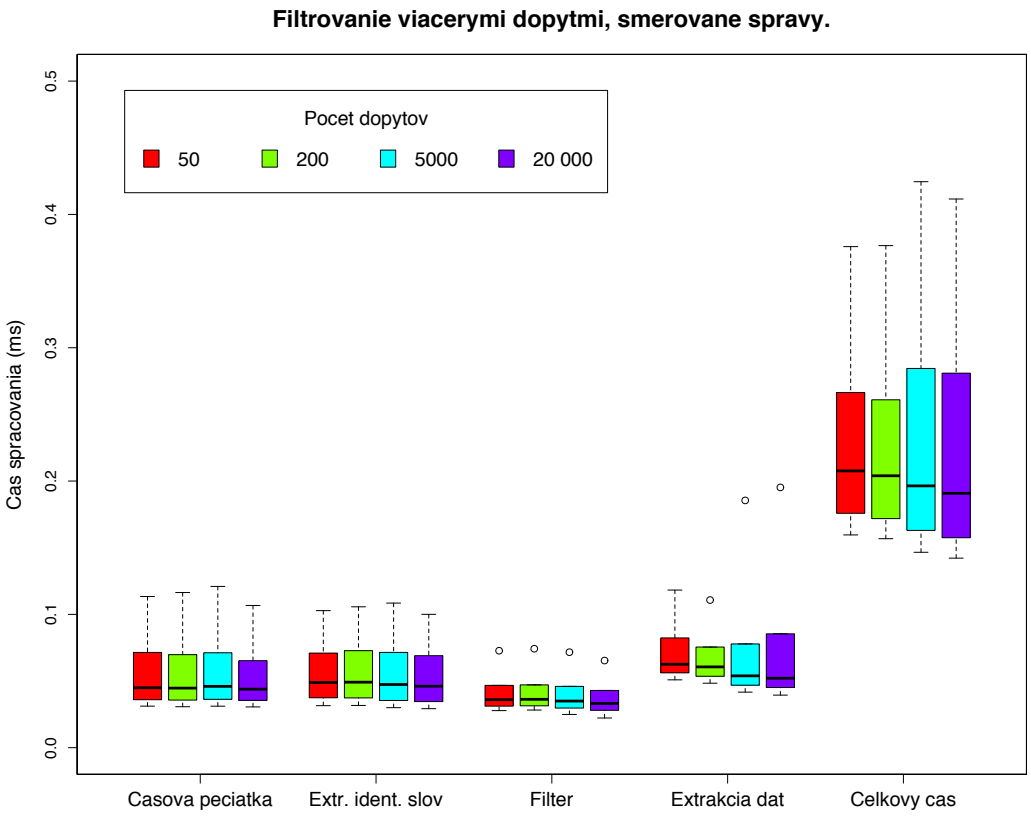
Ďalšie čiarové grafy sú dostupné v prílohe C.2.

#	**Dopyty**	**Trvanie (s)**	**Dáta**	**Výsledok(medián)**
1	50	160	1GB dataset cyklicky	0.2076 ms
2	200	160	1GB dataset cyklicky	0.2039 ms
3	5000	160	1GB dataset cyklicky	0.1964 ms
4	20000	200	1GB dataset cyklicky	0.19080 ms

Tabuľka 5.2: Testovacie scenáre.



Obrázok 5.9: Filtrovanie smerovanými dopytmi a správami po dobu 200 sekúnd.



Obrázok 5.10: Zobrazuje čas spracovania správy v uzle topológie a celkový čas. Na Y osi sú zobrazené kvantily v tomto poradí *10%*, *25%*, *50% (medián)*, *75%* a *90%*.

5.5 Zhodnotenie

Z výsledkov experimentov vyplýva, že na správanie systému vplýva veľa faktorov. Obzvlášť testovanie tohto rozmeru, kde na jednom počítači simulujeme prúd dát a zaroveň aj spracujeme je náročné. Často môže nastať preplánovanie procesov operačným systémom, môže sa spustiť Java GC (nad jedným JVM beží viac nástrojov ako Kafka, ZooKeeper, atď). Tieto udalosti môžu do spracovania zaviesť latenciu (viď. obr. 5.5 Filtrovanie prednastavenym filtrom). V reálnom nasadení by boli časti pred a postspracovanie prúdu oddelené na iných fyzických strojov, často takých o ktorých nevieme nič. Samotné riešenie by bežalo v klastri strojov, takže by sa výskyt takýchto udalostí ešte viac minimalizovali. Dosiahli sme spracovanie v reálnom čase, tak ako sme si to na začiatku práce definovali.

6. Zhodnotenie a budúca práca

V našom projekte sme podrobne analyzovali súčasné možnosti spracovania veľkých objemov dát (angl. Big Data). Vymedzili sme dva hlavné prístupy, dávkové a prúdové spracovanie. Pozornosť sme venovali najmä prúdovému spracovaniu a existujúcim nástrojom, ktoré spĺňajú požiadavky na prúdové spracovanie. Venovali sme sa analýze prúdu údajov zo sociálnej siete Twitter, pričom sme sa zamerali na filtračnú časť spracovania.

Hlavným príspevkom našej práce je to, že sa nám úspešne podarilo overiť rôzne topológie implementované nad zvoleným programovacím rámcom pre prúdové spracovanie. Najprv sme sa snažili identifikovať potenciálne limitácie samotného rámca, ktoré sa neprejavili ako kritické. Vyskúšali sme a podrobne overili štyri topológie. V poslednej sme dosiahli nami stanovené požiadavky na prúdové spracovanie. Zistili sme, že koexistencia viacerých systémov na jednom fyzickom stroji môže spôsobovať zvýšenie odozvy pre krátky časový okamih. Aj napriek tomuto zvýšeniu sme stále poskytovali riešenie odolné voči chybám.

Iteratívnym overovaním a vylepšovaním riešenia sme ukázali, že distribúcia správ môže mať zásadný dopad na fungovanie a správanie systému. Ukázali sme navrhnutá topológia v poslednej iterácii poskytuje dostatočne stabilné riešenie pre spracovanie prúdov. Pozorujeme tiež, že snaha rozdeľovať operácie nad prúdom do viacerých operácií znižuje šancu na vzniknutie úzkeho hrdla naprieč topológiou. Rovnako tvrdíme, že s čo najmenším prepojením systému do externých systémov v priebehu spracovania prúdu znižuje riziko vzniku úzkeho hrdla.

Implementované riešenie sme overili na dátach zo sociálnej siete Twitter, pričom sme museli simulovať prúdiace údaje. Napriek komplexite testovania sa nám úspešne podarilo overiť a ukázať, že naše riešenie spĺňa požiadavky stanovené v počiatku práce. Tiež spĺňa všeobecné požiadavky na prúdové spracovanie.

Ďalším zaujímavým smerom by bola kombinácia dávkového a prúdového spracovania (pozn.: Lambda architektúra). Budúcim krokom bude tiež návrh a im-

plementácia klientskej aplikácia spolu so snahou získať prístup k prúdu údajov reálneho sveta.

Literatúra

Aggarwal, C. C. (2007). *Data streams: models and algorithms*, volume 31. Springer Science & Business Media.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM.

Babu, S. and Widom, J. (2001). Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120.

Chaudhuri, S. and Dayal, U. (1997). An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74.

Dean, J. and Ghemawat, S. (2008). MapReduce : Simplified Data Processing on Large Clusters. 51(1):107–113.

Dong, X. L. and Srivastava, D. (2013). Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1245–1248. IEEE.

Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., and Tuulos, V. H. (2010). Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, page 32. ACM.

Franek, B. L. (2013). Importy dat z relační databáze do olap datovỳch kostek.

Higginbotham, S. (2010). Sensor networks top social networks for big data. [Online; zverejnené 13-September-2010].

Hwang, J.-H., Balazinska, M., Rasin, A., Cetintemel, U., Stonebraker, M., and Zdonik, S. (2005). High-availability algorithms for distributed stream processing. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 779–790. IEEE.

Kaisler, S., Armour, F., Espinosa, J. A., and Money, W. (2013). Big data: Issues and challenges moving forward. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 995–1004. IEEE.

Kamburugamuve, S., Fox, G., Leake, D., and Qiu, J. (2013). Survey of distributed stream processing for large stream sources. Technical report, Technická správa. 2013. Dostupné na URL: http://grids.ucs.indiana.edu/ptliupages/publications/survey_stream_processing.pdf.

Kreps, J., Narkhede, N., Rao, J., et al. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece*.

Liu, X., Iftikhar, N., and Xie, X. (2014). Survey of real-time processing systems for big data. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 356–361. ACM.

Marz, N. and Warren, J. (2013). *Big Data: Principles and best practices of scalable realtime data systems*. O'Reilly Media.

Mathioudakis, M. and Koudas, N. (2010). Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1155–1158. ACM.

Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. (2010). S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177. IEEE.

Silvestri, C. (2006). Distributed and stream data mining algorithms for frequent pattern discovery.

Stankovic, J. A., Son, S. H., and Hansson, J. (1999). Misconceptions about real-time databases. *Computer*, 32(6):29–36.

Stonebraker, M., Çetintemel, U., and Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47.

Terry, D., Goldberg, D., Nichols, D., and Oki, B. (1992). *Continuous queries over append-only databases*, volume 21. ACM.

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. (2014). Storm@ twitter.

In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM.

Wu, X., Zhu, X., Wu, G.-Q., and Ding, W. (2014). Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):97–107.

Zaslavsky, A., Perera, C., and Georgakopoulos, D. (2013). Sensing as a service and big data. *arXiv preprint arXiv:1301.0159*.

Zikopoulos, P., Eaton, C., et al. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.

Prílohy

A Technická dokumentácia

Cieľom je navrhnúť a implementovať systém, ktorý umožní používateľovi dopytovanie do prúdu údajov z vybranej domény. Na jeho dopyt by mal poskytnúť odozvu resp. výsledok v takmer reálnom čase. Pojem reálny čas definujeme v úvode projektu. V podstate identifikujeme len jeden prípad použitia pre túto konkrétnu implementáciu: *Získaj výsledok podľa dopytu.* Účastníkom v tom prípade použitia je používateľ aplikácie s ktorou sa dopytuje do prúdu udalostí (naša implementácia).

Program sme implementovali v Jazyku Java verzie 1.8.25, pričom aplikácia je priamo závislá od ďalších knižníc, neobsahuje žiadne grafické prostredie. Webové grafické používateľské rozhranie implementuje priamo rámec Storm na monitorovanie vykonávaných topológií v klastri. Zdrojové súbory sme rozdelily do nasledujúcich balíčkov:

- *bolt* - obsahuje skruty (bolty) topológie, teda hlavnú logiku,
- *builder* - obsahuje triedu zodpovednú za konštrukciu topológie,
- *scheme* - prispôsobené schémy (potrebné pre vytvorenie niektorých prameňov),
- *spout* - prispôsobené pramene topológie,
- *util* - pomocné triedy.

Zoznam použitých nástrojov a ich verzií je v tabuľke 1. Rámce a knižnice, ktoré sú potrebné pre aplikáciu sú získané prostredníctvom závislostí nástrojom Maven. Diagram tried rámca Storm je zobrazený na obrázku 2, diagram na obrázku 3 zobrazuje najdôležitejšie vzťahy tried. Vo fáze overovania výrobku

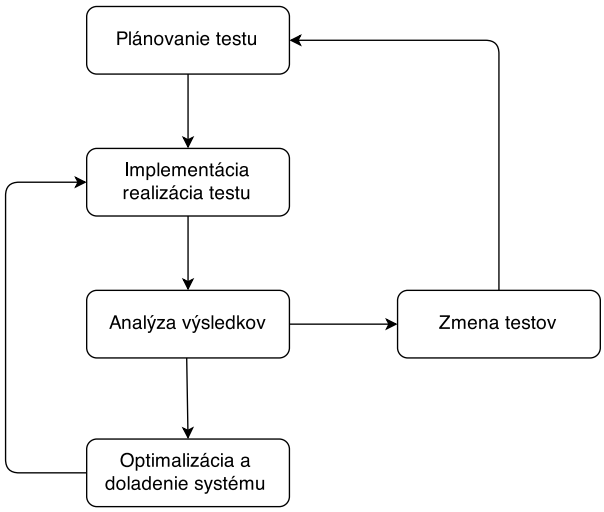
sme testovali na počítači v lokálnom klastri s parametrami v tabuľke 2. Proces testovania je zobrazný v diagrame na obr. 1.

Nástroj	**Verzia**
Apache Storm	0.9.4
Apache Kafka	0.8.2.1
Redis	2.8.19
ZooKeeper	3.4.6
Java	1.8.25
Ruby	jruby 1.7.19 (1.9.3p551)
twitter4j	4.0.3
org.json	20141113

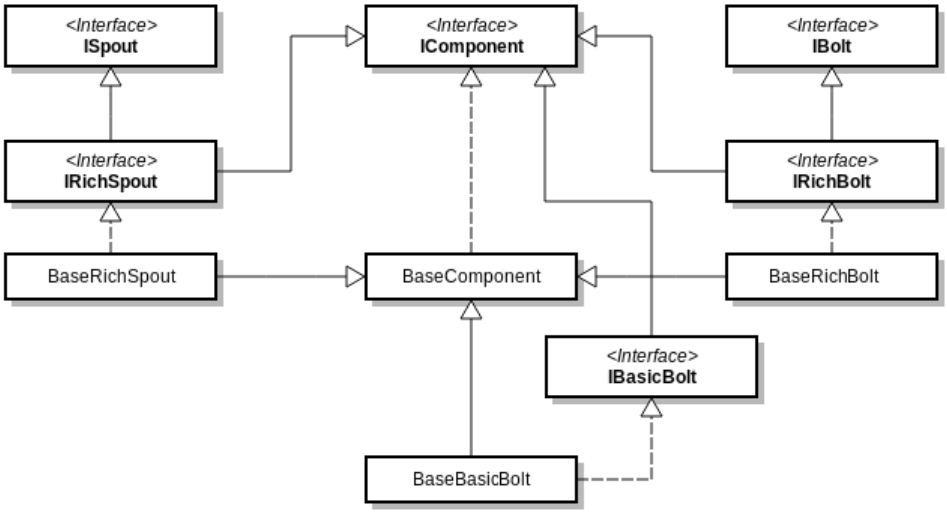
Tabuľka 1: Verzie použitých nástrojov, programovacích jazykov a rámcov.

Objekt	**Parametre**
Názov	MacBook Pro Late 2013
Operačný systém	OS X Yosemite 10.10.3
Procesor	2 x 2,4 GHz, Intel Core i5
Hlavná pamäť	8GB

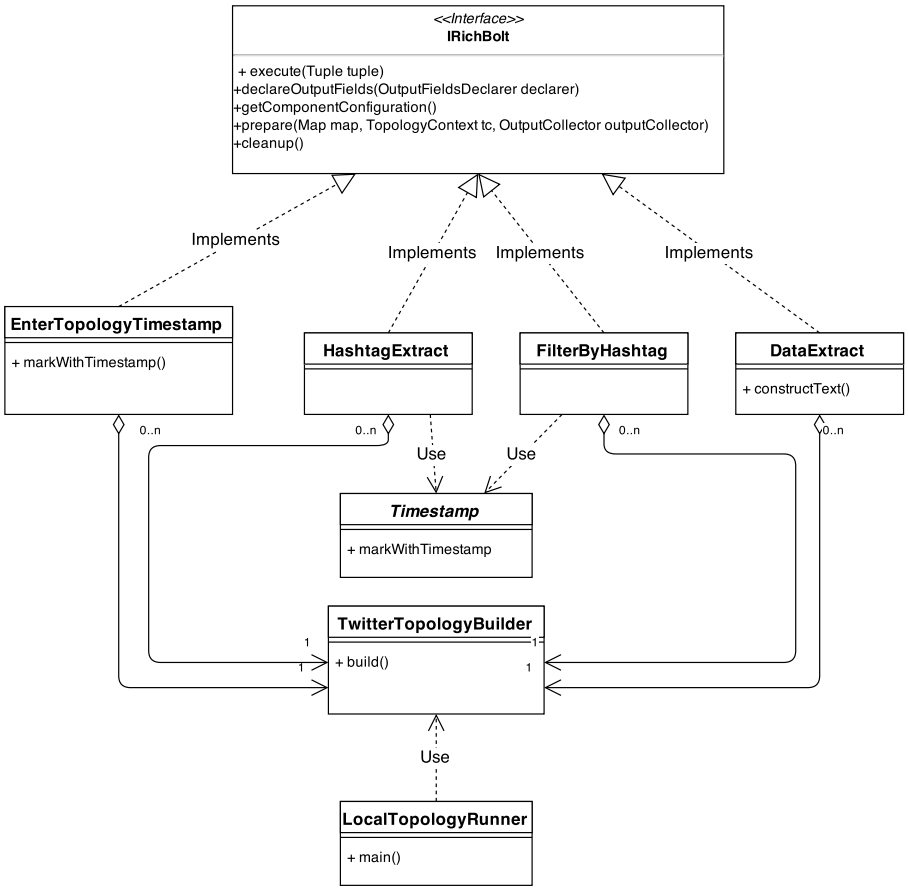
Tabuľka 2: Parametre počítača v lokálnom klastri.



Obrázok 1: Proces testovania.



Obrázok 2: Diagram tried rámca storm. Prevzaté z http://blog.zenika.com/.



Obrázok 3: Diagram tried.

B Inštalačná a používateľská príručka

Na spustenie programu na lokálnom klastri je potrebné najprv spustiť niekoľko iných systémov, tieto systémy su obsiahnuté na elektronickom médiu a sú potrebné pre spustenie aplikácie:

1. *./zookeeper-server-start.sh ../config/zookeeper.properties* pre štart ZooKeeper,
2. *./kafka-server-start.sh ../config/server.properties* pre štart Kafka,
3. *redis-server* Redis kľúč-hodnota databáza,
4. *./script/import.rb N dataset.in* simulácia prúdu dát, kde N je počet vlákien,
5. *./script/query.rb N queries.in* simulácia prúdu dopytov, kde N je počet vlákien.

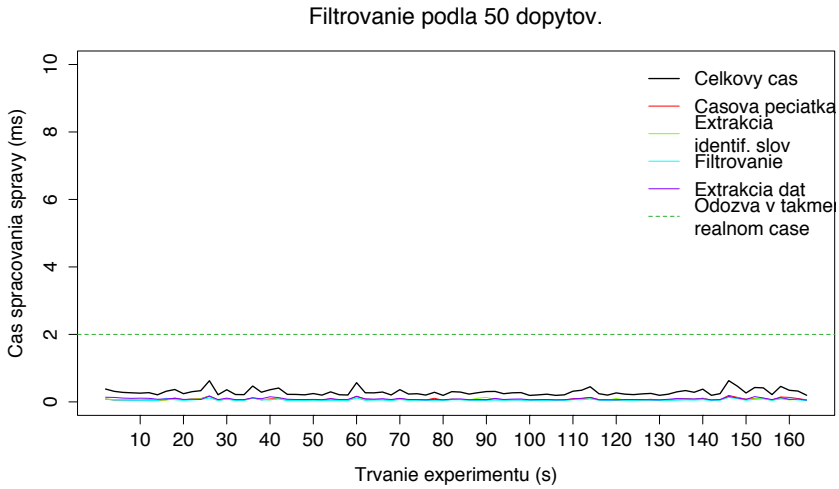
Pre spusteni samotnej aplikácie na spracovanie, resp. filtrovanie prúdu údajov je potrebné byť v koreňovom priečinku aplikácie a vykonať nasledujúce príkazy:

1. *mvn package*, kompilácia a vytvorenie spustiteľného súboru,
2. *java -cp StreamAnalysisFinal-2.0.2-jar-with-dependencies.jar edu.cimo.LocalTopologyRunner* pre spustenie topológie v lokálnom klastri.
3. *storm jar -c nimbus.host=nimbus.example.com StreamAnalysisFinal-2.0.2-jar-with-dependencies.jar edu.cimo.LocalTopologyRunner pre odoslanie topológie do vzdialeného klastra.*

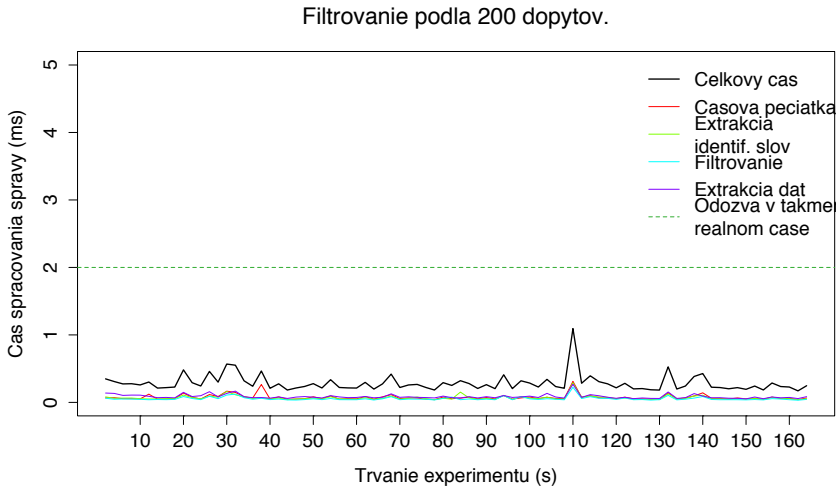
Po spustení program konzumuje správy z lokálnej inštancie Kafka komunikačného systému. V konfigurácii, ktorá je v návode konzumuje aj dopyty a simulovaný prúd dát z Twittru filtrouje podľa dopytov. Výsledky sú ukladané do databázy Redis na ďalšie spracovanie. My sme výsledky analyzovali ďalej pomocou programovacieho jazyka R.

C Grafy

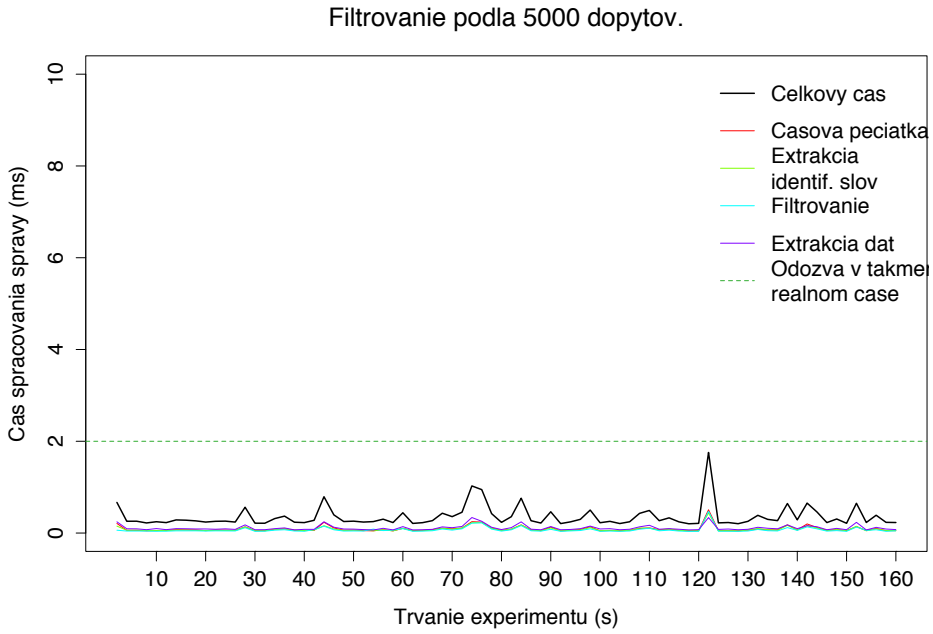
C.1 Filtrovanie viacerými dopytmi



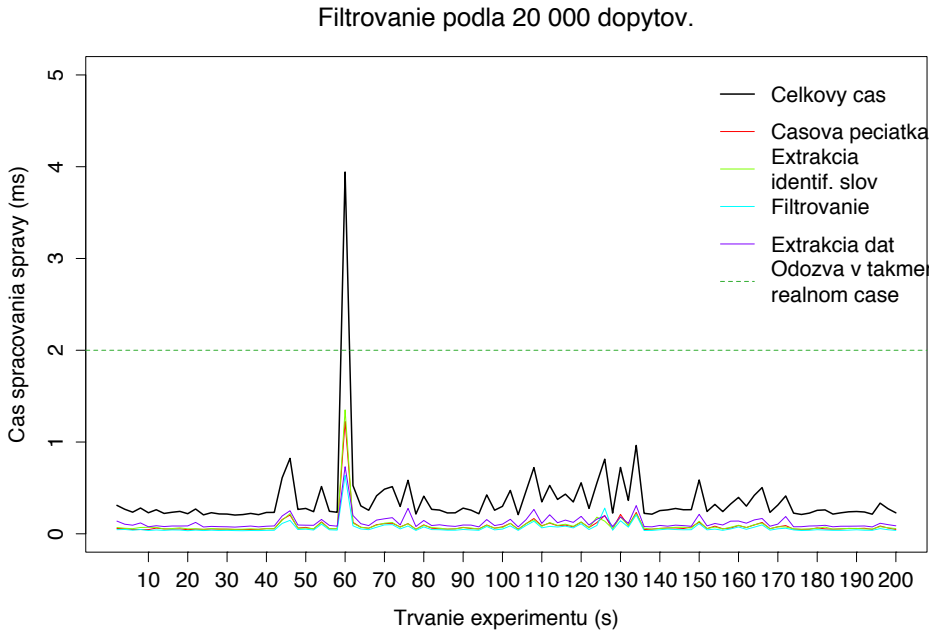
Obrázok 4: Filtrovanie 50 dopytmi pri duplikovaní dopytov naprieč topológiou.



Obrázok 5: Filtrovanie 200 dopytmi pri duplikovaní dopytov naprieč topológiou.

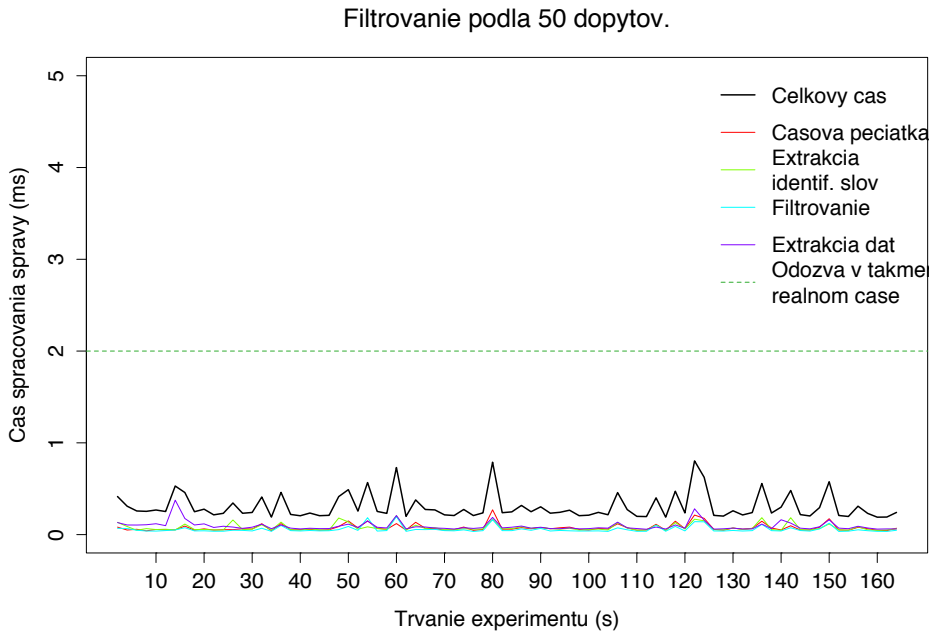


Obrázok 6: Filtrovanie 5000 dopytmi pri duplikovaní dopytov naprieč topológiou.

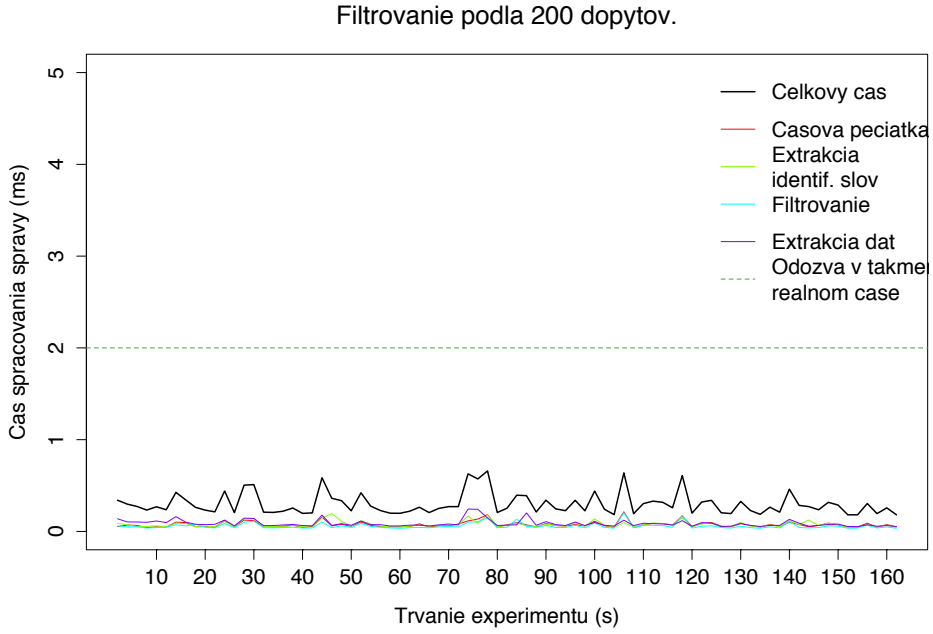


Obrázok 7: Filtrovanie 20 000 dopytmi pri duplikovaní dopytov naprieč topológiou. Špička v zobrazená v šesťdesiatej sekunda nastala pri spustení Java GC.

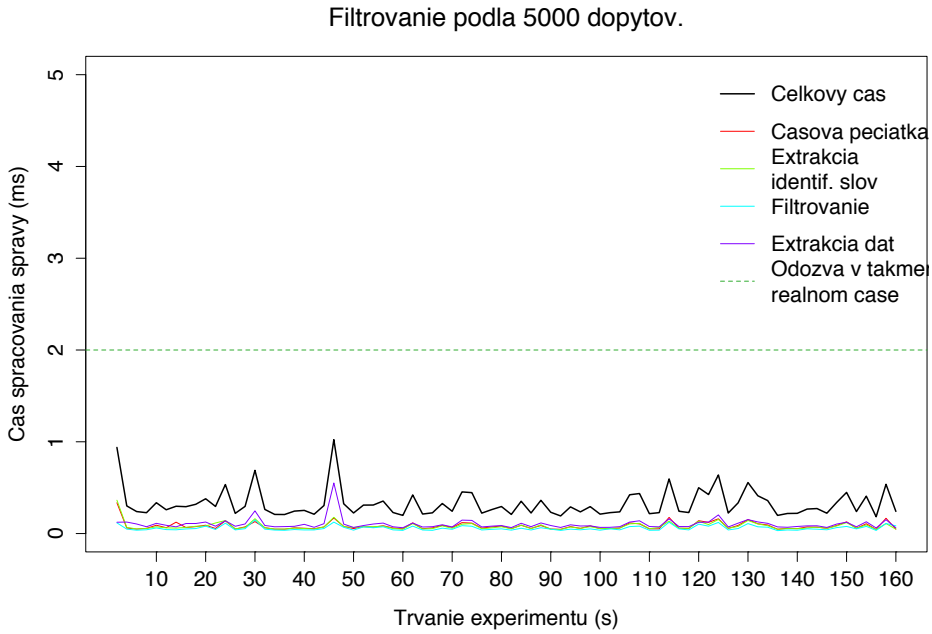
C.2 Smerovanie správ na základe dopytu



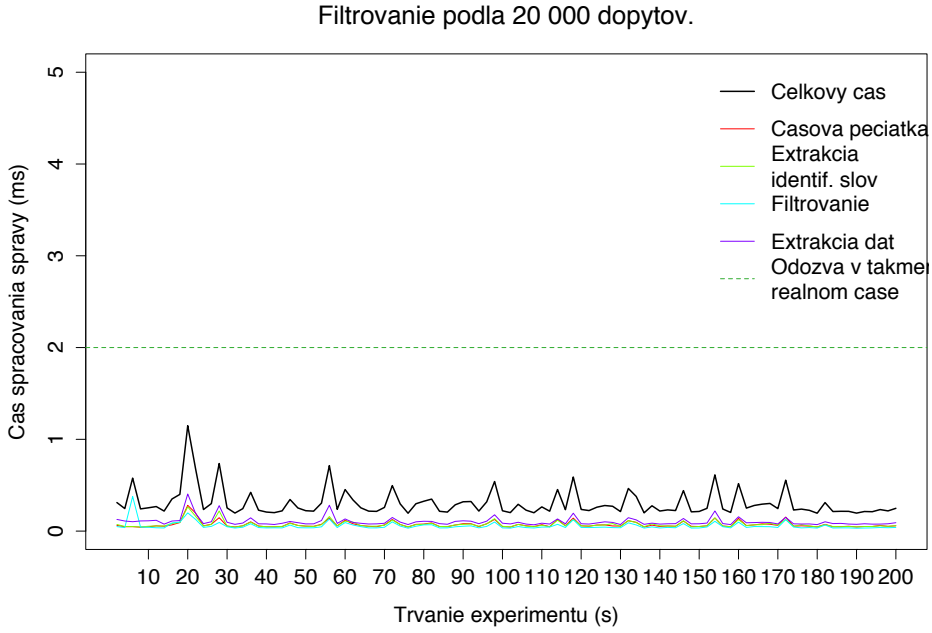
Obrázok 8: Filtrovanie 50 dopytmi, smerované správy naprieč topológiou.



Obrázok 9: Filtrovanie 200 dopytmi, smerované správy naprieč topológiou.



Obrázok 10: Filtrovanie 5000 dopytmi, smerované správy naprieč topológiou.



Obrázok 11: Filtrovanie 20 000 dopytmi, smerované správy naprieč topológiou.

D Obsah elektronického média

Prílohou tejto práce je elektronický nosič CD s nasledovným obsahom:

./files - množina testovacích dát a výsledky vo formáte CSV
./graphs - výsledné grafy experimentov
./res - nástroje potrebné pre spustenie produktu
./scripts - obsahuje skripty na simuláciu prúdu
./src - zdrojové súbory bez závislostí
./doc - obsahuje tento dokument v elektronickom formáte