VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

SPRACOVÁNÍ BIG DATA A APLIKACE NA OLAP

BIG DATA PROCESSING APPLIED ON OLAP

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — MATÚŠ BÚTORA

AUTHOR

VEDOUCÍ PRÁCE — Prof. Ing. TOMÁŠ HRUŠKA, CSc.

SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19109/2016/xbutor01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Bútora Matúš**
Obor: Informační technologie
Téma: **Big Data**
Big Data
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s agregačními operacemi OLAP pro podporu rozhodování.
2. Seznamte se s technologií zpracování Big Data Hadoop, případně SGE (Sun Grid Engine) ve výpočetním klastru fakulty http://www.fit.vutbr.cz/CVT/cluster/.
3. Navrhněte realizaci některých operací OLAP pro podporu rozhodování technologií distribuce velkých dat ve výpočetním klastru.
4. Navržený systém realizujte jednou z metod z bodu 2.
5. Na vhodném vzorku dat otestujte navržený systém a proveďte zhodnocení výkonnosti.

Literatura:

- White, T.: Hadoop - The Definitive Guide

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% 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 2016
Datum odevzdání: 17. května 2017

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom bakalárskej práca je popísať problematiku Big Data a agregačné operácie OLAP pre podporu rozhodovania, ktoré sú na ne aplikované pomocou technológie Apache Hadoop. Prevažná časť práce je venovaná popisu práve tejto technológie. Posledná kapitola sa zaoberá spôsobom aplikovania agregačných operácií a problematikou ich realizácie. Nasleduje celkové zhodnotenie práce a možnosti využitia výsledného systému do budúcna.

Abstract

The aim of the bachelor thesis is to describe the Big Data issue and the OLAP aggregate operations. These operations are applied using Apache Hadoop technology. Most of the work is focused on the description of this technology. The last chapter contains application of aggregate operations and their implementation, following the conclusion of the work and the possibility for future development.

Kľúčové slová

Big Data, OLAP, Hadoop, agregačné funkcie, MapReduce, HDFS

Keywords

Big Data, OLAP, Hadoop, aggregate functions, MapReduce, HDFS

Citácia

BÚTORA, Matúš. *Spracování Big Data a aplikace na OLAP*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Hruška Tomáš.

Spracování Big Data a aplikace na OLAP

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Prof. Ing. Tomáša Hrušku, CSc. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....

Matúš Bútora

17. mája 2017

Poďakovanie

Chcel by som poďakovať pánu Prof. Ing. Tomášovi Hruškovi, CSc. za vedenie, ochotu a odbornú pomoc na tejto bakalárskej práci.

Obsah

Kapitola 1

Úvod

Posledné roky je termín *Big Data* spomínaný veľmi často. Obrovský nárast objemu dát, ktorý je tak veľký, že 90 % dát sveta bolo vytvorených za posledné dva roky. Už v minulosti spoločnosti analyzovali dáta, ktoré mali k dispozícii, avšak problém veľkých dát je problémom posledných rokov. Cena hardvérov klesala a prichádzali nové možnosti ako uložiť veľké objemy dát. Spoločnosti pochopili, aké dôležité môžu byť dáta, ktorými disponujú, pre každodenný biznis a začali hľadať spôsoby ako s nimi naložiť. S príchodom nových technológii sa analýza veľkých dát stala jednoduchšia a čoraz viac firiem jej začalo pripisovať dôraz. To sú dôvody, prečo sú Big Data dnes tak populárne.

Jednou z technológií na spracovanie veľkých dát je *Hadoop*, ktorým sa zaoberá najväčšia časť bakalárskej práce. Ďalej je práca zameraná na všeobecný popis Big Data, ich analýzu a technológiu *Online Analytical Processing*. Systém Hadoop bol následné nainštalovaný a otestovaný. Hadoop sa skladá z distribuovaného súborového systému *Hadoop Distributed FIle System (HDFS)* a zo systému na paralelné spracovanie *MapReduce*. Používateľ nepotrebuje implementovať paralelizmus, pretože MapReduce ponúka veľké množstvo knižníc a frameworkov na ich implementáciu. V nasledujúcom texte sú popísané dva z nich, a z toho jeden bol využitý na implementáciu agregačných operácii vo výslednom systéme.

Kapitola 2

Big Data

Čo sú to Big Data? Aké veľké musia byť dáta, aby sa dali považovať za Big Data? Najčastejšie sú definované ako kolekcia dát tak objemných a komplexných, že ich nemožno spracovať a analyzovať aplikovaním tradičných spôsobov a využitím klasických databázových technológií. Big Data sú tvorené v rôznych formátoch, od vysoko štruktúrovaných, semi-štruktúrovaných až po úplne neštruktúrované. Najväčšiu časť tvoria posledné dve kategórie, ktorých analýza si vyžaduje nové riešenia. Každoročne máme prístup k novým typom dát, ktorých objem rapidne narastá. V roku 2013 bolo vyprodukovaných celkovo 4,4 zettbytov (10^{21} bytov) informácií a odhadovaný počet do roku 2020 je až 44 zettabytov. Ich zdrojmi sú online transakcie, sociálne siete, mobilné zariadenia a senzory vytvorené rôznymi spôsobmi, napríklad:

- New Yorkská burza vygeneruje denne 4-5 terabytov dát.
- Na Facebooku je nahraných viac než 240 miliárd fotografií, ktorých objem každý mesiac vzrastá o 7 petabytov.
- Urýchľovač častíc vo Švajčiarsku produkuje každoročne okolo 30 petabytov.

Platformy pre spracovanie, organizáciu a analýzu veľkých dát musia riešiť problémy ako sa vysporiadať s veľkým objemom dát obsahujúcim rôzne dátové typy, pričom je často potrebná ich analýza v reálnom čase [20] [29].

2.0.1 Charakteristika

Big Data sú typicky charakterizované ako 3 V:

- *Volume*, čiže množstvo dát.
- *Velocity*, rýchlosť produkovania dát.
- *Variety*, dáta pochádzajú z rôznych zdrojov v rôznych formátoch [20].

2.1 Štruktúra

Najdôležitejšia vlasnosť Big Data je ich zloženie. Ako vlastne Big Data vyzerajú a aké dáta do tejto skupiny patria? Odpoveď na túto otázku je zodpovedaná v nasledujúcej časti. Vo všeobecnosti sú to dáta, ktoré nemožno uložiť na jediné úložisko dát a na ich spracovanie nestačí jeden stroj. Dôvodov je hneď niekoľko – ich objem je príliš veľký a ich nárast príliš rýchly. Najpodstatnejšou vlastnosťou je ich štruktúra. Delia sa do dvoch hlavných kategórii, a to na dáta štruktúrované a dáta neštruktúrované. Neštruktúrované dáta sú ďalej rozdelené na semi-štruktúrované, kvázi štruktúrované a úplne neštruktúrované.

2.1.1 Štruktúrované dáta

Pojem štruktúrované dáta označuje dáta, ktoré dodržujú vysoký level organizácie. Sú závislé na vytvorenom dátovom modele. Model definuje ako sú dáta medzi sebou prepojené, ako sú spracovávané, ako sú uložené a ako sa k nim bude pristupovať [11]. Databázy a tabuľkové procesory (*spreadsheets*) sú typickým príkladom štruktúrovaných dát, sú organizované do stĺpcov a riadkov. Hlavnou výhodou štruktúrovaných dát je oddelenie kontextu od formátu, vďaka čomu sa ľahko vytvárajú databázové dotazy a dáta sú jednoduché na pochopenie. Zdroje štruktúrovaných dát zahŕňajú databázy, dátové sklady, tabuľkové procesory, emaily, metadáta a CRM systémy (systémy pre riadenie vzťahov so zákazníkom).

2.1.2 Semi-štrukturované dáta

Semi-štruktúrované dáta patria do kategórie dát medzi štruktúrovanými a neštruktúrovanými dátami. Reprezentujú dáta, ktoré nedodržujú striktnú schému z dôvodu častých zmien v ich štruktúre. Dáta obsahujú podobné entity spojené do skupín, ale entity obsahujúce rovnakú triedu sa môžu líšiť v atribútoch. Nie všetky atribúty sú povinné a typ

alebo veľkosť atribútov v rovnakej skupine sa môže líšiť. Do tejto kategórie patria dáta v textovom formáte s rozpoznateľnou schémou ako XML a JSON.

2.1.3 Kvázi štruktúrované dáta

Zahŕňajú dáta, ktoré nie sú presne štruktúrované. Kvázi štruktúrované dáta pozostávajú z textových dát s nepravidelným formátom, ktoré môžu byť sformátované využitím špeciálnych technológií. Príkladom sú webové clickstream dáta.

2.1.4 Neštrukturované dáta

Poslednou a najväčšou kategóriou sú dáta neštruktúrované, patrí sem 80 až 90 % dát vytvorených po celom svete. Neštruktúrované dáta nie sú identifikované žiadnou štruktúrou a tiež bez definície dátových typov. Rozdeľujú sa do dvoch základných kategórií:

- Textové objekty ako sú dokumenty vytvorené v Microsoft word, Excelovské dokumenty alebo telá emailov.
- Objekty médií ako sú obrázky, videá a audio súbory [20].

2.2 Analýza

V dnešnej dobe sú spoločnosti zaplavené množstvom dát. Možnosť ich analýzy s novými technológiami ako NoSQL databáze, Hadoop a MapReduce prináša nové príležitosti ako s nimi naložiť. Umožnenie podpory rozhodovania, predpovedanie rizík a vyhľadávania vzťahov medzi dátami zlepšuje servis pre zákazníkov a znižuje výdavky firiem. Analýza dát so sebou však prináša aj výzvy, treba vedieť čo v daných dátach hľadať a ako s objavenou hodnotou naložiť. Nižšie sú popísané tri základné analýzy veľkého objemu dát, ktoré sú v dnešnej dobe využívané dátovými analytikmi.

2.2.1 Deskriptívna analýza

Deskriptívna analýza využíva agregáciu dát a data mining, popisuje minulosť a odpovedá na otázku *„Čo sa stalo?“* Nezodpovedá ju však presne, vytvára iba približnú perspektívu na veľký objem dát. Vo všeobecnosti opisuje zdrojové dáta (sumarizuje) do podoby, z ktorej je človek schopný vyvinúť stratégie ďalšej analýzy. Umožňuje nám pochopiť správanie

z minulosti a na jeho základe zistiť, ako by mohlo ovplyvniť budúcnosť. Nie je vhodná pri detailnom skúmaní udalostí a pri popisovaní veľkého množstva dát s jediným indikátorom. Ponúka však užitočnú sumarizáciu dát potrebnú pri porovnávaní s iným systémom. Bežne sa používa pri zisťovaní vzťahu medzi zákazníkom a produktom.

2.2.2 Prediktívna analýza

Prediktívna analýza sa snaží zodpovedať otázku *„Čo by sa mohlo stať?“* Využíva sa na predpovedanie diania v budúcnosti na základe vyhľadávania vzťahov medzi dátami. Najvýhodnejšie uplatnenie má vtedy, keď je prístupný veľký počet dát zaznamenávaných postupom času. Čím viac dát je k dispozícii, tým presnejšiu analýzu dokážeme vykonať. Nikdy však nie je zaručená 100% úspešnosť.

2.2.3 Preskriptívna analýza

Základom preskriptívnej analýzy je zistiť ako reagovať na možné udalosti. Odpovedá na otázku *„Čo by sme mali robiť?“* Využíva poznatky z prediktívnej analýzy a aplikuje množinu riešení a odporúčaní za účelom objavenia najvhodnejších možností reakcie na dané problémy. Je to najnovšia z uvedených analýz a využitie spoločnosťami je zatiaľ veľmi malé [10] [25] [9].

Kapitola 3

OLAP

Online analytical processing, skrátene OLAP, je technológia využívajúca sa v multidimenzionálnych dátových modeloch. Tieto modely sú kľúčovým faktorom pri analýze veľkého množstva dát v systémoch pre podporu rozhodovania. Dáta sú zobrazované ako multidimenzionálne kocky, ktoré sú vhodné pre ich analýzu. OLAP systémy umožňujú rýchle zodpovedanie dotazov a prehľadne poskytujú agregačné operácie (väčšinou súčtové) nad pôvodnými dátami. Sú optimalizované skôr na vytváranie dotazov a zostavovanie údajov než na spracovávanie transakcií. Základom celého systému je OLAP kocka popísaná v podkapitole 3.2. Multidimenzionálne dátové modely sa okrem OLAP systému skladajú z ďalších dvoch častí, *dátových skladov* a *data miningu*. Dátové sklady sú veľké úložiská dát, ktoré integrujú dáta z rôznych zdrojov. Data mining je proces objavovania nových informácií z dát a vzťahov medzi nimi pri dátovej analýze [18] [5] [27].

3.1 Coddové pravidlá

V roku 1985 Edgar F. Codd spísal zoznam pravidiel pre SRBD (systém riadenia bázy dát). Zoznam dvanástich pravidiel definuje OLAP a umožňuje usporiadanie a analýzu dát v multidimenzionálnom priestore. Pravidlá sú nasledovné:

Multidimenzionálny konceptuálny model

OLAP by mal poskytovať užívateľovi multidimenzionálny model tak, aby zodpovedal jeho potrebám a aby tento model mohol využívať na analýzu zhromaždených údajov.

Transparentnosť

Technológie systému OLAP, architektúra výpočtu a databáza budú transparentné za účelom plného využitia produktivity a odbornosti užívateľa, pričom vstupné dáta sú heterogénne a ich homogenitu zaručíme pomocou ETL (Extract transformation loading).

Dostupnosť

Systém by mal pristupovať iba k údajom, ktoré sú potrebné pre analýzu. Ku všetkým takýmto údajom by mal systém pristupovať nezávisle na tom, z ktorého heterogénneho zdroja pochádzajú a ako často sú obnovované.

Stabilná výkonnosť

Pri narastajúcej veľkosti databázy by nemalo nastať zníženie výkonu. Užívateľ by zníženie výkonu nemal pocítiť.

Architektúra klient-server

Systém musí fungovať na základe architektúry klient-server. Server by mal umožniť pripojenie viacerým klientom, a to s využitím minimálneho úsilia.

Generická dimenzionalita

Každá dimenzia údajov musí byť ekvivalentná ako v štruktúre, tak aj v operačných schopnostiach.

Dynamické ošetrenie riedkych matíc

Systém OLAP musí byť schopný prispôsobiť svoju fyzickú schému konkrétnemu analytickému modelu, ktorý optimálne ošetrí riedke matice a súčasne udrží požadovanú úroveň.

Podpora pre viac užívateľov

Systém OLAP musí byť schopný podporovať viac užívateľov alebo skupiny užívateľov pracujúcich súčasne na konkrétnom modele.

Neobmedzené krížové dimenzionálne operácie

Systém musí rozoznať dimenzionálne hierarchie a automaticky vykonávať výpočty v rámci dimenzie a medzi dimenziami.

Intuitívne manipulácie s údajmi

Možnosť preorientovania sa na detailnú úroveň a späť. Užívateľské rozhranie musí umožňovať všetky manipulácie s dátami. *Napr.: drill-down* a *roll-up*.

Flexibilné vykazovanie

Schopnosť usporiadať riadky, stĺpce a bunky spôsobom, ktorý umožní analýzu a intuitívnu prezentáciu analytických zostáv.

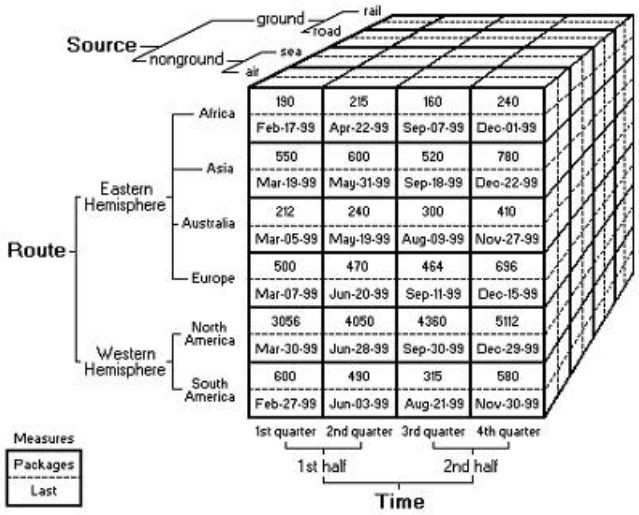
Neobmedzené dimenzie a úrovne agregácie

V závislosti na požiadavkach rozhodovania môže mať analytický model viac dimenzii, pričom každá z nich môže obsahovať viacúrovňovú hierarchiu. Analytický model by nemal byť umelo obmedzený počtom dimenzií a úrovní hierarchie [16] [1].

3.2 OLAP kocka

OLAP kocka, zvaná aj multidimenzionálna tabuľka, je dátová štruktúra prekonávajúca limity relačných databáz tým, že ponúka rýchlu analýzu dát. Táto dátová štruktúra agreguje *miery* podľa úrovní a *hierarchií* každej z *dimenzií*, ktorú chceme analyzovať. Kocka umožňuje zobrazovať veľké objemy dát a ponúka užívateľom flexibilný pohľad na dáta z rôznych perspektív [8]. OLAP kocky sú využívané biznis analytikmi. Ich dizajn využíva biznis logiku a sú optimalizované pre analytické účely [24]. Kocka pozostáva z niekoľkých kľúčových elementov: **Miera** predstavuje konkrétne číselné ohodnotenie daného aspektu [13]. Miery sú hlavné hodnoty, ktoré sa spracovávajú, agregujú a analyzujú. Zvyčajne sú to numerické hodnoty ako zisky, výnosy alebo náklady. **Dimenzia** je množina jednej alebo viacerých hierarchií úrovní kocky, ktorým používateľ rozumie a používa ich ako základ pre analýzu údajov. V prípade na obrázku 3.1 pozostáva kocka z troch dimenzií: produkt, lokalita a čas. **Hierarchia** je logická stromová štruktúra, slúžiaca na organizáciu členov dimenzie. Každý člen dimenzie má jeden nadradený a nula alebo viacero podradených členov. Podradený člen

na nasledujúcej nižšej úrovni v hierarchii priamo súvisí s nadradeným členom. Napríklad v hierarchii čas je to rok, štvrťrok, mesiac a deň. **Člen** je element v hierarchii. Napríklad v štandardnej časovej hierarchii bude člen 1. január 2009. Členmi sú taktiež január 2009 alebo rok 2009. Posledné dva ale budú agregáciou dní k ním patriacim [5] [7].



Obr. 3.1: OLAP dátová kocka [2].

3.2.1 Operácie nad OLAP kockou

OLAP poskytuje *user-friendly* prostredie pre interaktívnu analýzu dát. Existuje niekoľko operácií nad dátovou kockou, ktoré umožňujú vidieť dáta z rôznych pohľadov. Príklady uvedené v nasledujúcich operáciach sa vzťahujú na obrázok 3.1.

Roll-up

Roll-up vykonáva posun o jednu alebo viacero hierarchii v dimenzií nahor. Taktiež môže vykonať posun nadol, čo zapríčiní redukciu dimenzie. Využíva sa na generovanie štatistík na základe agregovanejších dát. Ak by sme zvýšili úroveň miery dimenzie *lokalita*, ako výsledok by sme dostali napríklad oblasť Európy. Na druhej strane odstránením dimenzie *čas* by výsledná analýza ukazovala štatistiku predaja produktov za celkové časové obdobie.

Drill-down

Drill-down operácia, ktorá vykonáva presný opak roll-up. Umožňuje nám detailnejší pohľad na dáta. Prevedením drill-down je konkrétna položka dimenzie znížená o jednu alebo viacero

hierarchií. Taktiež k nej môže byť pridaná dodatočná dimenzia. V prípade, kde berieme *lokalitu* štát sa miera dimenzie posunie na nižšiu úroveň, napríklad mesto. Pri pridaní dodatočnej dimenzie *zákazník* by bolo zoskupenie dát odstránené. Výsledok by tvorili údaje o predaji podľa skupín zákazníkov.

Slicing and dicing

Slice vytvára selekciu jednej dimenzie danej kocky, čoho výsledkom je podkocka. Pri selekcii dvoch a viacerých dimenzií hovoríme o operácii *dice*. Napríklad vybraním roku 2010 z dimenzie *čas* dostávame výsledok predaja v danom roku.

Pivoting

Pivot rotuje jednotlivými dimenziami, a tým mení orientáciu dátovej kocky a umožňuje na ňu pohľad z rôznych perspektív. Pivot združuje dáta s rôznymi dimenziami [4] [13].

3.3 Agregačné funkcie

Funkcionalita a analýza OLAP je založená na agregačných funkciách. Za účelom dosiahnutia správnych výsledkov musia byť dané funkcie presne stanovené. Agregácia musí brať do úvahy niektoré závislosti, ako napríklad stochastickú závislosť medzi dimenziami; nesmie porušovať pravidlo asociativity a komutativity. Agregácia znamená pripojenie, pričlenenie, prípadne zhrnutie či zhlukovanie, *tzn.* agregačné funkcie zhlukujú niekoľko množín údajov do jedinej hodnoty pomocou určitého výpočtu. Môžme ju vnímať ako mapovanie dát na dáta definované jednoduchou schémou za účelom vytvorenia jednoduchšieho pohľadu na originálne dáta. Agregačné funkcie delíme do troch nasledovných skupín.

3.3.1 Distributívne

Distributívne alebo induktívne agregačné funkcie sú také funkcie, ktoré sa dajú vypočítať distributívnym spôsobom. Predpokladajme rozdelenie dát do n množín. Aplikovaním funkcie na každú jednu množinu sa vo výsledku vytvorí n agregátov (výsledok agregácie). Ak je výsledok odvodený aplikovaním funkcie na n množín rovnaký ako výsledok pri aplikovaní funkcie na celý dátový set (bez rozdelenia na množiny), hovoríme o funkcii distributívnej. Medzi distributívne agregačné funkcie zaraďujeme: ***počet***, ***súčet***, ***minimum*** a ***maximum***.

Miera je distributívna, ak ju môže me dosiahnuť aplikovaním distributívnej agregačnej funkcie. Výpočet distributívnej miery je efektívny preto, lebo sa dá využiť distributívny spôsob výpočtu.

3.3.2 Algebraické

Agregačná funkcia je algebraická vtedy, ak môže byť spočítaná algebraickou funkciou s M argumentami (kde M je z konečného intervalu kladných čísel). Výpočet každej algebraickej funkcie je dosiahnutý aplikovaním distributívnej agregačnej funkcie. *Napr.* ***priemer*** môže byť vypočítaný pomocou distributívnych agregačných funkcií *súčet/počet*. Podobne `min_N()` a `max_N()`, ktoré vypočítajú N minimálnych alebo maximálnych hodnôt v danej dátovej množine. Taktiež ***smerodatná odchýlka*** je algebraická agregačná funkcia. Algebraickú mieru dostávame po aplikovaní algebraickej agregačnej funkcie.

3.3.3 Holistické

Funkcia je holistická, ak neexistuje konštanta obmedzujúca veľkosť subagregátu. *Tzn.* neexistuje algebraická funkcia s M argumentami (kde M je konštanta), ktorá by charakterizovala výpočet. Bežne používané holistické agregačné funkcie sú ***medián*** a ***modus***. Miera je holistická vtedy, ak na ňu aplikujeme holistickú agregačnú funkciu.

Väčšina aplikácií, ktoré využívajú dátovú kocku zloženú z veľkej sady dát, potrebuje efektívny výpočet distributívnej a algebraickej miery. Existuje veľa efektívnych techník, avšak vypočítať holistickú mieru je veľmi náročné. Využívajú sa techniky, ktoré sa približujú efektívnemu výpočtu holistickej miery, ako napríklad výpočet približného mediánu. V mnoho prípadoch je tento výpočet dostačujúci [19] [15] [16].

Kapitola 4

Hadoop

Hadoop je softvérový open source framework podporovaný Apache Foundation, ktorý umožňuje distribuované spracovanie veľkých množstiev dát použitím jednoduchých programovacích modelov. Zároveň zabezpečuje vysokú dostupnosť (*high availability*) dát, pričom dostupnosť nie je zabezpečená na hardvérovej, ale softvérovej úrovni [26]. Keď sa hovorí o veľkom množstve dát, myslia sa desiatky až stovky petabytov a exabytov. Hadoop je škálovateľný z jedného stroja na tisíce. Ponúka spôsob ako paralelne spracovať a spúšťať programy naprieč clustrom. Systém je navrhnutý tak, aby fungoval na *tzv.* „commodity hardware“, to znamená hardvér cenovo prístupný a ľahko dostupný. Systém sme schopní nainštalovať aj na bežnom osobnom počítači. Hadoop v súčasnej dobe využívajú technologické giganty ako Facebook, Yahoo! či Twitter. Hadoop vychádza zo systémov navrhnutých firmou Google, ktorými sú *Google File System (GFS)* a *MapReduce*. Skladá sa zo štyroch hlavných častí:

- **Hadoop Common** – obsahuje knižnice a služby potrebné k prevádzke iných modulov v rámci Hadoop
- **Hadoop Distributed File System** – distribuovaný súborový systém, ktorý zabezpečuje rozloženie súborov v clustroch serverov alebo počítačov. Uvedenému sa budeme podrobnejšie venovať v nasledujúcom texte tejto práce 4.1.
- **Hadoop Yarn** – riadiaca platforma, ktorá koordinuje jednotlivé zdroje, úlohy (*jobs*) a rozvrhuje užívateľské aplikácie. Podrobnejšie popísané v podkapitole 4.2.2

- **Hadoop MapReduce** – programovací model pre spracovanie veľkých objemov dát. MapReduce framework je podrobnejšie popísaný v podkapitole 4.2 [26] [29].

4.1 Hadoop Distributed Filesystem

Ako uložiť dátovú sadu, ktorej veľkosť je niekoľkonásobne väčšia ako kapacita jedného fyzického stroja? V takomto prípade je nutné rozdeliť sadu medzi viacero strojov a využiť súborový systém, ktorý zabezpečí ukladanie dát po sieti. Distribuovaný súborový systém je oproti klasickým diskom oveľa komplexnejší, a preto jednou z podmienok je zabezpečiť odolnosť voči chybám [29].

4.1.1 Dizajn HDFS

Hadoop File System (HDFS) je súborový systém určený k ukladaniu veľmi veľkých dátových sád. Nie je potrebný žiadny špeciálny počítač. HDFS sme schopní používať na bežných počítačoch.

Streamovaný prístup k dátam

Myšlienka HDFS je postavená na paralelnom streamovanom prístupe k dátam. Využíva sa prístup *write-once, read-many-times*, čiže dáta zapísané nemôžu byť ďalej modifikované a nedochádza tak k nežiadúcim zmenám. Komunikácia je založená na TCP/IP protokole. Dôraz je kladený na rýchly prechod celým dátovým setom. Na rozdiel od náhodného prehľadávania je optimalizované sekvenčné načítanie celého súboru [29] [17].

Veľké dátové sady

Súbory využívajúce HDFS majú typicky veľkosť stovky gigabytov až terabytov. Dnešné Hadoop clustre dokážu uložiť petabyty dát a poskytujú škálovateľnosť na stovky uzlov v jednom clustri. Taktiež je zabezpečená podpora desiatok miliónov súborov v jednej inštancii [29].

Prenositeľnosť

Ďalšou podmienkou je zaručiť jednoduchý prenos súborového systému z jednej platformy na inú. HDFS bol navrhnutý tak, aby nebol potrebný žiadny špeciálny hardvér. Softwarové

komponenty sú napísané v Jave, čo zaručuje bezproblémovú prenositeľnosť medzi platformami [28].

Detekcia chýb

HDFS je distribuovaný systém skladajúci sa z tisícok uzlov zložených z jednoduchých hardvérových komponentov. Dáta sú rozdeľované, replikované a distribuované cez rôzne uzly v clustri. Faktom je, že pri takomto veľkom počte strojov je zlyhanie skôr norma ako očakávanie. V clustri obsahujúcom 1000 uzlov priemerne zlyhá 2 až 10 uzlov za deň. Detekcia chýb a zotavenie je jeden z hlavných cieľov HDFS [28].

Nutné je spomenúť aj vlastnosti aplikácií, pre ktoré HDFS nie je optimálnym riešením.

Množstvo malých súborov

NameNode si v pamäti uchováva metadáta súborového systému. Limit počtu súborov, ktoré je schopný uschovať, závisí od veľkosti pamäte daného NameNode. Spravidla každý súbor, adresár a blok zaberá 150 bytov. Hoci uloženie miliónov súborov je ešte prijateľné, miliardy ďaleko prekonávajú kapacitu hardvéru [29].

Rýchla doba prístupu k dátam

HDFS je optimalizovaný na rýchly prechod celým dátovým setom. HDFS nie je vhodné riešenie pre aplikáciu, ktorá vyžaduje prístup k dátam pod desiatky milisekúnd. V takomto prípade je možnosť využiť *Hbase*, ktorý ponúka náhodný prístup k dátam v reálnom čase [12] [29].

4.1.2 Bloky

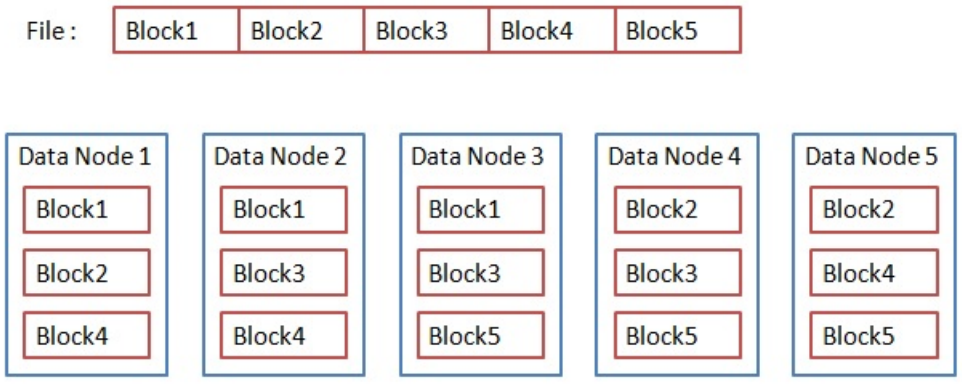
Dizajn HDFS spočíva v spoľahlivom ukladaní veľmi veľkých súborov medzi stroje v clustri. Každý súbor je uložený ako sekvencia blokov uložených nezávisle od seba. Veľkosť jedného bloku je nastaviteľná, štandardne je 128 MB. Na rozdiel od súborového systému pre jeden disk, súbor v HDFS, ktorého veľkosť je menšia ako veľkosť bloku, nezaberá jeho celú kapacitu. Napríklad súbor s veľkosťou 1 MB uložený do bloku s veľkosťou 128 MB, zaberá 1 MB miesta na disku, nie celých 128 MB. Každý blok je replikovaný naprieč clustrom so štandardnou replikačnou hodnotou 3 tak ako na obrázku 4.1. Bloky v HDFS sú veľké oproti

diskovým blokom z dôvodu minimalizácie hľadania v súbore. Ak je blok dostatočne veľký, čas pre prenos dát z disku je neporovnateľne dlhší ako čas vyhľadávania začiatku bloku. Napríklad ak je čas vyhľadávania okolo 10 ms a prenosová rýchlosť 100 MB/s, tak doba vyhľadávania je 1 % doby prenosu, a tým pádom potrebujeme blok o veľkosti 100 MB. Rozdelenie do blokov má svoje opodstatnenie a výhody [29] [12] [28].

- Súbor môže byť väčší ako veľkosť jedného disku v sieti. Bloky, do ktorých je súbor rozdelený, nemusia byť uložené na rovnakom disku. HDFS má možnosť vybrať akýkoľvek disk v clustri. [28]

- Bloková abstrakcia zjednodušuje ukladanie veľkých súborov. Jednoduchosť je podstatná v distribuovaných systémoch z dôvodu rýchleho vyhľadávania chýb. Veľkosť blokov je jednotná, čo uľahčuje výpočet voľnej pamäte na danom disku. Metadáta súboru sa do blokov neukladajú, spravované sú separátne iným systémom [29].

- Bloky sú vhodné pre replikačný proces, ktorý zabezpečuje prevenciu chybovosti a dostupnosť. Prevencia proti zlyhaniu stroja, chybovosti bloku či disku je zabezpečená replikáciou. Každý blok je replikovaný spravidla trikrát na fyzicky oddelené stroje. Ak nastala chyba a blok je nedostupný, použije sa kópia tohto bloku. Nedostupný blok je vymenený za jeho kópiu. Tá je presunutá na pozíciu pôvodne nedostupného bloku a dochádza k replikácií aby počet replikovaných blokov nadobudol opäť pôvodnú hodnotu [29] [28].



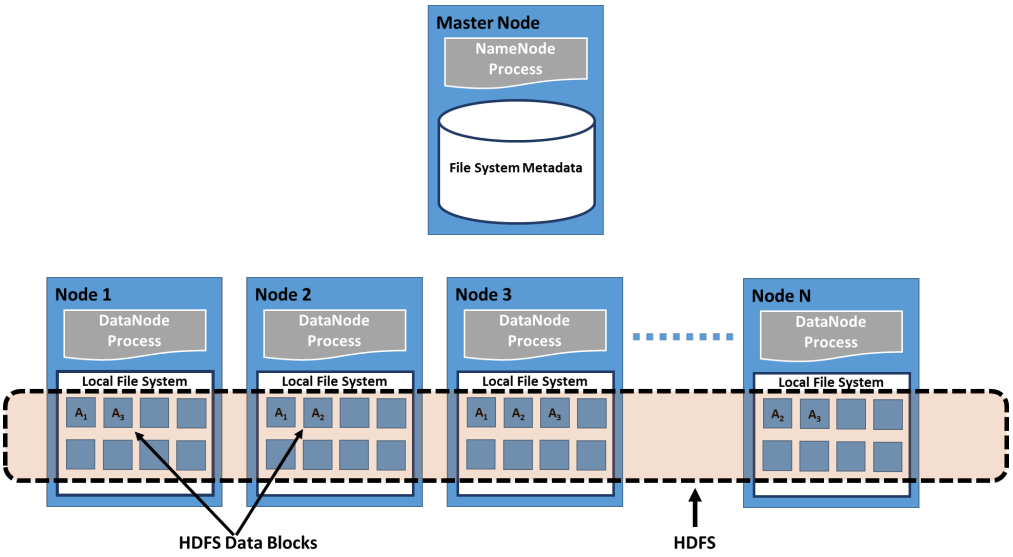
Obr. 4.1: Súbor rozdelený do blokov s replikačnou hodnotou 3 [3].

4.1.3 NameNode a DataNode

HDFS je založený na architektúre *master/slave*. Cluster sa skladá z jedného master serveru, ktorým je NameNode. Jeho úlohou je riadiť menný priestor (*namespace*) súborového systému. Jeden NameNode riadi vždy jeden menný priestor. NameNode uchováva metadáta dát uložených v HDFS. Samotné dáta su potom rozdelené do blokov a uložené na jednotlivých dátových uzloch (DataNodes). Menný priestor je oddelený od uložených dát z dôvodu rozdielnej rýchlosti prenosu. Toto rozdelenie dovoľuje, že NameNode sa môže správať ako konkurentný server a klienti využívajú priepustnosť celého clusteru. Rozhranie je klientovi prezentované podobne ako *Portable Operating System Interface (POSIX)*, takže užívateľský kód nemusí poznať funckiu NameNode a DataNode. Metadáta NameNode ukladá do pamäti RAM. Veľkosť menného priestoru teda závisí na veľkosti pamäte RAM NameNode. Metadáta obsahujú súborovú a adresárovú hierarchiu, mapovanie súboru do blokov na dátových uzloch, lokalitu blokov, vlastníka a oprávnenia súborov. Dátové uzly periodicky komunikujú s NameNode, vyzývané sú samotným NameNode alebo klientskou aplikáciou [17] [28] [12] [29].

Bez NameNode by systém nevedel ako opäť rekonštruovať súbor z blokov, do ktorých bol rozdelený. Ak nastane chyba, je potrebné, aby bol NameNode zálohovaný. Hadoop pre tento prípad ponúka dva mechanizmy.

- Prvý spôsob je zálohovať trvalý stav metadát súborového systému. Tie sú buď zálohované lokálne na disk, alebo pripojené na vzdialený sieťový súborový systém (NFS) [29].
- Druhá možnosť je vytvorenie *tzv. Secondary NameNode*. Jeho úlohou je periodicky spájať image menného priestoru a *edit log* za účelom obmedzenia neprimeranej veľkosti edit logu. V tomto logu sú zapísané všetky požiadavky, ktoré NameNode prijme (*create/update/delete*). Obvykle sekundárny NameNode je separátny stroj [29] [28].



Obr. 4.2: Hierarchia uzlov v HDFS [14].

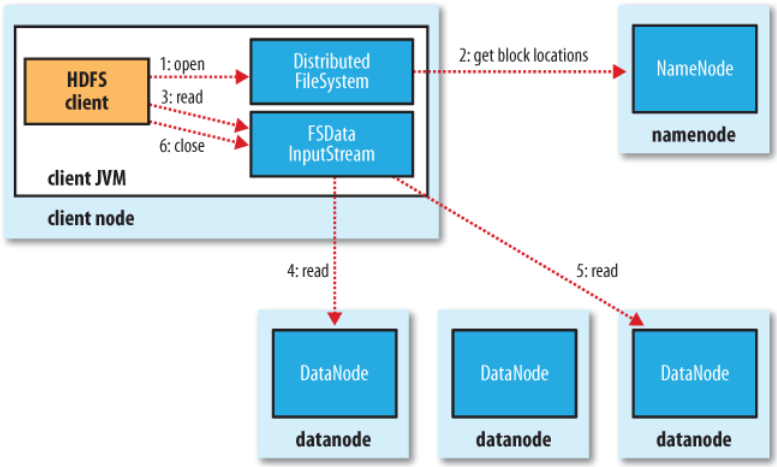
Replikácia dát

Súbory uložené v HDFS sú rozdelené na bloky a bloky sú replikované naprieč clustrom. Štandardný počet replík je tri. Pre každý súbor môžme počet replík nastaviť osobitne, pričom maximálny počet je 512. Minimálny počet replík je 1 [28] [12]. Dôvody, prečo sú bloky replikované, sú nasledovné:

- *Trvalosť dát.* Ak je replika bloku porušená, dáta nie sú stratené. Použijú sa iné dostupné repliky.
- *Tolerancia chýb.* Ak je replika bloku stratená, použije sa iný blok, a potom klient pristupuje ku kópii tohoto bloku. Zachovaný je aj prístup k súboru MapReduce aplikáciou. Tolerancia chýb poskytuje spoľahlivosť prístupu k dátam.
- *Poloha dát.* Označuje vzdialenosť vstupných dát od miesta, kde sú dáta potrebné na spracovanie. Čím väčší počet replík rôzne rozmiestnených po clustri, tým sú dáta bližšie k miestu spracovania.
- *Speculative execution (Špekulatívny výpočet).* Ak je vykonávaná úloha príliš pomalá, spustia sa redundantné kópie úlohy. Použije sa tá, ktorá skončila prvá [28].

4.1.4 Čítanie zo súboru

Ukážka toku dát pri komunikácii klienta s HDFS, NameNode a DataNode pri čítaní zo súboru je zobrazená na obrázku 4.3. Klient zavolá metódu `open()` objektu `FileSystem object`, ktorý je inštanciou `DistributedFileSystem` (krok 1 na obrázku 4.3). `DistributedFileSystem` pomocou vzdialených procedurálnych volaní RPC (Remote procedure calls) zavolá NameNode aby zistil polohu prvých blokov súboru (krok 2). NameNode vráti adresu dátových uzlov, ktoré obsahujú kópiu hľadaných blokov. Adresy sú zoradené podľa vzdialenosti v topológii siete clustru, od najbližšieho vzostupne. `DistributedFileSystem` vráti klientovi objekt `FSDataInputStream`, ktorý mu umožňuje čítať dáta zo súboru a zapúzdruje `DFSIinputStream`. Ten riadi vstupné/výstupné operácie DataNode a NameNode. Klient zavolá `read()` (krok 3). Následne `DFSIinputStream` streamuje dáta z dátových uzlov ku klientovi, ktorý volá `read()` opakovane (krok 4). Po načítaní konca bloku `DFSIinputStream` zatvorí spojenie s dátovým uzlom a vytvorí nové s najvhodnejším uzlom obsahujúcim nasledujúci blok (krok 5). Z pohľadu klienta sa operácie odohrávajú transparentne a sú vnímané ako jeden spojitý stream. Po skončení čítania súboru klient zavolá metódu `close()` objektu `FSDataInputStream` (krok 6). Pri čítaní porušeného bloku `DFSIinputStream` zastaví komunikáciu, použije repliku bloku a porušený blok nahlási NameNode [29] [21].



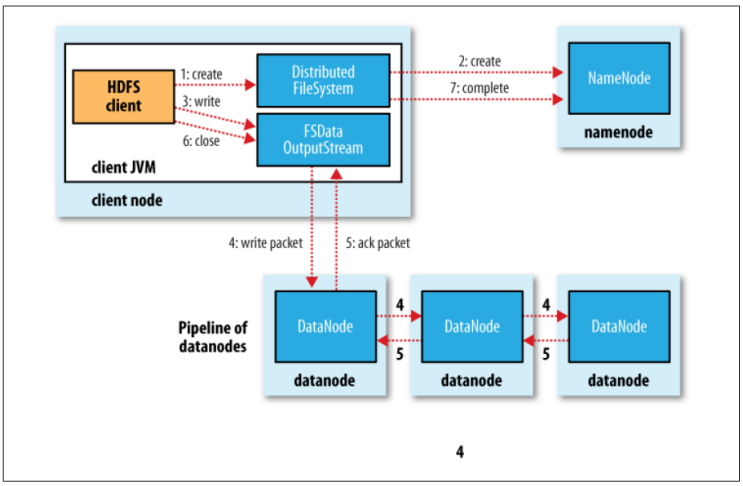
Obr. 4.3: Čítanie súboru v HDFS [29].

4.1.5 Zápis do súboru

Na obrázku 4.4 je zobrazené vytvorenie nového súboru a zápisu doň. Podobne ako pri čítaní zo súboru klient zavolá metódu `create()` (krok 1 na obrázku 4.4). NameNode vytvorí nový súbor v mennom priestore súborového systému ako odpoveď na volanie RPC z `DistributedFileSystem` (krok 2). Následne NameNode kontroluje oprávnenia klienta pre vytvorenie súboru a či daný súbor už v HDFS neexistuje. Ak kontrola prebehne úspešne, `DistributedFileSystem` vráti klientovi `FSDataOutputStream`, do ktorého klient zapisuje dáta. Komunikáciu s uzlami riadi `DFSOutputStream`. Zapisujúce dáta (krok 3) `DFSOutputStream` rozdeľuje na pakety, ktoré sú radené do internej fronty zvanej *data queue*. Fronta je zaslaná na `DataStreamer`, ktorý zasiela dotazy na NameNode o alokácii nových blokov. Ten vytvorí zoznam vhodných dátových uzlov pre uloženie replík. Uzly sú následne zreťazené. Medzi zreťazenými uzlami sa vytvorí *„pipeline“* na prenos dát. (Pipline of datanodes na obrázku 4.4 s replikačnou hodnotou 3). `DataStreamer` zasiela pakety na prvý dátový uzol v reťazci. Prvý uzol pakety uloží a pošle ďalej na druhý uzol. Obdobne sa pokračuje až na koniec reťazca (krok 4). `DataStreamer` vytvára aj vlastnú internú frontu paketov čakajúcich na potvrdenie. Fronta sa nazýva *ack queue*. Paket je z fronty odobraný až po potvrdení od všetkých uzlov v reťazci (krok 5). Po ukončení zápisu sa zavolá `close()` (krok 6). Zostávajúce pakety sú zaslané na dátové uzly a čakajú na potvrdenie. Po potvrdení NameNode obdrží signál, že súbor je kompletný. Ten pozná polohu blokov v súbore vďaka `DataStreamer`. V prípade zlyhania HDFS podnikne nasledujúce operácie:

- Uzavrie sa *pipeline* medzi dátovými uzlami. Všetky pakety v *ack queue* sú zaradené na začiatok *data queue*. Dátové uzly, ktoré sa nachádzajú za porušeným uzlom, takýmto spôsobom neprídu o žiadne pakety.
- Bloku, do ktorého sa aktuálne zapisuje, je pridelená nová identita. NameNode je o zmene identity bloku informovaný. V prípade zotavenia porušeného uzlu bude čiastočne zapísaný blok vymazaný.
- Porušený dátový blok je vyradený z potrubia a zbytok dát v bloku je prepísaný na ostatné nepoškodené uzly.
- NameNode je upozornený, že počet replík je menší a vytvorí novú repliku z iného uzlu. S nasledujúcimi blokmi je zaobchádzané ako pred chybovým stavom [22] [29].

Taktiež môže nastať prípad, že vyskytne chyba na viacerých dátových uzloch. Táto udalosť je však ojedinelá. Pokiaľ uspeje zápis aspoň jedného bloku v uzle, blok je asynchrónne replikovaný v clustri, kým sa opäť nedovŕši pôvodná replikačná hodnota (v našom prípade 3) [22].



Obr. 4.4: Zápis do súboru v HDFS s replikačnou hodnotou 3 [29].

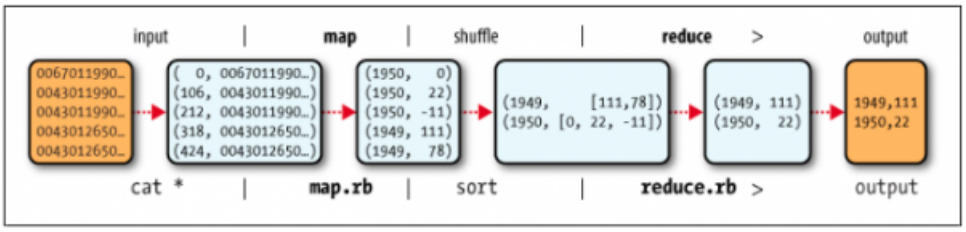
4.2 MapReduce

MapReduce je programovací model na spracovanie dát. Pod Hadoop môžme spúšťať MapReduce programy napísané v rôznych jazykoch *napr.:* Java, Ruby, Python, *atď.* Skladá sa z dvoch hlavných fáz: *Map fáza* a *Reduce fáza*. Dáta sa spracovávajú pomocou key/value párov. Dáta vstupujú do špecifickej mapovacej funkcie. Tá môže napríklad spočítať počet náhodných výskytov slova v súbore. Fáza Map generuje čiastočný množinu key/value párov. Výstup z Map fázy je poslaný na vstup Reduce funkcie. Výstup z Reduce funkcie je výstupom celej MapReduce aplikácie [26] [29].

- Map fáza – vstup rozdelí na menšie čiastkové úlohy. Tie distribuuje do TaskTrackerov. Úlohy môžu byť vykonávané znova, čo vedie k viacúrovňovej stromovej štruktúre. V tejto fáze je vykonávané filtrovanie a triedenie [26].
- Reduce fáza – zhromažďuje odpovede na všetky čiastkové úlohy, ktoré skombinuje a vytvorí výstup celej MapReduce aplikácie. Reduce vykonáva sumarizujúce operácie [26] [29].

4.2.1 Priebeh MapReduce

Na obrázku 4.5 je popísaná logika MapReduce na príklade hľadania najvyššej teploty v uvedených rokoch. Zo vstupu sú Map funkciou vyfiltrované roky a teploty v daných rokoch, ktoré sú jej výstupom. Pred vstupom do Reduce funkcie MapReduce framework tento výstup spracováva. Spracovanie triedi a zlučuje key/value páry podľa kľúča. V našom prípade sú ku každému roku priradené teploty, ktoré v ňom boli namerané. Teraz jediná úloha Reduce funkcie je iterovať cez celý zoznam a vybrať maximum v danom roku [29].



Obr. 4.5: Dátový tok MapReduce [29].

4.2.2 MapReduce Daemons

Existujú dva typy MapReduce frameworkov: *MapReduce 1* (MR1) pre Apache Hadoop 1 zobrazený na obrázku 4.6 a *YARN* (MR2) pre Apache Hadoop 2 zobrazený na obrázku 4.7. MR1 sa skladá z dvoch komponent: *JobTracker* a *TaskTracker* [28].

JobTracker

JobTracker je *master daemon* MapReduce aplikácie. Jeho funkciou je koordinovať bežiace MapReduce úlohy s tým, že klientovi poskytuje identifikačné číslo daných úloh. Ďalej prijíma požiadavky na úlohy od klienta, iniciuje ich a rozdeľuje úlohy na podúlohy medzi TaskTracker. Monitoruje podúlohy a v prípade zlyhania ich znova spúšťa čo najviac krát [29] [28].

Architektúra MR1 pozostáva z jedného JobTrackru a z viacerých TaskTrackrov. V nej má JobTracker nasledovné funkcie:

1. Klient zadá MapReduce úlohu JobTrackru.

2. JobTracker vyhľadá polohu dát.

3. JobTracker hľadá TaskTracker s vhodnými (Map alebo Reduce) voľnými slotmi v clustri čo najbližšie k dátam.

4. JobTracker pridelí úlohy TaskTreckerom, tie následne spúšťajú podúlohu úloh.

5. JobTracker monitoruje TaskTrackre.

6. Ak je to potrebné znova prideľuje úlohy, ktoré zlyhali.

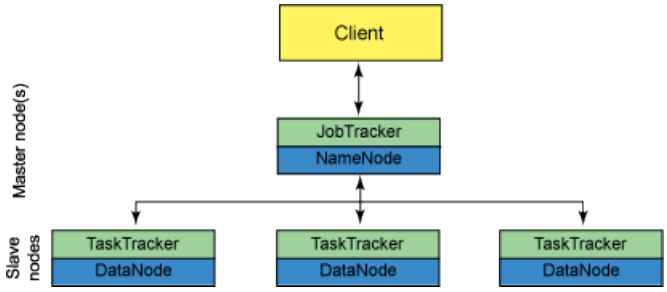
7. Sleduje ukončenie úloh a poskytuje o nich informácie klientovi [28].

V skratke JobTracker robí všetko, spravuje zdroje, rozvrhuje podúlohy a monitoruje ich. Podrobnosti o úlohách sú zobrazené na webovom rozhraní JobTrackeru. Opätovným spúšťaním nefunkčných podúloh umožňuje toleranciu voči chybám. TaskTrackre periodicky komunikujú s JobTrackerom. Ak JobTracker neprijme odpoveď počas určitého intervalu, štandardne 10 minút, TaskTracker je považovaný za stratenú a je znovu spustený. Ak bola úloha v mapovacej fáze, JobTracker znovu spustí všetky map podúlohy. Ak sa úloha nachádzala v reduce fáze, JobTracker znovu spustí iba zlyhané reduce podúlohy. Pre pomaly vykonávané podúlohy rozvrhne paralelné spracovanie zvané *speculative execution* 4.1.3 [28].

TaskTracker

TaskTracker je *slave daemon* MapReduce aplikácie. Cluster môže pozostávať z viacerých TaskTrackerov. Vykonáva Map a Reduce podúlohy úloh zadané JobTrackerom. Skladá sa zo *slotov*, ktoré sa špecifikujú buď na podúlohu Map, alebo Reduce. TaskTracker monitoruje svoje podúlohy a periodickou komunikáciou s JobTrackerom podáva o nich informácie [28].

V prípade MR1 sa JobTracker staral o spravovanie zdrojov aj o plánovanie podúloh. V MR2 boli funkcie JobTrackeru rozdelené medzi viacero daemonov. NodeManager nahradil TaskTracker [29]. YARN (MR2) predstavuje generickejší framework pre vytváranie nielen MapReduce aplikácií [28].



Obr. 4.6: Architektúra MR1 [23].

Nedostatky MR1:

- JobTracker riadi a monitoruje zdroje MapReduce úlohy a aj ich plánovanie. To z neho robí Single point of failure (SPOF), čiže bod zlyhania, ktorý zabraňuje efektívnemu škálovaniu clustru.
- Neúplné využitie zdrojov, sloty sú špecifické len pre Map alebo len pre Reduce podúlohy.
- Aplikácie, ktoré nie sú MapReduce, nemôžu byť spúšťané konkurentne [28] [26].

YARN prekonáva vyššie uvedené nedostatky nasledovne:

- Ľahšia škálovateľnosť clustru tým, že funkcionalita JobTrackeru je rozdelená medzi ResourceManager a ApplicationMaster. S YARN sme schopní spustiť Hadoop na omnoho väčších clustroch.
- Lepšie využitie clustru, kapacita zdrojov je nastavená pre každý uzol tak, že ho môžu využívať Map aj Reduce podúlohy.
- Podporuje konkurentné spúšťanie aplikácií, ktoré nie sú MapReduce.
- Lepšia priepustnosť, využíva sa iný systém plánovania [28].

ResourceManager

ResourceManager je *master MapReduce daemon* Yarn aplikácie. Globálne spravuje alokáciu zdrojov potrebných na výpočet a spúšťa *ApplicationMaster*. Každá aplikácia využíva jeden ApplicationMaster. ResourceManager neiniciuje ani nemonitoruje spúšťanie podúloh. Táto funkcia je ponechaná na ApplicationMaster. ResourceManager sa skladá z dvoch

komponent: *ApplicationsManager* a *Scheduler*. Scheduler má funkciu pre aplikáciu alokovať kontajnery zdrojov, ktoré slúžia ako náhrada za sloty. Funkcia ApplicationsManageru je prijímať požiadavky od klienta, spúšťať ApplicationMaster a monitorovať *kontajnery*. ApplicationMaster riadi podúlohy, na ktoré bola úloha od klienta rozdelená, spúšťa ich, monitoruje, a v prípade zlyhania znovu spúšťa.

NodeManager

Každý stroj obsahuje jeden NodeManager, ktorý spúšťa kontajnery zdrojov a monitoruje ich spotrebu. Zasiela správy o stave zdrojov ResourceManageru vždy, keď ApplicationMaster začne alebo dokončí danú podúlohu. NodeManager je *slave daemon*.

ApplicationMaster

ApplicationMaster je slave daemon vyskytujúci sa v každej MapReduce aplikácii. Zabezpečuje životný cyklus aplikácie, čo zahŕňa rozvrhovanie podúloh, spúšťanie podúloh, ich monitorovanie, a v prípade potreby znovu spustenie. Tiež riadi *speculative execution*.

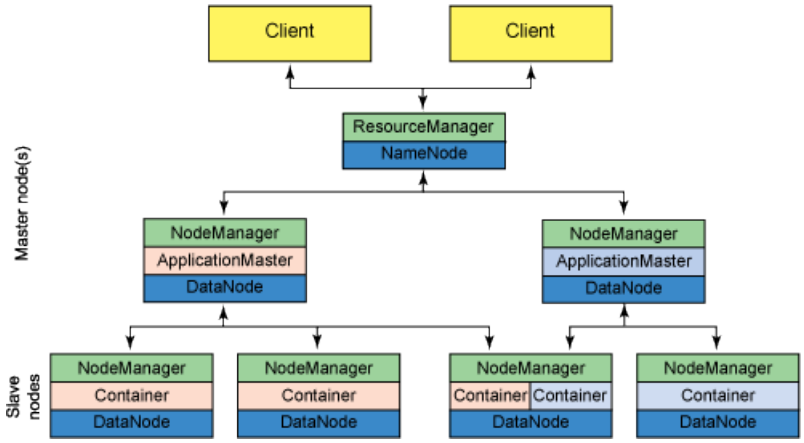
Vyjednáva zdroje od ResourceManageru alebo Scheduleru. V spolupráci s NodeManager spúšťa a monitoruje podúlohy nad zdrojovými kontajnermi. Každá map alebo reduce podúloha môže byť spustená v kontajneroch. Ten istý kontajner podporuje beh map ako aj reduce podúloh. Na rozdiel od slotov, TaskTrackeru, ktoré boli určené buď na map fázu, alebo reduce fázu. To poskytuje omnoho väčšiu flexibilitu a využitie clustru ako pri MR1.

Job HistoryServer

MR2 neobsahuje komponent JobTracker, ktorý by uchovával informácie o vykonaných úlohách. ApplicationMaster je ukončený vždy, keď dokončí prácu MapReduce aplikácie, či už úspešne alebo neúspešne. Job HistoryServer umožňuje uchovávanie informácií o jednotlivých úlohách [28][29].

4.3 Hive

Apache Hive je framework na vyhľadávanie a spravovanie veľkých dátových setov uložených v HDFS. Hive používa jazyk *HiveQL*, ktorý je podobný jazyku SQL a taktiež podporuje MapReduce. Bol vytvorený pre analytikov so skúsenosťami s jazykom SQL, ale slabými



Obr. 4.7: Architektúra YARN(MR2) [23].

znalosťami Javy, aby boli schopní písať MapReduce programy. Hive ukladá dáta do tabuliek. Tabuľka je abstrakciou dát, ktorej metadáta sú uložené v separátnej databáze zvanej *Derby metastore*. Hive shell je primárny spôsob interakcie užívateľa s Hivom pomocou zadávania príkazov v jazyku HiveQL [26] [29].

4.4 Pig

Apache Pig abstrahuje spracovanie veľkých dátových sád a podobne ako Hive je určený na ich analýzu. MapReduce umožňuje užívateľovi špecifikovať Map a Reduce funkcie, ale problém je napísať ich tak, aby pasovali na konkrétny dátový set. S využitím Pigu dokáže užívateľ vytvoriť omnoho bohatšie dátové štruktúry a aplikovať oveľa výkonnejšie transformácie nad dátami za neporovnateľne kratší čas. Pig sa skladá z dvoch častí [29] [26].

- *Pig latin* – jazyk pre vyjadrenie toku dát.
- Prostredie na vytváranie programov.

Obsahuje sadu operácii a transformácií, ktoré sú aplikované na vstupné dáta a z nich je vyprodukovaný výstup. Pig mení tieto transformácie na skupinu MapReduce operácií, pre programátora neviditeľných, a to mu umožňuje plne sa zamerať na dáta a nie na beh programu. Jeho výhodou je, že dokáže spracovať terabyty dát pomocou pár riadkov kódu. Pig Latin je skriptovací jazyk, ktorý je oproti HiveQL flexibilnejší. Programátorovi ponúka hneď niekoľko príkazov na preskúmanie dátovej štruktúry v programe už počas písania programu [29]. Podporuje nasledovné operácie:

- Načítanie a ukladanie.
- Triedenie.
- Filtrovanie.
- Streamovanie.
- Zgrupovanie a pridávanie.
- Spájanie a delenie dát [26].

Umožňuje vytvoriť užívateľom definované funkcie (*UDF*), ktoré operujú na vnorenom dátovom modele Pigu, čo umožňuje dobrú integráciu s jeho operátormi. Ďalšou výhodou UDF je lepšia znovupoužiteľnosť ako pri MapReduce knižniciach. V niektorých prípadoch Pig nepracuje tak dobre ako MapReduce programy, avšak postupom času sa tento rozdiel zmenšuje. Pig ponúka dva módy pre spúšťanie programov, a to lokálny a MapReduce mód.

Lokálny mód

V lokálnom móde Pig využíva jediný Java virtuálny stroj (*JVM*) a pristupuje k lokálnemu súborovému systému. Tento mód je vhodný len pre malé dátové sady alebo na učebné účely. Prístup k lokálnemu módu sa deje pomocou `-exectype` alebo `-x` nasledované možnosťou local [29].

MapReduce mód

V MapReduce móde sú Pig dotazy menené na MapReduce úlohy spúšťané na Hadoop clustri. Cluster môže byť plne alebo pseudo distribuovaný. Tento mód s plne distribuovaným clustrom sa využíva na analýzu veľkých dátových sád. Na využívanie MapReduce módu je potrebné skontrolovať verziu Pigu a Hadoopu, pretože nie všetky verzie sú kompatibilné. Ďalej je potrebné nasmerovať Pig NameNode a ResourceManager clustru. Pri spúšťaní nie je potrebné zadávať žiaden argument, MapReduce mód je nastavený ako primárny.

4.4.1 Pig program

Existujú tri spôsoby ako spúšťať Pig program. Všetky tri môžu byť spustené v lokálnom móde ako aj v MapReduce móde.

- *Skript* – Súbor obsahujúci príkazy jazyka Pig Latin. Alternatíva pre veľmi krátke skripty je `-e`, ktorá umožňuje spúšťať skript ako reťazec priamo z príkazového riadku.
- *Grunt* – Je interaktívny shell podobný ako *napr.* bash a je využívaný pre Pig príkazy. Skripty sú z Gruntu spúšťané pomocou príkazu `exec`. Grunt sa spustí ak nie je špecifikovaný žiadny súbor pre Pig a nie je využitá možnosť `-e`. Pomocou príkazu `grunt` pristúpime ku Grunt shellu z HDFS.
- *Embedded* – Pig program môže byť spustený z Javy využitím `PigServer class`. Pre programátorský prístup ku Gruntu sa používa `PigRunner` [29].

Kapitola 5

Výsledný systém

5.1 Hadoop a HDFS

Hadoop je softvérový open source framework vytvorený firmou Apache. Umožňuje distribuované spracovanie veľkého objemu dát. Zabezpečuje vysokú dostupnosť (*high availability*) dát, čo znamená, že dáta sú v podstate dostupné neustále. Problém nastáva pri zlyhaní NameNode. Tento stav sa nazýva *Single point of failure (SPOF)*, keďže bez NameNode Hadoop systém nie je schopný pokračovať vo výpočte. Hadoop 2.0 prekonáva SPOF zavedením pasívneho SecondaryNameNode. Veľkou výhodou je jeho škálovateľnosť z jedného stroja na tisícky a taktiež paralelné spracovanie dát naprieč týmito strojmi. Hadoop je navrhnutý pre *„commodity hardware“*, preto bolo možné aj výsledný systém nainštalovať na jednom notebooku.

5.1.1 Architektúra

Systém využíva tri virtuálne stroje s operačným systémom Ubuntu 16.04 a verziou Hadoop 2.7.3. Testovanie bolo vykonané na jednom fyzickom stroji, tým pádom pridelenie hardvérových zdrojov bolo obmedzené. Každý virtuálny stroj mal k dispozícií 4 GB pamäti RAM, 2 virtuálne jadrá procesoru a 60 GB pamäti na disku. Hadoop využíva architektúru master/slave. Z toho jeden virtuálny stroj slúži ako *Master* a dva ako *Slave*. Po spustení Hadoop pozostáva zo šiestich typov deamonov, rozdelených následovne:

Daemon	URL	Port
ResourceManager	*http://master:8088*	8088
NameNode	*http://master:50070*	50070
JobHistory Server	*http://master:19888*	19888

Tabuľka 5.1: Implicitne definované porty.

Master

- NameNode
- SecondaryNameNode
- ResourceManager
- JobHistoryServer

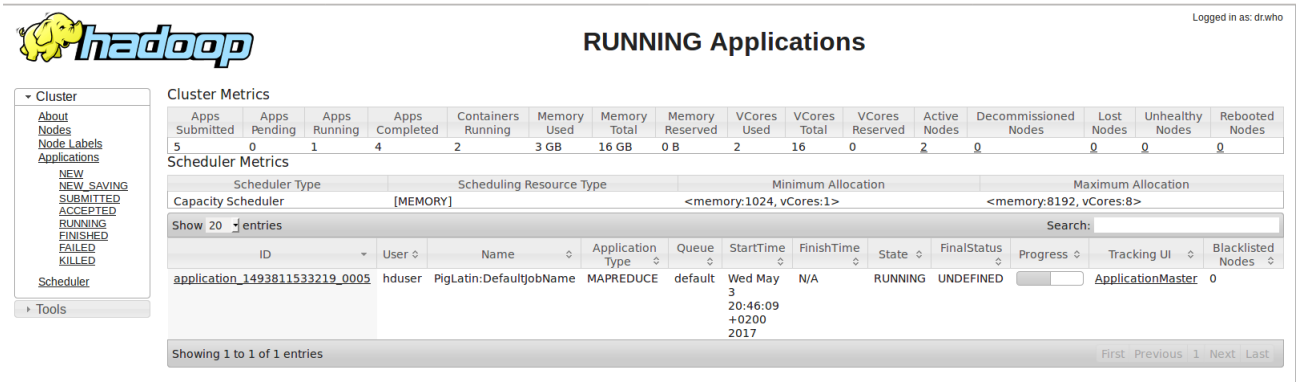
Slave

- DataNode
- NodeManager

Hadoop na správu systému a poskytnutie informácií ponúka webové užívateľské rozhrania. V našom prípade sú všetky nasledovné Hadoop deamony, ktoré su podrobnejšie popísané v sekcií 4.2.2, spúšťané mastrom, preto URL vždy obsahuje *master* a číslo portu. Implicitne definované porty k jednotlivým deamonom sú zobrazené v tabuľke 5.1.

RecourceManager

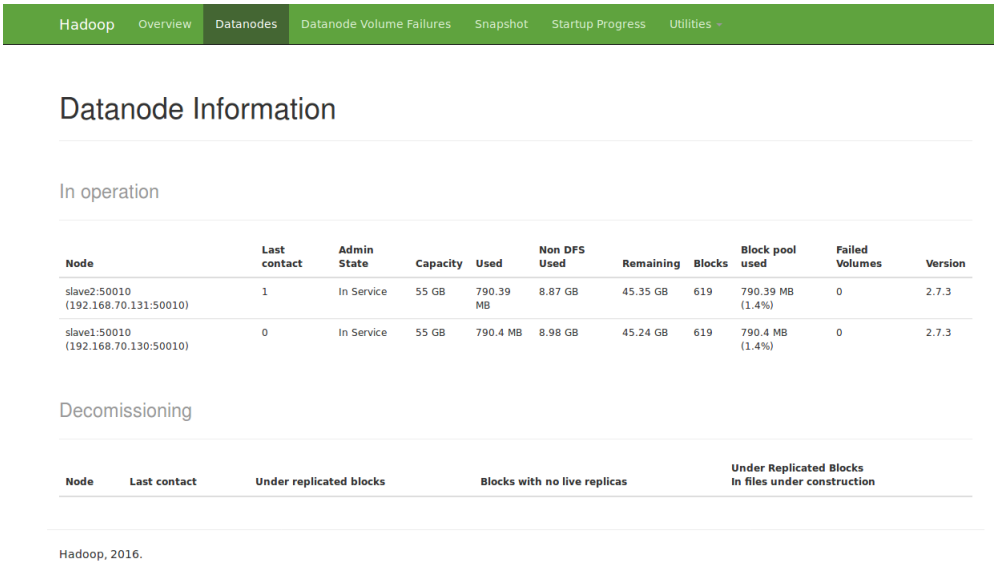
Master daemon spravujúci alokáciu zdrojov potrebných pre aplikáciu. Pozostáva z ApplicationsManager a Scheduler, ktoré je vidieť aj vo webovom užívateľskom rozhraní. Rozhranie poskytuje informácie o priebehu aplikácií a koľko jednotlivé uzly spotrebujú zdrojov počas jej behu. Tiež sú tu popísané informácie o konfigurácií serveru, log súbory a metriky. Ukážka je zobrazená na obrázku 5.1.



Obr. 5.1: Hadoop ResourceManager web UI.

NameNode

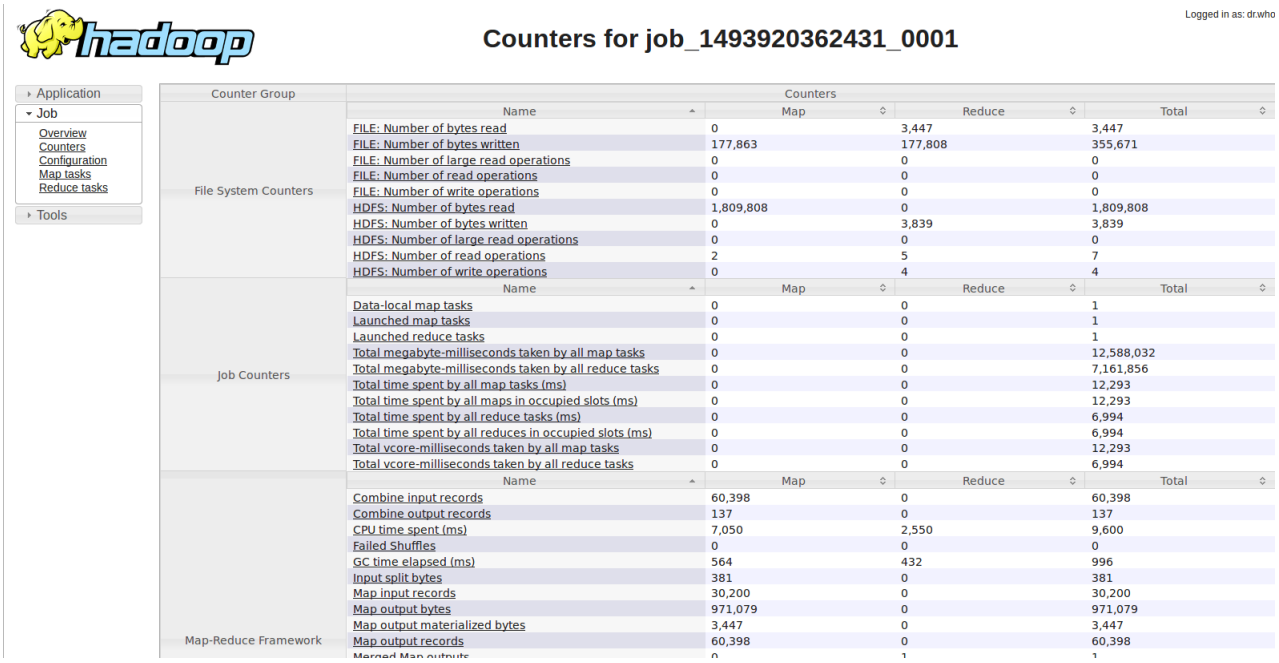
NameNode riadi menný priestor súborového systému. Uchováva metadáta dát uložených v HDFS, ktoré sú ukladané do pamäti RAM. NameNode periodicky komunikuje s DataNode, kde sú uložené dáta zo vstupného súboru. Hadoop webové rozhranie NameNodu poskytuje informácie o stave súborového systému ako detaily súborov a blokov, podrobné informácie o dátových uzloch a ich zlyhaniach. Ďalej umožňuje prehľadávanie súborov v HDFS a tiež log súborov master serveru ako je vidieť na obrázku 5.2.



Obr. 5.2: Hadoop NameNode web UI.

JobHistory Server

Umožňuje uchovávanie informácií o jednotlivých džoboch. Obsahuje zoznam všetkých džobov s časom ich prijatia, spustenia, ukončenia, ich identifikačné číslo, meno, používateľa, ktorý džob spustil, informácie o fáze Map a Reduce a celkový stav ukončenia. Uchováva aj detailnejšie informácie o jednotlivých džoboch ako ich konfigurácie, podrobnejšie detaily o Map a Reduce úlohách a počítadla času a bytov pri spracovávaní. Časť užívateľského rozhrania je zobrazená na obrázku 5.3.



Obr. 5.3: Hadoop JobHistory web UI.

5.1.2 Skripty Hadoop systému

Po načítaní virtuálnych strojov (stačí master a 1 slave) je potrebné Hadoop spustiť osobitne. Na spustenie slúži skript `start-all.sh`, ktorý však vypíše varovanie. Odporúča sa spustiť skripty postupne ako je uvedené na obrázku 5.4, kde prvá časť spustí *Job History Server*. Druhý skript spustí *NameNode*, *SecondaryNameNode* a dva *DataNode* na každom slave stroji. *YARN Daemons* (MapReduce Daemon), *ResourceManager* a *NodeManager* ku každému dátovému uzlu sú spustené tretím skriptom.

Skripty slúžiace na zastavenie činnosti systému vyzerajú obdobne, buď jeden príkaz `stop-all.sh`, alebo postupne ako je zobrazené na obrázku 5.5. Príkazy na správu HDFS

```
hduser@master:~$ mr-jobhistory-daemon.sh start historyserver &&
start-dfs.sh && start-yarn.sh
```

Obr. 5.4: Spustenie Hadoop.

```
hduser@master:~$ mr-jobhistory-daemon.sh stop historyserver &&
stop-dfs.sh && stop-yarn.sh
```

Obr. 5.5: Vypnutie Hadoop.

používane v Hadoop shelle sú podobné tým, ktoré sa využívajú v bežných linux shelloch. Ich zoznam je dostupný na stránkach Apache Foundation. Na obrázku 5.6 je zobrazená ukážka adresárov, ktoré sa nachádzajú v HDFS.

```
hduser@master:~$ hadoop fs -ls
Found 3 items
drwxr-xr-x   - hduser supergroup          0 2017-03-20 18:27 input
drwxr-xr-x   - hduser supergroup          0 2017-03-20 22:58 output
drwxr-xr-x   - hduser supergroup          0 2017-03-20 22:58 pig_scripts
```

Obr. 5.6: Súbory v HDFS.

5.2 Analýza

Agregačné operácie boli prevedené nad súborom vo formáte *.csv*, obsahujúcom export pšeničnej múky z Kanady do rôznych regiónov sveta od roku 1946 po rok 1987. Súbor pozostáva z 30200 riadkov a 7 stĺpcov, jeho veľkosť je 1.8 MB. Súbor takejto veľkosti nemôžme zaraďovať medzi zdroje veľkých dát, každopádne Hadoop sa používa aj pri spracovaní dát s veľkým počtom riadkov a stĺpcov. Príklad je zobrazený na obrázku 5.7. Cieľom však bolo testovať systém Hadoop a aplikovať základné agregačné operácie. Tie boli prevedená pomocou skriptov v jazyku *Pig Latin*. Aplikovaných bolo deväť skriptov obsahujúcich agregačné operácie: ***počet***, ***aritmetický priemer***, ***maximum***, ***minimum*** a ***medián***. Hadoop spoľahlivo funguje aj na bežnom počítači. K paralelnému spracovaniu však nedochádza, keďže využíva iba jeden fyzický stroj a nie desiatky až tisícky strojov v clustri. Doba spracovania skriptu, ktorý obsahuje výpočet aritmetického priemeru exportovaných hodnôt všetkých

krajín spolu a aritmetický priemer exportovaných hodnôt jednotlivých krajín, bola 3 minúty a 3 sekundy. Z toho Map fáza trvala 2 minúty a 33 sekúnd. Pri opakovanom spracovaní toho istého výpočtu je čas neporovnateľne nižší, spracovanie rovnaké skriptu po druhýkrát trvalo iba 12 sekúnd. Výsledkom je, že Hadoop má svoje uplatnenie nie len na spracovanie veľkého množstva dát na stovkách strojoch, každopádne až tam sa naplno prejaví jeho potenciál.

	A	B	C	D	E	F	G
25641	1986/02	Canada	Wheat flour	Jamaica	v381014	1.1.117	27
25642	1986/03	Canada	Wheat flour	Jamaica	v381014	1.1.117	70
25643	1986/04	Canada	Wheat flour	Jamaica	v381014	1.1.117	73
25644	1986/05	Canada	Wheat flour	Jamaica	v381014	1.1.117	10
25645	1986/06	Canada	Wheat flour	Jamaica	v381014	1.1.117	43
25646	1986/07	Canada	Wheat flour	Jamaica	v381014	1.1.117	274
25647	1986/08	Canada	Wheat flour	Jamaica	v381014	1.1.117	70
25648	1986/09	Canada	Wheat flour	Jamaica	v381014	1.1.117	104
25649	1986/10	Canada	Wheat flour	Jamaica	v381014	1.1.117	143
25650	1986/11	Canada	Wheat flour	Jamaica	v381014	1.1.117	1877
25651	1986/12	Canada	Wheat flour	Jamaica	v381014	1.1.117	55
25652	1987/01	Canada	Wheat flour	Jamaica	v381014	1.1.117	1813
25653	1987/02	Canada	Wheat flour	Jamaica	v381014	1.1.117	125
25654	1987/03	Canada	Wheat flour	Jamaica	v381014	1.1.117	89
25655	1987/04	Canada	Wheat flour	Jamaica	v381014	1.1.117	34
25656	1987/05	Canada	Wheat flour	Jamaica	v381014	1.1.117	54
25657	1966/01	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1796
25658	1966/02	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1492
25659	1966/03	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1007
25660	1966/04	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	691
25661	1966/05	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	2994
25662	1966/06	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1675
25663	1966/07	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	562
25664	1966/08	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	2833
25665	1966/09	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	2110
25666	1966/10	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	2011
25667	1966/11	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1634
25668	1966/12	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1850
25669	1967/01	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1710
25670	1967/02	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1286
25671	1967/03	Canada	Wheat flour	Leeward-Wind Islands	v380996	1.1.118	1388

Obr. 5.7: Príklad vstupného súboru.

5.3 Pig

Apache Pig je platforma na analýzu veľkých dátových sád. Veľkou výhodou Pig programov je ich štruktúra, ktorá dovoľuje paralelné spracovanie, tým pádom je program schopný spracovávať veľmi veľké dátové súbory [6]. Spracovanie začína Map fázou, kde sú uskutočnené jednotlivé Map úlohy. Ich výstup prichádza na vstup fázy Reduce, vykonávajúcej Reduce

úlohy. Pri písaní Pig skriptov nie je nutné vedieť ako presne prebiehajú MapReduce fázy, pretože Pig ich výpočet rieši za užívateľa.

5.3.1 Popis pig skriptu

Na obrázku 5.8 je vytvorená relácia `input_file`, ktorá načíta súbor uložený v HDFS pomocou operátoru `LOAD`. Vstavaná funkcia Apache Pig `PigStorage()` rozdelí jednotlivé hodnoty zo vstupného súboru. Ako parameter funkcie sa zadáva oddeľovač, v tomto prípade čiarka, podľa ktorého sú dáta zo súboru rozdelené. Ďalej bola definovaná schéma pre uloženie dát. Schéma z príkladu je ukázaná v tabuľke 5.2.

```
input_file = LOAD '/user/hduser/input/wheat_exports.csv' using PigStorage (',')
as (date:chararray, geo:chararray, export:chararray, dest:chararray, vec:chararray,
coordinat:chararray, value:int);
```

Obr. 5.8: Načítanie vstupného súboru pomocou Pig Latin.

stĺpec	date	geo	export	dest	vec	coordinat	value
dátový typ	char array	char array	char array	char array	char array	char array	int

Tabuľka 5.2: Schéma input_file.

Vytvorená relácia `export_raw` odstráni hlavičky zo vstupného súboru. Relácia `avg_dest_grp` zoskupí reláciu `export_raw` podľa poľa `dest` v definovanej schéme, ktoré obsahuje destinácie exportu. Ukážka relácií sa nachádza na obrázku 5.9.

```
export_raw = FILTER input_file BY geo != 'GEO';
avg_dest_grp = group export_raw by dest;
```

Obr. 5.9: Operácie filter a group.

Relácia je zložená z *tzv. bag of tuples*. Tuple je zoradená množina polí ohraničená „()“. V uvedenom príklade tuple obsahuje sedem polí. Kolekcia takýchto tuple sa nazýva *Bag*, ohraničených „{}“. Na príklade na obrázku 5.10 je zobrazená Bag s dvomi tuple. Relácie v Pig skriptoch sú podobné reláciam v databázach, kde jednotlivé tuple predstavujú riadky databázovej tabuľky. Narozdiel od relačných databázových tabuliek, Pig relácie nevyžadujú,

aby každá tuple obsahovala rovnaký počet polí. Tak isto nieje nutné, aby poliam na rovnakej pozícií (stĺpcoch) zodpovedal rovnaký dátový typ [6].

```
(Jamaica,{(1986/02,Canada,Wheat flour,Jamaica,v381014,1.1.117,27),
(1986/03,Canada,Wheat flour,Jamaica,v381014,1.1.117,70), ... });
```

Obr. 5.10: Bag of tuples.

Na obrázku 5.11 je znázornené použitím agregačnej funkcie `AVG`, ktorá iteruje reláciou `avg_dest_grp` dostávame výsledok, ktorý predstavuje aritmetický priemer množstva exportovaného do jednotlivých regiónov. Relácia je uložená do súboru v HDFS, ktorý bol vytvorený v adresári so zadanou cestou pomocou operátoru `STORE`. Po úspešnom výpočte sú vytvorené dva súbory: *part-r-00000*, ktorý obsahuje výsledok výpočtu a súbor *_SUCCESS*, ktorý je prázdny, ale signalizuje úspešné ukončenie výpočtu. Časť výsledného súboru je zobrazená na obrázku C.2.

```
avg_dest = foreach avg_dest_grp generate group as grp, AVG(export_raw.value);
STORE avg_dest INTO '/user/hduser/output/avg_dest';
```

Obr. 5.11: Použitie agregačnej funkcie AVG.

Pre ostatné agregačné operácie je možné použiť funkcie `COUNT()`, `MAX()`, `MIN()` a `SUM()`. Pre medián bola použitá UDF z kolekcie *Apache DataFu*, príklad je zobrazený v prílohe C.1.

Kapitola 6

Záver

Cieľom bakalárskej práce bolo zoznámiť sa s technológiu spracovania Big Data Hadoop a realizovať niektoré operácie OLAP pre podporu rozhodovania na vhodnej vzorke dát. Technológia Hadoop bola nainštalovaná na tri virtuálne stroje osobného počítača. Z operácii OLAP boli realizované agregačné funkcie: počet, aritmetický priemer, maximum, minimum a medián pomocou MapReduce frameworku Pig. Priebeh a funkcionalita skriptov s realizovanými operáciami je bližšie popísaný v kapitole päť.

Pri podmienkach s jedným fyzickým strojom nebolo možné paralelne spracovanie reálne veľkých dát o veľkosti terabytov či petabytov. Agregačné operácie boli však realizované technológiou Hadoop, ktorý je ľahko škálovateľný na viacero strojov. V prípade zabezpečenia výpočtového clustru je možné tento systém jednoducho prispôsobiť a využiť na analýzu Big Data.

Literatúra

[1] *Codd's paper*. [Online; navštívené 24.04.2017].
URL http://olap.com/learn-bi-olap/codds-paper/

[2] *Cube Cells (Analysis Services - Multidimensional Data)*. [Online; navštívené 25.04.2017].
URL https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models-olap-logical-cube-objects/cube-cells-analysis-services-multidimensional-data

[3] *HDFS Architecture*. HDFS Tutorial, [Online; navštívené 12.04.2017].
URL http://www.hdfstutorial.com/hdfs-architecture/

[4] *OLAP operations*. [Online; navštívené 25.04.2017].
URL http://athena.ecs.csus.edu/~olap/olap/OLAPoperations.php

[5] *Overview of Online Analytical Processing (OLAP)*. [Online; navštívené 25.04.2017].
URL https://support.office.com/sk-sk/article/Preh%C4%BEad-technol%C3%B3gie-OLAP-Online-Analytical-Processing-15d2cdde-f70b-4277-b009-ed732b75fdd6#bmwhat_is_on_line_analytical_processing

[6] *Pig Latin Basics*. The Apache Software Foundation., [Online; navštívené 28.04.2017].
URL https://pig.apache.org/docs/r0.10.0

[7] *The basic structure of a cube*. [Online; navštívené 25.04.2017].
URL https://blog.xlcubed.com/2008/11/the-basic-strucure-of-a-cube/

[8] *Understanding OLAP cube*. [Online; navštívené 24.04.2017].
URL https://technet.microsoft.com/en-us/library/hh916543(v=sc.12).aspx

[9] *Understanding Your Business With Descriptive, Predictive And Prescriptive Analytics*. [Online; navštívené 28.04.2017].
URL https://datafloq.com/read/descriptive-predictive-prescriptive-analytics/151

[10] *Descriptive, Predictive, and Prescriptive Analytics Explained*. Červenec 2016, [Online; navštívené 28.04.2017].
URL https://halobi.com/2016/07/descriptive-predictive-and-prescriptive-analytics-explained/

[11] Beal, V.: *Structured Data*. [Online; navštívené 24.04.2017].
URL http://www.webopedia.com/TERM/S/structured_data.html

[12] Borthakur, D.: *HDFS Architecture Guide*. The Apache Software Foundation, [Online; navštívené 06.04.2017].
URL https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[13] Chamula, M.: *Manažerské informační systémy a jejich úloha v řízení podniku*. Diplomová práce, Fakulta informatiky Masarykovy univerzity, 2011.

[14] Gunarathne, T.: *Hadoop and MapReduce*. Packt Publishing Limited, [Online; navštívené 25.04.2017].
URL https://www.packtpub.com/books/content/hadoop-and-mapreduce

[15] H.Lenz, B. T.: *A Formal Framework of Aggregation for the OLAP-OLTP Model*. *Journal of Universal Computer Science*, 2009.

[16] Hruška, T.: *BIG DATA - VELKÁ DATA*. FIT VUT v Brně, [Online; navštívené 24.04.2017].
URL
https://www.fit.vutbr.cz/study/courses/WAP/private/opory/PIS3000LAP.pdf

[17] Hruška, T.: *Hadoop*. FIT VUT v Brně, [Online; navštívené 12.04.2017].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/pis310bigdata.pdf

[18] Jensen, T. B. P. C. S.: *Multidimensional Database Technology*. FIT VUT v Brně, Prosinec 2001, [Online; navštívené 28.04.2017].
URL http://infolab.usc.edu/csci599/Fall2002/paper/I1_pederson_p40.pdf

[19] Jiawei Han, M. K., Jian Pei: *Data Mining Southeast Asia Edition*. Elsevier / Morgan Kaufmann, druhé vydání, 2006, ISBN 978-0-08-047558-5.

[20] Khosrow-Pour, M.: *Big data : concepts, methodologies, tools, and applications*. Big data : concepts, methodologies, tools, and applications, 2016, ISBN 9781466698406.

[21] Kumar, D.: *Hadoop Tutorial: Part 3*. [Online; navštívené 15.04.2017].
URL http://www.bigdataplanet.info/2013/10/Hadoop-Tutorial-Part-3-Replication-and-Read-Operations-in-HDFS.html

[22] Kumar, D.: *Hadoop Tutorial: Part 4*. [Online; navštívené 16.04.2017].
URL http://www.bigdataplanet.info/2013/10/Hadoop-Tutorial-Part-4-Write-Operations-in-HDFS.html

[23] M. Jones, M. N.: *Moving ahead with Hadoop YARN*. [Online; navštívené 28.04.2017].
URL https://www.ibm.com/developerworks/library/bd-hadoopyarn/

[24] Rouse, M.: *OLAP cube*. [Online; navštívené 24.04.2017].
URL http://searchdatamanagement.techtarget.com/definition/OLAP-cube

[25] Stimmel, C. L.: *Big Data Analytics Strategies for the Smart Grid*. Auerbach Publications, 2014, ISBN 9781482218282.

[26] Suja, R.: *Big Data*. Inštitút informatiky a štatistiky, [Online; navštívené 11.04.2017].
URL http://www.infostat.sk/web2015/sk/_publikacie/Big_Data.pdf

[27] Tomáš Hruška, Z. K.: *Studijní opora Informační systémy (IIS,PIS)*. FIT VUT v Brně, Únor 2012, [Online; navštívené 28.04.2017].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaIIS2PISPojemDataProcesyTransakce.pdf

[28] Vohra, D.: *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. Apress Berkely, CA, USA, 2016, ISBN 978-1-4842-2199-0.

[29] White, T.: *Hadoop: The Definite Guide*. O'Reilly Media, Inc., Čtvrté vydání, 2015, ISBN 978-1-491-90163-2.

Prílohy

Príloha A

Obsah DVD

- text bakalárskej práce vo formáte `pdf` a LaTeXsúbory
- `input`, obsahujúci vstupný súbor
- `output`, obsahujúci výstupné súbory
- `pig_scripts`, obsahujúci jednotlivé skripty

Príloha B

Základne Hadoop príkazy

Príklady sú uvedené pre používateľa *hduser*.

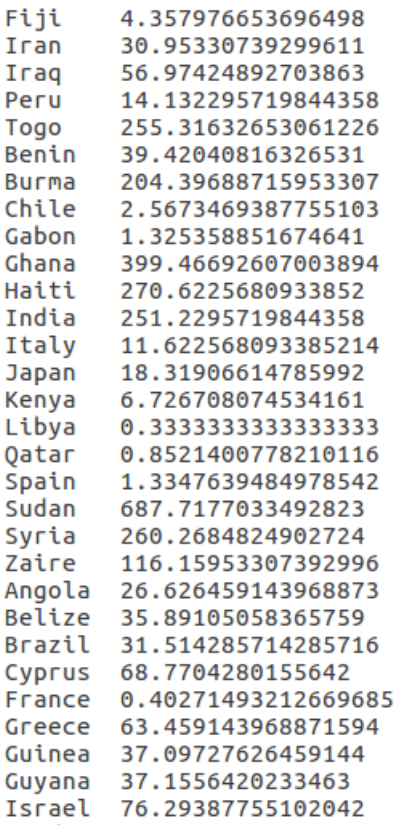
- Prehľadávanie súborov v HDFS:
  `hadoop fs -ls`.
- Pre vytvorenie adresáru v HDFS sa používa príkaz:
  `hadoop fs -mkdir -p /user/hduser/názov_adresáru`.
- Pridanie súboru do adresára v HDFS:
  `hadoop fs -put súbor /user/hduser/adresár`.
- Vymazanie súboru z HDFS:
  `hdfs dfs -rm -r hdfs://master:9000/user/hduser/súbor`.
- Skopírovanie súboru z HDFS do lokálneho FS:
  `hadoop fs -get /hdfs/source/path /localfs/destination/path`
  `hadoop fs -copyToLocal /hdfs/source/path /localfs/destination/path`
- Prístup ku Grunt shellu pre využívanie Pigu, pomocou príkazu `pig`.
- Spúšťanie skriptu pomocou *napr.:*
  `exec hdfs://master:9000/user/hduser/pig_scripts/wheat-avg.pig`

Príloha C

Obrázky

```
register /usr/local/pig-0.16.0/datafu-pig-incubating-1.3.1.jar
DEFINE Median datafu.pig.stats.StreamingMedian();

input_file = LOAD '/user/hduser/input/wheat_exports.csv' using PigStorage (',') as (date:chararray,
geo:chararray, export:chararray, dest:chararray, vec:chararray, coordinat:chararray, value:int);

export_raw = FILTER input_file BY geo != 'GEO' AND value != 0;

grp_median = group export_raw all;
median = foreach grp_median generate Median(export_raw.value);

STORE median INTO '/user/hduser/output/median';
```

Obr. C.1: Príklad agregačnej funkcie medián.



Obr. C.2: Príklad výstupného súboru.