Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Zpracování velkého objemu dat pomocí technologie Hadoop

2019

Mgr. Petr Zifčák

Vedoucí práce: RNDr. Stanislav Opichal, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Mgr. Petr Zifčák
Název práce:	Zpracování velkého objemu dat pomocí technologie Hadoop
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	RNDr. Stanislav Opichal, Ph.D.
Počet stran:	61
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Mgr. Petr Zifčák
Title:	Processing large volumes of data using Hadoop technology
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Applied Computer Science, combined form
Supervisor:	RNDr. Stanislav Opichal, Ph.D.
Page count:	61
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Apache Hadoop zahrnuje technologie pro zpracování velkých objemů dat v distribuovaném prostředí výpočetních zdrojů. V úvodních partiích práce jsou popsány principy fungování systému a jeho základních komponent. Následuje porovnání s jinými datovými úložišti, zejména relačními databázemi. Stěžejní částí je řešení praktického případu užití systému — zpracování dat z oblasti geografických informačních systémů. Výsledkem je návrh vlastního clusteru, instalace systému včetně potřebných komponent a porovnání výkonu s relační databází PostgreSQL.

Synopsis

Apache Hadoop includes technologies for processing large volumes of data in a distributed computing resource environment. The introductory parts of the thesis describe the principles of the system and its basic components. The following is a comparison with other data stores, especially relational databases. The key part is the solution of the practical use case of the system — processing of data from geographic information systems. The result is a custom cluster design, system installation, including required components, and performance comparisons with the PostgreSQL relational database.

Klíčová slova: Hadoop; Big Data; Hive; Gis Tools for Hadoop; PostGIS; distribuované úložiště; cluster

Keywords: Hadoop; Big Data; Hive; Gis Tools for Hadoop; PostGIS; distributed storage; cluster

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Poděkování
Děkuji vedoucímu bakalářské práce RNDr. Stanislavu Opichalovi PhD. a jeho kolegovi Jindřichu Pospíšilovi ze společnosti Behaim IT Solutions s.r.o. za jejich rady a čas, které vedly ke zkvalitnění této práce.

Prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Žijeme v digitální době, pro kterou je charakteristická velká rychlost růstu objemu dostupných dat a jejich různorodost. Každé dva roky se množství digitálních dat zdvojnásobí [1]. Hnacím motorem jsou nové technologie a služby, jako např. mobilní zařízení, senzorové sítě, sociální média, která každou sekundu generují obrovské množství údajů, které je potřeba uložit a efektivně zpracovat. Jedním z nástrojů na zpracování velkých objemů dat je i ekosystém Apache Hadoop, který bude představen v této práci.

Předloženou práci je možné rozdělit do tří navazujících částí:

1. Úvodní kapitola 3 vysvětluje základní pojmy používané v oblasti distribuovaného zpracování a ukládaní velkých objemů dat. Dále uvádí základní pojmy z oblasti geografických informačních systémů a použitých vstupních formátů dat.

2. V kapitolách 4 a 5 jsou teoreticky popsány principy fungování systému Apache Hadoop a jeho komponent, s důrazem na komponenty pro budování datových skladů a databázových technologií — Hive a HBase. Vlastnosti vybraných komponent jsou srovnány s přístupem obvyklým u relačních databází.

3. Prakticky zaměřené kapitoly 6 a 7 obsahují osobní poznatky při hledání vhodného řešení clusteru, postup instalace systému Hadoop v cloudu MetaCentra/Cesnet. Na závěr je provedeno srovnání postupu a výkonu ve zpracování prostorových dat knihovnou GIS Tools for Hadoop v prostředí Apache Hive a rozšířením PostGIS relační databáze PostgreSQL.

2 Cíle

Cílem práce je ukázat možnosti ekosystému Apache Hadoop v oblasti distribuovaného ukládání a zpracování velkého množství dat. Budou popsány principy, na kterých je postaveno fungování systému Hadoop a uveden přehled jeho základních komponent. Detailněji budou představeny komponenty Hive a HBase a jejich porovnání s běžnými datovými úložišti pro strukturovaná data — relačními databázemi. A nakonec na příkladu užití (*use case*) při zpracování geografických dat bude demonstrována výpočetní síla Hadoop clusteru.

3 Základní pojmy

3.1 Big Data

Data, jejichž velikost (*volume*), rychlost nárůstu (*velocity*) a různorodost (*variety*) neumožňuje zpracování pomocí doposud známých a ověřených technologií v rozumném čase byla definována jako **„Big Data“**[1][2].

Podle počátečních písmen vlastností bývají označovány jako **„3V“** a jejich význam je tento:

- ***Volume***, neboli velikost, se vztahuje k objemu zpracovávaných dat. S příchodem nových komunikačních technologií, sociálních médií, autonomních systémů sběru dat senzorových sítí a podobných systémů vzniká ohromné množství dat, které je potřeba uložit a efektivně zpracovat. Data jsou ukládána na stovkách až tisících serverech s celkovou úložnou kapacitou peta až exabyty.

- ***Velocity***, souvisí s rychlostí, s jakou jsou data generována. Množství dat, zejména nestrukturovaných, roste v čase až exponenciálně. Podle rychlosti růstu se volí vhodný nástroj pro jejich zpracovaní — od periodického nárůstu dat, které lze efektivně analyzovat pomocí úloh založených na MapReduce paradigmatu až po analýzu proudů dat v reálném čase (např. Apache Storm).

- ***Variety***, neboli různorodost, poukazuje na skutečnost, že uchovávaná a analyzovaná Big Data mohou být jak ve strukturované (např. tabulková data), tak semi-strukturované (např. XML, JSON) a nestrukturované formě. Odhaduje se [3], že 80–90 % všech dat má nestrukturovanou formu. Často v nich dochází k mísení textového a multimediální obsahu, jako např. emaily, audio/video soubory, obrazové soubory, prezentace, webové stránky apod.

Postupně byly k definici přidávány další **„V“**, jako např. *Veracity* — nižší věrohodnost dat, *Validity* a *Volatily* — s ohledem na rychlost změny vstupu omezená doba platnosti a použitelnosti informací [4].

Jedním z nástrojů umožňující zpracování a analýzu Big Dat ve velmi krátkém čase je systém **Apache Hadoop**. Byl navržen pro zvládnutí peta až exabytů dat uložených ve více uzlech současně. Dotazy a zpracování úloh probíhají paralelně a samostatně v každém jednotlivém uzlu, přičemž výsledky jsou shromažďovány v úložné vrstvě a odtud dále odesílány k analýze do dalších nástrojů.

3.2 Prostorová data

Prostorová data jsou data, která se vztahují k určitým místům v prostoru a pro která jsou známé lokalizace těchto míst. Prostor je charakterizován 2 a více dimenzemi, přičemž pro jednoznačné určení polohy objektu v prostoru se používají

[1] Termín Big Data se do češtiny doslovně nepřekládá.

souřadnice (označované nejčastěji symboly X, Y, Z pro délky, nebo zeměpisná šířka a délka pro úhlové velikosti). Kromě geometrické složky, která popisuje polohu a tvar objektu, jsou data opatřena atributy, které obsahují vlastnosti objektu.

Pro analýzu, transformaci a vizualizaci aj. operace s prostorovými daty jsou používány geografické informační systémy, zkráceně GIS.

Prostorová data mohou být prezentována analogově nebo digitálně. Datových formátů pro přenos, zpracování a prohlížení prostorových dat existuje velké množství. V této práci budou použita pouze data digitální, a to ve vektorovém datovém modelu, příp. v souborovém datovém formátu CSV nebo variant JSON/XML struktur. Z tohoto důvodu jsou níže popsány jen ty, které budou dále v práci použity.

Shapefile

Souborový vektorový datový formát shapefile byl vyvinut společností Esri, kterou byl v roce 1998 představen v technické dokumentaci [5] jako otevřený standard pro datovou interoperabilitu mezi Esri a ostatními softwarovými produkty. Lze v něm ukládat geometrické objekty v podobě bodů, linie, ploch (pouze rovné hrany, nikoliv křivky) a multipatch (tento objekt slouží k definování reprezentace obalu 3D objektů). Geometrie každého prvku je popsána vektorovými souřadnicemi jeho vrcholů — X a Y kartézského systému a volitelně Z reprezentující nadmořskou výšku.

Shapefile je složen minimálně ze třech samostatných souborů umístěných společně ve stejném adresáři. Tyto tři soubory se stejným názvem se liší pouze v příponě *.shp, *.shx a *.dbf. Informace v shp souboru popisují geometrie každého prvku. Dbf soubor je tabulka standardu dBASE, ve které je ke každému samostatnému prvku nebo skupině prvků stejného typu jeden řádek v tabulce. Propojení mezi geometrií prvku v shp a příslušným řádkem v dbf tabulce je uloženo v souboru shx. Kromě výše uvedených povinných souborů lze mít volitelně definovaný souřadnicový systém v souboru s příponou *.prj, kódování znakové stránky dbf tabulky v souboru *.cpg, případně indexy a metadata v dalších doplňkových souborech.

I přestože má shapefile řadu nevýhod, jako je např. nutnost pracovat s minimálně třemi soubory, nemožnost modelovat prostorové vztahy (topologie), patří k nejrozšířenějšímu GIS vektorovému formátu pro práci s geodaty. Proto byl i převládajícím vstupním formátem pro tuto práci.

CSV

Formát CSV (*Comma-Separeted Values*, hodnoty oddělené čárkami) je jednoduchý textový soubor pro výměnu tabulkových dat. Sestává z řádků, ve kterých jsou jednotlivé hodnoty odděleny předem definovaným znakem. Tímto znakem je většinou čárka, ale může jím být i jiný, např. středník, tečka, tabulátor. Jednotlivé hodnoty mohou být uzavřeny do uvozovek (“hodnota“) a první řádka může obsahovat jména sloupců. Pro formát CSV neexistuje závazná specifikace

struktury dat, proto je při importu/exportu dat mezi různými softwary nutné často manuálně nastavit jeho vlastnosti (kromě výše uvedených i např. ukončovač řádků CR/LF na Win32 platformě vs. LF na ostatních) a popsat datové typy hodnot ve sloupcích a jejich omezení např. max. délku řetězců nebo počet desetinných míst čísel (lze specifikovat ve volitelném přidruženém *.csvt souboru).

CSV formát byl v této práci využit jako výměnný formát bodových prvků do/z Hadoop, neboť ve srovnání s formáty založených na SHP/JSON je jeho velikost nejmenší.

Níže je uveden příklad struktury souboru s oddělovači v podobě čárky, ve kterém je popsána geometrie čtyř bodových prvků pomocí souřadnic X, Y a dva atributy ID a DESC.

```
X,Y,ID,DESC
16.892270,48.849290,602da86c7d8cc328dbd,souřadnice sídla
16.908690,48.751010,edbaeeae53e9a4a0c81,souřadnice sídla
16.908690,48.751010,d985354ef8e22493d00,souřadnice sídla
15.131930,49.662780,8cde2892118fae6bad9,souřadnice sídla
```

JSON a GeoJSON

JSON (*JavaScript Object Notation*) je na platformě nezávislý výměnný textový formát odvozený z programovacího jazyku JavaScript, ale díky své značné oblibě je široce podporován i v jiných jazycích (lze snadno číst a parsovat a ve srovnání s XML je datově úspornější). Obsah je organizován do dvou struktur, a to uspořádaných seznamů hodnot (pole) nebo objektů (kolekce párů klíč-hodnota). K podporovaným základním datovým typům patří čísla, řetězce (uzavřené do složených uvozovek), pravdivostní hodnoty, pole, objekty a prázdná hodnota null.

Pro snadnější práci s prostorovými daty byl z formátu JSON odvozen formát GeoJSON, který zavádí podporu geometrických datových typů – bodů, linií, polygonů a jejich skupin. Podle normy RFC 7946 — The GeoJSON Format, 2016[2] je jediným podporovaným souřadnicovým systémem WGS84 a hodnoty souřadnic se vyjadřují v desetinných stupních.

Níže je uveden příklad struktury souboru GeoJSON pro jeden bod.

```
{
   "type":"Feature",
   "properties":{
      "ID":"b7be92e078c72fb4baec6301bb94bfc7",
      "PUVOD":"souřadnice sídla"
   },
   "geometry":{
      "type":"Point",
```

[2]https://tools.ietf.org/html/rfc7946#page-12

```
      "coordinates":[
         16.17277,
         49.43754
      ]
   }
}
```

Společnost Esri, která je celosvětovým lídrem v oblasti geoinformačních technologií, používá ve svých produktech svou vlastní verzi formátu (Esri) JSON[3], ve které přidává podporu dalších geometrických objektů a jejich povolených vlastností.

Níže je pro srovnání uveden zápis téhož bodu jako pro GeoJSON, ale ve formátu Enclosed (Esri) JSON. Pokud z něho vypustíme záhlaví a ponecháme jen specifikaci geometrie a atributů, vznikne zjednodušená varianta formátu Unenclosed (Esri) JSON. Ta umožňuje snadnější přidávání (*append*) prvků k již existujícím, byť na úkor ztráty popisných informací ze záhlaví.

```
{
   "geometryType":"esriGeometryPoint",
   "spatialReference":{
      "wkid":4326
   },
   "fields":[
      {
         "name":"ID",
         "type":"esriFieldTypeString",
         "alias":"ID"
      },
      {
         "name":"PUVOD",
         "type":"esriFieldTypeString",
         "alias":"PUVOD"
      }
   ],
   "features":[
      {
         "geometry":{
            "x":16.17277,
            "y":49.43754
         },
         "attributes":{
            "ID":"b7be92e078c72fb4baec6301bb94bfc7",
            "PUVOD":"souřadnice sídla"
```

[3]https://github.com/Esri/spatial-framework-for-hadoop/wiki/JSON-Formats

```
            }
        }
    ]
}
```

Datové soubory ve formátu Geo- a Esri- JSON byly v této práci využity jako vstupní soubory pro liniové a plošné prvky frameworku Gis Tools for Hadoop.

3.3 Distribuované databáze

Distribuovaná databáze je logicky související kolekce sdílených dat a jejich popisů, fyzicky rozmístěná v síti počítačů [6]. V distribuovaných systémech se počítače označují jako uzly a jejich skupiny jako cluster nebo grid[4]. Přes rozmístění dat na více uzlech se databáze pro uživatele jeví jako jeden logický celek a síťová struktura systému je pro něho transparentní. Uživatel tak může používat syntakticky shodné příkazy nad lokálními i vzdálenými daty. Na každém uzlu je instalován vlastní systém pro správu dat (SŘBD), který umožňuje (alespoň z části) běh služeb nezávisle na ostatních. Práce jednoho uzlu, který nepřetržitě pracuje, nenarušuje celý systém.
K výhodám distribuovaných systémů patří [6]:

- Zátěž je díky paralelismu rozdělena na více uzlů a tím je dosaženo vyšší výkonosti systému.
- Úspora síťové komunikace efektivním rozmístěním dat do míst, kde jsou nejčastěji využívána.
- Díky replikám větší spolehlivost a tolerance k chybám.
- Snadná škálovatelnost hardwaru (bude vysvětleno dále).

K nevýhodám patří:

- Větší složitost – návrhu databáze, katalogu dat a jeho správy, transakčního zpracování s problémy uváznutí, optimalizaci dotazů, územní distribucí dat, šíření aktualizace při replikaci, heterogennost dat na jednotlivých uzlech vede k transformacím, komunikace v počítačových sítích – minimalizace počtu a velikost zpráv, výpadky.
- Obtížnější dosažení bezpečnosti a utajení.
- Distribuce řízení.
- Relativní nedostatek standardů, nástrojů a zkušeností.

[4]Termínem cluster a grid se označují kolekce uzlů, které uživatelům poskytují sadu služeb. Navenek se jeví jako jeden superpočítač s tím rozdílem, že jejich výpočetní síla je postavena na spolupráci skupiny uzlů. Uzly v clusteru jsou propojeny lokální sítí (LAN) a hardwarově homogennější, kdežto v gridu jsou propojené počítače geograficky odlehlejší a jsou hardwarově heterogenní [29].

4 Hadoop a jeho komponenty

AH je sada nástrojů s otevřeným zdrojovým kódem (*opensource*) určených pro zpracování velkého množství strukturovaných i nestrukturovaných dat v distribuovaných úložištích – clusterech. Jeho počátky byly položeny v roce 2003 [7] společně s vývojem souborového systému „Google File System“ společností Google a projektem Nutch (webový fulltextový vyhledávač), od kterého byl v roce 2006 oddělen do samostatného projektu. Na jeho základě byly vytvořeny snadněji instalovatelné a administrovatelné distribuce vyvíjené zejména společnostmi Hortonworks, Cloudera[5] a MapR. Některé společnosti, jako například Microsoft nebo Amazon, nabízí AH jako cloudovou službu. Hortonworks nabízí svůj produkt jako plně otevřený, Cloudera a MapR ve funkčně omezených verzích zdarma příp. ve zpoplatněných enterprise verzích bez omezení. Microsoft rovněž nabízí na zkoušku svůj ekosystém s počátečním vkladem $100 na uživatele (pro studenty $200).
Mezi hlavní výhody systému založeného na AH patří:

- **rozšiřitelnost** – přidáním nového hardwaru lze snadno zvýšit výpočetní sílu a diskovou kapacitu systému,
- **odolnost** proti jeho pádu, neboť při výpadku uzlu/ů jsou data a výpočty automaticky přesměrovány na jiný uzel/jiné uzly clusteru,
- **opensource** licence základních i velké části rozšiřujících komponent,
- **nízké pořizovací náklady** hardwaru, který je založen na běžně dostupných komponentách.

Jádro frameworku AH napsané převážně v jazyce Java se skládá z následujících komponent:

- **Hadoop Common:** Obsahuje základní knihovny a funkce, které jsou využívány v ostatních modulech.
- **Hadoop Distributed File System (HDFS):** Distribuovaný souborový systém pro organizaci dat na „běžně dostupných“ počítačových sestavách.
- **Hadoop YARN:** Moduly pro správu zdrojů a plánovaní úloh v celém clusteru.
- **Hadoop MapReduce/Tez:** Implementace modelu MapReduce nad daty uloženými v HDFS, resp. jeho rychlejší řešení pomocí Tez frameworku.

Pro tuto práci jsou podstatné komponenty HDFS, YARN a MapReduce/Tez, které jsou popsány níže v textu. Popis se bude vztahovat k aktuální verzi **3.1.** a rozšířené verzi **2.7.**

[5]Společnosti Hortonworks a Cloudera oznámily na počátku roku 2019 své sloučení.

4.1 Souborový systém HDFS

Souborový systém HDFS navržen pro ukládání a zpracování velkých souborů v distribuovaném prostředí na „komoditním hardwaru[6]“. Lze ho provozovat na jednotkách až tisících uzlech (počítačích), kdy každý z nich má svou operační paměť, CPU, diskové úložiště a instalovaný operační systém. Je založen na platformě Java.

Mezi klíčové vlastnosti HDFS patří [8]:

- **Odolnost k chybám** – detekce chyb a jejich automatická oprava je nutností, neboť reálné nasazení Hadoop clusteru sestává z mnoha „komoditních“ počítačů, u kterých se selhání některé z komponent nelze vyhnout. HDFS proto replikuje stejná data na více uzlech, aby i v případě výpadku některého z nich, byla data dostupná.
- **Zpracování dat pomocí proudů** (*streaming*) – cílem je zjednodušení a zpřehlednění hromadných operací nad posloupnostmi prvků.
- **Zaměření na velké objemy dat** – clusteru by nemělo činit potíže zpracovat desítky milionů souborů, přičemž velikost každého souboru je typicky >0,5 GB.
- **Jednoduchá datová soudržnost** – pro přístup k datům HDFS používá model *Write Once / Read Often*, což znamená, že je zaměřen na rychlé čtení souborů z úložiště. Soubory mohou být zapsány pouze jednou, vyjma operací přidání dat na konec souboru (*append*) nebo zkrácení souboru (*truncate*), aby se eliminoval přenos dat sítí při replikacích. K dalším podporovaným operacím se soubory patří vytvoření souboru (*create*), mazání (*erase*), přejmenování souboru (*rename*) a změna atributů (*get/set attribute*).
- **Provádění výpočtu blízko dat** – cílem je minimalizace vytížení sítě a zvýšení propustnosti systému.
- Relativně **snadná přenositelnost** mezi různými platformami a pro heterogenní HW[7].

[6]Termín „komoditní hardware“ označuje HW jehož komponenty jsou obecně dostupné z více zdrojů a neobsahují proprietárního řešení od jednoho dodavatele. Co se týče HW specifikace pro jeden datanode nebo YARN manager, byla v roce 2014 doporučena [19] sestava se dvěma 6–8 jádrovými 3 GHz procesory, pamětí RAM 64–512 GB s ECC korekcí, 12–24x 1 až 4 TB SATA disky a gigabitovým ethernetovým připojením.

[7]Výchozí platformou je GNU/Linux. OS Windows již není nativně podporován, nicméně lze v něm stále některé aktuální komponenty AH zprovoznit, např. souborový systém HDFS.

4.1.1 Architektura HDFS

HDFS architektura je postavena na modelu master/slave, ve které se jednotlivé uzly dělí do dvou kategorií:

- **DataNode** (=klient/slave) a
- **NameNode** (=server/master).

NameNode spravuje jmenný prostor souborového systému, řídí přístupy k souborům a spravuje jejich metadata (místo uložení bloků dat, názvy souborů, replikační faktor a oprávnění uživatelů). Dále určuje umístění bloků ve spravovaných uzlech a zajišťuje souborové operace – otevření, zavření, přejmenování souborů a adresářů a změna jejich atributů.

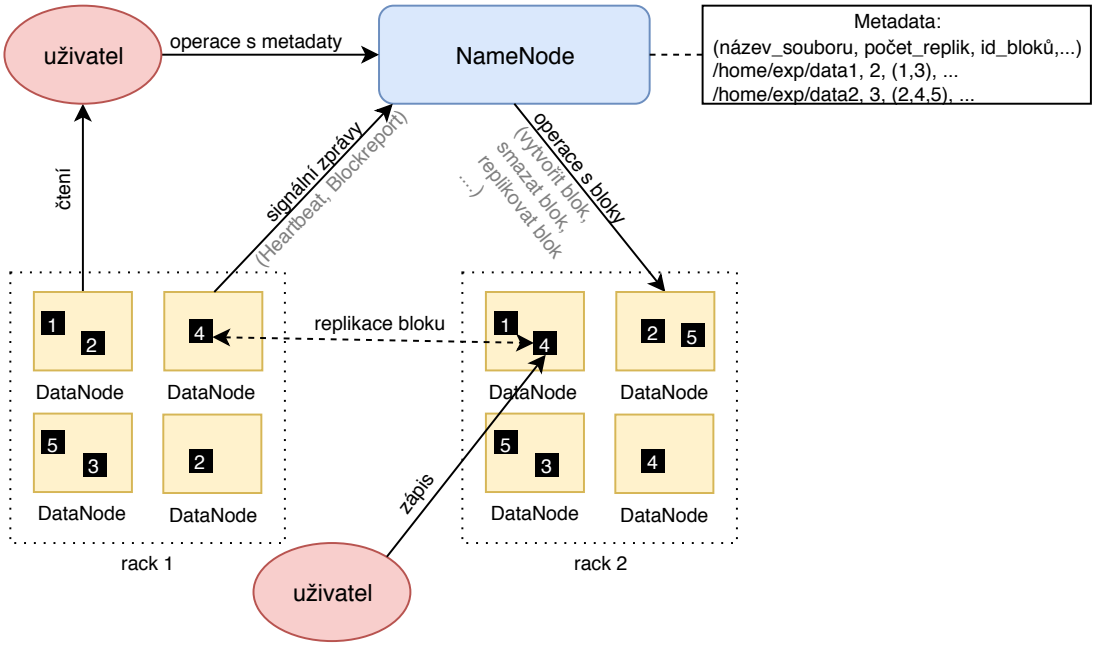
Využití NameNodu jako centrálního systémového katalogu má ale nevýhodu v tom, že se tento uzel stává nejužším článkem systému, neboť při jeho poruše havaruje celý systém. To se týkalo verze AH 1.x, od verze AH 2 byla přidána podpora alternativního pasivního NameNodu v jednom clusteru (sekundární NameNode, zkráceně SNameNode), který v případě poruchy výchozího aktivního NameNodu se stává tím hlavním a fungování HDFS pak není ohroženo. Aby se ještě zvýšila odolnost systému proti selhání, byla do verze AH 3 přidána podpora více pasivních NameNodů vjednom clusteru [9].

V uzlech kategorie **DataNode** jsou uložena vlastní data. Svazek disku je logicky rozdělen do bloků pevné velikosti, které jsou nejmenší adresovatelnou jednotkou z uzlu NameNode. Všechny bloky, vyjma posledního, mají stejnou velikost. Předdefinovaná velikost bloku je 64-128 MB podle verze AH[8] (pokud nebyla v konfiguračním souboru přenastavena na jinou vlastní hodnotu). Dělení na tyto logické bloky je postaveno nad existujícím souborovým systémem uzlu (např. ext3/4). Od verze AH 2.7 byla přidána podpora proměnlivé délky bloku, kdy nová data mohou být přidána do posledního nezaplněného bloku a tím zcela využít „kapacitu“ každého bloku. DataNody zajišťují čtení a zápis dat z/do souborového systému. Podle instrukcí NameNodu vytváří nebo maže bloky a provádí replikaci bloků (obr. 1). Jeden DataNode obvykle představuje jeden fyzický uzel v clusteru.

Jmenný prostor souborového systému podporuje tradiční hierarchickou organizaci do adresářů a složek. Jakákoliv změna položky nebo atributů je zaznamenána v NameNode uzlu.

4.1.2 Replikce bloků

Z důvodu minimalizace problémů při poruchách na jednom uzlu, které by způsobily nedostupnost lokálních souborů, je každý nově zapsaný blok v HDFS replikován podle nastavené hodnoty replikačního faktoru do dalších DataNodů. Hodnotu lze převzít z globálního nastavení nebo individuálně nastavit při vytvoření souboru, eventuálně upravit dodatečně. Replikaci bloků řídí NameNode. Ten v pravidelných intervalech přijímá a ukládá signální zprávy *Heartbeat* (defaultně



Obrázek 1: Architektura HDFS a replikace bloků.

3x za sekundu) a *Blockreport* (defaultně 1x za 6 hodin) od DataNodů z clusterů [10]. Vysíláním zpráv *Heartbeat* signalizuje DataNode svou správnou funkčnost a ve zprávách *Blockreport* zasílá uzlu NameNode seznam hostovaných datových bloků. Pokud do výchozího intervalu (5-10 minut) neobdrží NameNode zprávu *Heartbeat*, je DataNode považován za nefunkční a NameNode naplánuje replikaci bloků, které byly umístěny na nefunkčním DataNodu z ostatních DataNodů. Při umísťování bloků/replik v DataNodech je zohledňováno i jejich umístění v rámci racku[8] (*Rack Awereness*, [11]), aby v případě výpadku racku byla data dostupná i odjinud. Při replikačním faktoru = 3 jsou 2 bloky umístěny na dvou DataNodech v jednom racku (aby se zmenšil přenos dat mezi racky) a jedna replika na jiném racku. Replikace bloku na stejném DataNodu není povolena, proto je maximální počet replik roven celkovému počtu DataNodů v clusteru.

4.1.3 Čtení dat z HDFS

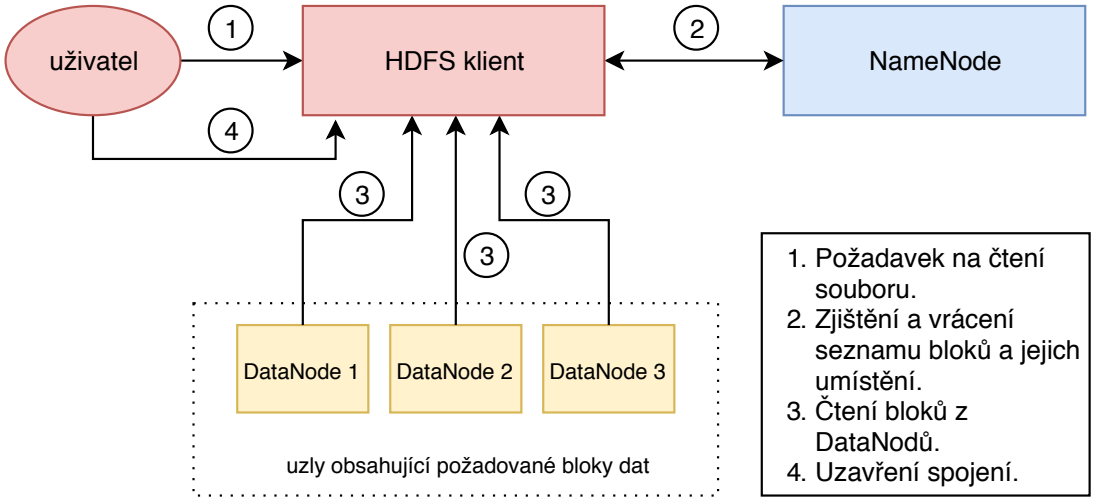
Při čtení souboru z HDFS (obr. 2) je sled událostí následující [12]:

1. Uživatel skrze klienta HDFS kontaktuje NameNode na umístění hledaného souboru.

2. Pokud má klient dostatečná oprávnění, NameNode mu z uložených metadat vrátí adresy uložených bloků hledaného souboru na DataNodech, které jsou nejblíže ke klientovi.

[8]Rack je ocelový rám umožňující instalaci různých prvků výpočetní techniky (např. routery, switche, servery, kabely).

3. Poté klient kontaktuje jednotlivé DataNody a paralelně načítá obsah boků z DataNodů.



Obrázek 2: Čtení dat v HDFS.

4.1.4 Zápis dat v HDFS

Při zápisu dat do HDFS je sled událostí následující [12]:

1. Uživatel skrze klienta HDFS pošle požadavek na zápis na NameNode.

2. NameNode provede ověření požadavku (např. zda již neexistuje stejný soubor, oprávnění uživatele k zápisu dat). Pokud kontrola proběhla v pořádku, NameNode vrátí odpověď se seznamem DataNodů pro zápis bloků souboru; v opačném případě nevytvoří nový soubor a klientovy vrátí výjimku (*IOException*).

3. Klient poté začne posílat data s žádostí o zápis prvnímu DataNodu ze seznamu. První DataNode začne po částech přijímat data, zapisuje do lokálního úložiště a odesílá je druhému DataNodu ze seznamu; ten je zapisuje do svého lokálního úložiště a odesílá dalšímu v řadě atd. Data se tak přenášejí od klienta pouze jednou.

4.2 MapReduce a Tez

MapReduce je programový model (paradigma) pro paralelní, distribuované a dávkové zpracování dat představený společností Google v roce 2004 [12]. Skládá se z metody *Map*, která provádí filtrování a třídění nad vstupními daty, a metody *Reduce*, která provádí spojování výsledků z předchozí fáze a sestavuje souhrnný výsledek. Je inspirováno procedurami *Map* a *Reduce* přítomnými ve funkcionálních jazycích, nicméně ve srovnání s nimi není při jedno vláknové implementaci

tolik výkonná. Jeho výhoda se projeví až při paralelním řešení úloh ve více vláknech na vícejádrovém hardwaru, u kterých je zátěž rozložena mezi více strojů v clusteru či gridu[9].

Řešení úloh pomocí MapReduce je možné rozdělit do pěti kroků:

1. Načtení vstupu a příprava dat – z úložiště dat (např. HDFS) jsou načítána vstupní data, rozdělena na menší části (např. po řádcích, slovech apod.) a přeposílána zpracovávajícím Map procesům.

2. Vykonávání kódu Map funkce – podle kódu funkce jsou mapovány (převáděny) vstupní dvojice klíčů a hodnot na seznam výstupních klíčů a dočasných hodnot. Formálně:

$$Map(in_{key}, in_{value}) \rightarrow list(out_{key}, temporary_{value})$$

3. Promíchání a seřazení – výstupní seznamy dvojic z předchozí fáze jsou rozdělovány, dvojice podle klíče seřazeny a seskupeny v nové dvojice. Výstupem je index funkce Reduce, která bude daný klíč zpracovávat.

4. Vykonávaní kódu Reduce funkce – zkombinuje/zredukuje všechny dočasné hodnoty pro každý klíč podle kódu Reduce funkce. Formálně:

$$Reduce(out_{key}, list(temporary_{value})) \rightarrow list(out_{value})$$

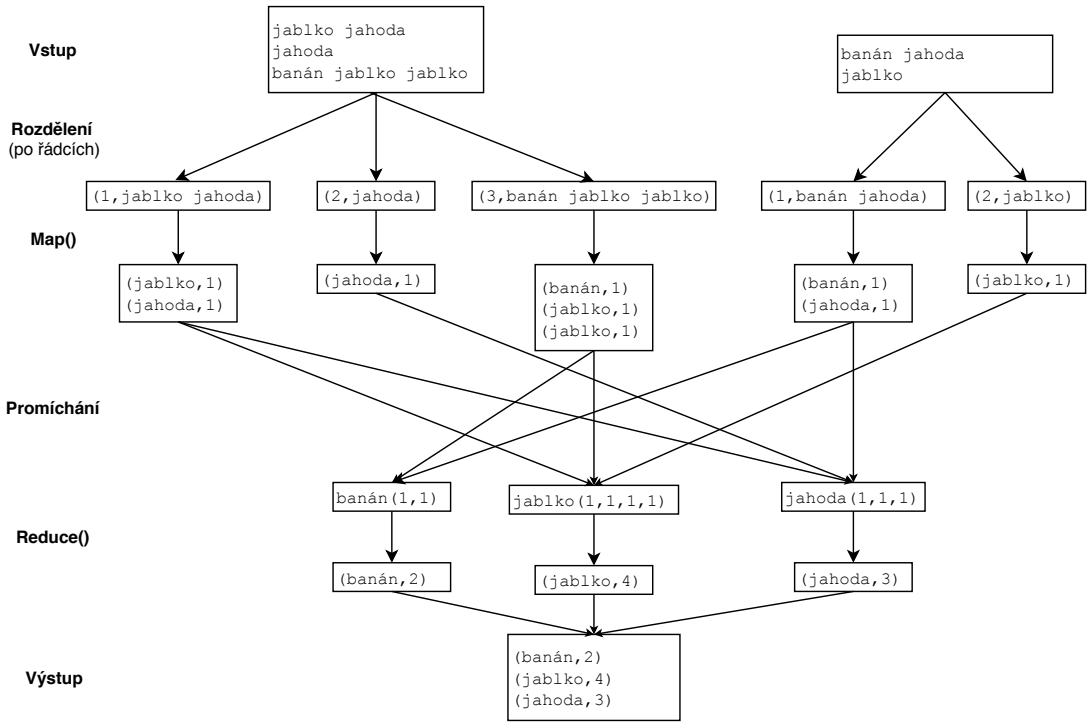
   Každé volání Reduce funkce obvykle vrátí jednu výstupní hodnotu nebo prázdný výsledek, ačkoliv je možné vrátit v jednom volání i seznam hodnot.

5. Zápis výsledku do úložiště.

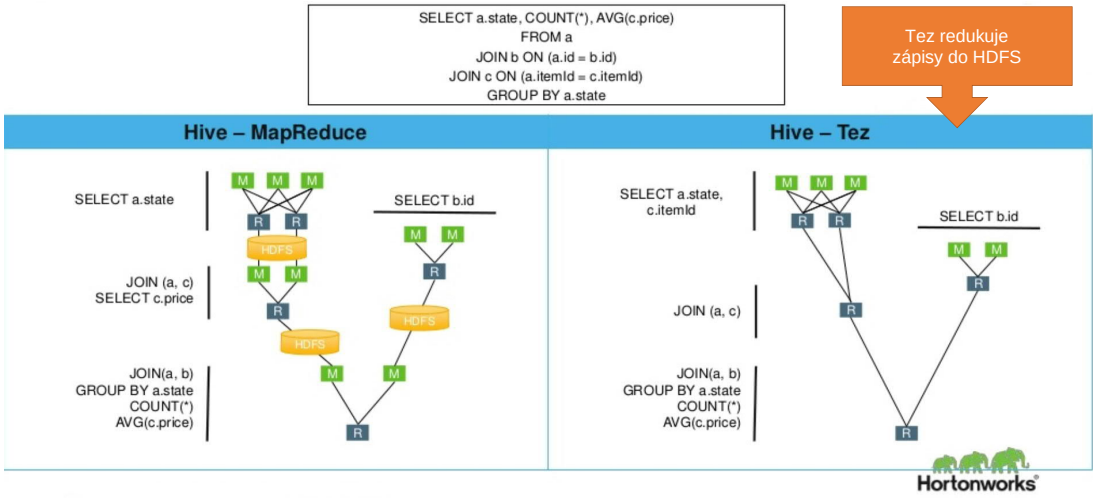
Posloupnost kroků je možné demonstrovat na příkladu zjištění počtu stejných slov, který je graficky zachycen na obr. 3.

AH vytváří rámec pro spuštění MapReduce úloh, které jsou primárně napsané v Javě (pomocí nástroje *Hadoop Streaming* lze použít i jiný programovací jazyk, např. Ruby nebo Python). Hlavní výhodou implementace MapReduce modelu v AH je, že úlohy mohou být prováděny na stejném fyzickém uzlu, na kterém jsou v HDFS uložena data. To významně snižuje síťové I/O operace a ponechává většinu I/O operací na lokálním disku nebo ve stejném racku [13]. Apache Tez představuje alternativu k dávkově orientovanému výpočetnímu modelu MapReduce, který má značně urychlit zpracování dat a posunout analýzu dat v AH k interaktivnímu modelu. Při zpracování úlohy v MapReduce se fáze Map a Reduce vždy střídají a jejich mezivýpočty jsou ukládány do HDFS a následně opět čteny, což zpomaluje dosažení výsledku, kdežto v Tez může několik Reduce fází následovat po sobě a mezivýsledky jsou drženy v operační paměti (obr. 4). Tez je implementován do AH od verze 2, od verze 3 je výchozím přístupem výpočtu pro vykonávání dotazu HiveQL v Hive.

[9]Termínem cluster a grid se označuje kolekce počítačů, která uživatelům poskytuje sadu služeb. Navenek se může tvářit jako jeden superpočítač s tím rozdílem, že jeho výpočetní



Obrázek 3: Počet výskytů slov pomocí úlohy MapReduce.



Obrázek 4: Zpracování stejného SQL dotazu modelem MapReduce a Tez [14].

4.3 YARN

YARN (*Yet Another Resource Negotiator*), jako další součást jádra AH, je modul pro plánování úloh a správu zdrojů celého clusteru. Byl zaveden jako rozšíření

síla je postavena na spolupráci skupiny počítačů. Počítače v clusteru jsou propojeny lokální sítí (LAN) a hardwarově homogennější, kdežto v gridu jsou propojeny počítače geograficky odlehlejší a jsou hardwarově heterogenní [29].

MapReduce paradigmatu, avšak podporuje i jiné způsoby zpracování dat (grafové, proudové, aj.; obr. 5). Jeho hlavní výhodou je spouštění různých výpočtových modelů na stejném hardwaru a nad jedněmi daty z úložné vrstvy.

Fungování YARN modulu je postaveno na komunikaci:

- master démonu[10] „Resorce Manager“,
- slave démonu „Node Manager“,
- a procesu „Application Master“ (jeden na běžící aplikaci).

Resource Manager (správce prostředků) je centrální autoritou, která přiděluje na globální úrovni zdroje (CPU, paměť) všem aplikacím, které mezi sebou o ně soupeří [15]. Běží obvykle na NameNodu. Skládá se ze dvou komponent a to plánovače (*Scheduler*) a správce aplikací (*Application Manager*). Plánovač alokuje zdroje běžícím aplikacím (způsobem FIFO, spravedlivě rovným dílem nebo dle oprávnění uživatele). Správce aplikací je zodpovědný za (re)startování a monitorování procesů Application Master na uzlech.

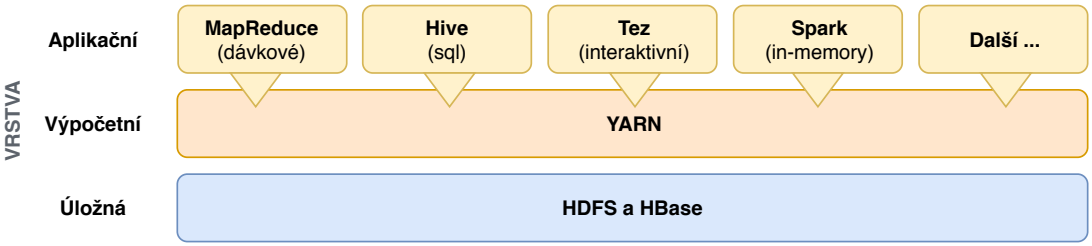
Slave démon **Node Manager** (správce uzlů) běží v jedné instanci na každém podřízeném uzlu (slave node). Monitoruje a spravuje úlohy uživatelů a zasílá jejich přehled Resource Manageru [15].

Proces **Application Master** je spuštěn pro každou aplikaci samostatně a řídí její životní cyklus. Běží v kontejnerech[11], které na vyžádání povoluje Resource Manager.

Proces spuštění aplikace v YARN začíná kontaktováním správce prostředků (*Resource Manager*) klientem a požádáním o spuštění hlavního procesu aplikace (*Application Master*). Správce prostředků poté vyhledá správce uzlů (*Node Manager*), který může spustit aplikaci v kontejneru. Aplikace poté provede úlohu podle svého vnitřního kódu (např. provede výpočet a vrátí výsledek), nebo požádá o další zdroje správce prostředků.

[10]Démon je označení programu, který je spuštěn dlouhodobě a není v přímém kontaktu s uživatelem. Jeho úkolem je vyčkávat v nečinnosti na nějakou událost a tu posléze bez nutnosti interakce s uživatelem obsloužit [27].

[11]YARN kontejner je označení pro úspěšné přidělení zdrojů (např. CPU, RAM) konkrétního uzlu Resource Managerem [28]. Na jednom uzlu může běžet několik kontejnerů současně.



Obrázek 5: Zapojení YARN modulu ve zpracování úlohy.

5 Další komponenty systému Hadoop

Komponent (projektů) založených na AH existuje celá řada. Většina je podporována organizací Apache Software Foundation[12]. Pro tuto práci jsou podstatné Hive a HBase, které jsou zaměřeny na práci s databázemi a datovými sklady, proto jsou blíže popsány dále v textu. Některé další komponenty jsou souhrnně popsány v poslední podkapitole.

5.1 Apache Hive

Apache Hive je sada nástrojů určených pro dotazování, sumarizaci a analýzu nad velkými datovými sadami v distribuovaných úložištích pomocí jazyka HiveQL (*Hive Query Language*). Představuje softwarovou infrastrukturu pro budování datových skladů v prostředí AH.

Příkazy HiveQL, které z velké míry kopírují standard SQL-92, jsou kompilátorem překládány do MapeReduce (do verze Hive 3), Tez (výchozí překlad od verze Hive 3) nebo Spark úloh a dále postupovány Hadoop frameworku ke zpracování. Překlad zvyšuje čas odezvy systému, na druhou stranu ale zkracuje čas potřebný na vývoj úloh, které nemusejí být zdlouhavě psány jako MapReduce úlohy. Hive se hodí pro dávkové zpracování dat a v případě využití Tez pro interaktivní, naopak není vhodný na zpracování příkazů v reálném čase (*real-time*). Hive podporuje tvorbu a znovu použití uživatelsky definovaných funkcí (UDF), které rozšiřují jazyk HiveQL.

5.1.1 Architektura Hive

Architektura Hive [16] se skládá z těchto komponent:

1. **Metastore** – je centrální úložiště metadat, ve které se ukládají informace o struktuře tabulek, relacích, databázích a jejich místě uložení. Metadata jsou ukládána v relačních databázích. Při instalaci služby Hive má administrátor možnost volby z Derby, MySQL, MS SQL Server, Oracle nebo PostgreSQL. Aplikací informací z metastoru získávají uložená (volně strukturovaná) data formu tabulek obvyklých z RDBMS.

2. **Uživatelského rozhraní** – pro interakci uživatele s daty v úložišti (např. HDFS/HBase) nabízí Hive přístup přes webové rozhraní prohlížeče (v distribuci Ambari do verze 3 „Shell In A Box“), příkazovou řádku Hive CLI/Beeline, nebo pomocí HDInsight v prostředí služby Microsoft Azure. Hive CLI komunikuje přímo s Metastorem, zatímco Beeline se připojuje k HiveServer2, který komunikuje s Metastorem. Proto při použití Hive CLI musí mít uzel přímý přístup k Metastore, což není bezpečné a komplikuje to nastavování přístupových práv k datům. Při použití Beeline se klient připojuje k HiveServer2, který komunikuje s Metastorem. Práva k tabulkám

[12] https://www.apache.org/foundation/

lze nastavit přes Apache Ranger, což je Framework pro zabezpečení dat v Hadoopu.

3. **Řadiče (driver)** – přijímá HiveQL příkazy, zahajuje jejich zpracování vytvořením relací a monitoruje jejich životní cyklus. Dále ukládá metadata vzniklá během vykonávání HiveQL příkazů a sbírá výsledky po fázi Reduce.

4. **Kompilátoru** – provádí ověření syntaxe příkazu a jeho převedení na prováděcí plán. Ten obsahuje sled kroků nutných pro vykonání příkazu MapReduce paradigmatem nebo Tez úloh. Dále komunikuje s Metastorem, odkud získává metadata nezbytná pro ověření proveditelnosti příkazu (např. dotazováním na existenci požadované tabulky z příkazu).

5. **Execution Engine** – v této úrovni dochází k předání kompilovaných a optimalizovaných HiveQL dotazů jako MapReduce/Tez úloh modulu YARN. Po obdržení výsledku z YARN modulu jsou předány řadiči, který je předá zpět uživatelskému rozhraní.

6. **Úložiště HDFS/HBase**.

K datům spravovaným v Hive je možné se také připojit aplikacemi skrze ODBC/JDBC rozhraní nebo skrze univerzální platformu Apache Thrift.

5.1.2 Datový model

Organizace dat v Hive je členěna do tří složek: **Tabulky, oddíly a přihrádky**.

1. **Tabulky** – podobně jako v relačních databázích, slouží pro uložení logicky souvisejících dat. Struktura tabulek a další metadata jsou uložena odděleně od vlastních dat v relační databázi Metastoru. Tabulky mohou být dvojího typu:

   - **Externí tabulka** – systém pracuje s tabulkou v původním umístění, které může být lokální nebo vzdálené (např. HDFS/Azure Storage Volumes), a nepřesouvá ji do interního úložiště. Při odstranění tabulky Hive maže pouze metadata a datový obsah ponechává v úložišti.
   - **Interně spravovaná (managed) tabulka** – Hive ukládá jejich obsah do jím spravovaného systému a řídí její životní cyklus. Výchozím adresářem pro ukládání interních tabulek je v AH 3.1:

     `/warehouse/tablespace/managed/dtb_name.db/tbl_name`

     Při mazání interní tabulky odstraní data i metadata.

2. **Oddíly, neboli partitions,** jsou menší bloky dat vzniklé dělením „velké“ tabulky, do kterých se ukládají související data. Kritériem je hodnota z jednoho nebo více sloupců. Rozdělování probíhá automaticky, pokud bylo definováno při vytvoření tabulky. Pro každý oddíl je vytvořen samostatný podadresář tabulkového adresáře v této struktuře:

`.../table_name/partition_column=hodnota`

Výhoda rozdělování do oddílů spočívá v tom, že se zrychluje zpracování dat v MapReduce/Tez úlohách, neboť ty přistupují pouze k části tabulky (oddílu), která tato data obsahuje, a možnost mazání bloků dat (DROP) po celých oddílech.

3. **Přihrádky, neboli buckets,** jsou další možností, jak rozdělit tabulky nebo oddíly na menší podsoubory, aby se zvýšila rychlost dotazů. Rozdělení se provádí podle hašovací funkce nad sloupci tabulky, která byla definována při vytvoření tabulky. Přihrádky jsou uloženy jako soubory v adresáři tabulky. Na rozdíl od oddílů, které mohou mít ve výsledku různou velikost souborů, jsou přihrádky přibližně stejně velké.

Pro konkrétní představu výše popsané organizace dat v Hive je v HiveQL zdrojovém kódu 1 uveden příklad vytvoření interní tabulky.

```
1  -- HiveQL
2  -- vytvoření interní tabulky
3  CREATE TABLE gps_position (
4    x INT,
5    y INT,
6    year_ INT,
7    month INT,
8    day INT,
9    description VARCHAR(64))
10 -- dělení tabulky do~oddílů podle roku
11 PARTITIONED BY (
12   year INT)
13 -- dělení dat v~oddílů do~12 ti přihrádek podle měsíců
14 CLUSTERED BY (
15   month)
16 -- seřazení lokalizací v~přihrádkách podle sloupců x a~y
17 SORTED BY (
18   x,
19   y)
20 INTO 12 BUCKETS
21 -- hodnoty budou odděleny tabulátorem
22 ROW FORMAT DELIMITED
23 FIELDS TERMINATED BY '\t'
24 -- řádky ukončeny znakem nového řádku \n
25 LINES TERMINATED BY '\n'
26 -- tabulka bude uložena v~textovém formátu
27 STORED AS textfile;
```

Zdrojový kód 1: Příklad vytvoření interní tabulky v HiveQL.

Názvy sloupců, tabulek a složek spravovaných (managed) Hivem (tzn. netýká se externích tabulek) jsou automaticky překonvertovány na malá písmena. Hive

je *case-insensitive* a nezáleží tedy na velikosti znaků v názvech sloupců nebo tabulek.

5.1.3 Souborové formáty

Tabulky v Hive jsou uloženy jako datové soubory v úložišti (HDFS) a Hive podporuje práci s různými souborovými formáty. Ten se specifikuje klauzulí `STORED AS` při vytváření tabulky. Formáty lze rozdělit podle vhodnosti na práci se sloupcově orientovanými tabulkami, jako jsou RC, ORC, Parquet a řádkově orientovanými tabulkami, jako jsou Avro, sekvenční a textové. Sloupcově orientované (jeden sloupec je množinou řádků) jsou vhodnější na rychlé analytické zpracování, řádkově orientované jsou vhodnější pro transakční zpracování.

U nově vytvořených i existujících tabulek všech formátů Hive Metastore automaticky udržuje jejich základní statistiky, ke kterým patří:

- počet oddílů, ze kterých je tvořena datová sada,
- počet souborů, ze kterých je tvořena datová sada,
- celková velikost datové sady v souborovém systému,
- celková velikost datové sady nekomprimovaná,
- počet řádku datové sady.

V kapitole 6.5 Optimální souborový formát tabulek je provedeno srovnání výše uvedených souborových formátů nad daty použitých v této práci.

5.1.4 Databázová transakce

Databázovými transakcemi se označují příkazy, které převedou databázi z jednoho konzistentního stavu do druhého. Transakce musí splňovat tzv. vlastnosti ACID:

- Atomicita (angl. *Atomicity*, A) - neboli nedělitelnost znamená, že daná transakce se provede najednou, nemůže být přerušena něčím jiným a později dokončena.
- Konzistence (angl. *Consistency*, C) – transakce nesmí narušit integritní omezení.
- Izolovanost (angl. *Isolation*, I) – při současném vykonávání více transakcí nedochází k jejich vzájemnému ovlivňování.
- Trvalost (angl. *Durability*, D) – výsledek úspěšně provedené transakce je trvale uložen v databázi.

HDFS je postaven na principu *Write once, Read many* a proto ve starších verzích Hivu bylo jedinou možností změny dat (*UPDATE*) smazání celého oddílu a jeho znovuvytvoření vložením nového oddílu se změněnými daty [17]. Stejný proces musel být absolvován i pro operaci *DELETE* v případě mazání jednotlivých záznamů. Hive od verze 0.14[13] umožňuje transakční zpracování dotazů *INSERT/UPDATE/DELETE* i pro jednotlivé záznamy (řádky) a s podporou ACID vlastností. Tabulky ale musí být interní (*managed*), měly by být rozděleny do přihrádek (povinnost je pro Hive v. 2 <) a musí být uloženy v ORC formátu. Operace *BEGIN*, *COMMIT* a *ROLLBACK* nejsou podporovány, prováděné operace jsou automaticky ukončovány a ukládány. V případě aktualizace záznamů operací *UPDATE* je upravovaný řádek uzamčen pro jiné aplikace, jiné části tabulky ve stejném oddílu jsou ale přístupné ke čtení [18].

5.1.5 Srovnání s tradiční RDBMS

Mezi hlavní rozdíly Hive a tradičního relačního databázového modelu (RDBMS) patří:

Hive	**RDBMS**
Zpracování velkých objemů dat - PB	Zpracování řádově TB dat.
Zpracování v clusteru se snadným horizontálním škálováním HW a distribuovanými výpočty na mnoha uzlech.	Škálování HW složité, distribuované zpracování dotazů omezené.
Vhodný na analýzu nad statickými objemnými daty – zápis jedenkrát, ale opakované čtení (Write once, Read many).	Rychlý pro vyhodnocování dynamicky se měnících dat, které vyžadují opakované čtení i zápisy (Read and Write many times).
Vhodný na analytické zpracování dat (OLAP). Přestože podporuje transakce, není s ohledem na rychlost vhodný pro transakční zpracování dat (OLTP).	Zvládá OLAP i OLTP.
Obecné indexování tabulek od v. Hive 3 není podporováno. Namísto nich podporuje materializované pohledy a selektivní prohledávání v souborových formátech Parquet a ORC pomocí tří úrovňového indexování (statistik): nad souborem, skupinou sloupců (strip) a skupinou řádků (=10.000).	Pro rychlejší zpracování podporuje indexování i materializované pohledy.

[13]Verze 0.14 byla vydána 12.12.2014.

Je spíše infrastrukturou nad rozdílnými souborovými formáty s volnější datovou strukturou. Manipulace s daty skrze HiveQL.	Pracuje s databází s jasně strukturovanými daty a širokou podporou jazyka SQL.

5.2 Apache HBase

Apache HBase je open-source, distribuovaná, verzovaná a nerelační databáze postavená nad HDFS úložištěm. Je vhodná pro operace čtení/zápisu objemných semi/strukturovaných dat s náhodným přístupem v reálném čase [19]. Jak je obvyklé u produktů AH, běží v distribuovaném prostředí výpočetních prostředků a vykazuje velkou míru odolnosti proti výpadkům jednotlivých uzlů. Výchozím programovacím jazykem je Java (pomocí knihovny *HappyBase*[14] je možné využít i Python). Na rozdíl od *Apache Hive* nemá přímou podporu SQL, lze ji ale zavést skrze nadstavbu *Apache Phoenix*[15].

5.2.1 Architektura HBase

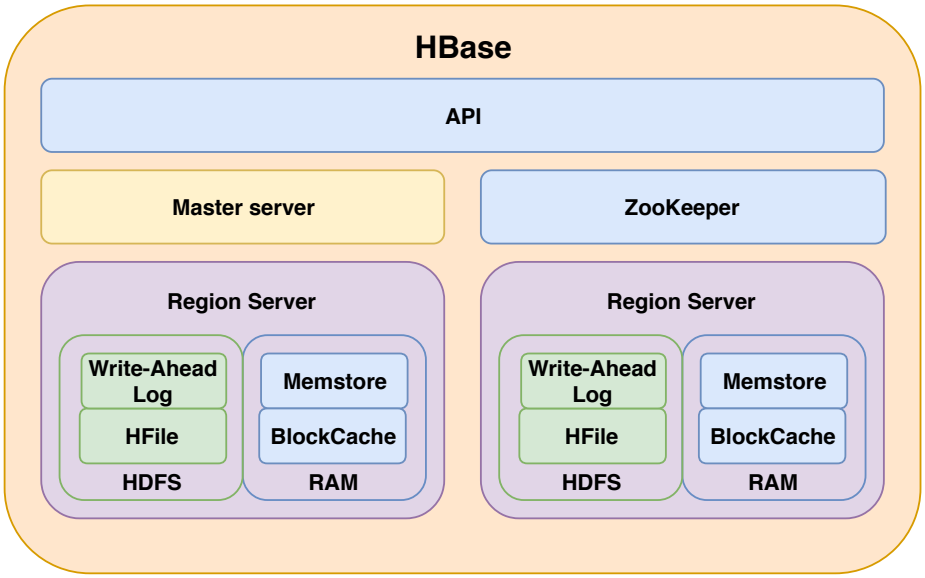
HBase je založena na master/slave architektuře a skládá se z těchto 3 hlavních komponent (obr. 6):

1. **Java API.**
2. **Jeden master server.**
3. **Více region serverů.**

Java API je komunikačním nástrojem s HBase. Dělí se na administrační a klientskou část. Do administrační spadají metody pro vytváření a mazání tabulek, mazání sloupců, přidávání rodin sloupců a úprav atributů tabulek [20]. Klientská část zahrnuje operace přidávání, úprav a mazání řádků a jednotlivých sloupců (kvalifikátorů) a spočívá především na metodách *Get*, *Put* a *Scan*. *Get* a *Put* pracují s řádky jako svým parametrem, metoda *Scan* je vykonávána nad posloupností řádků vymezenou od počátečního do koncového klíče řádku nebo (pokud není definován počátek a konec) nad celou tabulkou.

Master server je zodpovědný za přidělování regionů region serverům a řízení operací vytváření, úprav a mazání tabulek. Dále monitoruje provoz na region serverech v clusteru tak, aby byla data dostupná a zátěž na region serverech byla vyvážená. K tomuto úkolu master server používá koordinační službu ZooKeeper[16]. Tato služba poskytuje master serveru přehled o dostupných region serverech a jejich chybách.

[14] https://happybase.readthedocs.io/en/latest/
[15] https://phoenix.apache.org/
[16] https://zookeeper.apache.org/



Obrázek 6: Komponenty systému HBase.

Region servery jsou uzly v clusteru, na kterých jsou ukládány datové regiony. Jeden region je uložen na jednom region serveru a na každém z těchto serverů je uloženo více regionů (obvyklá výchozí velikost jednoho regionu je 128–256 MB) [21]. Datové regiony sestávají z menších souborů (64 kB) označovaných jako *HFile*. Při zápisu dat se nejprve zapíše informace do logovacího souboru *Write-Ahead Log*, který slouží pro obnovu dat v případě selhání zápisu. Dále jsou data zapsána do cach paměti v RAM MemStore a poté je proveden zápis do HFile souborů v HDFS úložišti DataNode. Při čtení dat se nejprve prohledává MemStore a BlockCache (místo v RAM s často čtenými daty), pokud nejsou nalezeny, dochází ke čtení z pevného úložiště.

Na každém region serveru funguje klient *Zookeeper*, který zasíláním zpráv informuje centrální *Zookeeper* o svém aktuálním funkčním stavu.

5.2.2 Datový model

Datový model HBase je odvozen z popisu Bigtable společnosti Google [22]. Ten umožňuje více způsobů reprezentace uložených dat [23]. Jeden z nich staví na připodobnění organizace dat s relačními databázemi a pracuje s těmito pojmy:

1. **Tabulka** - data v HBase jsou uspořádány do pojmenovaných tabulek.
2. **Řádek** – tabulky jsou tvořeny z řádků, kdy každý řádek je jednoznačně identifikovatelný podle klíče řádku. Klíč nemá definován datový typ a hodnoty v něm jsou uloženy jako pole bytů. Podle tohoto klíče (pořadí bytů v něm) je tabulka seřazena (HBase nepodporuje sekundární indexy nad jinými sloupci) [19].

3. **Rodina sloupců** – data v řádku jsou seskupeny do tzv. rodin (anglicky column families). Ty jsou specifikovány na počátku a za chodu je lze obtížně měnit. Každý řádek má stejnou sadu rodin sloupců, ačkoliv nemusí mít pro každou z nich uloženu nějakou hodnotu (podpora řídkých tabulek). Rodinám sloupců lze specifikovat Time To Live hodnotu v sekundách, podle které HBase automaticky odstraní řádky po jejich expiraci. Dále se na úrovni rodin sloupců specifikuje celkový počet verzí hodnot v buňce.

4. **Sloupcový kvalifikátor** – hodnoty uvnitř rodiny sloupců jsou dále adresovaná přes svůj sloupec. Ty mohou být mezi řádky jedné tabulky různé. Podobně jako u rodin sloupců, není u nich specifikován datový typ. Na rozdíl od rodin sloupců lze jednotlivý sloupec vytvářet za chodu na žádost klienta/aplikace. Název sloupce je složen z názvu rodiny, oddělovače „:“ a názvu sloupce (např. *obsah:img*).

5. **Buňka** – kombinací klíče řádku, rodiny sloupce a sloupcového kvalifikátoru je jednoznačně identifikována buňka tabulky.

6. **Časové razítko** – k hodnotám uvnitř buňky je přidruženo časové razítko, která ve výchozí konfiguraci obsahuje čas (v milisekundách) vložení konkrétní hodnoty do buňky a využívá se při verzování.

Další pohled na strukturu dat v HBase pojímá uložené hodnoty jako víceúrovňové seznamy (někdy též označovaný jako asociativní pole). Jednotlivé úrovně jsou (1) klíč řádku, (2) rodina sloupců, (3) sloupcový kvalifikátor, (4) časové razítko. Níže je jeden řádek tabulky HBase zobrazený jako víceúrovňový seznam.

```
{
"001": // (1) klíč
    {
    "obsah": // (2) rodina sloupců
        {
        "img": // (3) sloupcový kvalifikátor
              {
              "1542894158":<<soubor>> // (4) čas. razítko
              "1542895555":<<soubor>> // a hodnota buňky
              }
        }
    "info":  // (2) rodina sloupců
        {
        "format":  // (3) sloupcový kvalifikátor
              {
              "1542894158": "jpeg" // (4)
              }
        "geo":  // (3) sloupcový kvalifikátor
              {
```

```
                "1542894158": "49.8,17.2" // (4)
                }
            }
        }
}
```

Další možností reprezentace uložených dat v HBase je ve tvaru klíč-hodnota. Klíčem k získání konkrétní hodnoty buňky je pak seznam složený z prvků:

(klíč řádku, rodina sloupců, sloupcový kvalifikátor, časové razítko)

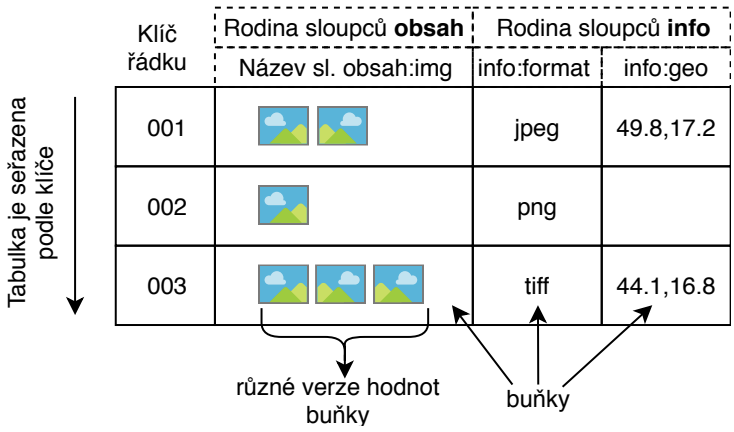
Např. pro získání souřadnic fotografe z obr. 7 by se použil klíč ve tvaru:

```
{001, info, geo, 1542894158} => {49.8,17.2}
```

V případě potřeby vrácení více sloupců řádku lze seznam zkrátit zprava, např.:

```
{001, info, geo} => {{1542894158:49.8,17.2},
{1542894158:44.1,16.8}}
```

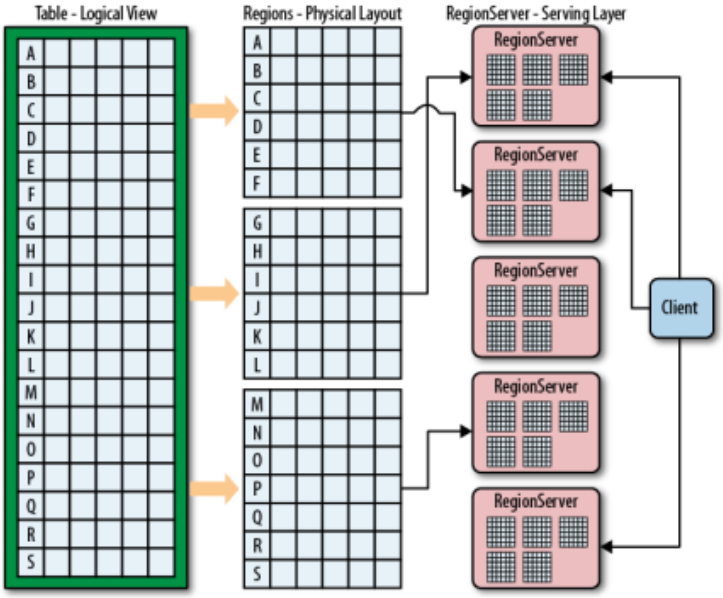
Pro vrácení dat celého řádku by pak postačil např. klíč: *{002}*



Obrázek 7: Příklad reprezentace datového modelu „klíč-hodnota“ HBase v podobě tabulky [19].

5.2.3 Distribuce dat

Základní technikou pro zajištění škálovatelnosti a vyvažování zátěže v distribuovaném prostředí HBase je dělení tabulek na menší části (*sharding*), které se označují jako regiony. Regionem je souvislá seřazená posloupnost řádků (záznamů) uložená na jednom uzlu. Na počátku je tabulka v jednom regionu a jak se postupným přidáváním dat zvětšuje, systém ji automaticky rozdělí na dva přibližně stejně velké regiony, aby nebyla překročena předefinovaná maximální velikost ([19]. Současně platí, že pro daný řádek zůstanou sloupce z jedné rodiny uloženy na jednom uzlu v souboru označovaným jako HFiles, pro která jsou data uložena jako páry klíč-hodnota. Na obrázku 8 je ukázán rozdíl mezi logickou a fyzickou strukturou uložených dat v HBase.



Obrázek 8: Logická a fyzická struktura řádků v regionech HBase [24].

5.3 Ostatní komponenty

Existuje řada dalších komponent, které je možné využít pro zpracování velkých objemů dat v distribuovaném prostředí AH. Většina je vyvíjena pod hlavičkou Apache Software Foundation. Níže uvádím vybrané opensource komponenty se stručným popisem jejich využití.

Kudu

Apache Kudu je kompromisem mezi čistě souborově orientovaným, pro sekvenční čtení dat vhodným, distribuovaným souborovým systémem (HDFS) a pro přímé přístupy vhodným HBase úložištěm. Jde o samostatné úložiště, které uchovává sloupcově uložená data v relačních tabulkách a ty dělí na menší části nazvané tablety (tablets), které umožňují paralelní uložení i zpracování dat. Záznamy jsou pak uloženy v tabletu definovaném na základě rozsahu nebo hashe primárního klíče.

Pig

Apache Pig je určen pro dávkové zpracování a transformaci semi-strukturovaných dat. Skládá se z vlastního jazyka Pig Latin, ve kterém je zapsán kód příkazů, a kompilátoru, který překládá kód do posloupnosti MapReduce/Tez úloh. Pig obsahuje sadu vestavěných funkcí pro manipulaci s daty jako s tabulkami, např. filtrování, spojování, řazení. Dále umožňuje tvorbu uživatelsky definovaných funkcí podobně jako v Hive.

Impala

Apache Imapala je dotazovacím SQL enginem a analytickou databází pro

Apache Hadoop. Přináší škálovatelné databázové techniky, které umožňují uživateli používat SQL dotazy s nízkou latencí nad daty uloženými v HDFS a HBase. Díky tomu, že Impala používá stejné soubory a datové formáty, metadata atd. jako již existující ostatní Hadoop frameworky (např. Hive, Pig), není potřeba přesunů dat do jiného frameworku nebo jejich transformací do jiného formátu [25].

Kafka

Apache Kafka je open-source platforma pro streamování dat. Disponuje vysokou propustností, nízkou latencí a masivní škálovatelností. Využití této platformy je široké – od sběrnice zpráv, přes monitoring všeho druhu, extenzivní sledování uživatelů na webu až například po agregaci logů. Platforma původně vznikla ve firmě LinkedIn, přes Apache Incubator se dostala do hlavních projektů Apache Software Foundation a do praxe jí nejvíce pomohla firma Confluent, založená několika vývojáři platformy [26].

6 Použitý Hadoop cluster

V této kapitole popisuji způsob hledání, a nakonec i nalezení vhodného technického řešení pro provoz frameworku Hadoop. Mezi kritéria výběru patřily co nejnižší (ideálně nulové) provozní náklady na potřebné výpočty, běh aktuální verze frameworku, dostupnost požadovaných služeb (Hive, Beeline, Tez, ...), kapacita minimálně stovek GB a možnost operativního přizpůsobování clusteru svým požadavkům (horizontální škálování, změna výchozího nastavení apod.).

6.1 Testování dostupných Hadoop clusterů

Původně jsem zamýšlel provozovat framework Hadoop ve virtualizační aplikaci Oracle VM VirtualBox 5.2.2 na platformě Windows 10 x64bit s konfigurací 16 GB RAM, 4 jádra CPU Intel i5 7200 2,71GHz, 500 GB SSD. Framework běžel v distribuci Hortonworks Data Platform v. 2.6.5[17] a v. 3.0 a Cloudera QuickStarts VM v 5.12.0. Výkon těchto řešení byl použitelný pro testování nad malými vzorky dat, řádově tabulky o stovkách záznamů. Při použití násobně větších objemů nebyly řešení použitelné, i jednoduché SQL dotazy se vykonávaly desítky hodin.

Proto jsem přešel k testování cloudového řešení frameworku Hadoop na platformě *Microsoft Azure*. Zde už byl výkon znatelně vyšší, problém byl v ceně předplatného za použité služby. To s výchozím testovacím sponzorovaným vložným $200 umožňovalo provoz frameworku Hive na maximálně 3 uzlech. Na více uzlů, nebo opakované počítání nad jednotkami GB dat cenový limit nestačil. Rovněž možnosti úprav výchozího nastavení uzlů bylo minimální. Proto jsem i tuto možnost opustil.

Na semináři *„Gridové počítání 2019“* konference e-infrastruktury Cesnet[18] jsem se seznámil se službami pro zpracování a ukládání dat v MetaCentru[19]. Ty jsou všem akademickým pracovníkům a studentům členů sdružení Cesnet poskytovány pro jejich výzkumné účely zdarma. Infrastruktura mj. zahrnuje Hadoop cluster o této HW specifikaci[20]:

- CPU: 24(+3) strojů x 16 jader x 2 hyperthreading (Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz),
- RAM: 24(+3) strojů x 128 GB,
- disk: 240 x 3.6 TB (data), 78 x 0.91 TB (OS),

a v této SW specifikaci:

- Hadoop 2.6.0,

[17] https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.5/bk_release-notes/content/comp_versions.html
[18] https://konference.cesnet.cz/
[19] https://metavo.metacentrum.cz/
[20] https://wiki.metacentrum.cz/wiki/Hadoop

- HBase 1.2.0,
- Hive 1.1.0,
- Hue 3.9.0,
- Pig 0.12.0,
- Spark 1.6.0.

Cluster jsem testoval, úlohy v Hive rozběhl, ale podle mé zkušenosti se systém hodí hlavně na déle trvající dávkové úlohy. Při snaze o změnu výpočetního enginu z MapReduce na Tez jsem neuspěl, použitá verze Hadoop 2.6.0 a Hive 1.1.0 z r. 2014 nepodporuje aktuální verzi *GIS Tools for Hadoop*, pouze funkčně omezenější, starší. Rovněž jsem nebyl úspěšný ve snaze uzpůsobit výchozí nastavení clusteru, a proto jsem nakonec i toto řešení opustil. Nakonec jsem pro účely této práce zvolil sestavení vlastního clusteru z virtuálních strojů cloudové infrastruktury MetaCentra/Cesnet, která (s nemalým úsilím na mé straně) splňovala očekávané požadavky definované v úvodu této kapitoly.

6.2 Použité řešení pro Hadoop clusteru

Všechny příklady uvedené v následující kapitole byly realizovány v cloudu MetCentra – MetaCloud. Jedná se o *IaaS* (Infrastruktura jako služba) a je obdobou známých IaaS cloudů, např. *Amazon Elastic Compute Cloud* (EC2), *Google Compute Engine*, *Microsoft Azure*. Je spravovaný pomocí softwarového rozhraní *OpenNebula* (v roce 2019 plánuje MetaCentrum přechod na *OpenStack*). MetaCloud mohou využít všichni uživatelé MetaCentra nebo e-Infrastruktury CESNET, tedy zaměstnanci a studenti vysokých škol a ústavů Akademie věd. Postačilo zaslat elektronickou přihlášku o členství v MetaCentru, vygenerovat si osobní *ssh klíč* na čelním uzlu[21] a požádat o zařazení do skupiny metacloud. Po schválení, které proběhlo obratem, se již bylo možné přihlásit do webového rozhraní *OpenNebula* (obr. 9) a provést spuštění předpřipraveného virtuálního stroje.

Operační systém jsem pro všechny stroje vybral *Cent OS Linux 7 x86-64*, který patří mezi podporované systémy *Hortonworks Data Platform* (HDP). Další specifikace zvolených virtuálních strojů je uvedena v tabulce 2.

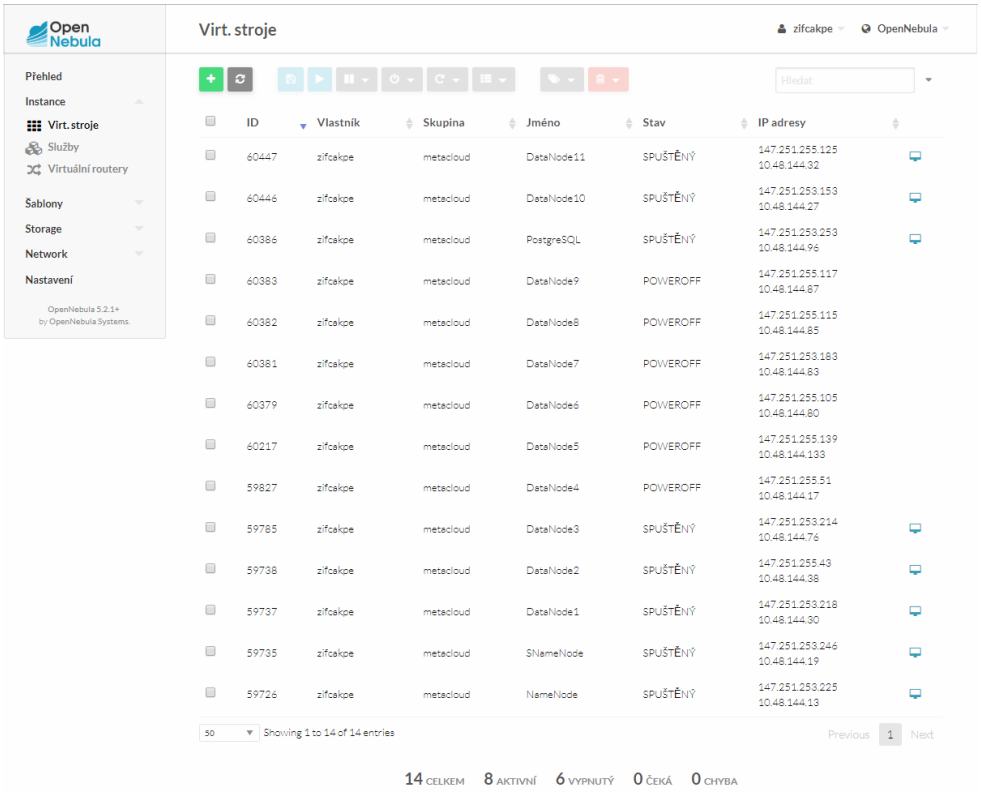
Parametry strojů v podobě velikosti úložného prostoru, počtu jader nebo velikosti paměti RAM jsou volitelné. Nicméně vyhrazení žádaných prostředků není okamžité, ale je závislé na jejich dostupnosti v rámci infrastruktur. Nadto vytvářením dalších instancí virtuálních strojů se doba čekání na přidělení prodlužuje (odebírá se kapacita) a prodlužuje se tím více, čím jsou požadované nároky na počet jader atd. vyšší. Přidělení zdrojů pro prvních 5 strojů z Tabulka 1 proběhlo

[21] Čelní uzel (*frontend*) je v MetaCentru stroj, na který se uživatelé mohou hlásit přímo, bez rezervace. Na čelní uzly se přihlašuje pomocí protokolu *ssh* (*secure shell*), který šifruje veškerou komunikaci.

Tabulka 2: Specifikace virtuálních strojů zapojených v clusteru.

Pojmenování virtuálního stroje	Počet CPU* /VCPU	Velikost paměti RAM GB	Úložiště v GB (SCSI)	Hostitel**
NameNode	4/4	32	500	zebra3a.priv.cerit-sc.cz
SNameNode	4/4	16	500	zebra2a.priv.cerit-sc.cz
DataNode1	4/4	16	500	zebra1b.priv.cerit-sc.cz
DataNode2	4/4	16	500	zebra6b.priv.cerit-sc.cz
DataNode3	4/4	16	500	zebra6b.priv.cerit-sc.cz
DataNode4	4/4	16	500	hdd2.priv.cerit-sc.cz
DataNode5	4/4	16	200	zebra2a.priv.cerit-sc.cz
DataNode6	4/4	16	200	zebra8b.priv.cerit-sc.cz
DataNode7	4/4	16	200	zebra9b.priv.cerit-sc.c
DataNode8	4/4	16	200	zebra9a.priv.cerit-sc.cz
DataNode9	4/4	16	200	zebra8b.priv.cerit-sc.cz
DataNode10	4/4	16	200	zebra9a.priv.cerit-sc.cz
DataNode11	4/4	16	200	zebra8b.priv.cerit-sc.cz
Celkem	44/44	192	4200	

* Model CPU:
- DataNode5 a 6: Intel Xeon E3-12xx v2 (Ivy Bridge, IBRS), 2799.998 MHz
- ostatní: Intel Xeon E312xx (Sandy Bridge, IBRS update), 2261.940 MHz
** Síťová konektivita: 1x InfiniBand 4xQDR, 1x 10 Gbit/s Ethernet (uzly A), 2x 1 Gbit/s Ethernet



Obrázek 9: Webové rozhraní OpenNebula se seznamem virtuálních strojů.

téměř okamžitě, ale např. pro DataNode5 po 10 dnech. Zdroje pro DataNode8 byly opět přiděleny okamžitě. V tabulce uvedené kapacity jsou tedy maximem možného, co jsem byl schopen získat v rozumném čase (v součtu cca 2 týdny „cíleného“ čekání). Jakmile jsou ale prostředky přiděleny, je požadovaný výkon a kapacita vyhrazena výlučně pro uživatele. Dokud není instance stroje ze strany uživatele zrušena, má ji k dispozici.

Pro přihlášení na spuštěný virtuální stroj s OS Linux jsem využil program Putty v. 0.70 ovládaný z PC s OS Windows 10. Přihlášení probíhalo nejdříve na čelní uzel skirit.ics.muni.cz[22], na který není potřeba předem žádat o přidělení zdrojů, a poté odtud prostřednictvím ssh v příkazové řádce jako uživatel root na běžící instanci virtuálního stroje ve tvaru:

```
ssh root@ip_adresa
```

kde `ip-adresa` je veřejná IPv4 adresa přidělena virtuálnímu stroji, např. `ssh root@147.251.253.225`. Pro pohodlnější přenos souborů z PC s OS Windows 10 do clusteru jsem si na uzel NameNode nainstaloval a nakonfiguroval FTP server.

[22]Lze si vybrat i jiný, geograficky bližší ze seznamu na https://wiki.metacentrum.cz/wiki/Frontend

6.3 Instalace Hadoop frameworku

Pro instalaci AH frameworku jsem použil aktuální distribuci Hortonworks Data Platform 3.1 (HDP). Ta obsahuje Apache Ambari 2.7.3 umožňující správu, monitorování a zabezpečení clusteru. Dovoluje instalaci v podstatě celého ekosystému AH včetně konfigurace. Má práce se primárně nezabývá instalací AH, proto ji nebudu dopodrobna rozebírat a uvedu jen stěžejní partie ze svých několikastránkových poznámek[23].

Proces instalace sestává ze dvou kroků, a to:

- instalace přes příkazovou řádku
- a následná (do)instalace přes webové rozhraní.

Instalace přes příkazovou řádku

Pro instalaci HDP je na začátku potřebné, aby centrální uzel měl přístup do podřízených uzlů přes *ssh* bez hesla. Proto jsem na nejvýkonnějším uzlu (budoucí NameNode a master uzel pro většinu služeb AH) vygeneroval přes příkazový řádek `ssh-keygen` veřejný klíč (*id_rsa.pub*), nakopíroval do ostatních uzlů a zavedl do seznamu autorizovaných klíčů. Po následném přihlášení se z centrálního uzlu přes *ssh* byl povolen přístupu bez hesla. Dále jsem upravil na každém virtuálním stroji název počítače, aby byl v souladu s absolutním doménovým jménem počítače (*FQDN*) a vypnul jeho automatickou změnu na výchozí hodnoty po restartu virtuálního stroje (soubor */etc/cloud/cloud.cfg*). Dále jsem svázal *FQDN* a zkrácený název počítače na přidělenou IP adresu stroje v */etc/hosts*. Reálný příklad obsahu souboru *hosts*:

```
127.0.0.1 localhost.localdomain localhost
127.0.0.1 localhost4.localdomain4 localhost4
147.251.253.225 cloud30b.cerit-sc.cz cloud30b

# The following lines are desirable for IPv6 capable hosts
::1 localhost.localdomain localhost
::1 localhost6.localdomain6 localhost6
```

Provoz Ambari a Hive vyžaduje instalaci relační databáze na jednom z uzlů. V ní jsou spravována metadata obou služeb. Na výběr je integrovaná PostgreSql databáze, případně již existující PostgreSql, MySql/MariaDB nebo Oracle. Zvolil jsem oddělenou MySql instalaci, abych měl k databázi administrátorská práva pro možnost úprav konfigurace. Poté již stačilo instalovat Ambari Server, zadat přístupové údaje do MySql databáze a pokračovat v instalaci clusteru ve webovém rozhraní.

[23]Podrobný návod společnosti Hortonworks je k dispozici v pdf dokumentu pod odkazem https://docs.hortonworks.com/HDPDocuments/Ambari-2.7.3.0/bk_ambari-installation/bk_ambari-installation.pdf

(Do)Instalace přes webové rozhraní

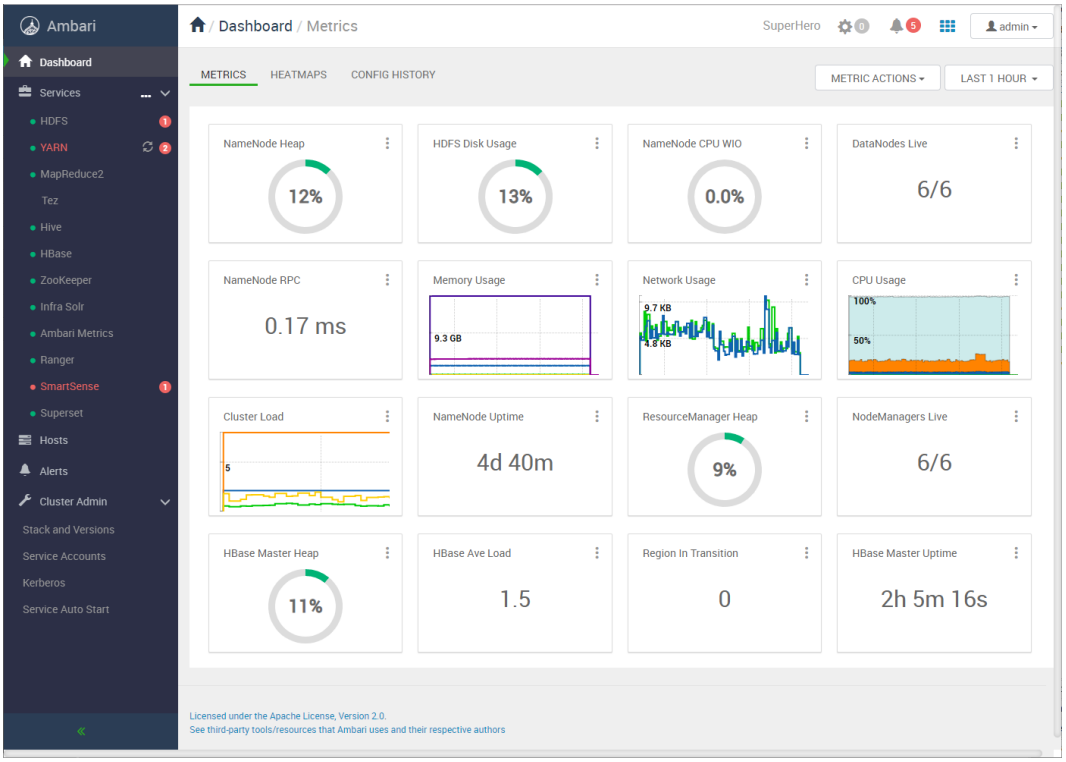
Ve webovém prohlížeči na URL adrese centrálního uzlu a portu *:8080* pokračuje instalace skrze průvodce v několika krocích. Nejdříve je potřeba vyplnit pojmenování clusteru, v mém případě nese název *SuperHero*, následně FQDN všech uzlů a privátní ssh klíč centrálního uzlu. Pokud následné otestování konfigurace proběhne bez závažných chyb, je možné vybrat požadované služby/moduly clusteru a jejich umístění. Klíčové moduly použité dále v této práci byly v této verzi:

- Hadoop 3.1.1.3.1.0.0-78,
- Hive 3.1.0.3.1.0.0-78,
- Tez 0.9.1.3.1.0.0-78.

Fungování většiny modulů je postaveno na master-slave/klient-server architektuře, proto je při jejich instalaci potřeba uvést, na kterém uzlu bude běžet serverová služba a na kterých klienti. Seznam modulů a jejich rozmístění v clusteru *SuperHero* je uveden v tabulce 3. Replikační faktor pro HDFS jsem ponechal na výchozí hodnotě 3. Na každém DataNodu a sekundárním uzlu NameNode (=SNameNode) byly vyhrazeny 4 CPU a 12 GB RAM pro úlohy ze služby Hive. Na každém uzlu tak bylo možné spouštět 4 kontejnery pro výpočet s alokovanými 3 GB RAM a 1 CPU/kontejner. Protože uzly DataNode a uzel SNameNode měly každý 16 GB RAM, zbyly 4 GB na režii systému.

Tabulka 3: Instalované moduly v clusteru.

Modul	**Virtuální stroj**		
	NameNode	**DataNode1 až X**	**SNameNode**
HDFS	NameNode Master	DataNode Slave	SNameNode Master
	Client	Client	Client
Hive	Metastore Mster	Client	Client
	HiveServer2 Master		
MapReduce2	Client	Client	Client
	History Server Master		
Tez	Client	Client	Client
YARN	Client	Client	Client
	ResourceManager Master		
	Registry DNS Master		
ZooKeeper	Client	Client	Client
	Server		



Obrázek 10: Webové rozhraní Ambari.

Webové administrační rozhraní Ambari hraje v obsluze frameworku HDP důležitou roli, neboť skrze něj lze přehledně monitorovat stav celého clusteru, přidávat/odebírat uzly, instalovat a konfigurovat služby, atd. Úvodní okno Ambari Dashboard je zachyceno na obr. 10.

Pro monitoring aktuální zátěže clusteru spuštěnými úlohami jsem vyžíval grafické webové rozhraní nástroje *YARN ResourceManager UI*. To zobrazuje seznam úloh podle jejich stavu, dokončenost v %, zapojení uzlů v úloze, vytíženost CPU a paměti RAM, a další.

6.4 Přenos souboru do clusteru

Pro přenos souborů z osobního počítače se systémem Windows do ulzu NameNode, na kterém běží operační systém CentOS 7, jsem používal protokol FTP nebo SCP. Např. příkaz pro přenos CSV souboru definičních bodů parcel měl tvar:

```
scp -P 2222 c:\Hadoop\Data\Cuzk\Merge\PARCELY_KN_DEF.csv
root@147.251.253.225:/home/zifcakpe/data/
```

Protože se soubory po přenosu pomocí programu *scp* ukládají do lokálního souborového systému CentOS a nikoliv přímo do HDFS, je nutné je ještě překopírovat do HDFS pomocí příkazu `hdfs dfs -put <localsrc> ... <dst>`, např.:

`hdfs dfs -put /home/zifcakpe/data/PARCELY_KN_DEF_WGS.csv /user/zifcakpe /data`.

Další možností přenosu souborů z lokálního PC přímo do HDFS vede přes webové rozhraní Ambari. To umožňuje vkládání souboru z osobního PC přímo do HDFS bez nutnosti použití *scp* a `hdfs dfs -put`. Toto řešení jsem využíval pro menší objemy dat, neboť při větších objemech nad 600 MB byl přenos nestabilní.

6.5 Optimální souborový formát tabulek

Jak je zmíněno v kapitole 5.1.3, Hive umožňuje ukládat (tabulková) data v různých souborových formátech, které se mj. liší v rychlosti zpracování HiveQL dotazů nad těmito soubory. Abych zvolil co možná nejvhodnější formát pro následné prostorové analýzy, sestavil jsem několik dotazů pro porovnání jejich rychlosti zpracování dotazů. Vstupními daty byla CSV tabulka definičních bodů s doplněnými sloupci z polygonových vrstev administrativních jednotek. Její vlastnosti jsou:

- Velikost na disku: 3,57 GB
- Typ tabulky: externí
- Formát: textový CSV (formátování ve shodě se standardem knihovny OpenCSVSerde)
- Počet řádků: 22 772 841
- Počet sloupců: 18
- Datové typy sloupců: všechny String
- Komprese: žádná

Výstupními srovnávanými daty byly interní tabulky v těchto formátech a vlastnostech:

- Textový, Parquet, ORC, Sequence a Avro.
- Datové typy sloupců: změněny na původní dle specifikace správce dat.
- Komprese: žádná.
- Typ tabulek: interní.

Kritéria hodnocení byly

1. Čas potřebný na převod z externí CSV tabulky do cílového formátu.
2. Výsledná velikost souboru.

Tabulka 4: Výsledné hodnoty kritérií pro různé formáty tabulek.

Kritérium	Textový	Parquet	ORC	Sequence	Avro
1.	162,0 s	150,1 s	149,6 s	171,4 s	173,8 s
2.	2942,3 MB	722,8 MB	460,0 MB	3227,5 MB	2144,7 MB
3.	25,5 s	330,1 s	36,4 s	45,4 s	56,9 s
4.	49,2 s	37,6 s	22,3 s	51,1 s	53,5 s
5.	30,6 s	32,8 s	27,7 s	53,1 s	57,5 s

3. Rychlost zpracování dotazu na výběr jediné hodnoty podle jednoho kritéria, př.: `SELECT text_km FROM parcely_kn_def_zsj_orc where id_2 = 821251833;`

4. Rychlost zpracování sumarizačního dotazu nad dvěma sloupci typu double, př.: `SELECT (sum(x)+ sum(y))suma FROM parcely_kn_def_zsj_orc;`

5. Rychlost zpracování dotazu vracející celkový počet bodů v jednotlivých krajích, př.:

```
SELECT kraj.nazev_nut3 AS nazev_kraje, parcely.cnt AS pocet
FROM kraj_orc AS kraj left join (SELECT kod_nut3, count(*)AS cnt
FROM parcely_kn_def_zsj_orc GROUP BY kod_nut3)AS parcely
ON kraj.kod_nut3 = parcely.kod_nut3;
```

Při vykonávání příkazu přes Beeline nebyl cluster zatížen jinou úlohou. Všechny tabulky byly při importu dat systémem automaticky rozděleny do 27 přihrádek.

Jak je patrné z výsledné srovnávací tabulky 4, nejlépe si vedl formát ORC. Pouze u výběru konkrétní hodnoty dle kritéria 3 byl překonán textovým formátem. Proto byl formát ORC vybrán jako výchozí souborový formát pro prostorové analýzy v následující kapitole[24].

[24]Důvod nezvykle vysokého času 300,1 s na zpracování úlohy dle kritéria 3 pro formát Parquet nejsem schopen vysvětlit.

7 Příklad zpracování prostorových dat

Po vytvoření clusteru a instalaci HDP jsem přistoupil k řešení modelové úlohy v prostředí AH z oblasti zpracování prostorových dat. Touto úlohou bylo získaní souhrnných statistik ze spojení bodových a polygonových dat – agregace bodů v polygonu, a to při zapojení různého počtu uzlů clusteru. Přístup a dosažené výsledky jsem srovnal s řešením stejné úlohy v relačním databázovém systému PostgreSQL.

7.1 Použitý software

Na straně AH jsem použil nástroj GIS Tools for Hadoop[25], který lze integrovat pro práci s daty uloženými v Hive tabulkách. Jako relační databázový systém jsem vybral PostgreSQL s extenzí Postgis.

Důvodem volby Hive bylo, že je zaměřen na analýzu statických objemných dat, který se hodí na řešení modelové prostorové úlohy. Dále podporuje jazyk SQL (HiveQL), s daty manipuluje jako s tabulkami, a proto je možné ho snadno porovnat s přístupem z relačních databází.

Důvodem volby databázového systému PostgreSQL bylo, že se jedná o robustní databázový systém s pokročilými funkcemi, ve kterém je integrován nástroj pro práci s prostorovými daty — PostGIS. Produkty jsou značně rozšířené a dobře zdokumentované.

Nakonec nástroje GIS Tools for Hadoop i Postgis implementují obdobné prostorové operace (konstruktory) a lze je tak přímo porovnat. Nadto všechny výše uvedené produkty spojuje open-source licence, která umožňuje využít jejich plnou funkcionalitu bez omezení.

7.1.1 GIS Tools for Hadoop

GIS Tools for Hadoop je sada open source GIS nástrojů od společnosti Esri, která využívá možnosti Hadoop frameworku pro analýzu objemných prostorových dat. Celý projekt a stručné návody k jeho používaní jsou volně dostupné na adrese https://github.com/Esri/gis-tools-for-hadoop. Nástroj se skládá ze tří částí:

1. **Knihovny Esri Geometry API for Java** – zavádí podporu geometrických objektů (body, linie, polygony) a prostorových operací nad daty uloženými v Hadoop clusteru. Obsahuje sadu tříd v jazyce Java jak pro konstrukci geometrických objektů, operací s nimi, tak popis struktury podporovaných GIS formátů a I/O operací s nimi.

2. **Knihovny Spatial Framework for Hadoop** – jedná se o rozšíření Apache Hive o uživatelsky definované funkce (UDF[26]) postavených nad předchozí

[25] http://esri.github.io/gis-tools-for-hadoop/

[26] https://github.com/Esri/spatial-framework-for-hadoop/wiki/UDF-Documentation

knihovnou. Pomocí ní je možné zadávání podporovaných GIS operací v HiveQL a jejich překlad do MapReduce/Tez úloh. Dále poskytuje nástroje pro čtení a zápis JSON formátu.

3. **Geoprocessing Tools for Hadoop** – integruje se jako toolbox v komerčním programu ArcMap ArcGIS, který umožňuje tomuto programu spojení s Hadoopem. Obsahuje nástroje pro kopírování dat do/z HDFS, konverzi prvků z/do formátu JSON a spouštět naplánované úlohy v Apache Oozie.

Knihovny si lze zkompilovat ze zdrojových kódů nebo použít předpřipravený JAR archiv. Pro finální práci ve „svém“ clusteru[27] jsem použil předpřipravené JAR archivy ve verzích: *esri-geometry-api-2.2.2.jar, spatial-sdk-hive-2.1.0.jar, spatial-sdk-json-2.1.0.jar*.

Instalace nástroje GIS Tools for Hadoop v HDP se realizuje prostým vložením tří Java knihoven do souborového systému HDP a pro jejich použití v Hive (např. přes Beeline) je potřeba na počátku definovat cestu k těmto knihovnám příkazem `add jar <path>` (zdrojový kód 2).

```
1  // vytvoření adresáře v příkazovém řádku
2  mkdir /home/esri-git
3  // přesun do nově vytvořeného adresáře
4  cd /home/esri-git
5  // stažení repozitáře gis-tools-for-hadoop z GitHubu
6  git clone https://github.com/Esri/gis-tools-for-hadoop.git
7  // v příkazovém řádku Beeline specifikování cesty ke knihovnám
8  add jar /home/zifcakpe/jar/esri-geometry-api-2.2.2.jar
9          /home/zifcakpe/jar/spatial-sdk-hive-2.1.0.jar
10         /home/zifcakpe/jar/spatial-sdk-json-2.1.0.jar;
```

Zdrojový kód 2: Instalace nástroje GIS Tools for Hadoop v HDP.

Pro eliminaci opakovaného uvádění odkazu na externí knihovny je lze přesunout do složky „*/usr/hdp/current/hive client/auxlib*“, jejihž obsah v podobě *.jar souborů je automaticky přístupný výpočetnímu enginu (MapReduce, Tez, LLAP).

7.1.2 PostGIS

PostGIS je rozšíření objektově-relačního databázového systému PostgreSQL. Zavádí podporu geografických objektů – definuje datový typ Geometry a z něho odvozené typy, např. Point, Line, Polygon, Curve nebo Geometrycollections, a operací nad těmito datovými typy – např. počítání plochy, délky, topologické vazby (modul PostGIS Topology) nebo síťové analýzy (např. vyhledávání nejkratší cesty). Pro urychlení dotazů implementuje indexovaní dat. Aby mohly být

[27] V rámci testování Hadoop clusteru Cesnetu (kap. 6.1) jsem kompiloval JAR soubory ze zdrojových kódů ve frameworku Eclipse.

tabulky s daty použity jako prostorová data, je potřeba jejich dalšího popisu ve 2 přidružených metadatových tabulkách – SPATIAL_REF_SYS pro specifikaci souřadnicového systému, a GEOMETRY_COLUMNS pro popis tabulky jako prostorové vrstvy (např. její název, dimenze prvků, geometrický typ) a určení sloupce obsahujícího geometrii.

Oproti GIS Tools for Hadoop má PostGIS mj. výhodu v podpoře a transformaci z/do libovolného souřadnicového systému a možnosti importu z nebo exportu do 95 vektorových formátů[28] skrze knihovnu/nástroj OGR (pomocí nástroje *ogr2ogr* ovládaného přes příkazový řádek). V této práci budou využita jen vektorová data a operace s nimi, nicméně PostGIS podporuje i rastrová data.

Instalace a zprovoznění databáze PostgreSQL 9.2.24 s rozšířením PostGIS 2.0.7 bylo, na rozdíl od clusteru s AH, jednoduché a přímočaré. V prvním kroku jsem vyhradil v OpenNebula CERITu samostatný virtuální stroj s 10 CPU/V-CPU (Intel(R) Xeon(R) CPU E7- 4860 @ 2.27GHz), 32 GB RAM a s 200 GB diskovým úložištěm. Operační systém jsem vybral opět CentOS 7. A v dalším kroku provedl instalaci databáze v příkazovém řádku virtuálního stroje pomocí těchto několika příkazů:

```
// instalace PostgreSQL serveru 9.2.24
yum install postgresql-server postgresql-contrib

// vytvoření výchozí PostgreSQL databáze a~nastartování
// serveru
postgresql-setup initdb
systemctl start postgresql

// nastavení automatického startu PostgreSQL po~spuštění
// virtuálního stroje
systemctl enable postgresql

// instalace Postgis 2.0.7
yum install postgis
```

V rámci instalace PostGISu se automaticky doinstaluje i knihovna OGR s nástrojem *ogr2ogr*.

7.2 Vstupní data

Za vstupní testovací data pro prostorové analýzy jsem vybral:

- polygonové administrativní hranice územních jednotek České republiky
- a definiční body parcel Katastru nemovitostí.

[28] https://www.gdal.org/ogr_formats.html

Důvodem vybrání těchto dat bylo, že pokrývají rozsáhlé území a jedinečných definičních bodů parcel existuje relativně velké množství (cca 23 miliónů). Data jsou volně stažitelná a spravuje je Český úřad zeměměřický a katastrální.

7.2.1 Administrativní hranice územních jednotek

Aktuální administrativní hranice s vymezením hranic katastrálních území, obcí, okresů, krajů a státu existují k volnému stažení ve formátu SHP a souřadnicovém systému S-JSTK Krovak East North (EPSG 5514/102607) pod odkazem http://services.cuzk.cz/gml/inspire/au/epsg-5514/. Proces konverze a importu hranic je pro všechny administrativní úrovně obdobný, proto níže popíši proces jen pro katastrální území.

GIS Tools for Hadoop

Formát SHP a souřadnicový systém 5514 není přímo podporovaný zvoleným nástrojem, proto jsem data překonvertoval knihovnou *ogr2ogr* do GeoJSON formátu a natransformoval do souřadnicového systému WGS84 (EPSG 4326). Vzor pro transformaci hranice krajů:

```
ogr2ogr -f "GeoJSON" -s_srs EPSG:5514 -t_srs EPSG:4326
-s_srs EPSG:5514 -t_srs EPSG:4326
"d:\Hadoop\Data\Cuzk\ZSJ\GEOJSON\katuzemi_p.geojson"
"d:\Hadoop\Data\Cuzk\ZSJ\SHP\KATUZE_P.shp"}
```

Pro přenos z OS Windows do Hadoop clusteru jsem využil webové rozhraní Ambari. Cesta uložených souborů v HDFS byla `/user/zifcakpe/data/`. Následně se přihlásil na NameNode uzel přes příkazovou řádku (`ssh root@147.251.253.225`) a přes Beeline provedl import do Hive. Ten sestává ze dvou kroků, a to definice schématu tabulky a vlastního importu dat:

- definice schématu tabulky *katuzemi_p* pro polygonové vymezení katastrálních území; data budou načítána z formátu GeoJSON; sloupec *boundaryshape* obsahuje lomové body polygonu v binárním tvaru – jeho obsah je automaticky generován knihovnou.

```
drop table if exists katuzemi_p;
CREATE TABLE katuzemi_p (
kod int, nazev string, obec_kod int, pou_kod int,
orp_kod int, okres_kod int, lau1_kod string,
vusc_kod int, prares_kod int, boundaryshape Binary)
ROW FORMAT SERDE
'com.esri.hadoop.hive.serde.GeoJsonSerDe'
STORED AS INPUTFORMAT
'com.esri.json.hadoop.EnclosedGeoJsonInputFormat'
OUTPUTFORMAT
```

```
'org.apache.hadoop.hive.ql.io.
HiveIgnoreKeyTextOutputFormat';
```

- import dat z GeoJSON souboru v HDFS do Hive tabulky *katuzemi_p*

```
LOAD DATA INPATH
'/user/zifcakpe/data/katuzemi_p.geojson'
OVERWRITE INTO TABLE katuzemi_p;
```

Pro srovnání rychlosti vykonávání dotazu v následující kapitole byla tabulka katastrálních území uložena ve formě dělené do 14 přihrádek podle názvu kraje a ve formě bez dělení.

PostGIS

Import dat do databáze PostgreSQL s rozšířením Postgis je znacně jednodušší, neboť knihovna *ogr2ogr* do ní umí číst i zapisovat. Import GeoJSON souboru katastrálních území do databáze „*parcely*“ pod uživatelským účtem „*root*“ do tabulky „*katuzemi_p*“ na virtuálním stroji měl tvar:

```
ogr2ogr -f "PostgreSQL" PG:"dbname=parcely user=root"
"/home/katuzemi_p.json" -nln katuzemi_p
```

Schéma cílové tabulky ani sloupec s geometrií není potřeba předem zadávat, vše obstará knihovna *ogr2ogr* ze vstupních dat a předaných parametrů.

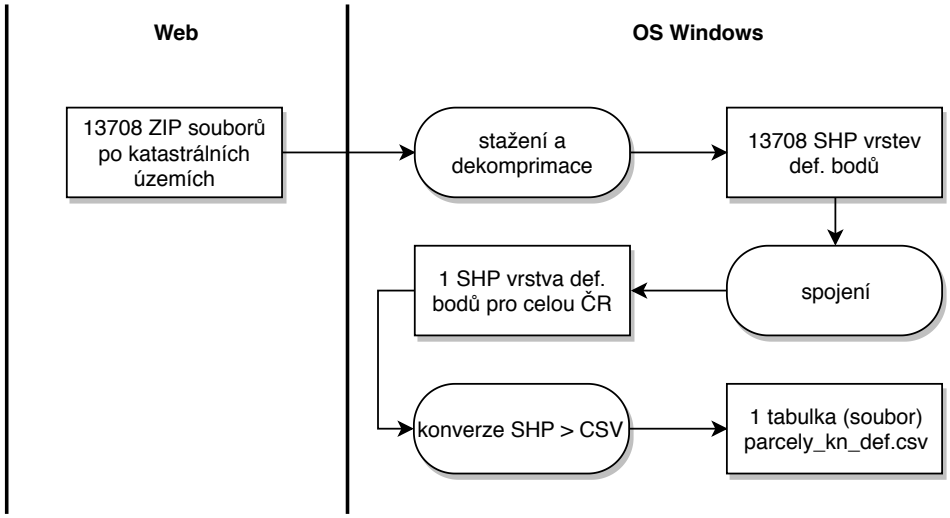
7.2.2 Definiční body parcel Katastru nemovitostí

Definiční bod parcely je bod určený souřadnicemi v S-JTSK[29], který je umístěn přibližně uprostřed parcely vedené v evidenci Katastru nemovitostí. Na rozdíl od plošné vektorové reprezentace parcely, která není v digitální podobě pro území celé České republiky k dispozici, je pokrytí území státu definičními body kompletní.

Abych s definičními body mohl pracovat, musel jsem si je nejdříve stáhnout a poté pomocí knihovny *ogr2ogr* transformoval do cílového výměnného formátu CSV (obr. 11). Definiční body parcel pro celou Českou republiku ve formátu ESRI Shapefile jsou ke stažení pod odkazem http://services.cuzk.cz/shp/ku/epsg-5514/. Definiční body (společné s dalšími vrstvami) jsou rozděleny po katastrálních územích do 13708 komprimovaných ZIP souborů. V programovacím jazyce Java jsem si vytvořil program pro automatizované stahování a dekomprimaci souborů. Pomocí příkazu v OS Windows

```
for /f "tokens=*" %G in ("dir PARCELY_KN_DEF.shp /b /s")
do ogr2ogr -f "ESRI Shapefile" -update -append
..\Merge\parcely_kn_def.shp
```

[29]S-JTSK= Souřadnicový systém Jednotné trigonometrické sítě katastrální. Nejčastěji používanou variantou je „S-JTSK/Krovak East North“ značená kódem EPSG: 5514 (příp. 102067 v produktech fy ESRI).



Obrázek 11: Proces získaní tabulky s definičními body parcel Katastru nemovitostí.

jsem SHP soubory spojil do jednoho souboru o velikosti 9,3 GB. Poté pomocí příkazu

```
ogr2ogr -f "CSV" -lco GEOMETRY=AS_XY
-s_srs EPSG:5514 -t_srs EPSG:4326
C:/Hadoop/Data/Cuzk/Merge/PARCELY_KN_DEF_WGS.csv
C:/Hadoop/Data/Cuzk/Merge/PARCELY_KN_DEF.shp
```

natransformoval do souřadnicového systému WGS84[30], doplnil souřadnice X a Y a převedl do formátu CSV. Velikost CSV souboru *parcely_kn_def.csv* byla 1,7 GB, 22 772 841 řádků a 9 sloupců.

Na ukázku struktury a obsahu je níže záhlaví a první 4 řádky tabulky:

```
X,Y,ID,ID_2,TYPPPD_KOD,KATUZE_KOD,TEXT_KM,PAR_VYMERA,DRUPOZ_KOD,
ZPVYPA_KOD,BUD_ID
12.81451811,50.3692291,1,40190747010,100018,600016,256/10,497,7,,
12.81480643,50.3691176,2,40190746010,100018,600016,173/7,26,14,17,
12.82037326,50.3715411,3,20404797010,100018,600016,453/23,399,7,,
12.82077571,50.3714844,4,20404798010,100018,600016,1939/1,700,14,17,
```

Ke každému bodu, který je vytvořen z jednoho řádku ze souřadnic X a Y, jsou přidruženy další informace v sousedních sloupcích. Z nich jsou pro tuto práci podstatné PAR_VYMERA (= výměra parcely) a DRUPOZ_KOD (=druh pozemku parcely vyjádřený číselným kódem). Ty budou použity pro výpočet statistik v modelové úloze v další kapitole.

[30]WGS84=Světový geodetický systém 1984, značený kódem EPSG: 4326.

Počty záznamů v tabulce podle druhu pozemků měly tuto četnost:

Druh pozemku	Kód	Počet
chmelnice	3	24293
vinice	4	97543
ovocný sad	6	104071
vodní plocha	11	666656
lesní pozemek	10	1569506
zahrada	5	2666754
trvalý travní porost	7	2906289
zastavěná plocha a~nádvoří	13	4312585
orná půda	2	5087236
ostatní plocha	14	5336379

GIS Tools for Hadoop

CSV soubor „parcely_kn_def.csv“ jsem přenesl do clusteru, naimportoval do Hivu a z tabulek administrativních hranic do ní doplnil další sloupce s názvy jednotek, které původní CSV tabulka neobsahovala (SQL klauzule JOIN nad společnými klíči). Spojením jsem vygeneroval obsahově totožné 3 tabulky, které se ale lišily v těchto vlastnostech:

1. Tabulka s názvem „parcely_kn_def_zsj_part“ - interní tabulka ve formátu ORC, která byla dělena do 14 oddílů (partitions) podle hodnoty ve sloupci „nazev_nut3“ (=příslušnost definičního bodu v jednom ze 14 krajů)

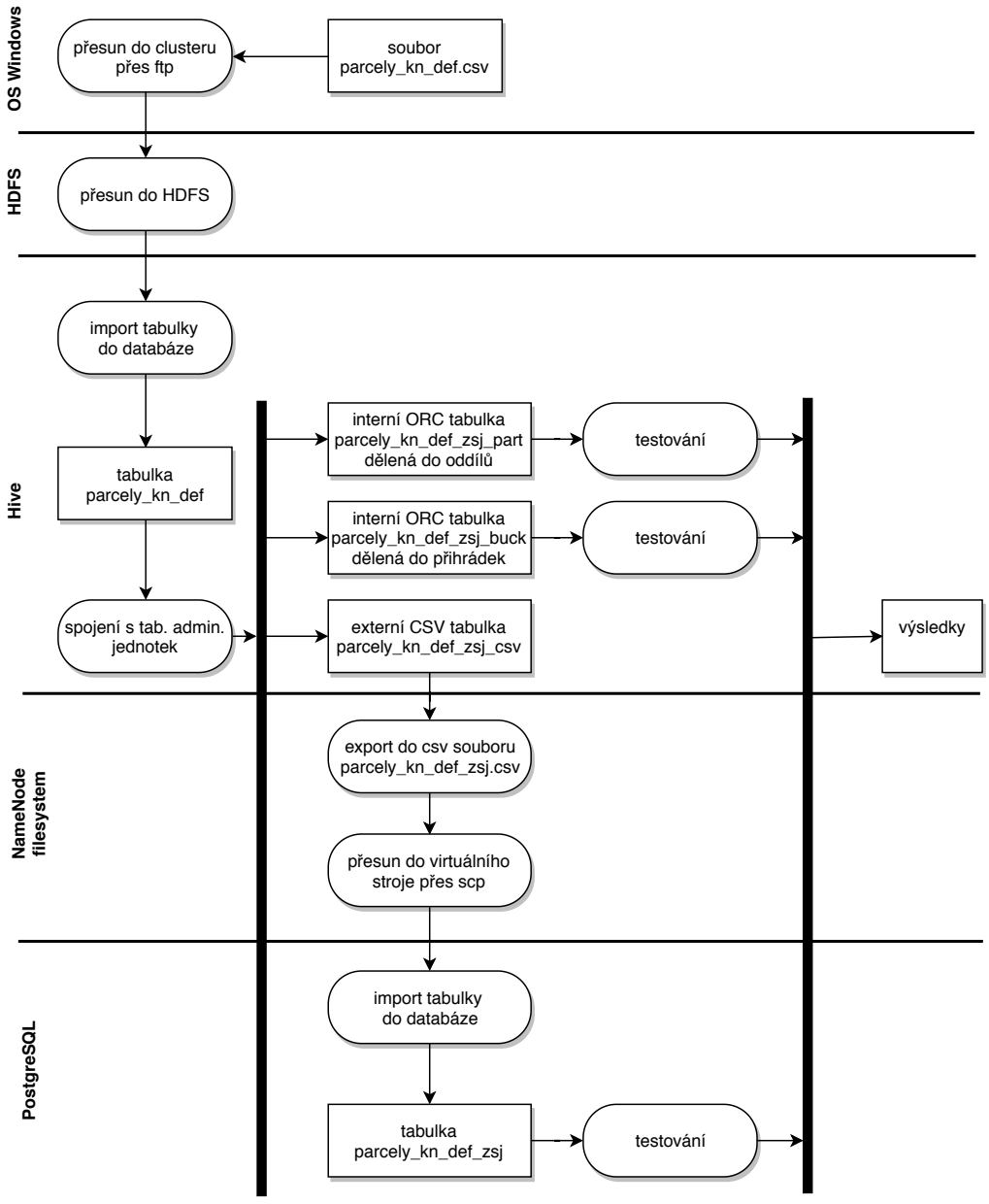
2. Tabulka s názvem „parcely_kn_def_zsj_buck“ - interní tabulka ve formátu ORC, která byla dělena do 27 přihrádek (buckets). Daný počet přihrádek automaticky navrhl Hive podle struktury dat sám.

3. Tabulka „parcely_kn_def_zsj_csv“ – externí tabulka v textovém formátu, pomocí které jsem data „vykopíroval“ z datového skladu Hivu do samostatného CSV souboru a přenesl do virtuálního stroje s PostgreSQL databází (viz další odstavec „PostGIS“).

Diagram graficky zobrazující výše uvedený proces je uveden na obr. 12.

PostGIS

Vyexportovaný soubor „parcely_kn_def_zsj.csv“ z Hivu jsem pomocí scp přenesl přes čelní uzel MetaCentra do virtuálního stroje s databází PotsgreSQL a poté provedl import do databáze (obr. 12). Na rozdíl do importu formátu GeoJSON (příp. SHP) skrze knihovnu *ogr2ogr* byl přímý import CSV souboru dvoukrokový:



Obrázek 12: Diagram přenosu a zpracování tabulky definičních bodů v Hivu a PostgreSQL.

- V prvním kroku bylo nutné vytvořit schéma tabulky, protože CSV formát neobsahuje informaci o datových typech hodnot ve sloupcích (viz kap. 3.2 - CSV).
- Ve druhém kroku byl pomocí příkazu `copy` v příkazové řádce *psql* proveden vlastní import dat.

Oba příkazy měly tuto formu:

```
create table parcely_kn_def_zsj (x real, y real, id int,
id_2 int, typpp_kod int, katuze_kod int, text_km
varchar(200), par_vymera int, drupoz_kod smallint,
zpvypa_kod smallint, bud_id int, kod_nut3 varchar(20),
kod_lau1 varchar(20), kod_orp int, kod_opu int,
kod_lau2 int, kod_ku int, nazev_ku varchar(200));

\copy parcely_kn_def_zsj (X, Y, ID, ID_2, TYPPP_KOD,
KATUZE_KOD, TEXT_KM, PAR_VYMERA, DRUPOZ_KOD,
ZPVYPA_KOD, BUD_ID, KOD_NUT3, KOD_LAU1, KOD_ORP,
KOD_OPU, KOD_LAU2, KOD_KU, NAZEV_KU)
FROM '/home/parcely_kn_def_zsj.csv';
```

7.3 Testovací GIS úloha – agregace bodů v polygonu

V obecné rovině úloha zjišťuje, které body odvozené z tabulky definičních bodů parcel spadají do kterého polygonem vymezeného katastrálního území, a z informací o těchto bodech počítá souhrnnou statistiku.

Dotaz nad vstupními daty definičních bodů a hranic katastrálních území měl takto formulované zadání:

- *Vyber katastrální území, ve kterém je součet ploch parcel s druhem pozemku ... největší. Jako výsledek zobraz název tohoto katastrálního území a zjištěnou plochu.*

Přepis do SQL kódu pro GIS Tools for Hadoop a PostGIS měl stejnou formu a je uveden ve zdrojovém kódu č. 3. Za `katuzemiX` jsem dosadil název tabulky s katastrálními územími. Za `tabX` jsem dosadil název tabulky s def. body dané databáze. Za `kodX` jsem dosadil kód druhu pozemku — hodnotu 3 (chmelnice), 5 (zahrada) nebo 14 (ostatní plocha). Ty odpovídají výběru 24.293, 2.666.754 a 5.336.379 záznamů (minimum, střed a maximum celkových počtů záznamů rozdělených podle kategorií „druh pozemku“).

V dotazu se vyskytují konstruktory ST_Point, ST_SetSRID a ST_Contains mají tento význam:

- ST_Point = vrací geometrii bodu z předaných souřadnic x a y.

```
1  SELECT poly.nazev_ku nazev,
2         sum(bod.par_vymera) vymera
3  FROM katuzemiX poly
4  INNER JOIN
5    (SELECT x,
6            y,
7            par_vymera
8     FROM tabX
9     WHERE drupoz_kod=kodX) AS bod ON ST_Contains(poly.wkb_geometry,
10          ST_SetSRID(ST_Point(bod.x, bod.y), 4326))
11 GROUP BY nazev_ku
12 ORDER BY vymera DESC
13 LIMIT 1;
```

Zdrojový kód 3: Zadání GIS úlohy v SQL/HiveQL.

- ST_SetSRID = nastavuje souřadnicový systém předaného geometrie bodu, v daném případě na WGS84 (kód 4326).
- ST_Contains = vrací true, pokud je bod uvnitř předané polygonové geometrie katastrálního území, jinak false.

7.3.1 Výsledky pro GIS Tools for Hadoop

SQL dotazy ze zadání byly pokládány skrze příkazový řádek *Beeline* nad ORC tabulkami definičních bodů a katastrálních území. Testování probíhalo při zapojení 4, 6, 8, 10 a 12 virtuálních strojů fungujících jako DataNody s výpočetními Hive klienty a jedním strojem s funkcí NameNode s instalovaným HiveServer2 Masterem (jejich parametry tab. 5). Na každém DataNodu byly alokovány 4 CPU/VCore a 4 kontejnery po 3 GB RAM na zpracování úlohy. Výpočetním modelem byl Tez. Cachování výsledků bylo zakázáno. Po změně počtu uzlů clusteru jsem nechal dostatečný čas na automatickou replikaci bloků před zahájením vykonávání dalšího dotazu (*HDFS Read/Write* aktivitu jsem monitoroval skrze rozhraní Ambari).

Abych nalezl co možná nejlepší čas zpracování úlohy, byly v dotazech použity různé kombinace optimalizovaných tabulek dělených do oddílů nebo přihrádek. Pro srovnání byla úloha zpracovávána i nad neoptimalizovanými tabulkami. Výsledné časy jsou uvedeny v tabulce 6 a použité zkratky optimalizovaných tabulek mají tento význam:

DF:14/26 Tabulka definičních bodů dělená do 14 oddílů a 26 přihrádek.

DF:0/26 Tabulka definičních bodů dělená do 26 přihrádek.

DF:0/0 Tabulka definičních bodů nedělená do oddílů ani přihrádek.

Tabulka 5: Parametry clusteru při vykonávaní dotazu v GIS Tools for Hadoop.

Počet uzlů v clusteru	Počet uzlů s úložištěm HDFS	Počet uzlů zpracovávajících dotaz Hive	Počet CPU/V-Core na úlohu	Velikost paměti RAM na úlohu (GB)
13	12	12	48	144
11	10	10	40	120
9	8	8	32	96
7	6	6	24	72
5	4	4	16	48

KU:14/0 Tabulka katastrálních území dělená do 14 oddílů.

DF:0/0 Tabulka katastrálních území nedělená do oddílů ani přihrádek.

Z tabulky 6 je zřejmý trend zkracování času při použití dělení tabulek do oddílů a přihrádek, které byly u milionových záznamů v průměru o 41 % nižší než u tabulek bez použití této optimalizace. Dále se čas zkracoval po přidání dalších uzlů do clusteru. Jedná se tak o potvrzení výhody použití AH, kdy lze zvýšit rychlost systému přidáním dalších uzlů.

7.3.2 Výsledky pro PostGIS

SQL dotaz ze zadání byl pokládán skrze příkazový řádek *psql* nad tabulkami definičních bodů a katastrálních území uložených v databázi PostgreSQL s extenzí PostGIS. Přestože jsem virtuální stroj, na kterém bežel databázový systém, škáloval ze 4 až na 20 CPU (Intel(R) Xeon(R) CPU E7- 4860 @ 2.27GHz) a 8–64 GB RAM, nemělo to žádný vliv na snížení dosažených časů uvedených v tabulce 7. Databázový systém stále využíval jen 1 CPU a maximálně 5 GB RAM. Ve verzích PostgreSQL 10 a 11 je ale už možnost paralelního výpočtu implementována, ale protože se pro OS CentOS zatím jedná o nestabilní verze, nebyly v této práci testovány.

Srovnáním výsledků dosažených časů pro PostGIS a nejlepších časů z AH pro miliónové počty bodů je možné říci, že pro zvolenou úlohu (SQL dotaz) byl výkonnostně srovnatelný 1 stroj s databází PostgreSQL+PostGIS, který měl parametry 4 CPU/8 GB RAM, a 11 uzlů v prostředí Hive AH s celkem 40 CPU/120 GB RAM.

Tabulka 6: Časy zpracování dotazu v GIS Tools for Hadoop.

Počet uzlů v clusteru	**Tabulky**	**Čas zpracování (s) pro daný počet definičních bodů**		
		∼24 tis.	**∼2,66 mil.**	**∼5,33 mil.**
13	DF:14/26 a KU:14/0	50	1181	2358
	DF:0/26 a KU:14/0	70	1145	2239
	DF:0/26 a DF:0/0	65	1150	2370
	DF:0/0 a DF:0/0	71	2007	4083
11	DF:14/26 a KU:14/0	69	1518	2825
	DF:0/26 a KU:14/0	82	1249	2407
	DF:0/26 a DF:0/0	71	1268	2679
	DF:0/0 a DF:0/0	73	2211	5063
9	DF:14/26 a KU:14/0	83	1634	4061
	DF:0/26 a KU:14/0	96	1547	3681
	DF:0/26 a DF:0/0	85	1582	3489
	DF:0/0 a DF:0/0	85	2689	5503
7	DF:14/26 a KU:14/0	91	1994	5110
	DF:0/26 a KU:14/0	113	2075	5327
	DF:0/26 a DF:0/0	106	2007	4324
	DF:0/0 a DF:0/0	95	3562	7553
5	DF:14/26 a KU:14/0	158	3360	6817
	DF:0/26 a KU:14/0	170	3411	6568
	DF:0/26 a DF:0/0	135	3496	6951
	DF:0/0 a DF:0/0	151	5853	12826

Tabulka 7: Časy zpracování dotazu pro PostGIS.

Počet CPU a velikost RAM virtuálního stroje	**Čas zpracování (s) pro daný počet definičních bodů**		
	∼24 tis.	**∼2,66 mil.**	**∼5,33 mil.**
4–20 CPU a 8–64 GB	19	1213	2606

8 Diskuze

Přestože byl příklad na zpracování prostorových dat v kapitole 7 postaven na analýze strukturovaných dat, předpokládal jsem lepší výsledek pro AH, než kterého nakonec dosáhl — 1 stroj s databází PostgreSQL+PostGIS se vyrovnal 11 uzlům v prostředí Hive AH u milionových počtů bodů. Pro 24 tisíc bodů nebylo ani se 13 uzly v clusteru dosaženo lepšího času, než kterého dosáhl PostGIS. Nakolik může stát za tímto skóre výkonnost použitých knihoven — GIS Tools for Hadoop vs. PostGIS, nejsem schopen posoudit. V tomto ohledu mohu pouze doufat, že řešení postavené na Hivu bylo optimálně naprogramováno, neboť je vyvíjeno společností ESRI, která je leaderem v oboru GIS řešení. Pravdou také je, že použitá vstupní data mají přesnou datovou strukturu a na jejich zpracování se více hodí relační databáze. Z toho vyplývá, že nasazení AH bude ve srovnání s relační databází přínosné v případech, kdy je požadavek na zpracování velkého množství dat, ale současně je k dispozici velké množství spolupracujících uzlů.

Na druhou stranu existuje velký prostor pro snižování času v AH přidáním dalších uzlů do systému, protože při zpracovávání dotazů byla vstupní úloha rozdělena až na cca 1 000 mapovacích podúloh, ale v jeden okamžik zpracovávána na max 48 z nich (cluster se 13 uzly). Při dostatečném počtu zdrojů by teoreticky mohly být rozdistribuovány na stovky uzlů a zpracovávány současně v jeden okamžik. Trend snižování časů po přidání uzlů je patrný z tabulky č. 6. Každým přidáním 2 uzlů do systému došlo ke snížení doby zpracování dotazu o 7–70 %, přičemž k největšímu zrychlení docházelo v clusterech s malým počtem uzlů a tento trend se se vzrůstající velikostí clusteru zpomaloval — pravděpodobně z důvodu, že klesal význam nově přidaných zdrojů vzhledem k celkové kapacitě clusteru.

Závěr

Tato práce v úvodních kapitolách představila ekosystém Apache Hadoop a jeho možnosti v oblasti distribuovaného ukládání a zpracování velkého množství dat. Detailněji byly popsány komponenty Hive a HBase a provedeno jejich porovnání s relačními databázemi. Ostatní komponenty byly popsány zevrubně s úmyslem alespoň z části nastínit šíři možností, ke kterým je možné je využít.

Cílem práce bylo také na příkladu zpracování prostorových dat demonstrovat výpočetní sílu Hadoop clusteru. Tento cíl se nakonec stal nosným tématem celé práce. Na jeho základě jsem rozšířil úvodní kapitolu o popis prostorových dat, provedl selekci nejvhodnějšího Hadoop clusteru a poté přistoupil k přímému srovnání výkonnosti zpracování prostorových dat v AH a relační databázi PostgreSQL.

Přestože existuje celá řada obsáhlých zdrojů o AH i jeho jednotlivých komponentách, kterým nemůže tato páce konkurovat, považuji za nejužší místo testování systému obtížnost jeho zprovoznění v reálných podmínkách. Proto považuji za přínos této práce v představení způsobu, jak si sestavit vlastní cluster a v reálných podmínkách provádět testování AH, to vše s nulovými náklady a profesionálním servisem cloudu MetaCentra/Cesnet. V průběhu hledání optimálního prostředí pro řešení příkladu jsem si totiž několikrát ověřil, že to, co je funkční na malém vzorku dat, v jisté verzi systému nebo za použití dané komponenty AH, není funkční při použití větších dat, v novější verzi nebo komponentě.

Možnosti rozšíření nebo navázání na tuto práci spatřuji v otestování obdobného příkladu užití v prostředí HBase nebo Impala, neboť výsledné časy Hadoop clusteru s využitím komponenty Hive a výpočetního modelu Tez zaostaly za očekáváním. Vhodné by také bylo pomocí nástroje Pig zautomatizovat transformaci a import vstupních geodat (SHP, JSON) do vybrané databáze AH. Z vlastní zkušenosti bych pouze upozornil na to, že AH je značně „košatý“ strom mnoha technologií a přístupů, které mají některé věci společné, ale řadu specifických a pro podrobné seznámení se s nimi je potřeba spousta času na studium a testování.

Conclusions

This work introduced the Apache Hadoop ecosystem and its capabilities in distributed data storage and processing. Components of Hive and HBase were described in more detail and compared with relational databases. Other components have been described in detail with the intention of at least partially outlining the range of options that can be used.

The aim of the work was also to demonstrate the computational power of the Hadoop cluster on the example of spatial data processing. This goal eventually became the main topic of the whole work. On the basis of this I extended the introductory chapter on the description of spatial data, selected the most suitable Hadoop cluster and then proceeded to directly compare the performance of spatial data processing in AH and the relational database PostgreSQL.

Although there are a number of extensive sources of AH and its individual components that cannot be competed with, I find the closest place to testing the system to the difficulty of setting it up in real terms. That is why I consider the contribution of this work to be a presentation of a way to build our own cluster and perform AH testing in real terms, all with zero costs and professional service of MetaCentrum / Cesnet cloud. For example, during the search for the optimum solution environment, I have verified several times that what works on a small sample of data, in a particular version of a system, or using a given AH component, is not functional when using larger data, in a newer version or component.

I see the possibilities of extending or continuing this work to test a similar example of use in HBase or Impala, because the resulting Hadoop cluster times using the Hive component and the Tez computational model have fallen short of expectations. It would also be useful to use the Pig tool to automate the transformation and import of input geodata (SHP, JSON) into a selected AH database. From my own experience, I would just point out that AH is a very big tree of many technologies and approaches that have some things in common, but a number of specific and a lot of time to study and test to get to know them.

A Obsah přiloženého CD/DVD

`data/`

Testovací data použitá v práci (kap. 7.2):
- Polygonové administrativní hranice územních jednotek České republiky ve formátech JSON, GEOJSON a SHP.
- Definiční body parcel Katastru nemovitostí ve formátu CSV.

`doc/`

Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`install/`

Knihovnu *Esri Geometry API for Java — esri-geometry-api-2.2.2.jar*.
Knihovny *Spatial Framework for Hadoop — spatial-sdk-hive-2.1.0.jar* a *spatial-sdk-json-2.1.0.jar*.

`readme.txt`

Webová adresa a instrukce pro přihlášení do administračního rozhraní Ambari použitého clusteru.

Literatura

[1] ANONYMOUS. *Executive Summary - Data Growth, Business Opportunities, and the IT Imperatives* [online]. 2014. [cit. 2018-10-22]. Dostupné z: https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[2] LANEY, Doug. 3-D Data Management: Controlling Data Volume, Velocity, and Variety.*META Group Res Note 6* [online]. 2001, roč. 6 [cit. 2018-11-23]. Dostupné z: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[3] ANONYMOUS. *Welcome to the Cognitive Era. A new era in technology, a new era in business.* [online]. 2016 [cit. 2019-01-01]. Dostupné z: https://www.ibm.com/blogs/insights-on-business/oracle-consulting/wp-content/uploads/sites/13/2016/02/cognitive-white-paper.pdf

[4] NORMANDEAU, Kevin (ed.). *Beyond Volume, Variety and Velocity is the Issue of Big Data Veracity.* 2013-09-12 [cit. 2018-11-23]. Dostupné z: http://www.evskp.cz/SD/4c.pdf

[5] ESRI - TECHNICAL SUPPORT. ESRI Shapefile Technical Description. An ESRI White Paper. 1998-07 [cit. 2019-01-03]. Dostupné z: https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[6] HRONEK, Jiří. *Databázové systémy.* Olomouc: Univerzita Palackého, Přírodovědecká fakulta, Katedra informatiky, 2007. 135 s. Dostupné také z: https://phoenix.inf.upol.cz/esf/ucebni/databa.pdf

[7] WIKIPEDIA. *Apache Hadoop — Wikipedia, The Free Encyclopedia.* [online]. 2018 [cit. 2018-10-22]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Hadoop

[8] ANONYMOUS. *HDFS Architecture* [online]. 2018 [cit. 2018-10-23]. Dostupné z: https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[9] ANONYMOUS. *NameNode High Availability in Hadoop HDFS.* [online]. 2018-11-16 [cit. 2018-11-07]. Dostupné z: https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/

[10] ANONYMOUS. *hdfs-default.xml (soubor)* [online]. 2018-10-30 [cit. 2018-10-30]. Dostupné z: https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

[11] ANONYMOUS. *Rack Awareness.* [online]. 2018-08-02 [cit. 2018-10-30]. Dostupné z: https://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-common/RackAwareness.html

[12] ANONYMOUS. *Hadoop HDFS Data Read and Write Operations*. [online]. 2016-06-13 [cit. 2018-10-29]. Dostupné z: https://data-flair.training/blogs/hadoop-hdfs-data-read-and-write-operations/.

[13] MURTHY, Arun. *Apache Hadoop YARN – Background and an Overview*. [online]. 2012-08-12 [cit. 2018-10-31]. Dostupné z: https://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/.

[14] COSS, Rafael. *Hadoop Crash Course*. [online]. 2015 [cit. 2019-05-09]. Dostupné z: https://www.slideshare.net/Hadoop_Summit/hadoop-crash-course-workshop-at-hadoop-summit.

[15] ANONYMOUS. *Hadoop Yarn Tutorial for Beginners*. [online]. 2016-12-13 [cit. 2018-11-01]. Dostupné z: https://data-flair.training/blogs/hadoop-yarn-tutorial/.

[16] ANONYMOUS. *Apache Hive Tutorial – A Single Best Comprehensive Guide for 2018*. [online]. 2018-03-01 [cit. 2018-11-08]. Dostupné z: https://data-flair.training/blogs/hadoop-yarn-tutorial/.

[17] ANONYMOUS. *ACID support*. [online]. [cit. 2019-04-26]. Dostupné z: https://orc.apache.org/docs/acid.html.

[18] ANONYMOUS. *Hive Transactions*. [online]. 2018-12-15 [cit. 2019-01-09]. Dostupné z: https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions.

[19] WHITE, Tom. *Hadoop: the definitive guide*. Sebastopol, CA: O'Reilly, 2015. ISBN 978-1-491-90163-2.

[20] ANONYMOUS. *HBase - Admin API*. [online]. 2018 [cit. 2018-11-17]. Dostupné z: https://data-flair.training/blogs/hadoop-yarn-tutorial/.

[21] APACHE HBASE TEAM. *Apache HBase ™ Reference Guide. Version 3.0.0-SNAPSHOT*. [online]. 2018 [cit. 2018-11-14]. Dostupné z: http://hbase.apache.org/book.html.

[22] CHANG, Fay; DEAN, Jeffrey; GHEMAWAT, Sanjay aj. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 2008-06, roč. 26. Dostupné také z: http://dx.doi.org/10.1145/1365815.1365816.

[23] KHURANA, Amandeep. Introduction to HBase schema design. *The Usenix Magazine*. 2012, roč. 37, č. 5.

[24] GEORGE, Lars. *HBase : the definitive guide*. Sebastopol, CA: O'Reilly, 2011. ISBN 978-1-449-39610-7.

[25] WIKIPEDIA. *Apache Impala — Wikipedia, The Free Encyclopedia* [online]. 2019 [cit. 2019-04-30]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Impala.

[26] JELÍNEK, Lukáš. *Apache Kafka se rychle stává dominantní streamovací platformou* [online]. 2017-02-06 [cit. 2019-04-30]. Dostupné z: https://www.linuxexpres.cz/novinky/apache-kafka-se-rychle-stava-dominantni-streamovaci.

[27] WIKIPEDIA. *Démon (software) — Wikipedia, The Free Encyclopedia* [online]. 2018 [cit. 2018-11-01]. Dostupné z: https://cs.wikipedia.org/wiki/D%C3%A9mon_(software).

[28] ANONYMOUS. *ApplicationMaster. MapR 6.1 Documentation.* [online]. 2018 [cit. 2018-11-05]. Dostupné z: https://mapr.com/docs/61/MapROverview/c_application_master.html.

[29] ANONYMOUS. *What is the difference between Cloud, Grid and Cluster?* [online]. 2018 [cit. 2018-10-30]. Dostupné z: https://stackoverflow.com/questions/9723040/what-is-the-difference-between-cloud-grid-and-cluster.