VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO BIG DATA

APPLICATION FOR BIG DATA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE **Bc. MATÚŠ BLAHO**

AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. TOMÁŠ HRUŠKA, CSc.**

SUPERVISOR

BRNO 2018

Zadání diplomové práce/19307/2017/xblaho03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Blaho Matúš, Bc.**
Obor: Informační systémy
Téma: **Aplikace pro Big Data**
Application for Big Data
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s metodami pro podporu rozhodování.
2. Seznamte se s technologií zpracování Big Data (nejlépe systém Hadoop)
3. Navrhněte vhodné aplikace pro zpracování v systému pro Big Data.
4. Vybrané aplikace implementujte
5. Na vhodném vzorku dat aplikace otestujte a proveďte zhodnocení.

Literatura:

- White, T.: Hadoop - The Definitive Guide

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hruška Tomáš, prof. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá popisom a analýzou konceptu Big Data a ich spracovaním a využitím v procese podpory rozhodovania. Navrhované spracovanie vychádza z konceptu MapReduce navrhnutého pre spracovanie Big Data. Teoretická časť tejto práce z veľkej časti, pojednáva o systéme Hadoop, ktorý poskytuje implementáciu tohoto konceptu. Jeho pochopenie je kľúčovou vlastnosťou pre správny návrh aplikácií spúšťaných v tomto systéme. Práca tiež obsahuje návrh konkrétnych aplikácií na spracovanie Big Data. V implementačnej časti práce sa nachádza popis správy systému Hadoop, popis implementácie aplikácií MapReduce a popis ich testovania nad testovacími sadami dát.

Abstract

This work deals with the description and analysis of the Big Data concept and its processing and use in the process of decision support. Suggested processing is based on the MapReduce concept designed for Big Data processing. The theoretical part of this work is largely about the Hadoop system that implements this concept. Its understanding is a key feature for properly designing applications that run within it. The work also contains design for specific Big Data processing applications. In the implementation part of the thesis is a description of Hadoop system management, description of implementation of MapReduce applications and description of their testing over data sets.

Kľúčové slová

Big Data, Podpora rozhodvania, Hadoop, MapReduce, HDFS

Keywords

Big Data, Decision Support, Hadoop, MapReduce, HDFS

Citácia

BLAHO, Matúš. *Aplikace pro Big Data.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Tomáš Hruška, CSc.

Aplikace pro Big Data

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením Prof. Ing. Tomáša Hrušku, CSc. Uviedol som všetky zdroje, z ktorých som čerpal.

.....

Matúš Blaho

22. mája 2018

Poďakovanie

Chcel by som poďakovať Prof. Ing. Tomášovi Hruškovi, CSc. za odbornú pomoc, vedenie, ústretovosť a ochotu a mojím rodičom za nekončiacu podporu.

Obsah

Kapitola 1

Úvod

Pojem Big data sa v posledných rokoch začína v oblasti informačných technológií skloňovať čoraz častejšie a začína mu byť pripisovaná čoraz väčšia dôležitosť. Samotný pojem Big data vznikol v období 90 rokov minulého storočia, no k jeho rozšíreniu a väčšej popularizácií dochádza až v priebehu posledných rokov. Tento jav je spôsobený tým, že s postupom času cena hardvéru potrebného na uloženie masívnych dátových sád klesá, a preto sa ukladanie obrovských objemov dát stáva jednoduchšie uskutočniteľným. Samotné ukladanie takýchto dát bez ich následného spracovania by samozrejme strácalo zmysel. Avšak spracovanie a aj samotné ukladanie a prístup k takýmto dátam je mimo možnosti klasických systémov.

Pojem podpory rozhodovania nie je v oblasti informačných technológií žiadnou novinkou. Stretávame sa s ním už viac ako 30 rokov. Tak ako sa vyvíjajú informačné technológie, tak vývojom prechádzajú aj samotné systémy na podporu rozhodovania. Preto sa aktuálne ponúka, ako jeden z možných smerov rozvoja, využitie vyššie zmienených Big Data dát pre podporu rozhodovania.

2. kapitola práce obsahuje popis podpory rozhodovania z pohľadu informačných technológií. Obsahuje taktiež popis metódy Online Analytical Processing, ktorá ma momentálne veľmi široké využitie. Súčasťou tejto kapitoly je aj popis dátových skladov, čo je koncept a technológia definujúca, ako sú dáta v systémoch na podporu rozhodovania ukladané, a ako sa do nich dostávajú.

V kapitole číslo 3 tejto práce sú popísané vlastnosti konceptu Big Data. Hoci sa v názve nachádza slovo dáta nejedná sa len o samotné dáta, ale pod túto definíciu patria aj zdroje týchto dát, systémy na ukladanie a spracovanie veľkých dátových sád.

Pre správny návrh a implementáciu algoritmov na spracovanie veľkých dátovych sád v rámci Big Data je nutné detailne porozumieť platformám a konceptom ich spracovania. Preto kapitola MapReduce obsahuje detailný popis paradigma MapReduce, jeho výpočtový model a jeho pôvod z funkcionálneho programovania.

V kapitole číslo 5 je podrobne rozobraný konkrétny systém na ukladanie a spracovanie Big Data a to Apache Hadoop. Tento systém pozostáva z dvoch častí, kde prvou je Hadoop distributed file system, ktorý slúži na ukladanie dát a z časti, ktorá implementuje paradigma

MapReduce, pomocou ktorého spracováva tieto dáta. Obidve časti systému sú komplexné a zložité, z toho dôvodu sú podrobne rozoberané, lebo ich pochopenie, správna konfigurácia a správne využitie je nevyhnutným predpokladom pre tvorbu dobrých a efektívnych aplikácií na spracovanie Big Data.

Kapitola číslo 6 obsahuje návrh a popis konceptu systému, ktorý spája klasické systémy na podporu rozhodovania a Hadoop reprezentujúci systémy na spracovanie Big Data. V tejto kapitole je možné nájsť aj návrh a popis algoritmov na spracovanie Big Data, navrhnutých, tak aby ich výstupy bolo možné použiť v podpore rozhodovania.

Systém Hadoop predpokladá nasadenie vo výpočtovom klastri. Preto jeho správa a využívanie nie je triviálny úkon. Popis týchto aktivít spolu s implementačnými detailami navrhovaných aplikácií je možné nájsť v kapitole 7.

Kapitola 8 je zameraná na samotné testovanie výsledných aplikácií, ktoré boli spúšťané v reálne nasadenom systéme. Súčasťou kapitoly sú funkčné testy a experimentovanie so vstupnými dátovými sadami. Na konci kapitoly sa nachádza ich vyhodnotenie.

Kapitola 2

Podpora rozhodovania

Informácia a znalosť sú najhodnotnejšie prvky, potrebné pre tvorbu rozhodnutí. Informácia a znalosť vznikajú interpretáciou dát. Interpretáciu môže vykonávať človek alebo špecializovaný systém. V prípade, že sa o interpretáciu stará špecializovaný systém, hovoríme o systéme na podporu rozhodovania. Systém na podporu rozhodovania je špeciálny informačný systém, ktorý je navrhovaný tak, aby podporoval tvorbu riešení pre problémy v oblasti rozhodovania. Systémy na podporu rozhodovania sú zodpovedné za spracovanie, analyzovanie, zdieľanie a vizualizáciu dôležitých informácií s cieľom agregovať a transformovať znalosti, čím dochádza k zlepšeniu znalosti v rámci organizácie.

To, ako majú byť dáta spracované a transformované na znalosti, definuje Business Inteligence (BI). BI pokrýva všetky procesy zapojené do extrakcie hodnotných a užitočných informácií z masy dát, ktoré slúžia pre podporu rozhodovania [16] [11]. Z technického hľadiska BI obsahuje:

- Dátové sklady – časť starajúca sa o modelovanie, ukladanie, manažment a poskytovanie dát
- ETL – proces extrakcie (extraction), transformácie (transformation) a nahrávania (loading) do dátových skladov.
- Vizualizácia – implementácia grafického zobrazenia výsledných správ (reports) a nástenie (dashboards)
- Datamining – dolovanie znalostí z dát
- OLAP – online analytical processing, slúži na spracovanie údajov v dátovom sklade podľa užívateľských dotazov [18] [26].

Ďalšia časť kapitoly sa zaoberá popisom a definíciou toho, čo je reportovanie, OLAP a ako fungujú dátové sklady.

2.1 Reportovanie

Reportovanie (angl. reporting) je proces zbierania a prezentovania dát tak, aby mohli byť analyzované. Reportovanie je nedeliteľnou súčasťou bussiness intelignece a podpory rozhodovania. Výstupom procesu reportovania sú výsledné správy (reports). Z pohľadu tvorby ich môžeme rozdeliť na dva druhy, manažované a ad-hoc. Manažované reporty sú pripravované technickým personálom organizácie. O ich prípravu sa starajú vývojári, ktorí dostanú dopredu zadanú úlohu. Ad-hoc reporty sú tvorené koncovými užívateľmi systému na podporu rozhodovania. Reportovanie je dôležitou prerekvizitou analýzy dát. Umožňuje totiž užívateľovi zrozumiteľne vidieť dáta a na ich základe tvoriť analýzy dát, z ktorých potom získava znalosť pre rozhodnutia.[9]

2.2 OLAP

Online analytical processing je technológia, ktorá poskytuje dimenzionálny rámec pre podporu rozhodovania. Cieľom nástrojov OLAP je poskytnúť multidimenzionálne pohľady na dáta, ktoré sa za nimi nachádzajú. Medzi základné koncepty OLAP patria dimenzie a ich hierarchie. Dimenziou rozumieme usporiadateľnú množinu základného dátového typu alebo ich hierarchicky usporiadaných štruktúr. Dimenziou môže byť napríklad čas alebo lokalita. Viacero dimenzií tvorí multidimenzionálny model nazývaný aj dátová kocka. Dátová kocka poskytuje formálny model pre operácie nad dátami, ako sú agregácie a podobné výpočty. Jednotlivé bunky kocky môžu niesť informácie, napríklad koľkokrát sa atribút vyskytol v dátach, alebo maximum, súčet, priemernú hodnotu nejakého atribútu.[3][18][16]

2.2.1 Materializácia dátovej kocky

Materializácia je vypočítanie agregovaných hodnôt vopred [16]. Doba odpovede na užívateľské dotazy vo veľkej miere závisí na efektívnom výpočte dátovej kocky. Tieto výpočty sú však časovo aj priestorovo náročné. Preto jednou z využívaných techník je predpočítanie agregovaných hodnôt. Existujú 3 možnosti materializácie:

1. Materializácia celej dátovej kocky : Jedná sa o najlepšie riešenie z pohľadu rýchlosti užívateľských dotazov. Predpočítanie všetkých hodnôt je ale priestorovo príliš náročné a s veľkosťou kociek ešte narastá.

2. Žiadna materializácia: Agregované hodnoty sú počítané v čase dotazu. V tomto prípade rýchlosť vybavenia užívateľského dotazu závisí najmä na systéme uchovávajúcom dáta.

3. Čiastočná materializácia: Dochádza k vypočítaniu len niektorých hodnôt. Problémom tejto možnosti je však správna voľba, toho čo má byť predpočítané. [2]

2.2.2 Coddove pravidlá

Dr. Edgar F. Codd definoval 12 pravidiel, ktoré popisujú vlastnosti OLAP:

1. Multidimenzionálny a konceptuálny model – OLAP by mal poskytovať užívateľovi muldimenzionálny model tak, aby zodpovedal jeho potrebám a aby tento model mohol používať na analýzu zhromaždených údajov.

2. Transparentnosť – To že užívateľ môže plne využiť svoju produktivitu, odbornosť a samotné prostredie docielime tým, že technológie systému OLAP, jeho databázy a architektúra výpočtu budú transparentné.

3. Dostupnosť – Systém OLAP by mal pristupovať len k údajom, ktoré sú potrebné pre analýzu. Systém by mal navyše byť schopný pristupovať k týmto dátam bez ohľadu na to, z ktorého heterogénneho podnikového zdroja pochádzajú a ako často sú obnovované.

4. Stabilná výkonnosť – Užívateľ nesmie pocítiť zníženie výkonu, aj keď veľkosť databázy postupom času narastá.

5. Architektúra klient/server – OLAP musí fungovať na základe architektúry klient/server. Dôležitá je cena, výkon, flexibilita, interoperabilita.

6. Generická dimenzionalita – Každá dimenzia údajov musí byť ekvivalentná v štruktúre a operačných schopnostiach.

7. Dynamická manipulácia s riedkymi maticami – Systém OLAP musí byť schopný prispôsobiť svoju fyzickú schému na konkrétny analytický model, ktorý optimálne ošetrí riedke matice za súčasného udržania požadovanej úrovne výkonu.

8. Systém OLAP musí byť schopný podporovať viacero užívateľov, prípadne skupín užívateľov, pracujúcich súčasne na konkrétnom modele.

9. Neobmedzené operácie naprieč dimenziami – Systém OLAP musí rozoznať dimenzionálne hierarchie a automaticky vykonávať výpočty v rámci dimenzií a medzi dimenziami.

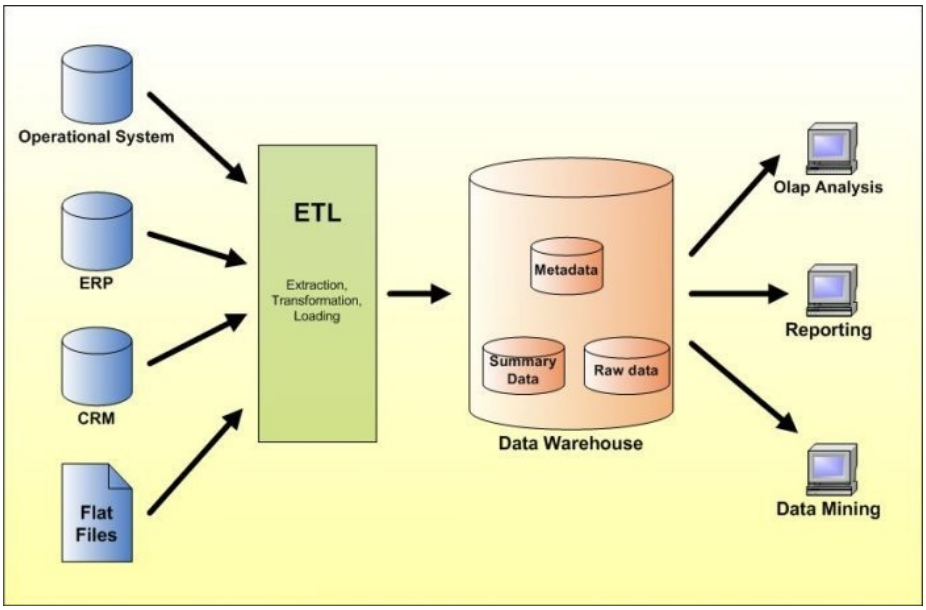
10. Intuitívna manipulácia s dátami – Užívateľské rozhranie musí zvládať intuitívne manipulácie v pre užívateľa prístupnom prostredí.

11. Flexibilný prístup – Schopnosť usporiadať stĺpce a bunky spôsobom, ktorý umožní analýzu a intuitívnu prezentáciu analytických zostáv.

12. Neobmedzené dimenzie a úrovne agregácie – V závislosti na požiadavkách procesu rozhodovania môže mať analytický model viacero dimenzií, pričom každá z nich môže mať viacúrovňovú hierarchiu. Analytický model by nemal byt umelo obmedzovaný počtom dimenzií a úrovňami hierarchie. [18]

2.3 Dátové sklady

Už zo samotného názvu dátové sklady (data warehouses) je možné odvodiť, že slúžia na ukladanie dát, avšak ich účelom nie je dáta len uchovávať. Dátovým skladom nazývame technológiu na nahrávanie, ukladanie a poskytovanie dát pre podporu rozhodovania vykonávanú analýzou informácií a vytváraním znalostí. [16]. Definícia dátového skladu Billa Inmona z roku 1990 hovorí, že dátový sklad je subjektovo orientovaná, integrovaná, časovo premenná a perzistentná kolekcia dát pre podporu procesu rozhodovania. Subjektová orientácia hovorí, že dáta sú zamerané na špecifický subjekt a nie na aktuálne prebiehajúce operácie v spoločnosti. Integrovanosť dátovej sady spočíva v tom, že dáta v rámci nej sú získané a integrované z viacerých zdrojov. Časová predmetnosť znamená uchovávanie historických dáta tak, že ich podoba je uchovávaná vzhľadom na jednotlivé časové body. Perzistentosť dátovej sady znamená to, že dáta uložené v rámci dátovej sady nie sú nijak modifikované ani mazané. [4] Z týchto vlastností vyplýva, že klasické produkčné databázy, nazývané aj OLTP (Online transactional processing), nie sú využiteľné ako dátové sklady. Vo väčšine prípadov slúžia totiž na uchovanie dát, ktoré zachytávajú stav organizácie v aktuálnom časovom bode. Pre analýzu dát sú potrebné dáta historické, ktorých ukladanie v rámci týchto systémov by viedlo k degradácií výkonu. Ich účelom je sprostredkovať rýchle spracovanie transakcií zameraných na vkladanie, odstraňovanie, upravovanie a podobné operácie. Systém OLTP býva väčšinou decentralizovaný, čo znamená, že dáta v rámci neho sú ukladané na viacerých heterogénnych serveroch. Pre analýzu dát by však bolo potrebné tieto dáta homogenizovať a integrovať, čo by bolo zložité. [16] Schému dátových skladov zobrazuje obrázok 2.1.



Obr. 2.1: Schéma dátového skladu [4]

2.3.1 Zdroje dát a ich spracovanie

Zdroje dát môžu byť produkčné, interné a externé. Medzi produkčné zdroje dát patria produkčné databázy, z ktorých sú dáta získavané pomocou databázových dotazov. Medzi dáta interné môžu patriť dáta uložené zamestnancami firmy v rôznych súboroch. Zdrojmi externých dát môžu byť napríklad weby slúžiace na interakciu firmy s vonkajším prostredím [16].

S pojmom dátových skladov je veľmi úzko spätý proces ETL. Ako už bolo vyššie zmienené, jedná sa o extrakciu, transformáciu a nahratie dát do dátového skladu. V následnej časti si vysvetlíme jednotlivé fázy ETL:

- **Extrakcia** dát je fáza, pri ktorej dochádza k extrahovaniu a zberu dát zo zdrojov. Extrakcie poznáme: Jednorázova extrakcia, kedy dochádza k extrakcii len raz. Občasná, počas ktorej dochádza k extrakcií dát v redších nepravidelnych intervaloch. Periodická, keď sú dáta získavané v periodicky sa opakujúcich časových momentoch.

- **Transformácia** dát je fáza, ktorej úlohou je zvýšiť kvalitu vstupných dát. Počas transformácie dát dochádza k ich čisteniu, zjednoteniu používanej terminológie, formátov dát, dopĺňanie časových údajov, keďže tieto údaje sa môžu pri rôznych zdrojoch líšiť alebo úplne chýbať.

- **Nahratie** je finálnou fázou ETL. Počas nej dochádza k samotnému uloženiu extrahovaných a transformovaných dát do tabuliek dátového skladu. Nahrávanie dát môže prebiehať tromi spôsobmi. Prvý spôsob je iniciálne nahranie, keď dochádza k nahratiu všetkých dát do prázdneho skladu. Inkrementálne nahratie je druhým spôsobom, pri ktorom dochádza k doplneniu zmien reflektujúcich zmeny v zdroji. Posledným spôsobom je prepis dát, kedy sú dáta v sklade kompletne zmazané a prepísané aktuálnymi dátami. [16]

Kapitola 3

Big Data

Ako už zo samotného názvu vyplýva, Big Data sú veľké dáta. Aby sme boli ešte presnejší dáta tak veľké a komplexné, že ich spracovanie dostupnými databázovými prostriedkami alebo klasickými aplikáciami sa stáva obtiažnym [16]. Preto je potrebné pre spracovanie Big Data hľadať nové spôsoby a technológie. Danú transformáciu a potrebu nových metód môže ilustrovať napríklad to, že podľa IBM bolo 90% dát, ktoré máme k dispozícií, vyprodukovaných v posledných 2 rokoch. S rozvojom informačných technológií a ich postupným prenikaním do ďalších a ďalších oblastí sa dá očakávať ešte ďalšie zvýšenie produkcie dát. Predpokladá sa, že v roku 2020 bude každá ľudská bytosť produkovať 1,7 megabajtu dát každú sekundu, v tom čase bude rozsah nami naakumulovaných dát 44 zettabajtov. Takéto objemy dát sú ťažko predstaviteľné, ale je potrebné si uvedomiť, že ďalšie prognózy hovoria, že v roku 2020 bude vlastniť smartfóny, so senzormi schopnými zaznamenávať a zbierať dáta, 6,1 miliardy užívateľov a do internetu bude pripojených cez 80 miliárd inteligentných zariadení. Ďalšou oblasťou produkujúcou obrovské objemy dát sú sociálne siete. Len samotný Facebook mal v treťom kvartáli roku 2017 viac ako 1,3 miliardy denne aktívnych užívateľov. Podľa štatistík bolo začiatkom roku 2017 vytvorených denne 500 miliónov takzvaných tweetov na sociálnej sieti Twitter. Ďalšími možnými zdrojmi rozsiahlych dát môžu byť rôzne transakcie, či už online alebo klasické no elektronicky archivované, záznamy o aktivitách užívateľov na webových stránkach, medicínske záznamy a podobne.

3.1 Vlastnosti Big Data

Najznámejšou skupinou vlastnosti Big Data je definícia vlastností podľa „V“. Táto definícia vychádza z charakterizovania dát anglickými slovami začínajúcimi na písmeno „V“. Ich počet sa podľa rôznych zdrojov líši. Základná charakteristika známa ako 3 V uvádza Volume, Velocity, Variety. Ďalšie charakteristiky rozširujú 3 V o Veracity a Value.[21][27]

- Volume – objem, popisuje dáta z pohľadu veľkosti ich rozsahu. Existuje zhoda, že keď sa jedná o objem dát vo veľkosti gigabajtov ešte sa o Big Data nejedná. No keď sa objem dostáva na úroveň terabajtov, už hovoríme o Big Data. Obrovský objem

dát je jeden z hlavných dôvodov, prečo na spracovanie Big Data už nestačia klasické databázové systémy.

- Velocity – rýchlosť produkcie dát. V predchádzajúcom odseku sú uvedené príklady rýchlosti vzniku dát.
- Variety – rôznorodosť. Zameriava sa na definovanie toho, že dáta môžu pochádzať z rôznych zdrojov, obsahujú rôzne dátové typy a sú ukladané v rôznych formátoch.
- Veracity – pravdivosť dát. Definuje to, že dáta musia pochádzať z dôveryhodného zdroja.
- Value – hodnota. Nejedná sa o vlastnosť, ktorú majú len Big Data, ale všeobecnú vlastnosť toho, že dáta, ktoré uchovávame by mali mať pre nás hodnotu a ich spracovanie by malo byť prínosom. Vzhľadom na veľkosť dát v rámci Big Data je ich uchovávanie náročnejšie ako pri malých dátach, a preto je to, aby produkovali pre nás hodnotu, veľmi podstatné [27][11].

3.2 Data Lake

S pojmom Big Data sa začína objavovať aj nový pojem Data Lake. Jedná sa o pomerne nový pojem, pre ktorý neexistuje presná definícia. Spoločností zameraná na spracovanie veľkých dát k nemu pristupuje rôzne. Väčšina prístupov sa však zhoduje v tom, že Data Lake je centralizovaný repozitár umožňujúci ukladanie štruktúrovaných a neštruktúrovaných dát a ich analýzu. [22][10] Plní podobnú funkciu ako dátové sklady popisované v kapitole 2.3. Data Lake a dátové sklady sa však líšia spôsobom, akým pristupujú k ukladaniu dát. Účelom Data Lake nie je nahradiť dátové sklady, ale fungovať ako ich doplnok. Vlastnosti Data Lake si zhrnieme na ich porovnaní s dátovými skladmi.

V rámci dátových skladov sú ukladané dáta, ktoré prešli procesom extrakcie, transformácie a uloženia. To je v kontraste so spôsobom ukladania dát v rámci Data Lake, kde sú dáta uchovávané „tak ako sú". To znamená, že dáta neprechádzajú procesom transformácie pri ukladaní a sú ukladané vo forme v akej boli získané. Preto sa v Data Lake môžu súčasne nachádzať dáta štruktúrované aj neštruktúrované. Dátové sklady využívajú princíp nazývaný schéma pri zápise. Najprv sa definuje schéma úložiska. Pri ukladaní sú dáta najprv štruktúrované podľa definovanej schémy a až následne ukladané. To vyžaduje znalosť dát, ich účelu, vstupného formátu a návrh ich transformácie už pri ich ukladaní. Pri Data Lake sa naopak využíva princíp nazývaný schéma pri čítaní. Dáta sa zapisujú bez transformácií. Schéma a štruktúra sú im priradené až pri ich čítaní za istým účelom [15]. To umožňuje ukladať dáta, ktorých účel nemusíme poznať, ale predpokladáme ich využitie až v budúcnosti. Tento prístup je vhodný pre ukladanie veľkých dát, pretože prichádzajú z rôznych zdrojov, v rôznych formátoch a vo veľkom množstve. Transformácia, čistenie a štrukturalizovanie dát už pri ukladaní, by pri Big Data bola výpočtovo príliš náročná, priam nezvládnuteľná

úloha. Rozdiel v uchovávaní dát má za následok rozdielne nároky na hardvér, používaný v jednotlivých systémoch. Dátové sklady sú postavené na nižšom množstve strojov s cenovo náročnejšom hardvérom, umožňujúcom rýchle obslúženie dotazov, zatiaľ čo koncept Data Lake je postavený na veľkom počte strojov s lacným hardvérom, umožňujúcim ukladanie obrovských dát a zrýchlenie dotazov vďaka paralelizmu. Takisto požívatelia týchto systémov sú rozdielni. Zatiaľ, čo užívateľmi dátových skladov sú biznis analytici, užívateľmi Data Lake sú dátoví vedci, dátoví developeri a až po výstupe ich práce sa nimi stávajú aj samotní biznis analytici. [13]

Vyššie popísané vlastnosti prinášajú výhody, ale aj nevýhody. Medzi výhody patrí flexibilita týchto systémov. Je jednoduché pridávať do nich nové zdroje dát, ktoré sa majú uchovávať. Takisto sa jednoducho rozširujú a implementujú nové požiadavky analytikov, alebo pridávajú nové analýzy nad dáta, ktoré boli doposiaľ len ukladané. Medzi nevýhody patrí to, že ukladanie obrovských objemov dát komplikuje hľadanie užitočných informácií a takisto môže v extréme dopadnúť tak, že síce budeme mať uložených mnoho dát, ale nebudú mať pre nás žiadnu hodnotu.

3.3 Metódy spracovania Big Data

Obrovské objemy dát si vyžadujú iné prístupy na spracovanie ako klasické dáta. Spracovanie Big Data môžeme rozdeliť na dve metódy a to podľa spôsobu, ako k spracovaniu dochádza a času, koľko trvá, kým dodajú výsledok, na:

- Batch processing – jedná sa o dávkové spracovanie. Aplikácie na dávkové spracovanie sú navrhnuté tak, aby spracovávali veľké množstvá historických dát distribuovane a paralelne. Výsledky týchto výpočtových operácií nie sú dostupné okamžite, ale ich spracovanie trvá niekoľko hodín, prípadne aj dní. Použitie tejto metódy spracovania veľkých dát je vhodné napríklad na plánovanie a určovanie stratégií, čo sú úkony, ktoré samé trvajú nejakú dobu a nepotrebujú výsledky analýz okamžite. Sila týchto systémov spočíva v tom, že dokážu doručiť výsledky požadovaných analýz nad historickými dátami, ktoré boli zbierané roky prípadne desaťročia.

- Stream processing – systémy na streamové spracovanie dát sú vhodné pre aplikácie, ktoré potrebujú neprestajne analyzovať veľké objemy živých dát s nízkou latenciou. Ich spôsob spracovania dát sa zameriava na to, že spracovávajú dáta ako prúdy dát. Prúd dát môžeme definovať ako kontinuálny prísun prichádzajúcich dátových záznamov porovnateľný s dátovým kanálom. Tieto systémy spracovávajú dáta v reálnom čase, prípadne čase blízkemu reálnemu času. Takéto systémy sú vhodné na spracovanie dát z rôznych snímačov pre systémy na detekciu hrozieb a iných okamžitých udalostí [14].

3.4 Vlastnosti systémov pre spracovanie a ukladanie Big Data

Big Data sa však nevzťahujú len na samotné dáta a ich definíciu, ale daný koncept zavádza aj technologické požiadavky na spracovanie a ukladanie takýchto dát. Ako už bolo vyššie zmienené, systémy na spracovanie Big Data sa musia od klasických databázových systémov líšiť, aby boli schopné spracovávať dáta v akceptovateľnom čase. Preto tieto systémy musia mať nasledujúce vlastnosti:

- Distribúcia dát – veľký set dát je najprv rozdelený na menšie časti alebo bloky, ktoré sú následne distribuované na viacero uzlov respektíve strojov.
- Paralelné spracovanie – dáta sú spracovávané paralelne na stroji, kde sú uložené, čo zvyšuje výpočtovú silu N-násobne podľa počtu strojov v klastri. Po spracovaní dát na jednotlivých strojoch dochádza k zlučovaniu dát do výsledku.
- Zotavenie z chýb – dochádza k replikácií jednotlivých dát na viacero strojov. V prípade zlyhania jedného stroja nám replikácia dát umožňuje prístup k dátam uloženým na inom stroji.
- Použitie cenovo dostupného hardvéru – pre vysokú schopnosť tolerancie chýb je možné prevádzkovať takýto systém na hardvéri nízkej ceny a nie je potrebný žiadny špecializovaný hardvér.
- Flexibilnosť a škálovateľnosť – architektúra systému umožňuje jednoduché pridávanie a odoberanie uzlov z a do výpočtového klastra podľa potreby [20].

Jedným z takýchto systémov je aj Apache Hadoop, o ktorom pojednáva kapitola číslo 5.

Kapitola 4

MapReduce

MapReduce je abstraktný model paralelného programovania, navrhnutý na spracovanie veľkých setov dát na klastri počítačov, pôvodne vyvinutý spoločnosťou Google.

4.1 Základný princíp

Základným princípom modelu MapReduce sú dve fázy. Prvou je fáza nazývaná map a druhá sa nazýva reduce. Zmieňované fázy majú pôvod v rovnomenných funkciách pochádzajúcich z funkcionálneho programovania. Zjednodušene sa dá povedať, že jednotlivé fázy MapReduce fungujú nad kľúčmi a im priradenými hodnotami [17] [24]. V nasledujúcich podkapitolách si vysvetlíme ako presne fungujú.

4.1.1 Fáza map

Funkcia map

Pre vysvetlenie fungovania funkcie map si použijeme jej implementáciu v programovacom jazyku Haskell. Funkcia map vracia zoznam prvkov vytvorený aplikovaním funkcie, ktorú dostane ako prvý parameter, na všetky prvky zo zoznamu, ktorý dostane ako druhý parameter [5]. Z formálneho hladiska môžeme funkciu map definovať tak, že jej vstupom je funkcia f a zoznam prvkov x_1, x_2 až x_n a výstupom je zoznam prvkov $f(x_1), f(x_2)$ až $f(x_n)$.

Princíp fázy map

Vstupom fázy map sú samotné dáta reprezentované ako dvojice kľúč a hodnota. Jej výstupom je zoznam párov kľúč a hodnota. Počas fázy map je nad každým párom kľúč-hodnota spustená užívateľom definovaná funkcia fázy map [17]. Pre lepšiu predstavu si vysvetlíme fázu map na príklade. Predpokladajme, že implementujeme v MapReduce počítanie výskytu slov v dokumente. V takomto prípade bude vstupom funkcie vo fáze map dvojica číslo riadku reprezentujúce kľúč a hodnotou bude obsah jedného riadku. Funkcia prejde riadok a pre každé slovo vytvorí dvojicu kľúč-hodnota nasledovne: kľúčom bude samotné

slovo a hodnota bude 1, čiže jeho výskyt. Po skončení fázy map teda bude v systéme zoznam dvojíc obsahujúcich každé slovo v dokumente ako kľúč a rovnakou hodnotou 1.

4.1.2 Fáza reduce

Funkcia reduce

Pre vysvetlenie fungovania funkcie reduce si použijeme jej implementáciu v programovacom jazyku LISP. Funkcia reduce využíva binárnu operáciu na skombinovanie prvkov zoznamu. Binárna operácia je teda prvým parametrom funkcie a samotný zoznam druhým [1]. Z vnútorného hľadiska si jej priebeh môžeme predstaviť nasledovne $f(..f(f(x_1, x_2)x_3)..x_n)$, kde f je funkcia, a x_1 až x_n sú prvky zoznamu.

Princíp fázy reduce

Vstupom fázy reduce je dvojica kľuč a zoznam hodnôt. Jej výstupom je dvojica kľúč-hodnota. Čiže funkcia fázy reduce je spúšťaná nad každou dvojicou kľúč zoznam hodnôt a vykonáva nad hodnotami asociovanými s jednotlivými kľúčmi redukciu alebo agregovanie [24]. Pre lepšie pochopenie si vysvetlíme fungovanie fázy reduce na pokračovaní príkladu použitom pri vysvetľovaní fázy map. Vstupom pre každú funkciu vo fáze reduce bude dvojica kľúč, ktorého hodnota bude unikátne slovo nájdené zo vstupného dokumentu, a zoznam hodnôt 1 s dĺžkou rovnej počtu výskytov slova v dokumente. Funkcia vykoná operáciu súčet nad týmito hodnotami a na výstup zapíše dvojicu slovo a výsledok súčtu ako dvojicu kľuč hodnota, ktoré sú aj výsledkom celého implementovaného algoritmu.

4.1.3 Fáza shuffle

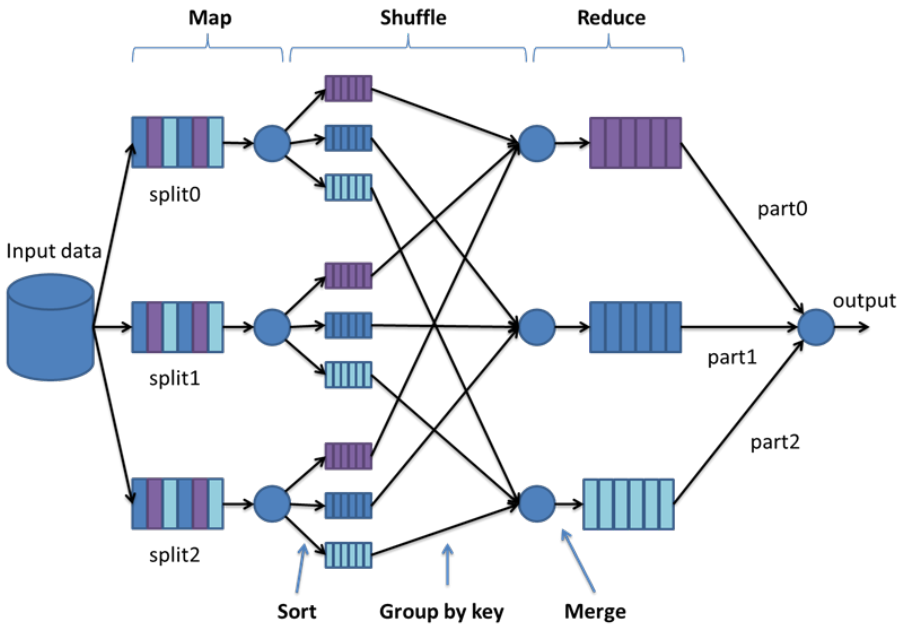
Z podkapitol 4.1.1 a 4.1.2 vyplýva, že výstup funkcie vykonávanej vo fáze map nie je totožný so vstupom funkcie fázy reduce. Preto je medzi fázu map a reduce ešte vložená fáza shuffle. Počas nej sa najprv celý zoznam dvojíc kľúč a hodnota triedi podľa kľúča. Po utriedení sa všetky hodnoty s totožným kľúčom zlúčia do jedného zoznamu, čím vznikne dvojica kľúč a zoznam hodnôt, vystupujúca na vstupe fázy reduce. Táto časť nie je navrhovaná ani implementovaná užívateľom a plne sa o ňu stará výpočtový model MapReduce.[24]

4.2 Paralelizmus v MapReduce

Návrh MapReduce umožňuje paralelný beh výpočtu vo veľkom výpočtovom klastri. Funkcia fázy map môže byť vykonávaná paralelne, pre každý pár kľúč-hodnota zo vstupu, a môže byť vykonávaná priamo na uzle, kde sa dané dáta nachádzajú. To značne redukuje čas potrebný na výpočet a taktiež množstvo prenášaných dát. Prakticky to znamená, že program sa posiela tam, kde sú dáta, a nie dáta k programu.

Časť triedenia fázy shuffle môže prebiehať paralelne nad jednotlivými výstupmi funkcií fázy map. Dáta sa prenášajú medzi strojmi v klastri až medzi krokmi triedenia a zlúčenia fázy shuffle. Časť zlúčenia môže byť vykonávaná paralelne pre každý unikátny kľuč.

Funkcia fázy reduce opäť môže byť vykonávaná paralelne pre každú unikátnu hodnotu kľúča.[17][24] Obrázok 4.1 zobrazuje paralelný beh MapReduce.



Obr. 4.1: Schéma paralelného behu MapReduce [24]

Kapitola 5

Apache Hadoop

Apache Hadoop je opensource projekt pod záštitou The Apache software foundation, ktorý umožňuje distribuované spracovanie a ukladanie rozsiahlych objemov dát. Ako bolo vyššie spomenuté, tento software spĺňa požiadavky na vlastnosti systémov pre spracovanie Big Data. Distribúciu dát, zotavovanie sa z chýb a replikáciu dát zabezpečuje časť systému nazývaná Hadoop Distributed File System (HDFS), ktorá implementuje paradigma distribuovaného súborového systému. Paralelné spracovania dát, v rámci platformy Hadoop, zastrešuje Hadoop YARN a implementácia MapReduce, preberaného v kapitole 4. Hadoop MapReduce obsahuje rôzne knižnice na spracovanie dát vo viacerých programovacích jazykoch. V rámci tohto diela bude preberaná ich implementácia v jazyku Java. Hadoop YARN sa stará o plánovanie a sledovanie úloh z MapReduce spúšťaných vo výpočtovom klastri. HDFS spolu s MapReduce a YARN sú navrhnuté tak, aby mohli byť spúšťané na výpočtových klastroch, ktoré sú zložené z cenovo dostupného hardvéru, nazývaného „*commodity hardware*“. Ďalšou z vlastností, ktoré Hadoop podporuje, je škálovateľnosť. Systém môže byť teoreticky nasadený aj len na jeden stroj. V tom prípade ale prichádzame o možnosť paralelného spracovania, distribúcie a replikácie dát. Nasadenie na mnoho strojov je preto bežnejšie, pričom môže ísť až o tisíce, ako napríklad v spoločnosti Yahoo, kde sa nachádza výpočtový klaster s veľkosťou 4500 strojov [8]. V ďalšom texte sa budeme podrobnejšie venovať jednotlivým častiam systému Hadoop.

5.1 Hadoop distributed file system

Ako bolo vyššie spomenuté, HDFS je implementácia distribuovaného súborového systému. Jednou z výhod tohto prístupu k ukladaniu dát je skutočnosť, že môžeme ukladať súbory, ktorých veľkosť presahuje veľkosti jednotlivých úložných zariadení distribuovaného súborového systému. Problémom je ale to, že so zväčšovaním výpočtového klastru sa bude zvyšovať pravdepodobnosť zlyhania niektorého zo strojov vo výpočtovom klastri. Preto je ďalšou z vlastností HDFS odolnosť voči chybám. Táto vlastnosť je zabezpečená replikáciou dát. HDFS preto jednotlivé súbory rozdeľuje na bloky a tie následne zapisuje a replikuje

na rôzne stroje v klastri. V podkapitole 5.1.1 je popísané akým spôsobom sú dáta ukladané z pohľadu rozdelenia a replikácie. Podkapitola 5.1.2 pojednáva o architektúre systému z pohľadu aktívnych prvkov, ktoré sa o tento zápis a replikáciu starajú. V podkapitole 5.1.3 sú zhrnuté silné a slabé stránky systému, ktoré z jeho návrhu a implementácie vyplývajú.

5.1.1 Ukladanie a replikácia dát

Spôsob ukladania súborov do blokov

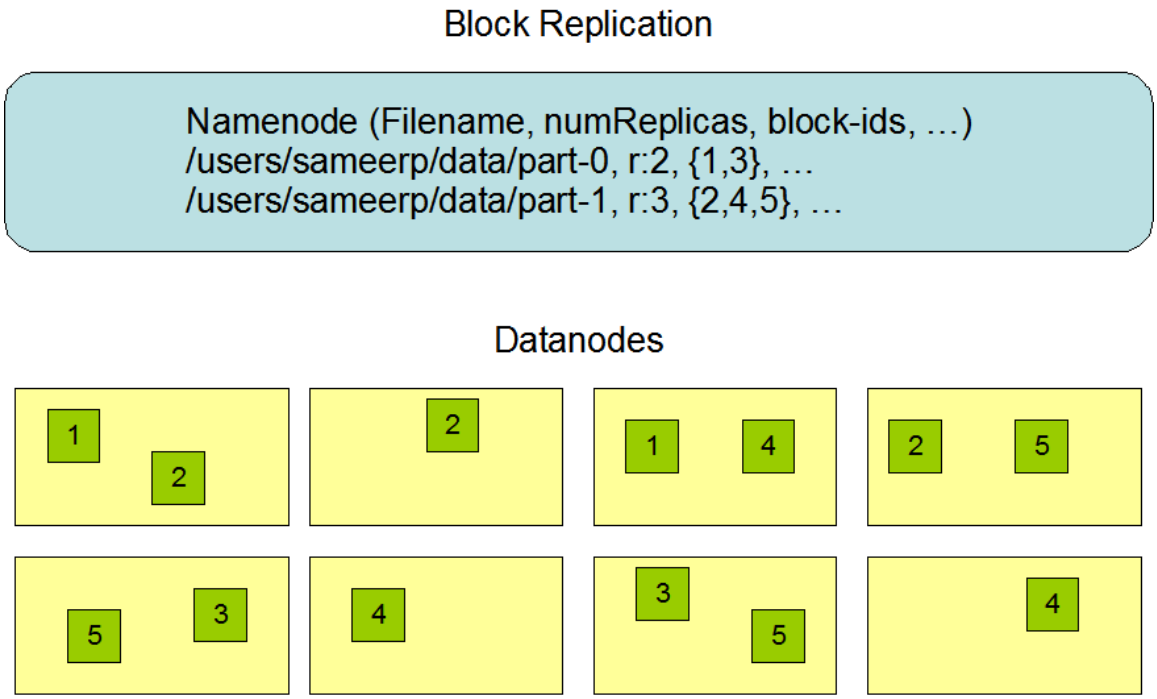
Tak, ako klasické súborové systémy, aj HDFS používa systém ukladania súborov spôsobom, že ukladaný súbor je rozložený do nezávislých blokov, ktoré sú následne zapísané do rôznych miest súborového systému. Rozdiel však spočíva vo veľkosti jedného bloku. Kým pre súborové systémy je veľkosť bloku v rozsahu kilobajtov, HDFS používa pri základnom nastavení veľkosť bloku 128 MB. Súbor menší ako veľkosť bloku, na rozdiel od klasických súborových systémov, nezaberá na úložnom zariadení celú veľkosť bloku, ale len skutočnú veľkosť súboru. Jeho zapísaním však dochádza ku konzumácií bloku, ktorých je limitované množstvo. K problému malých súborov sa vrátime pri zhrnutí vlastností systému v podkapitole 5.1.3.

Motiváciou k využívaniu veľkých blokov je redukcia ceny hľadania blokov. Preto pri dostatočne veľkom bloku je prenos samotného bloku mnohokrát dlhší ako jeho vyhľadanie. Ilustrujme to nasledujúcim výpočtom. Predpokladajme dobu hľadania 10 ms a prenosovú rýchlosť 100 MB/s. Ak chceme, aby doba hľadania bola 1 % z doby prenosu, jednoduchým výpočtom sa dostaneme k veľkosti bloku 100 MB. Preto je základná veľkosť bloku 128 MB, aj keď mnohé inštalácie HDFS používajú ešte väčšie bloky. Za predpokladu zvyšovania sa rýchlosti čítania moderných diskov sa dá očakávať nárast veľkosti blokov [28][12].

Replikácia dát

Vyššie zmienený koncept ukladania dát do blokov je vhodný pre proces replikácie. Replikácia nám zabezpečuje toleranciu chýb a dostupnosť dát. Z cieľom zabezpečiť funkčnosť aj v prípade zlyhania stroja alebo poškodenia disku, respektíve jeho blokov, je každý blok replikovaný na viacero fyzicky separátnych strojov. V prípade, že je nejaký blok nedostupný, jeho kópia môže byť prečítaná z inej lokality spôsobom transparentným pre klienta. V prípade poškodenia alebo nedostupnosti stroja, na ktorom je blok uložený, môže byť daný blok replikovaný zo záložnej kópie, aby sa obnovil faktor replikácie na pôvodnú hodnotu [28]. Replikácia súboru naprieč klastrom je ilustrovaná na obrázku 5.1.

V rámci HDFS je v základnom nastavení hodnota replikácie nastavená na hodnotu 3. Toto nastavenie umožňuje prístup ku všetkým dátam aj v prípade, že dôjde k zlyhaniu dvoch strojov s replikovanými dátami. Hodnota nastavenia počtu replík je nastaviteľná v intervale od 1 do 512. Aplikácie môžu túto hodnotu nastavovať pre každý súbor osobitne [28].



Obr. 5.1: Replikácia blokov v HDFS [12]

5.1.2 Architektúra systému

HDFS klaster má dva typy uzlov operujúcich na základe konceptu master/slave. V role master vystupuje takzvaný NameNode a v rolách slave vystupujú uzly nazývané DataNode. NameNode sa stará o údržbu menného priestoru (namespace) súborového systému. Udržuje a spravuje strom súborového systému, všetky metadáta o súboroch a zložkách v systéme. DataNode naopak uchováva jednotlivé bloky v rámci súborového systému, sprístupňuje ich a oboznamuje NameNode so zoznamom uchovávaných blokov [28].

NameNode aj DataNode sú aplikácie implementované v jazyku Java, čo umožňuje ich vysokú prenositeľnosť. Systém je väčšinou navrhovaný tak, že NameNode beží na jednom dedikovanom serveri a zvyšné stroje v klastri majú spustenú inštanciu aplikácie DataNode [28]. Architektúra systému ešte obsahuje klienta, ktorý je zodpovedný za interakciu HDFS s užívateľom. Celá architektúra, aj s princípom komunikácie, je zobrazená na obrázku 5.2.

NameNode

Ako už bolo vyššie spomenuté, NameNode uchováva strom súborového systému, metadáta o súboroch a zložkách v súborovom systéme. Tieto informácie sú uložené na jeho lokálnom disku v dvoch súboroch, v súbore pre obraz menného priestoru a v súbore pre log zmien. NameNode uchováva znalosť o tom, na ktorých dátových uzloch (DataNodes) sú uložené bloky jednotlivých súborov. Táto informácia ale nie je uložená trvale, je získavaná z dátových uzlov pri spustení systému. Rovnako uchováva aj informáciu o počte replík da-

ných blokov. V pamäti je uchovávaná aj reprezentácia metadát súborového systému. Tie to informácie slúžia na obsluhu požiadaviek na čítanie zo súborového systému [28] [12].

Obraz súborového systému a log zmien

Keď klient vykoná operáciu zápisu do súborového systému, transakcia je najprv nahratá do logovacieho súboru zmien. Zmena v logu zmien vyvolá zmenu pamäťovej reprezentácie súborového systému. Z konceptuálneho hľadiska je log zmien jedna entita, ale reprezentovaný je viacerými súbormi na disku. Pre zápis je vždy otvorený len jeden logovací súbor a pri každej transakcií je okamžite vynútená operácia `flush` a jeho synchronizácia. Až po vykonaní týchto operácií je klientovi navrátený kód pre úspešný výsledok transakcie. Takto je zabezpečené, že nedôjde k strate žiadnej z transakcií.

Súbor uchovávajúci obraz systému sa nazýva `fsimage` a uchováva kompletný trvalý záznam metadát systému. Neaktualizuje sa však s každou transakciou, keďže jeho veľkosť môže dosahovať gigabajty, a jeho zápis by bol príliš pomalý. Toto však nijako neznižuje spoľahlivosť systému, keďže v prípade zlyhania NameNode dôjde po jeho opätovnom spustení k načítaniu `fsimage` súboru do pamäte a aplikovaniu relevantných zmien uložených v logu zmien [28][25].

NameNode ako slabina systému

Z definície NameNode vyplýva, že všetky údaje o súborovom systéme, súboroch v ňom a ich rozdelení na bloky, sú uložené na jednom stroji. To z neho robí SPOF (`Single point of failure`). V prípade výpadku alebo zlyhania stroja, na ktorom je spustená inštancia NameNode, dôjde k zneprístupneniu celého systému, keďže o lokalitách blokov má znalosť len NameNode. V Hadoop existujú, 2 spôsoby ako sa dá s týmto problémom vysporiadať.

Prvou z možností je vytvorenie zálohovania súborov, ktoré ukladajú perzistentné metadáta o súborovom systéme. Hadoop umožňuje nakonfigurovať NameNode tak, aby zapisoval tieto súbory na viacero súborových systémov. Tieto zápisy sú synchronizované a atomické. Obvykle sa využíva zápis na lokálny disk a do zdieľaného sieťového disku (NFS).

Druhou možnosťou je mať spustený sekundárny NameNode. Jeho názov by mohol napovedať, že sa jedná o druhý aktívny NameNode, ale nie je tomu tak. Jeho úlohou je periodicky vykonávať aplikovanie zmien uložených v logu zmien na súbor `fsimage`. Následne potom vyvolá uloženie vytvoreného obrazu na primárny NameNode a nechá si uloženú kópiu. Takto vytvára akúsi zálohu, a tiež zabraňuje tomu, aby log zmien na primárnom NameNode narástol do príliš veľkých hodnôt. Požiadavky na veľkosť pamäte a výkon CPU sekundárneho NameNode sa preto nelíšia od požiadaviek na primárny a dôležité je, aby bežal na oddelenom fyzickom stroji.

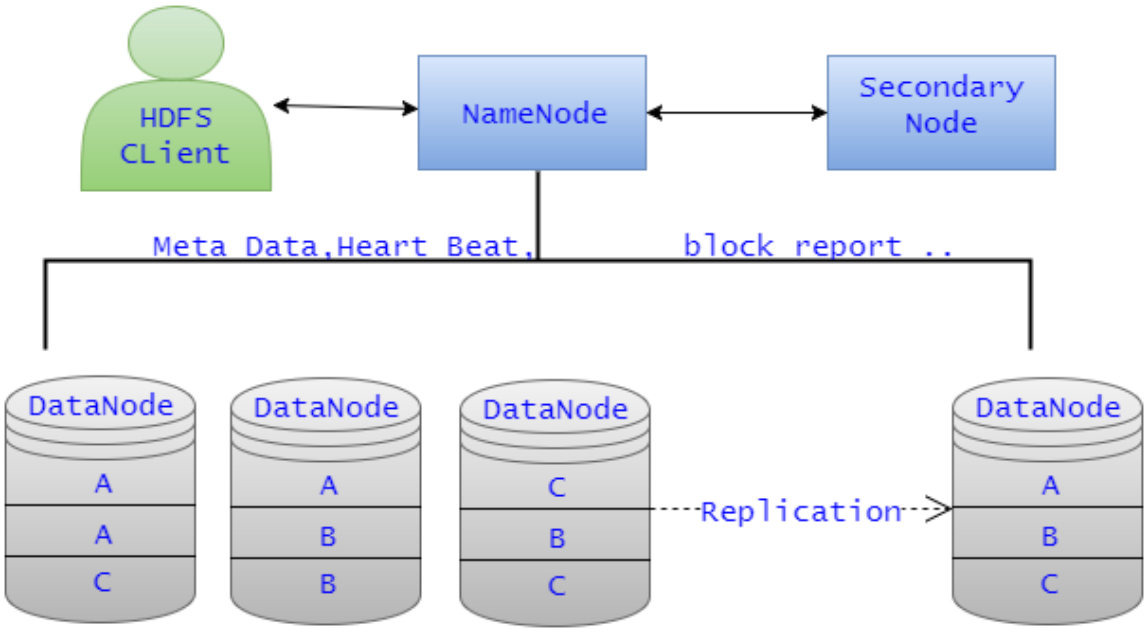
Vzhľadom na to, že sekundárny NameNode nemá uložené všetky zmeny, ale len sa na ne periodicky dopytuje, pri kompletnom výpadku primárneho NameNode môže stále dôjsť k strate dát. Preto sa odporúča kombinácia oboch prístupov [28].

DataNode

DataNode operuje ako slave. Jeho úlohou je ukladať bloky na svoj lokálny disk a periodicky oboznamovať NameNode o zozname blokov, ktoré ukladá. Pri štarte systému je dotazovaný na bloky dát, ktoré uchováva. Pre komunikáciu s NameNode používa TCP/IP protokol. DataNode posiela na NameNode takzvané „*hearthbeat*“ správy. Pomocou nich oboznamuje NameNode, že je dostupný. V základnom nastavení je tento interval 3 sekundy. Keď NameNode nedostane takúto správu od DataNode v priebehu 10 minút, predpokladá, že DataNode zlyhal a jeho bloky považuje za stratené. Tieto správy obsahujú aj informácie o DataNode. Obsahujú napríklad jeho vyťaženie, pomocou ktorého vykonáva NameNode rozloženie záťaže tzv. „*loadbalancing*“.

Bloky sú na DataNode ukladané do jeho normálneho súborového systému. Pre každý blok sa vytvoria 2 súbory. Prvý obsahuje samotné dáta bloku, druhý obsahuje kontrolný súčet pre blok. Kontrolný súčet je vytvorený klientom pri zapisovaní dát. Používa sa na overenie toho, že nedošlo k poškodeniu dát pri prenose, alebo chybou disku na DataNode. Veľkosť bloku nie je zaokrúhľovaná ani dopĺňaná, a preto blok zaberá na súborovom systéme DataNode veľkosť rovnú veľkosti dát, ktoré ukladá [12][25].

DataNode umožňuje ukladať kópie často čítaných blokov do svojej pamäte, čo ešte zlepšuje dobu prístupu. Z tejto vlastnosti môžu benefitovať aplikácie spúšťané v klastri, ktoré môžu vykonávať úlohy späté s blokmi uloženými v pamäti DataNode priamo na ňom [28].



Obr. 5.2: Schéma HDFS z pohľadu NameNode a DataNode [6]

5.1.3 Zhrnutie vlastností systému HDFS

Systém je navrhovaný pre ukladanie obrovských dátových setov, preto vyniká v nasledujúcich oblastiach:

Detekcia a zotavenie sa zo zlyhania hardvéru, cenovo dostupný hardvér

Pri veľkých klastrových riešeniach je možnosť zlyhania hardvérovej komponenty skôr pravidlom ako výnimkou. Vzhľadom na to, že v HDFS sa očakávajú stovky až tisícky uzlov, je systém navrhnutý tak, aby bol schopný sa s týmto javom vysporiadať. Detekciu zlyhania zabezpečuje posielanie takzvaných „*hearthbeat*“ správ. Zotavenie sa z takéhoto stavu zabezpečuje rozdelenie do blokov a ich replikácia. Celý systém je implementovaný v jazyku Java, ktorý sa vyznačuje dobrou prenesiteľnosťou. Vďaka týmto vlastnostiam nie je nutné používať drahý vysoko spoľahlivý a špecializovaný hardvér, ale je možné použitie cenovo dostupného tzv. „*commodity hardware*“.

Streamovaný prístup k dátam

HDFS je postavený na myšlienke, že spracovanie veľkých dátových sád spočíva v princípe zapíš raz, čítaj mnohokrát. Dátová sada je raz zapísaná zo zdroja a následne sa nad ňou vykonávajú mnohé analýzy. Predpokladá sa, že každá analýza vyžaduje veľkú časť ukladaných dát, prípadne dokonca celú. Preto sa kladie dôraz na sekvenčné čítanie, keďže pri čítaní veľkej sady dát je doba čítania celej sady dôležitejšia na úkor doby čítania prvého záznamu. Z toho vyplýva, že systém je vhodný na použitie pre dávkové spracovanie a nie pre interaktívne aplikácie [28][23].

Ukladanie veľkých súborov

Veľké súbory v tomto kontexte znamenajú súbory dosahujúce veľkosť stovky gigabajtov až terabajtov. Rozdelenie súborov do blokov umožňuje ukladať súbory presahujúce veľkosť diskov jednotlivých strojov, dokonca v teoretickej hladine je možné uložiť súbor s veľkosťou súčtu všetkých diskov v klastri.

Na dosiahnutie vyššie zmienených vlastností museli byť vykonané ústupky, preto má HDFS niektoré vlastnosti, ktoré ho robia nevhodným na vykonávanie iných špecifických úloh:

Ukladanie množstva malých súborov

Mapovanie uloženia jednotlivých blokov si NameNode uchováva v operačnej pamäti. Veľkosť dát uchovávaných o bloku, súbore alebo zložke v pamäti je približne 150 bajtov pre každú entitu [28]. Predpokladajme, že do systému ukladáme malé súbory menšie ako 1 blok. Pre každý takýto súbor musí byť vytvorený jeden blok, čiže 2 položky pamäti RAM stroja,

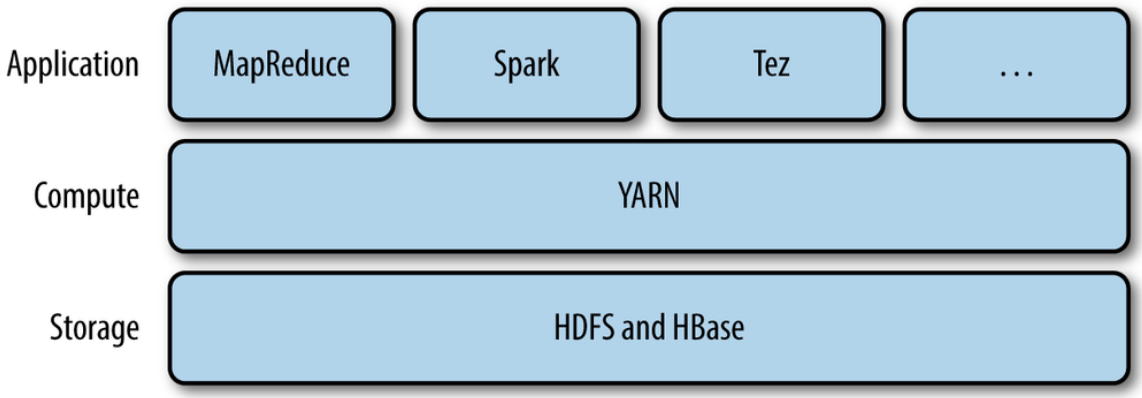
kde beží NameNode. Úložné kapacity HDFS z pohľadu diskov dosahujú veľkosti v stovkách až tisíckach terabajtov. Pri uložení jedného milióna súborov je potom potrebný v pamäti RAM priestor o veľkosti minimálne 300 MB. Preto sú milióny súborov ešte prijateľné, ale miliardy už prekračujú možnosti systému.

Rýchly prístup k dátam

Systém je optimalizovaný na priepustnosť pre veľké dátové sady. To znamená, že v prípade, že aplikácia potrebuje pristupovať k dátam v rádoch milisekúnd, nie je HDFS vhodné riešenie. Vzhľadom na to, že systém operuje na princípe blokov, už len ich samotný prenos môže trvať v rádoch sekúnd.

5.2 YARN a Hadoop MapReduce

YARN (`Yet Another Resource Negotiation`) je systém na manažovanie prostriedkov výpočtového klastra v systéme Hadoop. Bol uvedený vo verzií Hadoop 2, kde nahradil prechádzajúcu implementáciu manažmentu prostriedkov s názvom MapReduce 1 (MR1) používaného v Hadoop 1. MR 1 totiž trpela niekoľkými nedostatkami, ktoré sa snaží YARN odstrániť [19]. Keď by sme chceli návrh systému Hadoop 2 vertikálne rozvrstviť, tak najnižšiu vrstvu tvorí HDFS, ktorý slúži ako úložisko. Nad ňou operuje ako výpočtová vrstva, práve zmieňovaný YARN a najvyššiu vrstvu, aplikačnú, tvorí MapReduce. Tento návrh je ilustrovaný na obrázku 5.3. Univerzálnosť YARN umožňuje spúšťať aj iné aplikácie ako MapReduce. V ďalších podkapitolách sú popísané návrh a vlastnosti kľúčových prvkov YARN a MapReduce.



Obr. 5.3: Architektúra YARN [28]

5.2.1 Architektúra YARN

Architektúra YARN bola navrhnutá tak, aby umožnila lepšie zdieľanie zdrojov, lepšiu škálovateľnosť a spoľahlivosť. YARN aj MR1 vychádzajú z konceptu master/slave. MR1 má

v role master JobTracker uzol, ktorý zodpovedá za prijímanie úloh (`Job`) od klienta a za prideľovanie, monitorovanie podúloh (`Task`). V role slave vystupuje TaskTracker, ktorý zodpovedá za spúšťanie jednotlivých map a reduce podúloh.

Princípom vykonávania je snaha zabezpečiť čo najvyššiu blízkosť dát k nad nimi vykonávanými podúlohami. Preto JobTracker spolupracuje s NameNode z HDFS a prideľuje úlohy TaskTrackeru, čo najbližšie prípadne priamo na DataNode, kde sa dáta nachádzajú. TaskTracker následne spustí podúlohu a po jej ukončení informuje JobTracker, že podúloha skončila. V momente keď JobTracker vyhodnotí, že všetky rozdelené podúlohy boli dokončené, informuje klienta o ukončení úlohy [19]. JobTracker takto zodpovedá za plánovanie aj monitorovanie úloh.

Základnou myšlienkou YARN je rozdeliť úlohu plánovania a úlohu monitorovania medzi globálny ResourceManager a pre aplikáciu špecifický AplicationMaster. O správu jednotlivých „pracovných" strojov sa stará NodeManager. Tieto stroje sú rozdelené na kontajnery (`containers`) [23].

5.2.2 Aplikačné rozhranie MapReduce

Aplikačné rozhranie v Hadoop 2 ponúka užívateľovi abstrakciu pre tvorbu a spúšťanie aplikácií. Ako už bolo zmienené, zavedením YARN je možné na Hadoop klastri spúšťať rôzne aplikácie nie len MapReduce.

Celá stavba Hadoop vedie k tomu, aby užívateľ pri tvorbe aplikácií na spracovanie dát nemusel riešiť paralelizáciu, synchronizáciu, prenos dát, ich čítanie respektíve zápis. O všetky tieto a mnohé ďalšie úkony sa stará aplikačné rozhranie. V MapReduce je prakticky minimálnou požiadavkou na užívateľa aby navrhnol a implementoval len samotné metódy map a reduce. Avšak toto rozhranie poskytuje aj ďalšie prvky, ktoré umožňujú efektívnejší a rýchlejší beh užívateľskej aplikácie. V nasledujúcich podkapitolách sú s istou mierou abstrakcie popísané jednotlivé prvky aplikačného rozhrania MapReduce a prepojenie aplikačného rozhrania MapReduce na YARN. Vzhľadom na to, že Hadoop je postavený na jazyku Java, jednotlivé názvy prvkov korelujú s názvami tried [28][7].

Mapper

Komponenta Mapper zaobaľuje funkciu map. Mapovanie beží ako individuálna úloha a zo vstupných záznamov vytvára medzivýsledky. Pre jednu dvojicu kľúč-hodnota na vstupe môže vygenerovať 0 až mnoho výstupov. Vstupné a výstupné záznamy nemusia mať rovnakú hodnotu. Jedinou podmienkou pre oba prvky v dvojici kľúč-hodnota je, že musia byť serializovateľné. Pre každý úsek vstupu (`InputSplit`) spustí MapReduce rozhranie jednu úlohu. Veľkosť jedného `InputSplit` je totožná s veľkosťou jedného bloku HDFS. Následne Mapper vykoná úlohu map nad každou z dvojíc kľúč-hodnota v `InputSplit`. Počet spúšťaných Mapper inštancií závisí na veľkosti vstupných dát. Napríklad, pre vstupné dáta o veľkosti desať terabajtov pri veľkosti bloku stodvadsaťosem, sa spustí osemdesiatdvatisíc

inštancií. Preto pre malé súbory môže dôjsť k tomu, že čas plánovania a spúšťania Mapper inštancií prekročí čas ich samotného vykonávania. MapReduce sa pri spúšťaní inštancie snaží o tzv. optimalizáciu lokálnosťou dát. Snaží sa teda spúšťať úlohu na uzle, kde sa nachádza blok, ktorý ma spracovať, prípadne čo najbližšie k nemu. Výstup je zapisovaný na lokálny disk nie do HDFS [7].

Reducer

Komponenta Reducer zaobaľuje funkciu reduce. Redukuje teda výstupy jednotlivých Mapper inštancií, ktoré zdieľajú rovnaký kľúč do menších skupín hodnôt. Reducer nedisponuje možnosťou lokálnosti dát, keďže spracováva výstupy mapovania naprieč celým klastrom. Jeho výsledok je zapisovaný do HDFS s prvou replikou zvyčajne na uzle, kde bola inštancia Reducer spustená. Počet spúšťaných Reducer inštancií môže užívateľ špecifikovať. V prípade, že uživateľ nešpecifikuje ich počet, postará sa o to MapReduce aplikačné rozhranie. V prípade, že je spustených viac Reducer inštancií, musí dochádzať k deleniu výstupov mapovania. Užívateľ môže toto delenie špecifikovať sám implementáciou triedy Partitioner. V prípade, že tak neučiní, používa sa delenie implementované aplikačným rozhraním, ktoré spája do jednej partície kľúče pomocou výsledku výpočtu hašovacej funkcie nad ich hodnotou. Je nutné poznamenať, že aj v takomto prípade idú všetky hodnoty s rovnakým kľúčom na spracovanie do jednej Reducer inštancie [7].

Partitioner

Komponenta Partitioner sa stará o delenie priestoru kľúčov do partícií. Jeho úlohou je implementovať funkciu, podľa ktorej sú kľúče rozdeľované. Partitioner má prístup aj k hodnote z dvojice kľúč-hodnota, čiže deliaca funkcia môže byť implementovaná aj nad hodnotou. Je nutné dodržať to, aby bol počet partícií zhodný s počtom spustených Reducer inštancií. V prípade, že bude počet paritícií menší, spracovanie bude neefektívne, keďže niektoré inštancie Reducer nebudú dostávať žiadny vstup. Ak bude tento počet vyšší, budú dvojice kľúč-hodnota pridelené do partície bez inštancie Reducer zahodené [28] [7].

Combiner

Komponenta Combiner slúži na optimalizáciu veľkosti prenášaných dát medzi inštanciami Mapper a Reducer. Inštancia triedy Combiner totiž beží na tom istom uzle ako Mapper a jeho úlohou je lokálna redukcia. Combiner implementuje tú istú funkciu ako Reducer. O tom, či bude Combiner spustený alebo nie, rozhoduje MapReduce aplikačná vrstva. Počas behu komponenty Combiner totiž dochádza k zápisom na disk a tie môžu mať efekt na spomalenie samotného behu aplikácie. Pri používaní inštancie triedy Combiner musíme byť opatrní, keďže jej nesprávne zapojenie môže ovplyvniť výsledok celej MapReduce úlohy. Jej zapojenie je možné len v prípade, že operácia, ktorú vykonáva Reducer, je komutatívna [7].

5.2.3 Spúšťanie užívateľskej MapReduce aplikácie nad YARN

V tejto podkapitole sú previazané jednotlivé prvky z predchádzajúcej časti textu, aby bol vykreslený celistvý pohľad na spúšťanie MapReduce aplikácií nad architektúrou YARN. Jedna aplikácia môže obsahovať viacero MapReduce úloh. Tie môžu byť reťazené a jedna úloha môže ako vstup využívať výstup inej. Jednu MapReduce úlohu stelesňuje jedna inštancia triedy Job. Pomocou tejto inštancie definujeme vstup, výstup a konfiguráciu úlohy, rovnako pomocou nej špecifikujeme, z akých tried majú byť vytvárané inštancie tried, definujúcich MapReduce úlohu. Po zavolaní metódy `submit()` inštancie triedy Job dôjde k spusteniu MapReduce úlohy [28]. Na obrázku 5.4 sa nachádza detailná schéma spúšťania MapReduce úlohy.

Spustenie MapReduce na YARN prebieha v nasledovných krokoch:

1. **Zadanie úlohy**: Po zavolaní submit(), prejde riadenie pod proces implementovaný v triede JobSubmitter a ten vykoná požiadavku na pridelenie aplikačného identifikátora od správcu prostriedkov, skontroluje existenciu vstupných a výstupných ciest MapReduce úlohy, vypočíta úseky vstupu. Následne prekopíruje potrebné prostriedky (`.JAR` súbor s úlohou, konfiguráciu úlohy atď.) do zdieľaného súborového systému (HDFS) s vysokou replikáciou. Ďalej zadá úlohu na spracovanie správcovi prostriedkov.

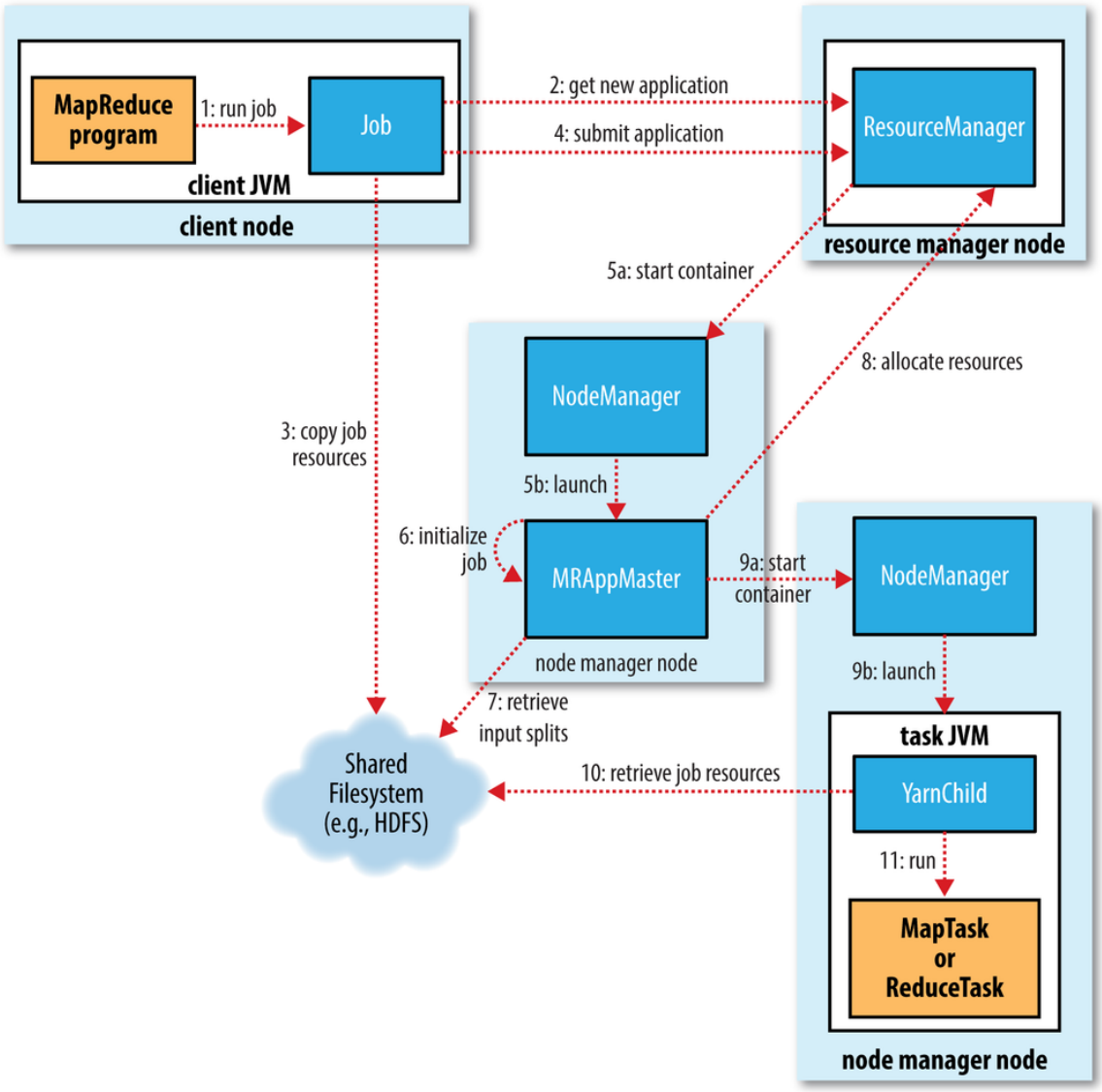
2. **Inicializácia úlohy**: Keď je správcovi zadaná úloha na spracovanie, predá ju plánovaču v YARN. Ten pre ňu vyhradí kontajner v klastri a spustí proces MRAppMaster pre danú úlohu. MRAppMaster je AplicationMaster pre MapReduce úlohy. Ten si po spustení pre úlohu vytvorí štruktúry, pomocou ktorých bude uchovávať informácie o prebiehajúcej úlohe. Následne si vyžiada informáciu o úsekoch vstupu. Pre každý úsek vstupu vytvorí jednu inštanciu map úlohy a pre celú úlohu definovaný počet inštancií úlohy reduce. Takisto rozhoduje o tom, ako bude úloha spúšťaná. V prípade, že sa jedná o úlohu, ktorá je malá a spúšťanie jej spracovania paralelne by svojimi nárokmi prekročilo samotné nároky spúšťanej úlohy, je úloha spúšťaná sekvenčne na tom Java virtuálnom stroji (JVM), ako beží samotný MRAppMaster proces. Posledným krokom pred spustením úlohy je vytvorenie jej výstupného súboru a dočasných súborov využívaných počas behu úlohy.

3. **Priradenie podúloh**: V prípade, že je úloha dosť veľká na to, aby bola vykonávaná paralelne, začne MRAppMaster vytvárať požiadavky na spustenie map podúloh. Map podúlohy sú vždy spúšťané ako prvé, keďže ich výstup je vstupom reduce podúloh. Požiadavky pre reduce podúlohy začne MRAppMaster posielať až potom, ako skončí aspoň 5% map podúloh.

   Plánovač sa snaží úlohy map priraďovať strojom, na ktorých sa nachádza blok dát, s ktorým budú pracovať. Úlohy reduce sú priraďované ľubovoľným strojom v klastri.

4. **Spúšťanie podúloh**: Posledným krokom je spustenie vykonávania podúlohy. To prebieha tak, že MRAppMaster kontaktuje NodeManager na uzle, ktorý bol priradený

plánovačom pre danú podúlohu. Ten vytvorí pre úlohu kontajner a v HDFS nájde a následne načíta do vytvoreného kontajnera prostriedky potrebné pre úlohu, ktoré tam v kroku číslo 1. uložil JobSubmitter. Následne NodeManager spustí úlohu. Úloha je spúšťaná v oddelenom Java virtuálnom stroji, vďaka čomu chyba v úlohe, respektíve jej pád, nijak neovplyvní plynulý chod procesu NodeManager [28].



Obr. 5.4: Spúšťanie MapReduce úlohy nad architektúrou YARN [28]

Kapitola 6

Návrh využitia technológie Big Data v podpore rozhodovania

V súčasnej dobe sa stáva získavanie a skladovanie veľkých objemov dát jednoduché a rozšírené. Môžeme preto predpokladať, že Big Data ako také, budú postupne nachádzať uplatnenie v rôznych oblastiach priemyslu alebo služieb. Jednou z oblastí služieb sú aj obchody a obchodné reťazce. V prípade, že sa naplnia prognózy o rozšírení inteligentných zariadení, je pravdepodobné, že nájdu využitie aj v týchto oblastiach. Napríklad vo forme inteligentných pokladní, ktoré budú schopne zbierať dáta o nákupoch alebo nakupujúcich. Pri predstave, koľko nákupov prebehne vo väčšom meste počas jedného dňa, je jasné, že takto získané dáta budú spĺňať charakter Big Data. Bude ich však nutné uchovávať a analyzovať. Analýzou týchto dát môžu reťazce získať veľmi cenné informácie o samotných produktoch, ale takisto o ich zákazníkoch a ich chovaní.

V tejto kapitole sa nachádza návrh systému, ktorý by mohol slúžiť na uchovávanie a spracovanie týchto dát. Navrhovaný systém je postavený nad systémom Apache Hadoop, ktorý je komplexne popísaný v kapitole 5. Systém Hadoop sa stal akýmsi nepísaným štandardom na uchovávanie veľkých dátových sád. Ponúka však aj rozhranie MapReduce, ktoré je vhodný na paralelné spracovanie veľkých dátových sád.

6.1 Popis konceptu navrhovaného systému

Jedným z nosných pilierov podpory rozhodovania je systém OLAP. V podkapitole 2.2.2 boli popísané vlastnosti, ktoré musí spĺňať. V prípade, že by sme chceli OLAP implementovať priamo nad veľkými sadami dát uchovávaných v rámci konceptu Big Data, došlo by k porušeniu týchto pravidiel. Napríklad, stabilná výkonnosť je ťažko dosiahnuteľná, keď dochádza k spracovaniu mnohých terabajtov dát. Aj v prípade použitia paralelizmu môže totiž spracovanie trvať hodiny. Takisto je ťažko predstaviteľná realizácia analytického modelu a užívateľských ad-hoc požiadaviek priamo nad samotnými dátami, ktoré môžu byť neštruktúrované. Avšak ak majú Big Data produkovať hodnotu, musíme byť schopní nad

nimi robiť analýzu s cieľom podpory rozhodovania. Preto sa ako možné riešenie ponúka materializácia dátovej kocky a predpočítanie vybraných agregovaných hodnôt.

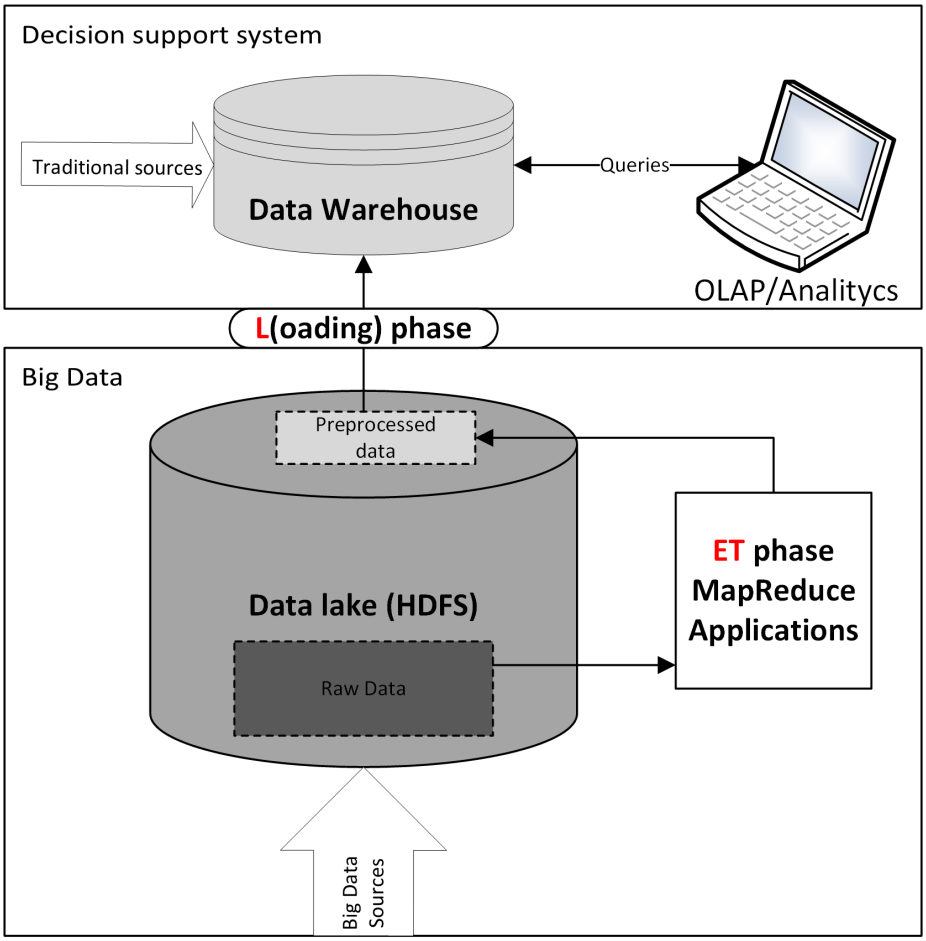
Materializácia môže prebehnúť priamo v platforme Hadoop, pomocou aplikácií MapReduce a do dátového skladu sú natiahnuté už predspracované a očistené dáta, prípadne samotné výsledné hodnoty agregačných funkcií. Nevýhodou tohto prístupu je vznik duplicitných dát, ukladaných v dvoch rôznych systémoch. Agregované dáta však zaberajú oveľa menej priestoru, a preto je ich priestorová náročnosť zanedbateľná v porovnaní s obrovskou dátovou sadou, ukladanou v Data Lake.

Obrázok 6.1 zobrazuje schému navrhovaného riešenia. Navrhovaný systém sa skladá z dvoch podsystémov. Prvým podsystémom je klasický systém pre podporu rozhodovania, ako bol popísaný v kapitole 2. Druhým podsystémom je platforma pre spracovanie Big Data Hadoop, popísaná v kapitole 5. Cieľom systému je umožniť vykonávanie analýz pomocou systémov na podporu rozhodovania nad obrovskými sadami dát ukladanými v Data Lake.

Vstupné surové dáta sú zapisované do HDFS, ktorý tvorí konkrétnu implementáciu konceptu Data Lake. Proces ETL je preto rozdelený na 2 fázy a to ET, fázu extrakcie a transformácie a L fázu nahratia. Extrakcia a transformácia prebieha priamo na platforme Hadoop a to tak, že výpočtovo a časovo náročné transformácia a agregácia dát z Data Lake, sú vykonávané pomocou Hadoop MapReduce rozhrania. Výsledky aplikácií bežiacich nad týmto rozhraním sú ukladané naspäť do HDFS. Spracované dáta sú potom, vo fáze L, nahraté do dátového skladu, odkiaľ sú sprístupnené užívateľom systému na podporu rozhodovania. Natiahnutie dát musí prebiehať až po dobehnutí MapReduce aplikácií. Spúšťanie aplikácií a natiahnutie dát môže byť periodické, prípadne vykonávané na požiadavku.

Druhou možnosťou využitia Big data v podpore rozhodovania je ich využitie v reportovaní. Reporty môžu byť totiž postavené priamo nad výstupmi aplikácií MapReduce. Ich výstup môže byť použitý v rámci webového rozhrania, ktoré ich natiahne priamo z HDFS. Výhodou reportovania ako takého je, že samotné reporty sú vytvárané periodicky alebo na požiadavku, preto na ich tvorbu nie je potrebná odpoveď v rádoch sekúnd.

Takto navrhovaný systém má svoje výhody aj nevýhody. Výhodou systému je, že umožňuje koncovému užívateľovi analyzovať veľké dátové sady. Nevýhodou je implementačná náročnosť systému a istá miera duplicity ukladaných dát.



Obr. 6.1: Schéma navrhovaného systému

6.2 MapReduce aplikácie pre predspracovanie dát

Navrhovaný systém z podkapitoly 6.1 je príliš rozsiahly na to, aby ho bolo možné implementovať celý v rámci tejto práce. Preto sa v ďalších častiach zameriame na najpodstatnejšiu časť tohto systému. Tou sú MapReduce aplikácie na spracovanie dát do formy, z ktorej ich je možné nahrať do dátového skladu alebo do formy, v ktorej ich je možné jednoducho spracovať do reportov. Každý systém na spracovanie veľkých dát, respektíve na podporu rozhodovania, je navrhovaný so špecifickým zámerom. Tento zámer sa odvíja od charakteru dát a zamerania organizácie, ktorá chce tento systém používať. Avšak dáta organizácií sú pre ne veľmi cenné, preto je ich problematické ich získať. Z toho dôvodu sú používané dáta používané v tejto práci vygenerované. No pri návrhu bol kladený dôraz na to, aby formát dát čo najviac reflektoval to, ako vyzerajú dáta, ktoré sú vytvárané v dnešnej dobe. Vstupné aj výstupné dáta sú v textovej forme, ako je obvyklé pre dáta ukladané v HDFS a spracovávané cez Hadoop MapReduce.

Navrhované aplikácie sa zameriavajú na spracovanie a analýzu dát o nákupoch ľudí. Prvá aplikácia sa zameriava na dvojice najčastejšie nakupovaných produktov. Druhá aplikácia sa

zameriava na analýzu záujmu o vybrané produkty. Obidve aplikácie fungujú v 2 krokoch, kde prvý je spracovanie čistých dát a druhým krokom je samotná analýza.

Aplikácie očakávajú vstupné dáta v textovej forme, kde sú jednotlivé nákupy a informácie uložené na separátnych riadkoch. Informácie o samotných nákupoch sú potom oddelené čiarkou. Ich formát je zobrazený na obrázku 6.2. Čas a lokalita nesú informácie o tom, kde a kedy nákup prebehol. Pohlavie a vek sú informácie o človeku, ktorý nákup vykonal a 1 až n je počet unikátnych položiek v košíku.

pohlavie	vek	čas	lokalita	položka č. 1	položka č. 2	...	položka č. n

Obr. 6.2: Formát vstupných dát

Takýto formát dát je veľmi podobný tomu, ako vyzerajú logovacie súbory, ktoré sú jedným z typických príkladov ukladaných dát v Data Lake. Vzhľadom na premenlivú dĺžku nákupu je náročné pre ne stanoviť pevnú štruktúru.

6.2.1 Aplikácia na zistenie najčastejšie nakupovaných dvojíc produktov

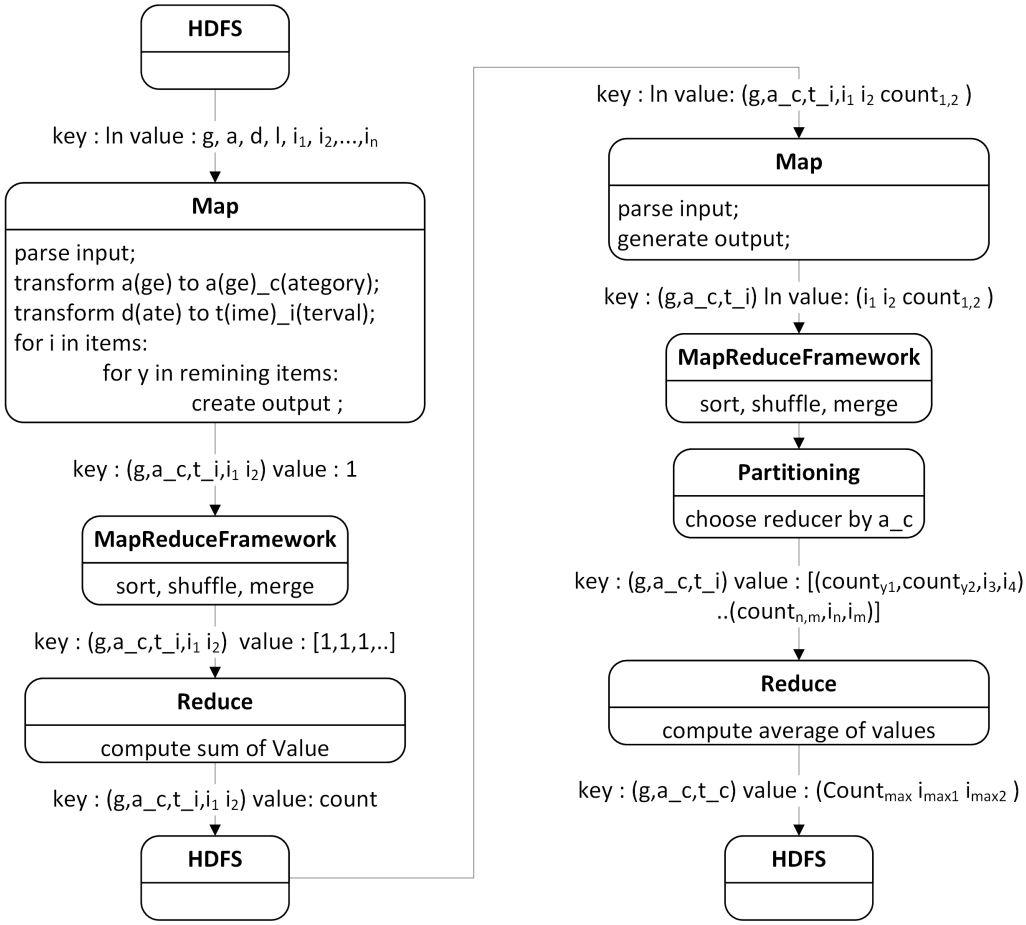
Výsledný pohľad vytvorený aplikáciou zobrazuje závislosť najčastejšie nakupovaných dvojíc produktov, na pohlaví a vekovej kategórií nakupujúceho a časového intervalu kedy nákup prebiehal.

Formát výstupných dát

Výstupné dáta aplikácie sú opäť uložené v textovej forme. Ich formát závisí na tom, či bude v aplikácii implementovaná vlastná komponenta Partitioner. V prípade, keď nebude využitá vlastná implementácia, bude formát výstupu nasledovný: časový úsek v týždni, veková kategória, pohlavie, položka x, položka y, počet. V prípade vlastnej implementácie komponenty Partitioner bude výstup uložený v súboroch, ktoré budú reprezentovať jednotlivé vekové kategórie a ich formát bude vyzerať nasledovne: pohlavie, časový úsek týždňa, položka x, položka y, počet. Najnižšiu vekovú kategóriu bude reprezentovať súbor s príponou 0001, druhú 0002 a tak postupne až po 0006 reprezentujúci najvyššiu vekovú kategóriu. V obidvoch prípadoch položka x a y reprezentujú najčastejšie nakupovanú dvojicu položiek, v rámci vekovej kategórie a času. Počet je číslo, koľkokrát boli nakúpené.

Návrh implementácie aplikácie nad MapReduce

Navrhovaná aplikácia pozostáva z dvoch zreťazených MapReduce úloh. Výstup prvej MapReduce úlohy bude zapísaný do HDFS a postavený ako vstup druhej úlohy. Po vykonaní druhej úlohy nebude výstup prvej úlohy zmazaný, môže totiž slúžiť na ďalšíu analýzu. Schéma algoritmu je ilustrovaná na obrázku 6.3.



Obr. 6.3: Schéma návrhu aplikácie na zistenie najčastejšie nakupovaných dvojíc produktov v kontexte MapReduce

Podúloha map prvej MapReduce úlohy bude mať ako vstup jeden riadok zo vstupu celej aplikácie. Ten bude reprezentovaný dvojicou kľúč-hodnota, kde kľúč bude číslo riadku a hodnota bude obsah samostatného riadku. Podúloha map bude prebiehať v nasledujúcich krokoch:

1. Vo vstupnej hodnote nájde čas nákupu a transformuje ho na časový úsek v rámci týždňa.
2. Vo vstupnej hodnote nájde pohlavie nakupujúceho.
3. Vo vstupnej hodnote nájde vek nakupujúceho a transformuje ho na vekový interval.
4. V cykle prejde položky nákupného košíka a každú položku uloží do zoznamu.
5. V cykle prejde každú položku zoznamu, pričom v tele prechodu sa bude nachádzať ďalší cyklus, v rámci ktorého prejde všetky položky, ktoré sa nachádzajú v danom zozname za aktuálne spracovávanou položkou. Na výstup v každej iterácií cyklu zapíše

dvojicu kľúč-hodnota nasledovne: kľuč bude tvoriť pätica vek, časový interval, pohlavie, aktuálne spracovávaná položka z prvého cyklu, aktuálne spracovávaná položka z vnoreného cyklu a hodnotu bude tvoriť číslo 1, znamenajúce výskyt daného kľúča.

Podúloha reduce prvej úlohy MapReduce dostane ako vstupný kľúč výstupný kľúč podúlohy map a ako hodnotu dostane zoznam jednotiek o dĺžke počtu výskytov jednotlivých dvojíc položiek v nákupoch, v rámci daného časového obdobia a vekovej kategórie. Tento zoznam v cykle prejde a vykoná nad ním operáciu súčet. Na výstup zapíše ako kľúč vstupný kľúč a hodnotu výsledku operácie súčet.

Vstupom podúlohy map druhej MapReduce úlohy bude jeden riadok výstupu prvej úlohy. V reprezentácii dvojice kľúč-hodnota, bude kľúč číslo riadku a hodnota bude šestica: vek, časový interval, pohlavie, položka číslo 1, položka číslo 2, a počet výskytov. Podúloha identifikuje v rámci textového vstupu jednotlivé údaje a na výstup zapíše dvojicu kľúč-hodnota nasledovne: kľúčom bude trojica vekový interval, časový interval, pohlavie a hodnotou bude trojica položka číslo 1, položka číslo 2 a počet výskytov.

Na vstupe podúlohy reduce druhej úlohy MapReduce bude kľúč totožný s výstupným kľúčom podúlohy map a hodnotu bude tvoriť zoznam trojíc z výstupov úlohy map s totožným kľúčom. Podúloha prejde tento zoznam a v rámci neho nájde maximálnu hodnotu výskytu. Na výstup zapíše šesticu vekový interval, časový interval, pohlavie, položka číslo 1 a položka číslo 2, maximálna hodnota, kde položka číslo 1 a položka číslo 2 sú položky z trojice v ktorej sa maximálna hodnota výskytu nachádzala. Tento výstup bude zároveň aj výstupom celej aplikácie.

Rozdelenie do vekových kategórií bude nasledovné; prvá kategória bude pokrývať vek nižší ako 18 rokov, druhá kategória bude vo veku od 18 do 25 rokov, tretia bude od veku 25 do 35 rokov, štvrtá kategória bude od 35 rokov do 50 rokov, piata kategória bude od 50 rokov do 70 a posledná kategória bude pokrývať vek vyšší ako 70 rokov. Rozdelenie do časový intervalov bude vychádzať z rozdelenia týždňa na dni s tým, že bude zjemnené ešte na hodinové úseky dňa. Takéto rozdelenie už v podúlohe map prvej úlohy MapReduce je dôležité, z dôvodu objemu dát produkovaných touto úlohou. Čas je súčasťou kľúča obidvoch podúloh tejto úlohy a redukcia jeho rozsahu na intervaly znižuje počet všetkých výstupných záznamov. V prípade, že by sa dátum netransformoval už v prvej časti, spôsobilo by to, že každý nákup by mal unikátny kľúč. Tiež by ho mali aj všetky dvojice položiek nachádzajúce sa v ňom. V prípade, keď by sme predpokladali veľkosť priemerného nákupu 20 položiek, tak by pre každý nákup bolo po prvej úlohe zapísaných 209 záznamov. To isté platí aj transformácií veku na vekovú kategóriu. Keď by sme predpokladali, že sa ako hodnota veku nakupujúceho môžu objaviť hodnoty od 10 do 85 rokov, vzniklo by tak 75 rôznych možností vekovej podzložky výstupného kľúča prvej úlohy. To by malo za následok, že by prvá úloha MapReduce vygenerovala obrovské množstvo dát a nevykonala nad nimi žiadnu agregáciu. Z pohľadu výkonnosti by to spôsobilo, že aplikácia by bola neefektívna a jej použitie obmedzené na malé dátové sady.

6.2.2 Aplikácia pre zistenie záujmu o vybrané produkty

Cieľom aplikácie je vytvoriť agregované dáta o záujme o vybrané produkty. Keďže spracovanie všetkých produktov vyskytujúcich sa v nákupoch by bolo časovo aj priestorovo príliš náročné, aplikácia očakáva ako jeden zo vstupov zoznam produktov, pre ktoré má byť agregácia zistená. Výsledný pohľad zobrazuje priemernú predajnosť daných produktov v jednotlivých dňoch týždňa v závislosti na lokalite a pohlaví zákazníka. Záujem o produkt je v tomto prípade vnímaný nie z počtu kusov, koľko sa produktu predalo, ale v koľkých nákupoch sa vyskytol.

Formát výstupných dát

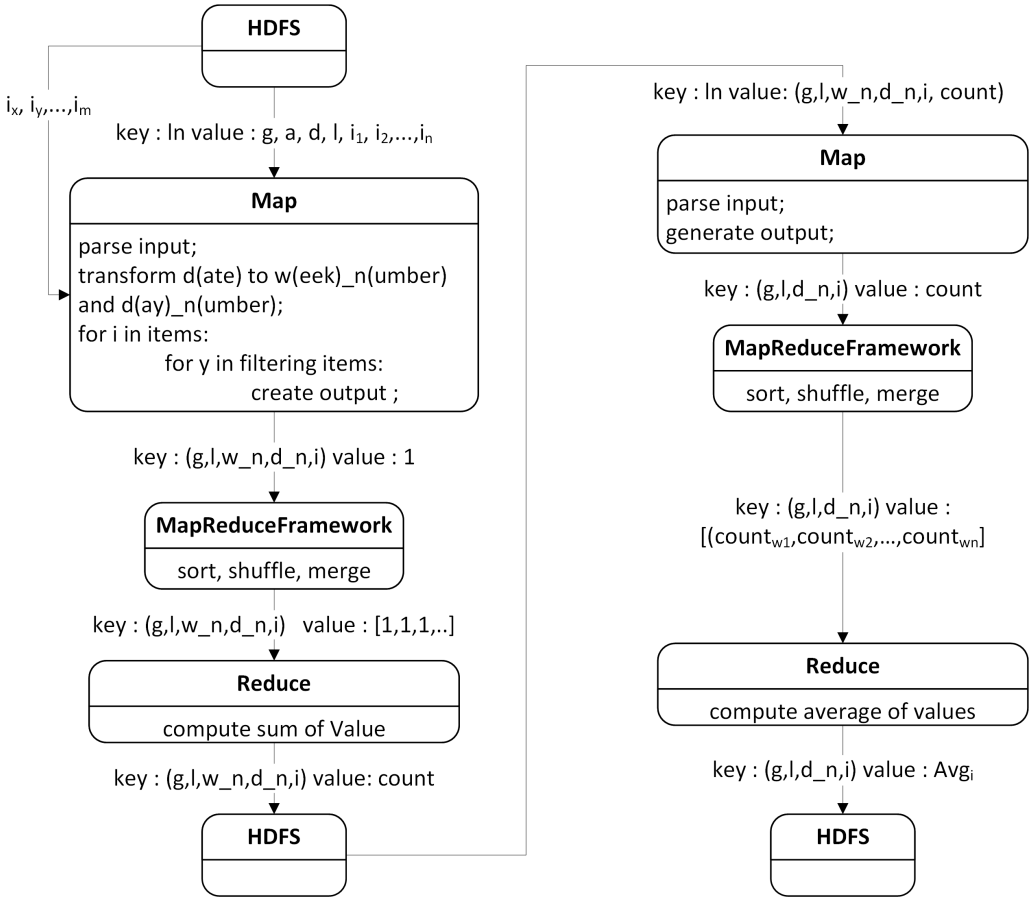
Výstupné dáta aplikácie sú uložené v textovej forme. Výstup je uložený celkovo v 2 podzložkách. V podzložke s názvom temp sú uložené dáta nesúce informáciu, o tom koľkokrát sa produkt objavil v nákupných košíkoch v závislosti na pohlaví nakupujúceho, lokalite, týždni v roku a dni v tomto týždni. V podzložke final sa nachádza priemerný počet výskytov vybraných položiek v závislosti na dni týždňa za obdobie roka. Samotný formát dát v súboroch s výsledkami je nasledovný: pohlavie, lokalita, položka, číslo dňa v týždni a hodnota. Hodnota reprezentuje agregovanú hodnotu.

Návrh implementácie aplikácie nad MapReduce

Navrhovaná aplikácia pozostáva z celkovo dvoch zreťazených MapReduce úloh. Výstup prvej MapReduce úlohy je zapísaný do HDFS a postavený ako vstup druhej úlohy. Schéma algoritmu je ilustrovaná na obrázku 6.4.

Podúloha map prvej MapReduce úlohy bude mať ako vstup jeden riadok zo vstupu celej aplikácie. Ten bude reprezentovaný dvojicou kľúč-hodnota, kde kľúč bude číslo riadku a hodnota bude obsah samostatného riadku. Podúloha map bude prebiehať v nasledujúcich krokoch:

1. Vo vstupnej hodnote nájde čas nákupu a transformuje ho na poradové číslo týždňa v roku a poradové číslo dňa v tomto týždni.
2. Vo vstupnej hodnote nájde pohlavie nakupujúceho.
3. Vo vstupnej hodnote nájde lokalitu nákupu.
4. V prvom cykle prejde všetky položky nákupného košíka.
5. V druhom vnorenom cykle prejde vstupný zoznam vybraných položiek, pričom porovnaním zistí, či sa aktuálne spracovávaná položka košíka nachádza vo vstupnom zozname vybraných položiek. V prípade že áno na výstup zapíše pohlavie, lokalitu, poradové číslo týždňa a dňa, čo bude výstupným kľúčom a hodnotu bude tvoriť číslo 1, znamenajúce výskyt daného kľúča.



Obr. 6.4: Schéma návrhu aplikácie pre zistenie záujmu o produkt v kontexte MapReduce

Podúloha reduce prvej úlohy dostane ako vstupný kľúč výstupný kľúč podúlohy map a ako hodnotu dostane zoznam jednotiek o dĺžke počtu výskytov filtrovaných položiek v nákupoch v závislosti na pohlaví, lokalite, týždni a dni v tomto týždni. Tento zoznam v cykle prejde a vykoná nad ním operáciu súčet. Na výstup zapíše ako kľúč vstupný kľúč a hodnotu výsledku sčítania.

Vstupom podúlohy map druhej úlohy aplikácie je jeden riadok výstupu prvej úlohy. Kľúčom je číslo riadku a hodnotou je šestica pozostávajúca z výstupného kľúča predchádzajúcej úlohy spolu s hodnotou. Úloha rozdelí hodnotu na jednotlivé časti a na výstup zapíše kľuč zložený z pohlavia, lokality, názvu produktu a poradového čísla dňa. Ako hodnota je na výstupe zapísaný počet výskytov produktu v nákupných košíkoch v tomto dni. Poradové číslo týždňa je vynechané z výstupného kľúča, keďže pre spočítanie priemeru chceme, aby sa všetky hodnoty s rovnakým dňom dostali na vstup jednej reduce podúlohy.

Na vstupe podúlohy reduce druhej úlohy bude kľúč totožný s výstupným kľúčom podúlohy map a hodnotu bude tvoriť zoznam celých čísel. Podúloha prejde tento zoznam a vypočíta pre neho aritmetický priemer. Na výstup zapíše vstupný kľúč a hodnotu aritmetického priemeru. Tento výstup bude zároveň aj výstupom celej aplikácie.

Kapitola 7

Výsledný systém

7.1 Nasadenie systému Hadoop

Vzhľadom na to, že systém Hadoop je klastrové riešenie, jednou z podstatných otázok je jeho nasadenie. Najpodstatnejším faktorom ovplyvňujúcim odpoveď na túto otázku je účel, na ktorý chceme klaster využiť. V tejto podkapitole si predstavíme riešenia a možnosti, ktoré boli počas tejto práce preskúmané a vyskúšané. Prvou z nich je nasadenie priamo na vlastnom počítači, druhou je nasadenie vo virtuálnom stroji, prípadne strojoch spúšťaných na vlastnom počítači a poslednou je nasadenie v cloudovom systéme. Medzi skúmanými riešeniami chýba nasadenie v klastri skutočných fyzických strojov, pretože skúmané riešenia boli zvolené na základe finančnej dostupnosti a realizovateľnosti.

Nasadenie na vlastnom počítači

Prvou možnosťou je nasadiť systém Hadoop priamo na vlastnom počítači. Keďže systém Hadoop dokáže fungovať aj na jednom stroji, nič nám nebráni spustiť si ho na vlastnom počítači. Ale takéto riešenie môže byť dosť riskantné a pracné, preto nebolo testované a skúmané v tejto práci.

Nasadenie vo virtuálnom stroji

Druhou možnosťou je systém Hadoop nasadiť na vlastnom stroji ale nie priamo, no pomocou virtualizačných prostriedkov. Tento spôsob nasadenia systému odstraňuje nedostatky prvej možnosti. Virtualizačná vrstva totiž zabezpečuje izoláciu nasadeného systému od systému vlastného počítača, čo pri chybových stavoch redukuje problémy, ktoré môžu vzniknúť. Výhodou tohto riešenie je aj to, že virtuálnych strojov môžeme mať na jednom počítači spustených viacero, a preto pri dostatočnej výpočtovej kapacite môžeme mať na vlastnom počítači spustený klaster s viacerými strojmi.

Ďalšou z výhod oproti predchádzajúcej možnosti je, že systém nemusíme konfigurovať sami. Riešenie od spoločností Cloudera alebo Hortonworks totiž umožňujú stiahnutie

predpripraveného obrazu pre virtuálny stroj. Obidve z týchto spoločností ponúkajú diskové obrazy pre Virtual Box, WMWare a pre Docker. Na základe skúseností sa ukázal Docker ako najlepšie riešenie, keďže spomalenie spôsobené virtuálizáciou bolo minimálne. Diskové obrazy sú však predkonfigurované na riešenie s jedným strojom. V prípade, že chceme mať k dispozícií klaster s viacerými strojmi, je nutné vykonať konfiguráciu.

Nevýhodou tohoto riešenia je to, že stroje sú iba virtuálne a spúšťané na jednom fyzickom stroji. To znamená, že prichádzame o hlavné benefity systému HDFS, čo sú distribuovaný súborový systém a tolerancia chýb. V prípade zlyhania disku totiž dôjde k strate všetkých replík dát v rámci virtuálnych strojov. Prichádzame taktiež aj o možnosť paralelného spracovania na viacerých strojoch, keďže virtuálne stroje sú limitované výkonom jedného fyzického procesoru, ktorý zdieľajú. Ďalšou z nevýhod riešenia vo virtuálnom prostredí je jeho nestabilita, preto sa môže stať, že dôjde k chybe priamo v programe zabezpečujúcom virtuálizáciu a uzol systému Hadoop sa už nepodarí naštartovať.

Z vyššie uvedených dôvodov je systém vo virtuálnom prostredí nasadenom na jednom fyzickom stroji vhodný pre počiatočné experimenty a prípadne vývoj vlastných algoritmov. Určite nie je vhodný pre produkčné prostredie, v ktorom by sme mali uchovávať skutočné dáta a vykonávať nad nimi analýzu.

Nasadenie v cloudovom systéme

Ďalšou z možností je nasadenie systému Hadoop v cloude. Toto riešenie so sebou prináša všetky výhody a nevýhody cloudových riešení. V rámci tohoto spôsobu nasadenia boli preskúmané dve platené riešenia a to Cloud x Lab a Google Dataproc.

- **Cloud x Lab** je riešenie zamerané na experimentovanie so systémom Hadoop, učenie sa dátovej analýzy a vývoja aplikácií na dátovú analýzu. Jeho cena sa v dobe písania práce nachádzala na úrovni 30 dolárov za trojmesačný, prakticky neobmedzený prístup. Užívateľ po zaplatení predplatného dostane k dispozícií výpočtový klaster s dvomi strojmi spusteným ako NameNode, jedným strojom primárnym a druhým sekundárnym. V role DataNode je spustených ďalších šesť strojov. Obmedzenie prístupu sa vzťahuje na konfiguračné zmeny a podobné úkony. Výhodou tohoto riešenia je že, okrem prístupu k systému Hadoop, dostane užívateľ k dispozícií aj prístup k iným technológiám na analýzu dát (Hadoop Spark, HBase, a iné). Nevýhodou systému je to, že celý výpočtový klaster je zdieľaný všetkými užívateľmi. To spôsobuje značný neporiadok v HDFS a vyčerpanie diskového priestoru na niektorých dátových uzloch. Preto sa ani tento systém nehodí na používanie v produkčnom prostredí. Nie je vhodný ani na testovanie vyvinutých aplikácií nad rozsiahlymi objemami dát, keďže pri prenose dát medzi fázami map a reduce môže, z dôvodu nedostatku miesta na stoji, kde ma prebiehať reduce, dôjsť k chybe a tým pádom aj zlyhaniu aplikácie. Naopak, riešenie je vhodné, pre svoju relatívnu cenovú dostupnosť, na fázu implemen-

tácie MapReduce aplikácií a ich testovanie na správnu funkcionalitu, keď je aplikácia spúšťaná viackrát nad malými dátovými sadami.

- **Google Dataproc** je riešenie od spoločnosti Google, ktoré je súčasťou Google Cloud Platform a je zamerané predovšetkým na produkčné systémy, preto ponúka širokú škálu cloudovej funkcionality. Výhodou tohoto riešenia je predovšetkým to, že je v rámci neho možné nasadiť vlastný klaster. Jeho veľkosť a parametre jednotlivých strojov, ako veľkosť diskov, operačnej pamäte, počet jadier procesoru je možné nastaviť pri nasadzovaní klastra. Klaster je v základom režime nasadzovaný s jedným Master uzlom, kde bežia YARN ResourceManager a HDFS NameNode. Na N uzloch, označovaných ako pracovníci, beží YARN NodeManager a HDFS DataNode. Toto prináša výhodu jednoduchej škálovateľnosti klastra a tiež to, že jeho nasadenie a konfigurácia prebieha z pohľadu užívateľa kliknutím na jedno tlačidlo. Nevýhodou je však to, že toto riešenie je spoplatnené. Je však možné získať skúšobnú licenciu, ku ktorej dostane užívateľ kredit tristo dolárov. Užívateľ, v rámci tejto platformy, platí za výpočtový čas skonzumovaný jeho klastrom a za priestor, ktorý zaberajú jeho dáta.

7.2 Práca so systémom Hadoop

Systém Hadoop je rozsiahle riešenie, ktorého konfigurácia a monitorovanie nie je triviálne. V tejto podkapitole budú popísané prostriedky používané na monitorovanie klastru systému Hadoop. Systém je možné manažovať cez prostredie systémového príkazového riadku, ale takisto aj cez užívateľské rozhranie, ktoré je realizované prostredníctvom webových aplikácií. Priamo so systémom Hadoop sú dodávané napríklad webové aplikácie k JobHistory serveru alebo k YARN ResourceManager-u. Rozšírené možnosti monitorovania klastru priamo z jednej webovej aplikácie ponúka open-source projekt Ambari. Jedná sa o samostatný projekt od Apache Foundation, ktorého cieľom je monitorovanie a správa Hadoop klastru. Open source webová aplikácia Hue sa zameriava na prácu s dátami uloženými v HDFS.

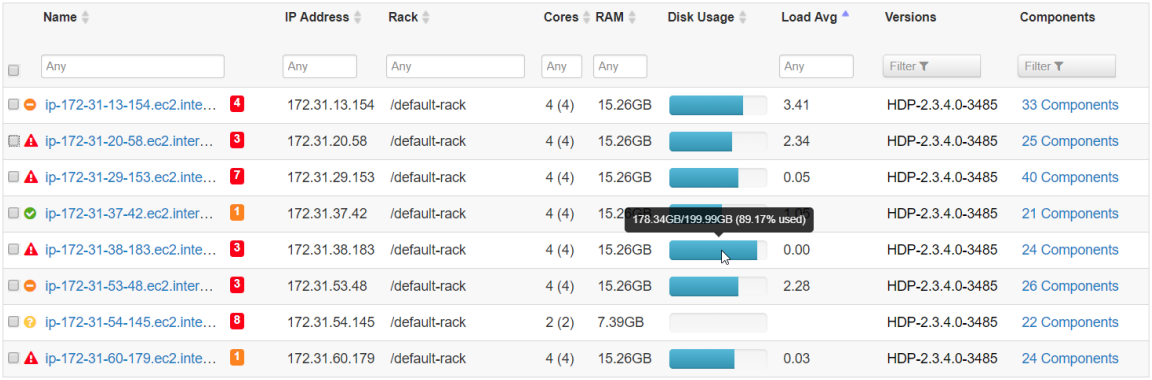
7.2.1 Spravovanie Hadoop klastra

Ako bolo vyššie zmienené, webová aplikácia Ambari sa zameriava na správu Hadoop klastra. Monitorovanie klastra môžeme rozdeliť na dve hlavné časti. Prvou je monitorovanie samotných strojov, kde náš klaster beží. Druhou monitorovanie služieb, ktoré sú v našom klastri spustené.

Monitorovanie strojov klastra

Na monitorovanie strojov Hadoop klastra v aplikácií Ambari slúži sekcia Hosts, zobrazená na obrázku 7.1. V tejto sekcií sa nachádza zoznam všetkých strojov a zoznam ich parametrov, ako je počet jadier procesora, veľkosť operačnej pamäte, veľkosť úložného priestoru a jeho využitie. Ďalšími informáciami zobrazenými o jednotlivých strojoch klastra je ich IP

adresa, zaradenie do konkrétneho racku a verzia systému. Poslednou kategóriou sú oznámenia o stave jednotlivých strojov, čiže informácie o tom, ktoré služby boli nakonfigurované, aby na nich bežali a ich stav, priemerné vyťaženie stroja a alarmy, ktoré slúžia na to, aby oboznámili administrátora systému o dôležitej udalosti, ktorá nastala na danom stroji.



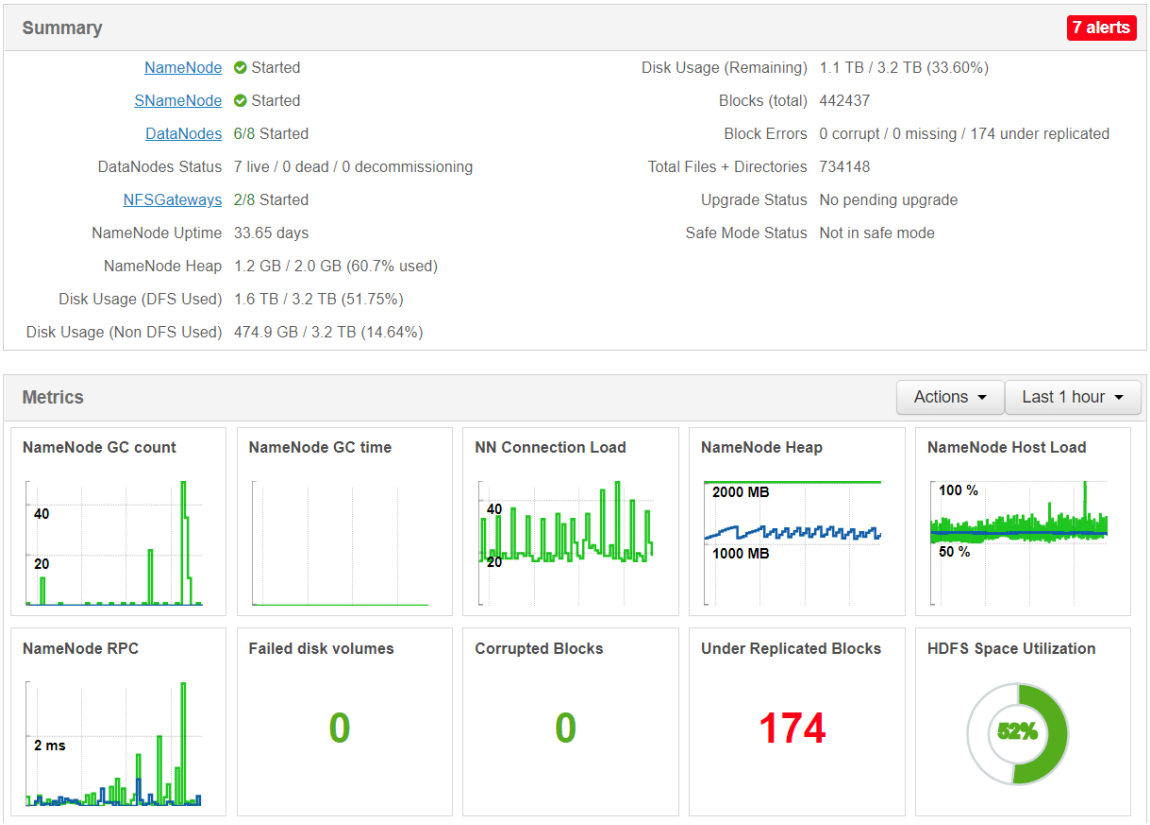
Obr. 7.1: Monitorovanie klastra

Po kliknutí na názov stoja sa otvorí detailné zobrazenie jednotlivých parametrov a prístup ku konfigurácií jednotlivých služieb, bežiacich na zvolenom stroji.

Monitorovanie služieb

Monitorovanie jednotlivých služieb prebieha po kliknutí na vybranú službu v ľavej strane aplikácie Ambari. V rámci tohoto projektu boli využívané 2 služby, ktoré môžu byť spúšťané na systéme Hadoop a to služba HDFS a služba YARN. Ich monitorovanie teda prebieha vo webovej aplikácií výberom rovnomenných položiek v zozname služieb alebo pomocou konzolových aplikácií. Na monitorovanie HDFS z príkazového riadku je možné použiť `dfsadmin` a na monitorovanie služby YARN rovnomennú konzolovú aplikáciu.

Na obrázku 7.2 je zobrazená sekcia správy HDFS. Obsahuje súhrn podstatných informácií predovšetkým o komponente NameNode o tom koľko dátových uzlov beží v rámci systému a informácie o distribuovanom súborovom systéme. V spodnej časti sa nachádzajú metriky, ktoré si môže užívateľ nakonfigurovať sám.



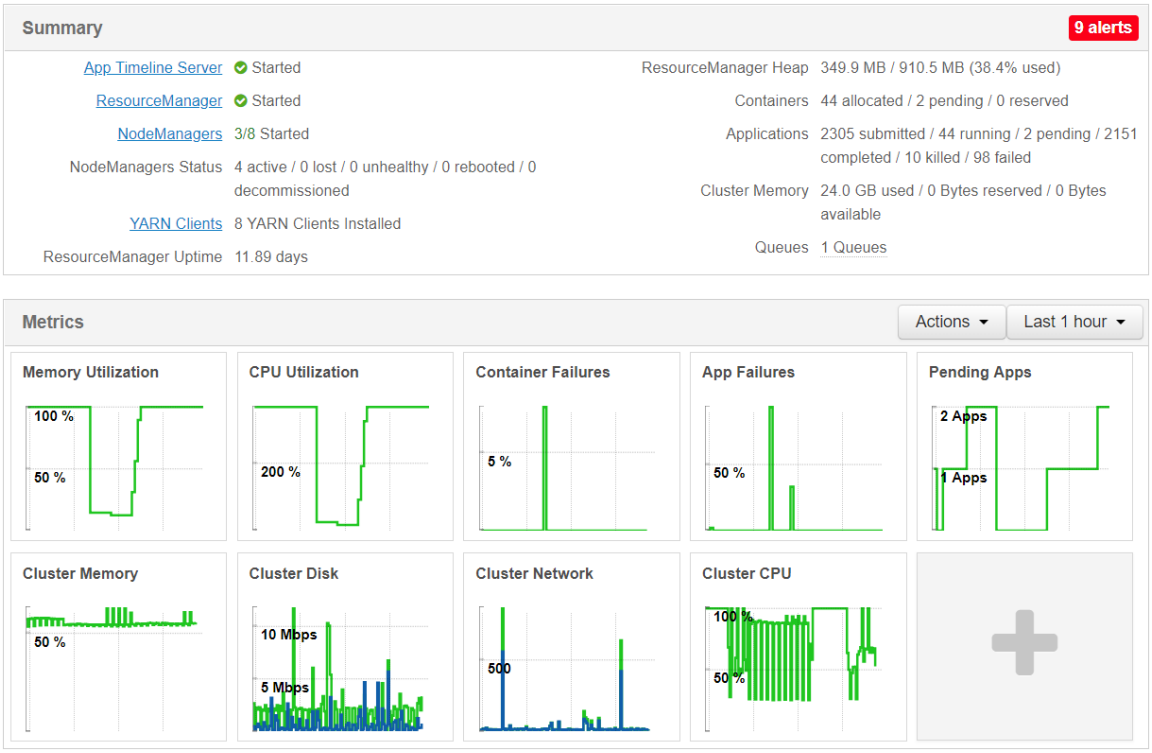
Obr. 7.2: Monitorovanie HDFS

Podobné informácie o stave distribuovaného súborového systému HDFS je možné získať pomocou príkazov z obrázku 7.3. Ich výstup je možné nájsť v prílohe A.

```
hdfs dfsadmin -report && hdfs dfsadmin -printTopology
```

Obr. 7.3: Príkazy na monitorovanie HDFS

Monitorovanie služby YARN je zobrazené na obrázku 7.4. V tejto sekcií môžeme nájsť informácie poskytované komponentom ResoureManager Hadoop klastra, informácie o celkovej výpočtovej sile klastra, krátku štatistiku aplikácií, ktoré v klastri aktuálne bežia alebo bežali.



Obr. 7.4: Monitorovanie YARN

V prípade monitorovania služby YARN z príkazového riadku je potrebné použiť príkazy na obrázku 7.5. Ich výstup je možné nájsť v prílohe A.

```
yarn node -list && yarn top
```

Obr. 7.5: Príkazy na monitorovanie YARN

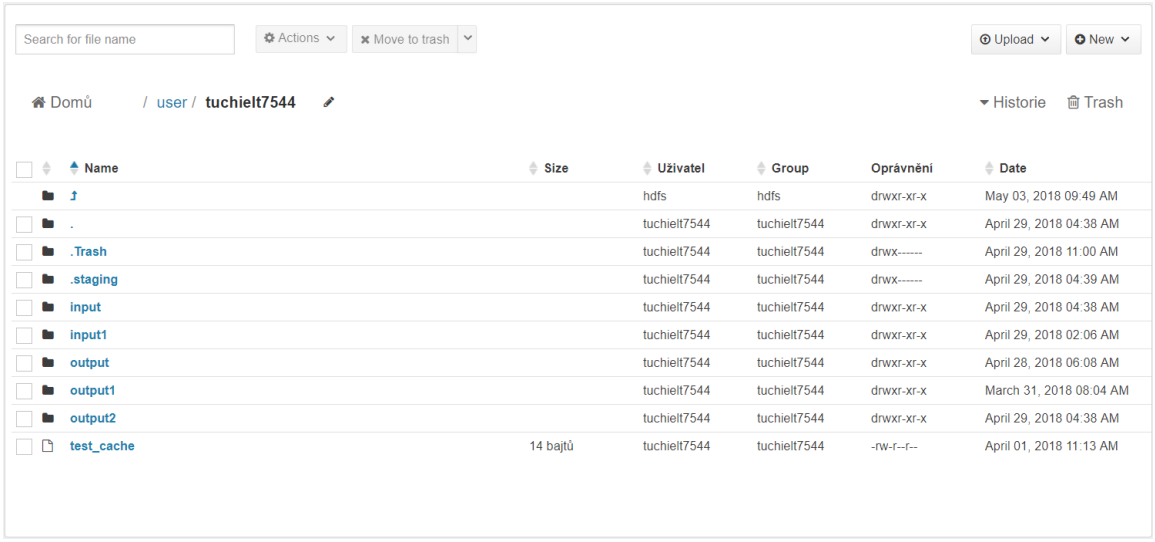
7.2.2 Práca s HDFS

Rovnako, ako monitorovanie systému, tak aj práca s distribuovaným súborovým systémom je možná priamo z príkazového riadku alebo pomocou webového klienta. Webové rozhranie zastrešuje open-source aplikácia Hue, vyvíjaná pod záštitou spoločnosti Cloudera.

Webová aplikácia Hue poskytuje grafické rozhranie pre prácu s dátami uloženými v zdieľanom súborovom systéme klastra. Na rozdiel od aplikácie Ambari sa zameriava na spravovanie a analyzovanie obsahu, ktorý sa nachádza v HDFS. Na samotnú správu súborov a zložiek uložených v HDFS slúži záložka File Browser. Na obrázku 7.7 je zobrazené prehliadanie súborov prostredníctvom aplikácie Hue. Toto zobrazenie je ekvivalentom príkazu z obrázku 7.6. Výstup tohoto príkazu je možné nájsť v prílohe A.

```
hadoop fs -ls
```

Obr. 7.6: Prehliadanie domovskej zložky v HDFS



Obr. 7.7: Prehliadanie súborov v HDFS

Aplikácia Hue prináša výhodu najmä v prípade využitia cloudového riešenia. Umožňuje totiž, prostredníctvom prehliadača súborov, nahrávať súbory priamo z užívateľovho počítača. Jedná sa o podstatné zjednodušenie oproti nahrávaniu z príkazového riadku, kde musí užívateľ nahrať súbory najprv do normálneho súborového systému stroja v klastri a potom nahrať do HDFS pomocou príkazu z obrázku 7.8. Uvedený príklad skopíruje súbor `file.txt` do zložky `input` v domovskej zložke užívateľa v HDFS.

```
hadoop fs -copyFromLocal file1.txt input/
```

Obr. 7.8: Nahratie súboru do HDFS

7.2.3 Spúšťanie aplikácií a správa úloh

Spúšťanie aplikácií je dôležitou súčasťou analýzy dát. Podrobný priebeh spúšťania aplikácií a ich úloh je popísaný v podkapitole 5.2.3. Ako bolo v danej časti textu zmienené ,aplikácia ako taká, žiadne výpočty neobsahuje a jej úlohou je nastavenie konfigurácie úloh a ich spustenie. Spustenie aplikácií prebieha cez konzolovú aplikáciu `yarn`. Demonštrácia spustenia výsledných aplikácií z príkazového riadku sa nachádza na obrázku 7.9. `MapReduceApplicationX.jar` špecifikuje `.jar` súbor, v ktorom sa nachádzajú triedy spúšťanej aplikácie. `MapReduceApplication` je názov triedy, v ktorej sa nachádza implementácia tela aplikácie. `Input`, `output` respektíve `items` sú vstupné parametre aplikácií. Parametre `input` a `output` v tomto prípade špecifikujú cesty k zložkám, kde sa nachádzajú vstupné súbory aplikácie a kam majú byť zapísané výstupné súbory. Druhá aplikácia očakáva na vstupe ešte jeden parameter `items` a to cestu k súboru, ktorý obsahuje položky, nad ktorými chceme analýzu vykonávať. Všetky tri parametre špecifikujú cesty v HDFS.

```
yarn jar MapReduceApplication1.jar MapReduceApplication input output
yarn jar MapReduceApplication2.jar MapReduceApplication input output items
```

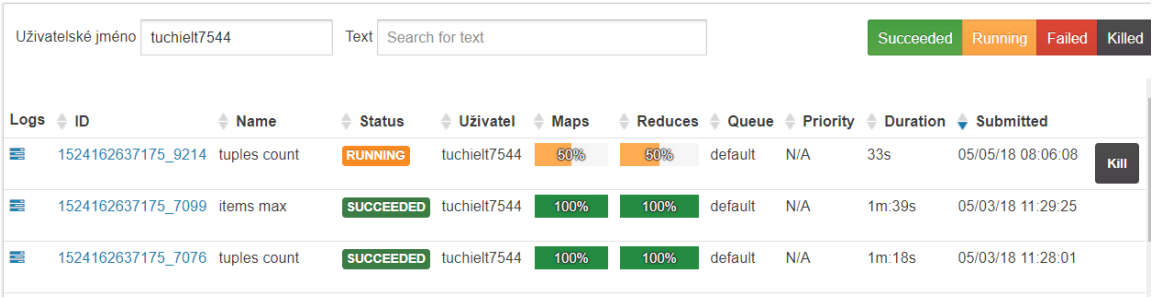
Obr. 7.9: Spúšťanie MapReduce aplikácií

Po spustení aplikácie dôjde k vypísaniu informácií o úlohách spúšťaných aplikáciou. Užívateľ je informovaný o počte vstupných ciest, ktoré budú spracované. Toto číslo je zhodné s počtom súborov vo vstupnej zložke. Druhou informáciou je počet vstupných úsekov. Podstatnými informáciami sú jednoznačné identifikátory aplikácie a úlohy. Na najspodnejšom riadku sa nachádza url adresa do webového rozhrania History serveru, kde je možné sledovať spustenú úlohu. Výpis po spustení aplikácie je možné nájsť na obrázku 7.10.

```
INFO impl.TimelineClientImpl: Timeline service address:
http://ip-172-31-13-154.ec2.internal:8188/ws/v1/timeline/
INFO client.RMProxy: Connecting to ResourceManager at
ip-172-31-53-48.ec2.internal/172.31.53.48:8050
INFO input.FileInputFormat: Total input paths to process : 1
INFO mapreduce.JobSubmitter: number of splits:1
INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524162637175
_7099
INFO impl.YarnClientImpl: Submitted application application_1524162637175
_7099
INFO mapreduce.Job: The url to track the job:
http://a.cloudxlab.com:8088/proxy/application_1524162637175_7099/
```

Obr. 7.10: Spustenie aplikácie

Zjednodušenie monitorovania a správy bežiacich aplikácií prináša aplikácia Hue, ktorej časť Job Browser sa na ňu zameriava. Užívateľ pomocou tejto aplikácie môže sledovať všetky úlohy, ktoré boli spúšťané alebo bežia v klastri. Pri bežiacich úlohach je možné sledovať ich progres alebo ich zabiť, čo ich okamžite ukončí. Zoznam úloh zobrazuje obrázok 7.11. Po kliknutí na konkrétnu úlohu sa užívateľovi zobrazia záložky, na ktorých môže sledovať informácie o zvolenej úlohe, ako napríklad metadáta, alebo veľmi zaujímavé počítadla, pomocou ktorých je možné sledovať, koľkokrát bola vykonaná podúloha map alebo reduce, veľkosti vstupov a výstupov týchto podúloh, a ďalšie informácie podobného charakteru.



Obr. 7.11: Prehliadanie úloh

7.3 Implementácia aplikácií MapReduce

7.3.1 Implementácia generátoru dát

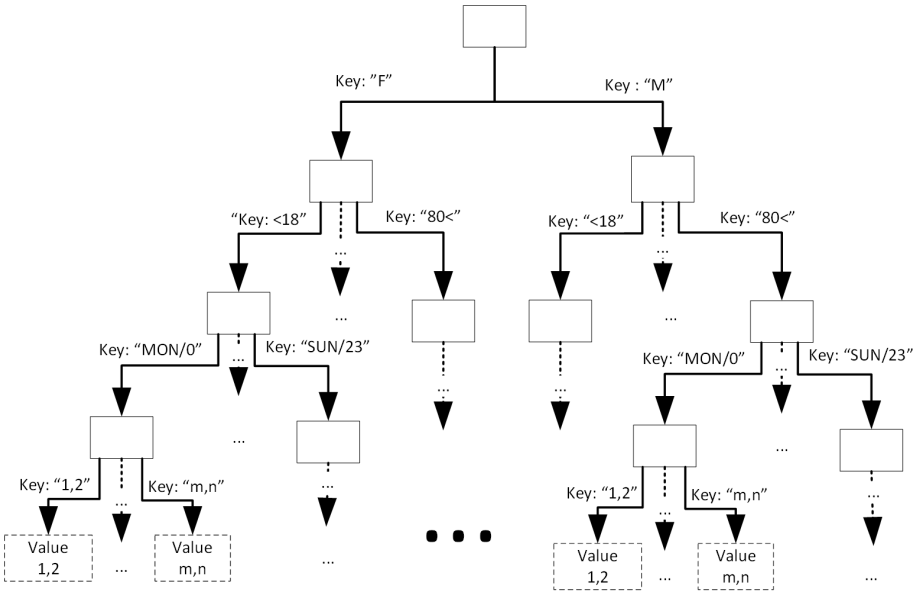
Generátor dát je dôležitou časťou tejto práce. Okrem toho, že vytvára vstup pre aplikácie implementované v tejto práci, slúži aj na kontrolu ich správneho fungovania. Jeho fungovanie ovplyvňujú rôzne parametre, ktorých popis je možné nájsť v prílohe B. Generátor dát je implementovaný v skriptovacom jazyku Python verzie 2.7. Jazyk Python bol zvolený pre svoju jednoduchosť, intuitívnu prácu s textovými reťazcami a jednoduchú prácu s dátovými štruktúrami.

Funkcionalitu generátoru dát môžeme rozdeliť na 2 časti, prvou je generovanie dátovej sady a druhou je vytvorenie výstupných štatistík nad touto dátovou sadou, proti ktorým je možné kontrolovať výstupy MapReduce aplikácií.

Generovanie dát je postavené na module `random`. Táto funkcionalita je z implementačnej stránky relatívne jednoduchá, keď sa opakovaným volaním nad rôznymi intervalmi, získavajú náhodné hodnoty pre dátum, veľkosť nákupného košíka, vek, pohlavie, a následne aj samotné položky. Dôležité je však to, aby sa v košíku nenachádzali duplicitné hodnoty položiek, preto po vygenerovaní dôjde ku kontrole, či sa daná položka v košíku už raz nenachádza. Preto sa na ukladanie košíka používa dátová štruktúra `Set`, ktorá umožňuje ukladať len unikátne položky. Po vygenerovaní každého záznamu o nákupe je záznam zapísaný do výstupného súboru. Ak by sme zapisovanie nechali až nakoniec, mal by generátor dát nezvládnuteľné pamäťové nároky.

Na ukladanie výsledných agregovaných hodnôt slúži zložená stromová štruktúra, kde uzly tvoria štruktúry typu `Dictionary` a listy tvoria numerické hodnoty. Na obrázku 7.12 sa nachádza schéma tejto zloženej štruktúry. Koreňová úroveň obsahuje len dva kľúče, ktoré sú rovné pohlaviam. V druhej úrovni sa nachádza šesť kľúčov, ktoré sú znamenajú zaradenie do vekovej kategórie. V tretej úrovni sa nachádzajú kľúče definujúce časový úsek týždňa. Vo štvrtej a poslednej úrovni tvorenej štruktúrami `Dictionary` sa nachádzajú kľúče reprezentované dvojicami čísel položiek, cez ktoré sa pristupuje už priamo k samotným hodnotám počtu výskytov daných položiek. Prvé tri úrovne stromu sú vytvorené pri spustení generátora, posledná úroveň je vytváraná počas behu, podľa toho ako vznikajú dvojice. Po úspeš-

nom ukončení generovania sú nájdené maximálne hodnoty a tie sú vypísané do výstupného súboru.



Obr. 7.12: Schéma stromovej štruktúry generátoru dát

Z dôvodu, že agregované hodnoty musia byť počas behu aplikácie uložené v pamäti, je pri veľkom počte unikátnych položiek beh skriptu pamäťovo náročný. Preto je pre generovanie rozsiahlych dátových sád s veľkým množstvom položiek nutné vykonávať bez výpočtu výsledkov agregácie.

7.3.2 Implementácia MapReduce aplikácií

Na obrázku 7.13 sa nachádza UML diagram tried implementovaných aplikácií. Diagram bol vytvorený s miernou abstrakciou, aby obsiahol obidve aplikácie, ktoré majú podobnú základnú kostru.

Jadro aplikácie

Aplikácie sú implementované v jazyku Java s využitím knižníc Hadoop. Každá aplikácia je implementovaná v rámci jednej triedy, ktorá implementuje rozhranie `Tool` a rozširuje triedu `Configured` z aplikačného rozhrania Hadoop. Vstupným bodom aplikácií je statická metóda `main`. Táto metóda obsahuje len zavolanie statickej metódy `run` triedy `ToolRunner`, ktorej sú predané ako parametre inštancia triedy `Configuration` a inštancia triedy aplikácie. Konfigurovanie a spúšťanie úloh prebieha až v metóde `run`. Jej telo je možné rozdeliť na dve časti a to konfiguráciu úloh a ich spúšťanie.

Jednotlivé MapReduce úlohy stelesňujú inštancie triedy `Job`. Ich vytváranie prebieha volaním statickej metódy `getInstance` triedy `Job`. Ako parametre očakáva inštanciu triedy

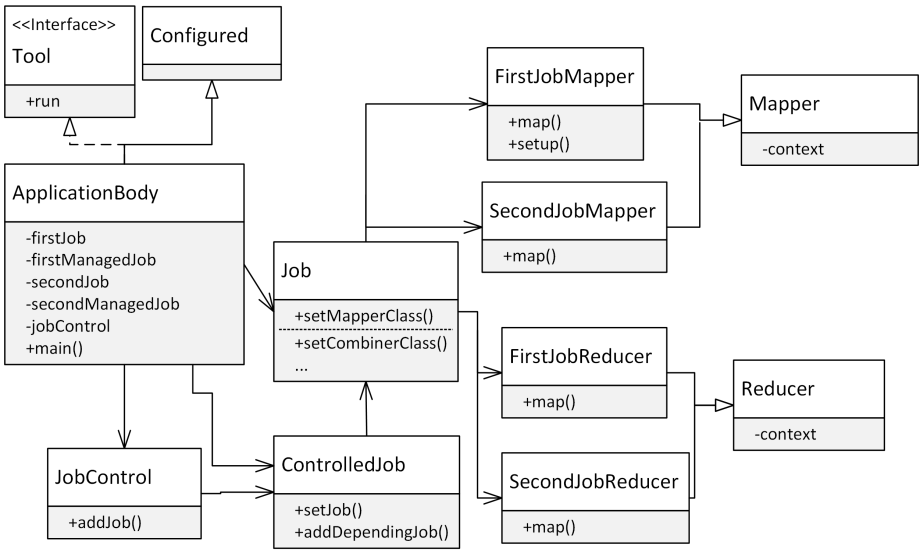
`Configration` a textový reťazec s názvom úlohy. Tento názov sa potom zobrazuje vo webovej aplikácií na sledovanie úloh popísanej v podkapitole 7.2.3. Následne je potrebné nastaviť podúlohy map a reduce. Tie sú tiež reprezentované samotnými triedami. Nastavenie podúlohy map prebieha zavolaním metódy `setMapperClass`, ktorej ako parameter odovzdáme názov triedy, ktorá implementuje požadovanú map podúlohu. Obdobne sa volaním `setReducerClass` nastavuje trieda s implementáciou reduce podúlohu. V prípade, že chceme použiť aj `Combiner`, zavoláme metódu `setCombinerClass`, ktorej odovzdáme totožnú triedu ako pri volaní `setReducerClass`. Ďalšími parametrami úlohy, ktoré je potrebné nastaviť, je formát výstupu, kde treba nastaviť výstupný kľuč a výstupnú hodnotu. Na nastavenie typu výstupného kľúča slúži metóda `setOutputKeyClass` a na nastavenie typu výstupnej hodnoty slúži metóda `setOutputValueClass`. V prípade, že sa typy výstupného kľúča alebo hodnoty podúlohy map líšia od výstupných typov celej úlohy, je potrebné ešte zavolať metódy `setMapOutputKeyClass`, respektíve `setMapOutputValueClass`, inak dôjde k výnimke a zlyhaniu celej úlohy. Celočíselné hodnoty sú reprezentované pomocou `IntWritable.class`, čísla s desatinnou čiarkou `FloatWritable.class` a textové hodnoty pomocou `Text.class`. Tieto triedy sú súčasťou knižníc rozhrania Hadoop. Posledným konfiguračným krokom je nastavenie vstupných a výstupných ciest pre úlohu. To prebieha volaním statickej metódy `addInputPath` triedy `FileInputFormat` pre nastavenie vstupu a `setOutputPath` triedy `FileOutputFormat` pre nastavenie výstupnej cesty. Pre úlohy vytvárajúce medzivýsledok je cesta k výstupu nastavená na do podložky temp. Táto cesta je potom vstupom nadväznej podúlohy. Jej výstup je nastavený do podzložky final.

Poslednou časťou je spustenie nakonfigurovaných úloh. Aby bolo možné úlohy reťaziť je potrebné pre každú z nich vytvoriť objekt triedy `ControlledJob`. Následne sa pre neho volaním metódy `setJob` nastaví úloha ktorú zapudruje. Závislosť úlohy sa potom nastavuje volaním metódy `addDependingJob`, ktorej sa ako parameter predá objekt triedy `ControlledJob`, ktorý zapuzdruje úlohu, na ktorej má byť daná úloha závislá. O správne spúšťanie úloh sa stará objekt vytvorený z triedy `JobControl`. Ten je následne spustený vo vlastnom vlákne, kde postupne spúšťa nami nakonfigurované úlohy. Aplikácia potom čaká na ukončenie všetkých úloh. Čakanie prebieha aktívne, kde aplikácia v cykle každých 5 sekúnd na výstup napíše počty úloh, ktoré bežia, čakajú, alebo boli ukončené, či už úspešne alebo neúspešne. Druhou časťou výpisu je informácia o aktuálne bežiacich úlohách a informácia o progrese jednotlivých podúloh v percentách.

Implementácia podúloh

Podúlohy map sú implementované rozšírením triedy `Mapper` a implementáciou metódy `map`. Celkovo boli implementované štyri podúlohy map, pre každú aplikáciu dve. Niektoré črty majú spoločné. V obidvoch podúlohach map, ktoré patria pod prvé úlohy každej z aplikácií sa nachádza inštancia triedy `Calendar` z balíčku `util`. Ten je potom použitý na transformácie časov na časové úseky. Pre prvú podúlohu map aplikácie na zistenie záujmu o vybrané produkty je implementovaná aj metóda `setup`. Tá sa volá ešte pred spustením podúlohy. V

nej sa z komponenty systému nazývanej zdieľaná cache prečítajú položky, ktoré majú byť filtrované zo vstupu. Podúlohy reduce sú implementované rozšírením triedy `Reducer`. Ich chovanie sa nachádza v implementácií metódy reduce. Všetky úlohy odovzdávajú dvojice hodnota a kľúč na ďalšie spracovanie volaním metódy `write` členskej premennej `context` vytvorenej z triedy `Context`. O jej vytváranie a správu sa stará implementácia základných tried priamo v aplikačnom rozhraní systému Hadoop.



Obr. 7.13: UML diagram tried aplikácií

7.3.3 Preklad a zabalenie aplikácií

Aby bolo možné implementované aplikácie spúšťať, je potrebné ich najprv preložiť a zabaliť do `.jar` archívu. Preklad prebieha zavolaním príkazu `hadoop com.sun.tools.javac.Main *.java` v zložke s implementáciou tried danej aplikácie. Pre úspešné preloženie aplikácie je nutné mať nastavené cesty prostredia. Príkazy pre ich nastavenie sa nachádzajú na obrázku 7.14.

```
export JAVA_HOME=/usr/java/default
export PATH=${JAVA_HOME}/bin:${PATH}
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Obr. 7.14: Príkazy pre nastavenie ciest prostredia

Príkaz `jar cf MapReduce Application1.jar *.class` zabalí preložené triedy do `.jar` archívu. Názov výstupného archívu špecifikuje `MapReduceApplication1.jar`. Príkaz musí byť spustený v zložke, kde bol vykonaný preklad aplikácie. Po zabalení aplikácie je dobré odstrániť všetky vytvorené `.class` súbory, pretože v prípade, keď je pri spúšťaní aplikácie `.jar` archív v tej istej zložke ako tieto vytvorené `.class` súbory, dôjde počas behu aplikácie k výnimke a jej beh je neúspešný.

Kapitola 8

Testovanie výsledných aplikácií

Testovanie výsledných aplikácií môžeme rozdeliť do dvoch fáz. Prvou z nich je testovanie na správnu funkcionalitu. Druhou fázou je experimentovanie s aplikáciami, pričom je pozorované, aký vplyv majú jednotlivé charaktery dátových sád na dĺžku behu aplikácií a veľkosť výstupných dát.

8.1 Funkčné testy

8.1.1 Popis postupu testovania

Cieľom funkčných testov je overiť správne chovanie aplikácie. V rámci funkčného testovania, preto nie je potrebné aplikácie testovať nad rozsiahlymi dátovými sadami. Testovanie je z toho dôvodu založené na porovnávaní výstupu aplikácie s výstupnými agregáciami vypočítanými generátorom dát. Pre funkčné testovanie bolo vygenerovaných 50 testovacích vstupov o veľkosti približne 20 megabajtov. Súbory boli následne nahrané do HDFS do zložiek `inputX`, kde X má hodnotu 1 až 20. Potom boli nad nimi postupne spustené testované aplikácie. Po ukončení behu aplikácií boli overené výstupy proti referenčným výstupom generátoru dát. V prípade nezhôd došlo k ladeniu aplikácie a proces testovania sa zopakoval.

8.1.2 Testovacie prostredie

Testovanie prebiehalo v cloudovom prostredí CloudxLab, ktoré je popísane v kapitole 7.1. Keďže testovacie sady dát nie sú príliš rozsiahle, je možné testy spúšťať aj v tomto prostredí. Výhodou využitia tohoto prostredia pre funkčné testy je aj to, že aplikácie je možné spustiť mnohokrát bez obmedzenia. To je vhodné pri opravách chýb a ladení aplikácií.

8.1.3 Výsledky testov

Testy pomohli odhaliť nedostatky v implementácií, ktoré boli opravené. Na konci tejto fázy testovania boli všetky výstupy korektné, preto možno obidve aplikácie považovať za správne fungujúce.

8.2 Experimentovanie s dátovými sadami

Zrejmým faktorom, ktorý ovplyvňuje dĺžku behu aplikácie je veľkosť jej vstupu. Keďže každý záznam zo vstupu je spracovaný práve raz vieme, že zložitosť aplikácií v závislosti na veľkosti vstupu je lineárna. Výstupy oboch aplikácií nezávisia na veľkosti vstupnej dátovej sady. Počet vekových kategórií, týždňov v roku aj dní v týždni, respektíve hodín v dni je totiž pevný a preto je veľkosť výstupu konštantná v závislosti na veľkosti vstupu. Pre skúmanie toho, ako charakter dát ovplyvňuje beh aplikácie, boli generované rôzne sady dát. Následne boli nad testovacími dátami aplikácie spustené. Aby nedošlo k ovplyvňovaniu výsledkov, bol počet bežiacich aplikácií vždy 1.

Charakter dát ovplyvňujúci aplikáciu na zistenie najčastejšie kupovaných dvojíc produktov

Vzhľadom na to, že výstup aplikácie vzniká dvakrát, môže dosahovať relatívne veľký rozsah. Jedná sa najmä o medzi výstup, pri ktorom je agregácia dát nízka a vytváraním dvojíc dáta vznikajú. Zistiť, aké veľké výstupy aplikácia produkuje, je nutné najmä z dôvodu správneho nastavenia výpočtového klastra. Ako faktory ovplyvňujúce dĺžku behu aplikácie a rozsah výstupných dát, boli určené:

- Veľkosť nákupného košíka: Pretože v prvej úlohe map sa nákupný košík prechádza v dvoch vnorených cykloch, je počet položiek v košíku hlavným faktorom ovplyvňujúcim dĺžku tejto podúlohy, a preto aj samotnej aplikácie.
- Počet unikátnych produktov: Vzhľadom na to, že v prvej úlohe aplikácie sú vytvárané dvojice produktov, ich celkový počet ovplyvňuje počet možných dvojíc, ktoré môžu vzniknúť. Pretože všetky unikátne dvojice, ktoré boli nájdené sú zapisované na disk ako výstup úlohy, je počet všetkých možných kombinácií hlavným faktorom rozsahu výstupných dát.

Charakter dát ovplyvňujúci aplikáciu na zistenie záujmu o vybrané produkty

Aj výstup tejto aplikácie vzniká dvakrát, ale miera agregácie už pri prvom výstupe je oveľa vyššia. Preto boli pre túto aplikáciu zvolené, ako skúmané, nasledovné vlastnosti:

- Veľkosť nákupného košíka: V prvej úlohe map sa nákupný košík prechádza v cykle a porovnáva sa s vybranými položkami, pre ktoré chceme robiť analýzu. Z tohoto dôvodu, bude dĺžka behu tejto aplikácie ovplyvnená priemernou veľkosťou košíka.

- Počet filtrovaných produktov: Tak isto, ako sa prechádza košík v cykle, prechádza sa v cykle pre každú položku košíka aj zoznam, podľa ktorého sa majú z košíka vyberať produkty na filtrovanie. Preto veľkosť zoznamu položiek môže vplývať na dobu behu aplikácie.

8.2.1 Popis postupu testovania

Postup testovania aplikácie na zistenie najčastejšie kupovaných dvojíc produktov

Pre testovanie toho, ako ovplyvňuje beh aplikácie veľkosť nákupného košíka, bolo vygenerovaných celkom 24 dátových sád. Každá obsahovala 1 milión záznamov a počet unikátnych položiek bol 1000. Priemerná veľkosť košíka bola v rozmedzí 5 až 31 položiek s prírastkom približne 5 položiek. Pre každú úroveň veľkosti košíka boli vygenerované 4 sady.

Pre zistenie toho, ako ovplyvňuje počet unikátnych položiek beh aplikácie pre zistenie najčastejšie nakupovaných dvojíc produktov bolo vygenerovaných 20 dátových sád. Každá obsahovala 10 miliónov záznamov. Priemerná veľkosť košíka bola nastavená na 15 položiek. Premenlivým faktorom bol počet unikátnych položiek, ktorý sa zvyšoval z 500 až na 2500 s krokom 500.

Postup testovania aplikácie na zistenie záujmu o vybrané produkty

Pre účel skúmania toho, ako ovplyvňuje dĺžku behu aplikácie počet položiek v košíku, bolo vygenerovaných celkom 24 dátových sád. Každá obsahovala 10 miliónov záznamov vytvorených z 2500 unikátnych položiek. Priemerná veľkosť košíka sa menila rovnako ako v prechádzajúcom prípade.

Pre zistenie toho, ako ovplyvňuje počet filtrovaných položiek beh aplikácie pre zistenie záujmu o produkty, bola vygenerovaná sada s 1 miliónom záznamov a 1500 unikátnymi položkami. 20 dátových sád. Priemerná veľkosť košíka bola nastavená na 15 položiek. Premenlivým faktorom bol počet filtrovaných položiek. Ten začínal na počte 50 a končil na hodnote 250 s krokom 50 položiek. Pre každý rozsah filtrovacieho súboru boli vygenerované 4 testovacie súbory s náhodnými položkami.

8.2.2 Testovacie prostredie

Tieto testy bolo nutné spúšťať v plne funkčnom klastri bez obmedzenia veľkosti spracovávaných dát. Preto bolo využité cloudové prostredie Google Dataproc. V rámci neho bol nakonfigurovaný výpočtový klaster s maximálnymi možnými parametrami, ktoré testovacia licencia umožňovala. Konfiguráciu klastra zobrazuje tabuľka na obrázku 8.1. Takáto konfigurácia poskytovala celkovo 1,5 TB úložného priestoru, pričom faktor replikácie bol nastavený na hodnotu 2. Na spúšťanie aplikácií bolo k dispozícií 6 YARN kontajnerov,

v ktorých sú spúšťané jednotlivé podúlohy, a preto mohlo naraz bežať až 6 simultánne spustených úloh.

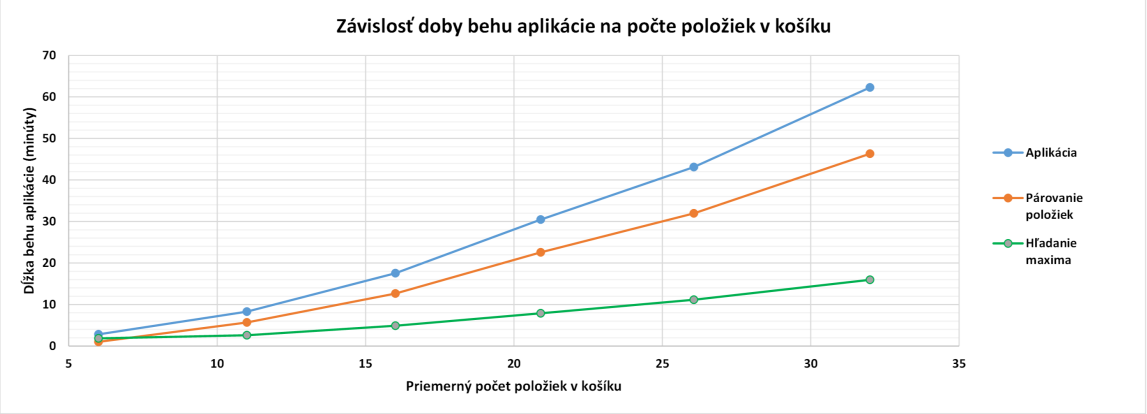
Rola	Jadrá CPU	RAM	Disk	Spustené služby	Počet
master	2	7,5GB	500GB	NameNode, ResourceManager	1
worker	2	7,5GB	500GB	DataNode, NodeManager	3

Obr. 8.1: Konfigurácia testovacieho klastra

8.2.3 Výsledky testov

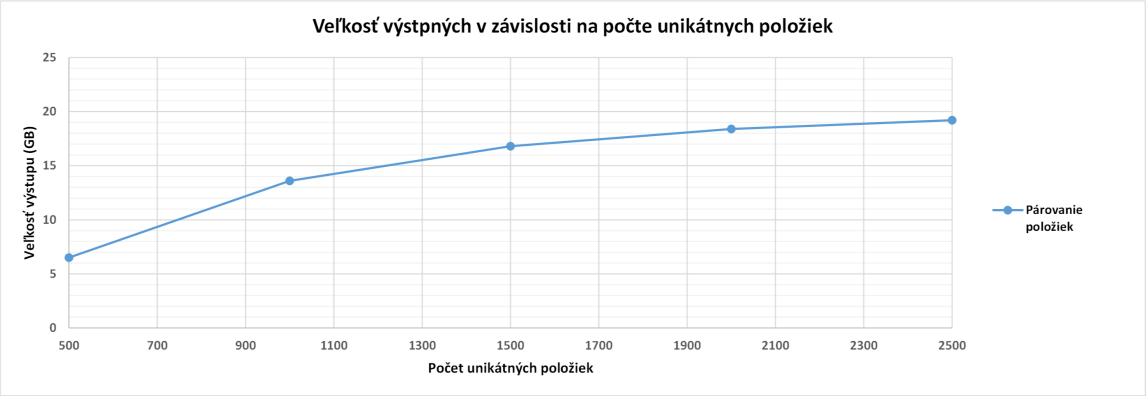
Výsledky testov aplikácie zaisťujúcej najčastejšie kupované dvojice produktov

Obrázok 8.2 zobrazuje závislosť dĺžky behu aplikácie na priemernej veľkosti nákupného košíka. Z grafu je zrejmé, že bol potvrdený predpoklad, že so zvyšujúcim sa priemerným počtom položiek v košíku rastie aj dĺžka behu aplikácie. Podľa priebehu grafu je možné konštatovať, že závislosť je kvadratická. Teoretická časová zložitosť dvoch vnorených cyklov je tiež kvadratická.



Obr. 8.2: Závislosť dĺžky behu aplikácie na veľkosti nákupného košíka

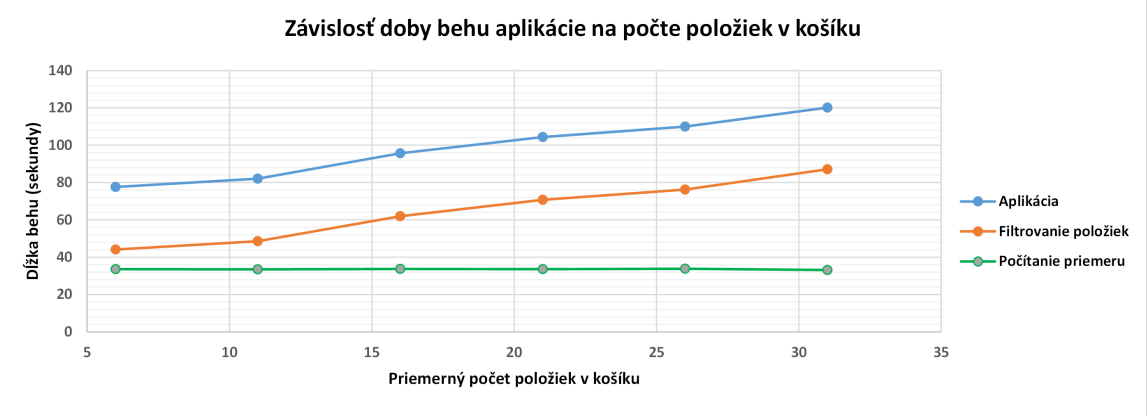
Na obrázku 8.3 sa nachádza graf zobrazujúci závislosť veľkosti výstupu aplikácie na počte unikátnych položiek. Závislosť z grafu sa môže javiť ako logaritmická. Toto je však nepravdepodobné, keďže funkciu pre počet všetkých dvojíc v nákupnom košíku o počte položiek n, môžeme vyjadriť ako $\sum_{x=2}^{n}(x-1)$, čo indikuje kvadratickú závislosť. Dôvodom, prečo graf funkcie vyzerá takto, môže byť to, že pri zvyšujúcom sa počte unikátnych položiek sa zvyšuje aj počet všetkých možných dvojíc a pri veľkosti vstupu, ktorý bol 10 miliónov záznamov, v ňom nemuseli byť obsiahnuté všetky možné kombinácie. To však istým spôsobom koreluje s realitou, keď je nepravdepodobné, že v nákupoch by vznikli všetky možné kombinácie predávaných produktov.



Obr. 8.3: Závislosť veľkosti výstupných dát na počte unikátnych položiek

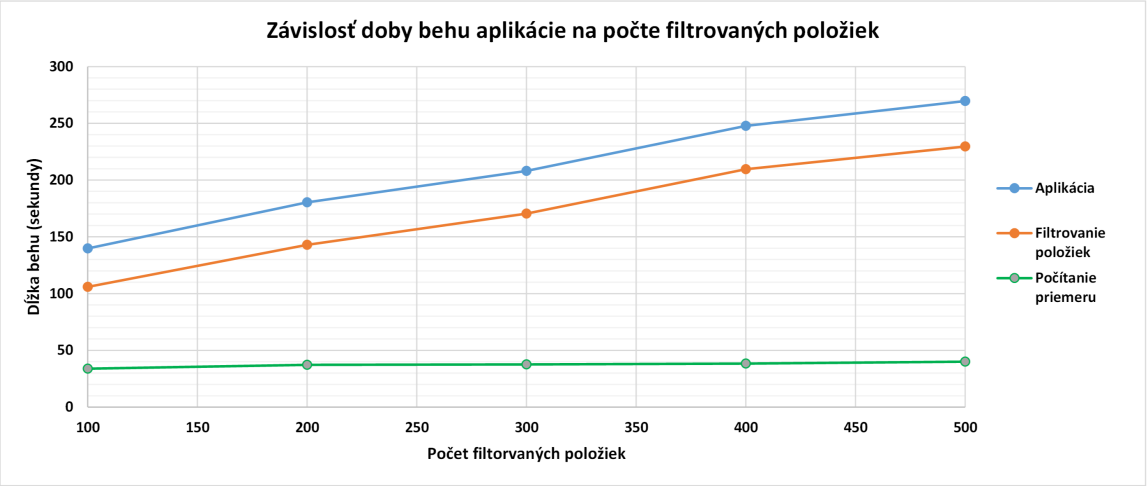
Výsledky testov aplikácie na zistenie záujmu o vybrané produkty

Obrázok 8.4 obsahuje graf, ktorý zachytáva výsledok skúmania vplyvu priemerného počtu položiek v nákupnom košíku na dĺžku behu jednotlivých úloh a celej aplikácie. Ako bolo predpokladané, počet položiek nákupného košíka ovplyvňuje len dobu behu prvej úlohy. Z grafu je možné zistiť, že aj napriek miernym odchýlkam, je časová zložitosť prvej úlohy a aj celej aplikácie na veľkos nákupného košíka lineárna. Druhá úloha má vzhľadom na dĺžku košíka konštantnú časovú závislosť.



Obr. 8.4: Závislosť dĺžky behu aplikácie na veľkosti nákupného košíka

Graf na obrázku 8.5 zobrazuje výsledok zisťovania, toho ako vplýva počet filtrovaných položiek na dĺžku behu aplikácie. Počet filtrovaných položiek ovplyvňuje najmä prvú úlohu. Jej časová závislosť na počte filtrovaných položiek je lineárna. Keďže počet filtrovaných položiek ovplyvňuje aj veľkosť výstupu prvej úlohy, je možné v grafe spozorovať, že mierne vplýva aj na dobu behu druhej úlohy. Vzhľadom na to, že tento výstup dosahuje veľkosť v rádoch kilobajtov maximálne megabajtov, je tento vplyv takmer nepatrný.



Obr. 8.5: Závislosť doby behu aplikácie na počte filtrovaných položiek

8.2.4 Zhodnotenie výsledkov experimentov

Cieľom experimentov s dátovými sadami bolo zistiť správanie aplikácií a overiť predpoklady získané analýzou ich implementácie. Experimentovanie prebehlo vo viacerých krokoch nad rôznymi dátovými sadami, ktorých súhrnný objem presiahol sto gigabajtov. Ako bolo predpokladané aplikácia na zistenie najčastejšie kupovaných dvojíc produktov dosahovala vysoké doby trvania behu a produkovala veľké množstvo výstupných dát. Toto správanie je však spôsobené netriviálnosťou analýzy, ktorú vykonáva. Analýzou jej výsledkov môžu byť získané veľmi cenné informácie o správaní zákazníkov, preto je cena za jej beh primeraná očakávanej výstupnej hodnote. Aplikácia je vhodná na prípravu reportov a materializáciu. Jednou z možností zrýchlenia aplikácie by bolo jej spustenie vo väčšom výpočtovom klastri.

Aplikácia na zistenie záujmu o vybrané produkty dosahovala relatívne krátke časy behu a výstupné dáta z obidvoch podúloh boli malé v porovnaní s veľkosťou analyzovanej dátovej sady. Aj pri veľkých objemoch vstupov dokázala vykonať analýzu v rozmedzí niekoľkých minút. Pri analyzovaní menších dátových vstupov, s využitím väčšieho klastra by, vzhľadom na krátku dobu dodania výsledkov, mohla byť použitá aj priamo koncovými užívateľmi.

Kapitola 9

Záver

Cieľom tejto práce bolo navrhnúť využitie technológie Big Data v podpore rozhodovania s využitím systému Apache Hadoop. Prvým krokom bolo naštudovanie podpory rozhodovania. V kapitole 2 sa nachádza popis tejto oblasti a zhodnotenie aktuálne využívaných technológií na podporu rozhodovania. Druhou oblasťou, ktorú bolo treba naštudovať, bol samotný koncept Big Data, ktorého popis sa nachádza v kapitole 3. V tejto kapitole boli zhrnuté aktuálne trendy v tejto oblasti a ich porovnanie s konceptom, využívaným v tradičných systémoch na podporu rozhodovania. Aby bolo možné navrhnúť a implementovať reálne aplikácie bolo potrebné naštudovať samotný systém Apache Hadoop, čo bolo do detailov vykonané v kapitole 5, pričom v kapitole 4 sa nachádza popis paradigma MapReduce, na ktorom je výpočtová časť systému Hadoop postavená. Získané poznatky boli využité v kapitole 6, ktorá obsahuje návrh systému spájajúceho Big Data s podporou rozhodovania a návrh aplikácií na spracovanie Big Data, ktoré umožňujú toto prepojenie. Pri návrhu aplikácií bolo využité viacnásobné spustenie MapReduce úloh v priebehu jednej aplikácie, čo zvyšuje analytické možnosti týchto aplikácií.

Druhou časťou práce bola samotná implementácia aplikácií. Aby mohli byť implementované aplikácie ladené a vykonané nad nimi experimenty s dátami, bolo potrebné preskúmať možnosti ich nasadenia a naštudovaná práca s výpočtovým klastrom, kde beží systém Hadoop. Popis možných nasadení systému sa nachádza v podkapitole 7.1. Práca so systémom Hadoop je demonštrovaná na praktických ukážkach v podkapitole 7.2. Aplikácie boli implementované v programovacom jazyku Java s využitím knižníc dodávaných v systéme Hadoop. Ten bol zvolený preto, že pracuje priamo nad aplikačným rozhraním MapReduce, čo umožňovalo určitú voľnosť pri implementácií a možnosť využitia klasických programových riešení. Napríklad možnosť vykonávať agregácie už v samotnej úlohe map. Preto, aby bolo možné vytvoriť tieto aplikácie bolo potrebné naštudovať aj samotné programátorské konštrukcie potrebné pre implementáciu aplikácií. Popis ich využitia je možné nájsť v podkapitole 7.3. Po implementovaní a overení správnej funkcionality boli aplikácie nasadené v reálnom výpočtovom klastri a boli nad nimi vykonané experimenty s dátovými sadami. Ich priebeh a zhrnutie je možné nájsť v kapitole 8. Nakoľko neboli k dispozícií reálne dáta,

museli byť testy vykonané nad generovanými dátami. To vyvolalo potrebu implementácie aj samotného generátora dát. Istou výhodou je to, že takto mohli byť aplikácie testované nad špecifickými sadami dát, čo by pri reálnych dátach nebolo možné.

Pokračovanie práce je možné rozvíjať viacerými smermi. Jedným z nich je vytvorenie celého navrhovaného systému prepojenia systémov na podporu rozhodovania a konceptu Big Data. Druhým smerom je možná implementácia ďalších aplikácií MapReduce slúžiacich na analýzu dát. Táto práca pootvorila v tejto oblasti pomyselné dvierka a je ju možné použiť ako študijný podklad pre implementáciu takýchto aplikácií a pre prácu so systémom Hadoop. Zrejme by ale bolo dobré pri pokračovaní zvážiť prípadnú spoluprácu so spoločnosťou, ktorá by bola schopná poskytnúť rozsiahle sady dát, pre ktoré by mohli byť tvorené nové aplikácie, a prípadne poskytnúť výpočtové kapacity pre zostavenie väčšieho Hadoop klastra.

Literatúra

[1] *CLHS: Function Reduce*. [Online; navšívené 30,12.2017].
URL http://clhs.lisp.se/Body/f_reduce.htm

[2] *Data Cube Materialization*. Shodhganga : a reservoir of Indian theses, [Online; navšívené 5.4.2018].
URL http://shodhganga.inflibnet.ac.in/bitstream/10603/97217/18/18_chapter%207.pdf

[3] *Data Cubes*. University of Regina, [Online; navšívené 5.4.2018].
URL http://www2.cs.uregina.ca/~dbd/cs831/notes/dcubes/dcubes.html

[4] *Data Warehouse Definition*. National University of Singapore, [Online; navšívené 5.4.2018].
URL https://www.comp.nus.edu.sg/~lingtw/cs4221/dw.pdf

[5] *Haskell reference*. [Online; navšívené 30,12.2017].
URL http://zvon.org/other/haskell/Outputprelude/map_f.html

[6] *HDFS*. [Online; navšívené 8.1.2018].
URL http://tutorialsdiary.com/our_courses/chapter-3-hdfs-introduction/

[7] *MapReduce Tutorial*. The Apache Software Foundation, [Online; navšívené 15.1.2018].
URL https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[8] *Powered by Apache Hadoop*. [Online; navšívené 4,1.2018].
URL https://wiki.apache.org/hadoop/PoweredBy

[9] *Reporting in BI*. [Online; navšívené 10.4.2018].
URL https://www.logianalytics.com/resources/bi-encyclopedia/reporting-bi/

[10] *What is a data lake?* [Online; navšívené 10.4.2018].
URL https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

[11] *Decision support systems v - big data analytics for decision making*. New York, NY: Springer Berlin Heidelberg, 2015, ISBN 978-3-319-18532-3.

[12] Borthakur, D.: *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, [Online; navšívené 10.1.2018].
URL https://svn.eu.apache.org/repos/asf/hadoop/common/tags/release-0.16.3/docs/hdfs_design.pdf

[13] Campbell, C.: *TOP FIVE DIFFERENCES BETWEEN DATA LAKES AND DATA WAREHOUSES*. [Online; navšívené 10.4.2018].
URL https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

[14] Chen, C. P.; Zhang, C.-Y.: Data-intensive applications, challenges, techniques and technologies. *Information Sciences*, ročník 275, 2014: s. 314–347, ISSN 00200255, doi:10.1016/j.ins.2014.01.015.
URL http://linkinghub.elsevier.com/retrieve/pii/S0020025514000346

[15] Henson, T.: *Schema On Read vs. Schema On Write Explained*. The Architecture of Open Source Applications, [Online; navšívené 10.4.2018].
URL http://www.aosabook.org/en/hdfs.html

[16] Hruška, T.: *BIG DATA - VELKÁ DATA*. FIT VUT v Brne, [Online; navšívené 30.12.2017].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/podklady/Prednasky/PIS/PIS306OLAP.pdf

[17] Hruška, T.: *Hadoop*. FIT VUT v Brne, [Online; navšívené 10.1.2018].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/podklady/Prednasky/PIS/PIS310BigData.pdf

[18] Hruška, T.; Ruttkay, L.; Šáfry, M.: *Nástroje pro podporu rozhodování OLAP, vizualizace a prezentace dat: Studijní opora*. FIT VUT v Brne, [Online; navšívené 2.1.2018].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/podklady/Opory/OPISOLAP.pdf

[19] Jones, M.; Nelson, M.: *Moving ahead with Hadoop YARN*. [Online; navšívené 15.1.2018].
URL https://www.ibm.com/developerworks/library/bd-hadoopyarn/

[20] Kunar, D.: *Understanding Big Data*. BigData Planet, [Online; navšívené 26.12.2017].
URL http://www.bigdataplanet.info/p/what-is-big-data.html

[21] Marr, B.: *Why only one of the 5 Vs of big data really matters*. IBM Big Data & Analytics Hub, 2015, [Online; navšívené 26.12.2017].
URL http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters

[22] Marvin, R.: *Data Lakes, Explained*. [Online; navšívené 10.4.2018].
URL https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

[23] Murthy, A.: *Apache Hadoop YARN*. Upper Saddle River, NJ: Pearson, [2014], ISBN ISBN9780321934505.

[24] Nielsen, F.: *Introduction to HPC with MPI for data science*. New York, NY: Springer Berlin Heidelberg, 2016, ISBN ISBN978-3-319-21902-8.

[25] Robert Chansler, k. a.: *The Hadoop Distributed File System*. The Architecture of Open Source Applications, [Online; navšívené 10.1.2018].
URL http://www.aosabook.org/en/hdfs.html

[26] Tutunea, M. F.; Rus, R. V.: *Business Intelligence Solutions for SME's*. *Procedia Economics and Finance*, ročník 3, 2012: s. 865–870, ISSN 22125671, doi:10.1016/S2212-5671(12)00242-0.
URL http://linkinghub.elsevier.com/retrieve/pii/S2212567112002420

[27] Vohries, W.: *How Many Vs in Big Data? The Characteristics that Define Big Data*. Data Science Central, [Online; navšívené 26.12.2017].
URL https://www.datasciencecentral.com/profiles/blogs/how-many-v-s-in-big-data-the-characteristics-that-define-big-data

[28] White, T.: *Hadoop*. Beijing: O'Reilly, fourth edition vydanie, [2015], ISBN 978-1-491-90163-2.

Príloha A

Výstupy vybraných príkazov

Výstup príkazu hdfs dfsadmin -report

```
Configured Capacity: 1584932302848 (1.44 TB)
Present Capacity: 1509140525056 (1.37 TB)
DFS Remaining: 1509140451328 (1.37 TB)
DFS Used: 73728 (72 KB)
DFS Used\%: 0.00\%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0
Pending deletion blocks: 0

-------------------------------------------------
Live datanodes (3):

Name: 10.132.0.2:9866 (cluster-6ff6-w-1.c.vernal-mantra-199709.internal)
Hostname: cluster-6ff6-w-1.c.vernal-mantra-199709.internal
Decommission Status : Normal
Configured Capacity: 528310767616 (492.03 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 3665764352 (3.41 GB)
DFS Remaining: 503046811648 (468.50 GB)
DFS Used\%: 0.00\%
DFS Remaining\%: 95.22\%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
```

```
Cache Used\%: 100.00\%
Cache Remaining\%: 0.00\%
Xceivers: 2
Last contact: Sun May 13 19:38:16 UTC 2018


Name: 10.132.0.3:9866 (cluster-6ff6-w-2.c.vernal-mantra-199709.internal)
Hostname: cluster-6ff6-w-2.c.vernal-mantra-199709.internal
Decommission Status : Normal
Configured Capacity: 528310767616 (492.03 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 3665756160 (3.41 GB)
DFS Remaining: 503046819840 (468.50 GB)
DFS Used\%: 0.00\%
DFS Remaining\%: 95.22\%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used\%: 100.00\%
Cache Remaining\%: 0.00\%
Xceivers: 2
Last contact: Sun May 13 19:38:16 UTC 2018


Name: 10.132.0.4:9866 (cluster-6ff6-w-0.c.vernal-mantra-199709.internal)
Hostname: cluster-6ff6-w-0.c.vernal-mantra-199709.internal
Decommission Status : Normal
Configured Capacity: 528310767616 (492.03 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 3665756160 (3.41 GB)
DFS Remaining: 503046819840 (468.50 GB)
DFS Used\%: 0.00\%
DFS Remaining\%: 95.22\%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used\%: 100.00\%
Cache Remaining\%: 0.00\%
Xceivers: 2
Last contact: Sun May 13 19:38:16 UTC 2018
```

Výstup príkazu hdfs dfsadmin -printTopology

```
Rack: /default-rack
   10.132.0.2:9866 (cluster-6ff6-w-1.c.vernal-mantra-199709.internal)
   10.132.0.3:9866 (cluster-6ff6-w-2.c.vernal-mantra-199709.internal)
   10.132.0.4:9866 (cluster-6ff6-w-0.c.vernal-mantra-199709.internal)
```

Výstup príkazu yarn node -list

```
Total Nodes:3
Node-Id Node-State Node-Http-Address Number-of-Running-Containers
cluster-6ff6-w-0.c.vernal-mantra-199709.internal:57325
RUNNING cluster-6ff6-w-0.c.vernal-mantra-199709.internal:8042 0
cluster-6ff6-w-1.c.vernal-mantra-199709.internal:54772
RUNNING cluster-6ff6-w-1.c.vernal-mantra-199709.internal:8042 0
cluster-6ff6-w-2.c.vernal-mantra-199709.internal:59881
RUNNING cluster-6ff6-w-2.c.vernal-mantra-199709.internal:8042 0
```

Výstup príkazu yarn top

```
NodeManager(s): 3 total, 3 active, 0 unhealthy,
0 decommissioned, 0 lost, 0 rebooted
Queue(s) Applications: 0 running, 0 submitted,
0 pending, 20 completed, 0 killed, 3 failed
Queue(s) Mem(GB): 18 available,
0 allocated, 0 pending, 0 reserved
Queue(s) VCores: 6 available, 0 allocated,
0 pending, 0 reserved
```

Výstup príkazu hadoop fs -ls

```
drwx------   - tuchielt7544 tuchielt7544 0 2018-05-14 00:00 .Trash
drwx------   - tuchielt7544 tuchielt7544 0 2018-05-13 12:58 .staging
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-13 12:38 input
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-08 14:50 input1
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-13 12:45 output
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-13 12:49 output1
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-13 12:57 output2
drwxr-xr-x   - tuchielt7544 tuchielt7544 0 2018-05-08 18:19 test
-rw-r--r--   3 tuchielt7544 tuchielt7544  373 2018-05-13 12:36 test_cache
```

Príloha B

Zoznam parametrov skriptu na generovanie dát

- -h,–help - výpis nápovedy.
- -o, –output= - špecifikuje názvu výstpného súboru
- -n, –number= - špecifikuje počet všetkých unikatnych položiek
- -c, –count= - špecifikuje počet záznamov, ktoré má generátor vytvoriť
- -g, –generate - upraví chovanie skriptu tak, že dáta len vygeneruje
- -l, –locations= - špecifikuje počet lokalít
- -p, –period= - špecifikuje počet dní, pre ktoré májú byť dáta vygenerované
- -m, –max= - špecifikuje maximálny počet položiek v košíku
- -f, –func= - špecifikuje, pre ktorú aplikáciu majú byť spočítané výsledné agregácie

Príloha C

Ukážka vygenerovaného vstupného súboru

```
F,59,1525947595,location19,item1665,item1987,item1171,item1121,
item1628,item939,item653,item627,item1332,item1079,item376,item1812,
item1884,item127
F,77,1524952189,location2,item1283,item104,item2346,item547,
item1396,item202
F,52,1523349329,location23,item2433,item2083,item484,item390,item2087,
item136,item1353,item748,item12,item44,item2096,item1937,item1505,item851,
item636,item350,item1596
F,20,1525194407,location16,item2113,item1346,item1955,item2436,item1990,
item905,item2419,item2242,item1584,item1713,item210,item1315,item1077,
item1001,item2043,item1951
M,44,1523439003,location25,item326,item582,item1287,item2344,item585,
item331,item974,item463,item2398,item218,item2237,item286
M,19,1524475607,location34,item402,item55,item892,item642,item2423
```