Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Paralelizace výpočtů pomocí modelu MapReduce

Bakalářská práce

Autor: Lukáš Trumm
Studijní obor: Informační management

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové listopad 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 8.11.2015 Lukáš Trumm

Děkuji vedoucímu bakalářské práce Ing. Pavlu Křížovi, Ph.D. za vedení práce.

Anotace

Tato práce se zabývá zpracováním velkého množství dat pomocí programovacího modelu MapReduce. Popisuje hlavní myšlenky tohoto přístupu a principy, ze kterých model vznikl. Vysvětluje jeho propojení s distribuovaným souborovým systémem, který je nezbytnou součástí efektivního využití modelu. Praktická část práce se zaměřuje na open-source implementaci MapReduce modelu – Hadoop a distribuovaný souborový systém HDFS. Cílem praktické části je vytvoření funkčního příkladu MapReduce úlohy implementované pomocí Hadoop frameworku. Záměrem je získat z velkého množství částečně strukturovaných dat o návštěvnosti české Wikipedie souhrnné statistické informace o nejnavštěvovanějších stránkách. Práce porovnává řešení tohoto problému na lokálním počítači s řešením pomocí paralelního zpracování na více serverech. Dochází k závěru, že využití paralelizovaných výpočtů je efektivní až při zpracování dat v řádech desítek GB a větších.

Annotation

Title: Parallel computing using MapReduce model

This thesis deals with processing large amount of data using MapReduce programming model. It describes main ideas of this approach and the origin of the model. The thesis explains interconnection between the model and distributed file system, which is necessary for effective usage of the model. Practical part of this thesis focus on open-source implementation of MapReduce – Hadoop and distributed file system HDFS. The goal of the practical part is creation of functioning example of MapReduce job implemented using Hadoop framework. The intention is to obtain statistical information about the most visited pages on Czech Wikipedia from large amount of semi-structured data. This thesis compares the solution of this problem using local computer and using parallel processing on multiple servers. It concludes that the parallel computing is effective starting with tens of GB of input data or more.

Klíčová slova

Paralelní výpočty, distribuované systémy, MapReduce, Hadoop

Key words

Parallel computing, Distributed systems, MapReduce, Hadoop

Obsah

1 Úvod

Čím dál více společností potřebuje zpracovávat velké množství dat. Záznamy detailního chování uživatelů při procházení webových portálů nebo informace ze senzorů na nejrůznějších zařízeních generují obrovské množství záznamů, které však obvykle jen čekají na své zpracování. Dat je často tolik, že je obtížné nebo nemožné je všechny zpracovat. Tento problém lze řešit prováděním výpočtů na více počítačích souběžně. Jeden z přístupů představuje model MapReduce.

MapReduce je programovací model určený pro distribuované výpočty nad velkým množstvím dat. Byl původně vyvinut společností Google, která své první výsledky shrnula ve studii z roku 2004 [1]. Model a jeho myšlenky se rychle ujaly – vznikla open-source implementace zvaná Hadoop, jejíž vývoj byl veden společností Yahoo a později se stal projektem organizace Apache.

Již v roce 2004 zpracovával Google s využitím MapReduce 100 TB dat denně [1] a v roce 2008 je již zmiňováno 20 PB dat denně [2]. Mnoho dalších velkých firem se začíná potýkat s podobným množstvím dat. Např. Facebook přijímal v roce 2009 15 TB nových dat denně [3]. V roce 2012 oznámil Jay Parikh, vice prezident pro infrastrukturu společnosti Facebook, že pomocí distribuovaných výpočtů zpracovává firma přes 100 TB dat každých 30 minut a denně přijímají 500 TB dat nových [4].

Množství dat však nemusí být jediným problémem. Nízká míra strukturovanosti dat představuje překážku i při menším objemu. Mezi taková data se řadí multimediální soubory, dokumenty nebo webové stránky, které jsou technologiemi jako jsou např. relační databáze špatně zpracovatelné. Data, která jsou obtížně zpracovatelná ať již z důvodu velkého množství, kvůli nedostatečné strukturovanosti nebo z obou důvodů bývají označována za Velká data (nebo častěji anglicky – Big data).

Pojem MapReduce je možné chápat ve třech mírně odlišných významech. Prvním je výše zmíněný programovací model a principy s ním spojené. Druhým významem je obecné označení pro framework, který řídí provádění programů napsaných tak, aby vyhovovaly modelu MapReduce. Do třetice je možné MapReduce chápat jako konkrétní softwarovou implementaci programovacího modelu, příkladem může být open-source projekt Hadoop, nebo uzavřená implementace společnosti Google [5].

Mým záměrem v rámci této práce je seznámit se s programovacím modelem MapReduce a jeho open-source implementací Hadoop. Přínosem pro veřejnost bude český popis teoretických i praktických aspektů MapReduce, jelikož většina významných zdrojů k této problematice je v angličtině. Principy budou aplikovány na konkrétní funkční příklad, který v rámci této práce vznikne.

2 Principy modelu MapReduce

2.1 Hlavní myšlenky

Model MapReduce stojí na několika myšlenkách, které sami o sobě nejsou nijak nové. Avšak až ve chvíli, kdy inženýři společnosti Google spojili patřičné postupy do jednoho celku, ukázala se síla tohoto systému.

Zvětšující se množství dat, které je třeba zpracovat, lze do jisté míry řešit pořizováním větších pamětí. Čísla uvedená výše ale naznačují, že množství dat může být větší, než paměti největších počítačových systémů se sdílenou pamětí. Jsou to systémy s mnoha procesory ale jen jednou operační pamětí, do které mají všechny procesory přístup. I kdyby se daly takové systémy pořídit, jejich cena by byla neúměrná výkonu. Barroso, Clidaras a Hölze [6] vidí řešení v systémech, které *škálují horizontálně*. Horizontální škálování označuje zvyšování celkového výkonu systému zvyšováním počtu výpočetních jednotek. Naopak vertikálním škálováním se rozumí zvyšování výkonu u jednotlivých počítačů. Ekonomicky nejvýhodnější je podle nich systém složený z většího množství serverů nižší třídy nebo osobních počítačů vyšší třídy. Implementace modelu MapReduce se zaměřují právě na takovéto systémy – větší klastr levnějších serverů.

Je třeba předpokládat, že *chyby v hardwaru jsou běžné*. Pokud by například průměrná doba mezi selháními jednoho zařízení byla 1 000 dní, znamená to, že v klastru o tisíci serverech se v průměru vyskytne chyba každý den. Implementace programovacího modelu MapReduce jsou schopné tyto výpadky výpočetních uzlů klastru automaticky řešit a přiřadit úlohy jiným jednotkám.

V tradičních vysoce výkonnostně náročných úlohách (např. meteorologické simulace) se uplatňuje princip rozdělení klastru na výpočetní a skladovací uzly, které jsou propojené vysoce propustným spojením. Řada výpočetních problémů s velkými daty však není závislá jen na výkonu procesorů. Rozdělení uzlů klastru na výpočetní a skladovací tak vytváří úzké hrdlo systému. Místo přesouvání dat k výpočetní síle je však možné *přesouvat výpočty k datům*. MapReduce využívá architektury, kde na jednotlivých uzlech klastru je jak výpočetní tak i skladovací jednotka. Tedy výpočty na procesoru určitého serveru probíhají nad daty na přidruženém disku.

Datově náročné úlohy jsou typické tím, že množství zpracovávaných dat je příliš velké na to, aby se data vešla do paměti počítače. Zpracovávání dat z pevných disků je pak výrazně rychlejší, pokud jsou data čtena sekvenčně. Náhodný přístup k datům je řádově pomalejší a to jak na mechanických discích tak i v případě SSD disků. Model MapReduce upřednostňuje *sekvenční čtení* před náhodným přístupem do paměti. Řada principů modelu upřednostňuje propustnost před odezvou.

Programátorské řemeslo vyžaduje značnou míru koncentrace a soustředění. Krátkodobá paměť programátora musí udržet jak obyčejné záležitosti jako názvy proměnných, tak složité principy algoritmu, který právě implementuje. Programování distribuovaných systémů je

o to náročnější, protože výsledný kód běží souběžně na mnoha procesorech. MapReduce se snaží řešit tento problém vrstvou abstrakce, kterou *odděluje programátora od nízkoúrovňových problémů* jako je např. zamykání datových struktur. MapReduce rozděluje problém na to co se má při výpočtech vyřešit a jak se má tento výpočet rozdělit mezi počítače v klastru. Co se bude řešit je čistě v rukou programátora, zatímco přenesení výpočtů do klastru obstarává MapReduce framework.

Ideální paralelní algoritmus by měl být schopen lineárně škálovat. Když se dvakrát zvětší objem dat, doba běhu by se měla prodloužit maximálně dvakrát. A nebo by měla doběhnout za stejný čas, pokud bychom zdvojnásobili velikost klastru. Většina dnešních algoritmů je však daleko od tohoto ideálu. Je zajímavé, že pro některé případy se využití modelu MapReduce ideálu velmi přibližuje [5, str. 15].

2.2 Vznik modelu MapReduce

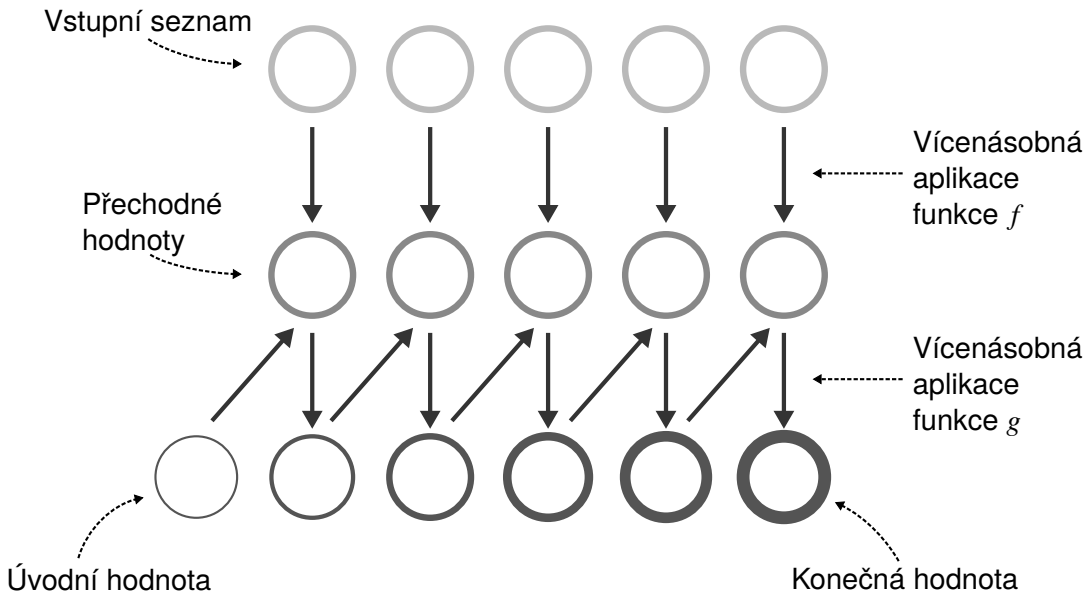
Základním konceptem jak se vypořádat s velkým množstvím dat je princip rozděl a panuj. Hlavní myšlenkou je rozdělit problém na více menších dílčích úloh, ideálně nezávislých. Tyto dílčí problémy pak mohou být zpracovávány paralelně na více počítačích, více procesorech, nebo jen ve více vláknech procesorového jádra.

Tento koncept je aplikovatelný na širokou škálu problémů. Konkrétní implementace ale může být pokaždé jiná a může vyžadovat velmi komplexní pojetí problematiky. Před programátorem takového systému stojí nelehké otázky:

- Jak rozdělit problém na menší úlohy, které by mohly být spouštěny paralelně?
- Jak distribuovat úlohy mezi větší množství uzlů klastru?
- Jak zabezpečit aby výpočetní uzly dostávaly data, která právě k výpočtu potřebují?
- Jak sdílet dílčí výsledky mezi uzly?

MapReduce poskytuje programátorovi abstrakci, která skrývá systémové detaily. Umožňuje tak soustředit se na to, *jaké* výpočty mají být provedeny a méně na to *jak* budou provedeny.

Název MapReduce vychází ze dvou základních logických součástí – *map* a *reduce*. Tvůrci modelu se inspirovali ve stejnojmenných funkcích, které se používají ve funkcionálních programovacích jazycích [1]. Tyto funkce mohou mít jako argument jiné funkce. Na obrázku 1 je vysvětlen princip využití funkcí *map* a *reduce*.



Obrázek 1: Znázornění funkcí *map* a *reduce*. Funkce *map* má jako svůj argument funkci f s jedním argumentem, která je aplikována na všechny položky vstupního seznamu. *Reduce* opakovaně aplikuje funkci g s dvěma parametry (aktuální přechodná a předchozí hodnota) a agreguje tak výsledky. Zdroj: Převzato z knihy Data-intensive Text Processing with MapReduce [5] a pozměněno.

2.3 Zpracování párů klíč-hodnota v MapReduce

MapReduce se zaměřuje na zpracování dat ve formátu klíč-hodnota. Klíč i hodnota mohou být nejrůznější datové typy nebo formáty. Dean a Ghemawat ve své studii [1] schematicky zapisují funkce *map* a *reduce*:

$$map: (k_1, v_1) \rightarrow [(k_2, v_2)]$$

$$reduce: (k_2, [v_2]) \rightarrow [(k_3, v_3)]$$

Funkce *map* je aplikována na každou dvojici klíč-hodnota na vstupu. Výsledkem je seznam (v hranatých závorkách) *přechodných* klíčů a hodnot[1]. *Reduce* převede klíče a seznamy hodnot, které k nim patří, na výsledný seznam klíčů a hodnot.

Dean, Ghemawat [1] a autoři dalších publikací, které se věnují MapReduce [5, 7] uvádějí jako úvodní příklad využití modelu MapReduce výpočet četnosti výskytu slov ve velké kolekci dat. Vstupem je velká množina textových dokumentů (klíče tvoří identifikační čísla dokumentů, hodnoty jsou texty dokumentů samotných). Mapovací funkce rozdělí text na slova a vytvoří (tzv. emituje) přechodné páry klíč-hodnota, kde klíče jsou jednotlivá slova a hodnoty jsou 1, značící jeden výskyt daného slova. Redukční funkce poté sečte všechny hodnoty náležící každému klíči. Algoritmus mapovací a redukční fáze je možné zapsat pseudokódem, viz výpis 1 na následující straně.

[1] anglicky *intermediate keys and values*

```
class Mapper
function Map(docid a, doc d)
    for all term t ∈ doc d do
        Emit(term t, count 1)

class Reducer
function Reduce(term t, counts [c1, c2, ...])
    sum ← 0
    for all count c ∈ counts [c1, c2, ...] do
        sum ← sum + c
    Emit(term t, count sum)
```

Algoritmus 1: Pseudokód MapReduce algoritmu pro výpočet četnosti slov. Zdroj: Převzato z knihy Data-intensive Text Processing with MapReduce [5].

Na obrázku 2 na následující straně je schéma znázorňující zpracování klíčů a hodnot v MapReduce. Vstup je nejdříve rozdělen mezi jednotlivé *mapovače*[2], které vykonají programátorem zadanou mapovací funkci. *Kombinovače* zajistí lokální agregaci klíčů a hodnot. *Rozdělovače* jsou odpovědné za rozdělení přechodných hodnot mezi *redukovače* podle klíčů. Určí tedy, kam budou zkopírovány které klíče a hodnoty. Následuje *přemisťování* dat, o které se stará MapReduce framework. Cílem této fáze je setřídit a seskupit data podle klíčů tak, aby hodnoty patřící jednomu klíči byly všechny přítomné na jednom uzlu klastru. Redukční úlohy mohou díky tomuto rozdělení zpracovávat jen data, která jsou umístěna na daném uzlu klastru.

Konkrétní průběh zpracování dat při řešení úlohy o výpočtu četnosti slov ve zdrojovém textu je znázorněn na obrázku 3 na straně 7. Vstup je rozdělen na jednotlivé servery klastru, kde probíhají fáze mapování a kombinování. Data jsou poté rozdělena a přemístěna na jiné servery, kde proběhne redukční fáze.

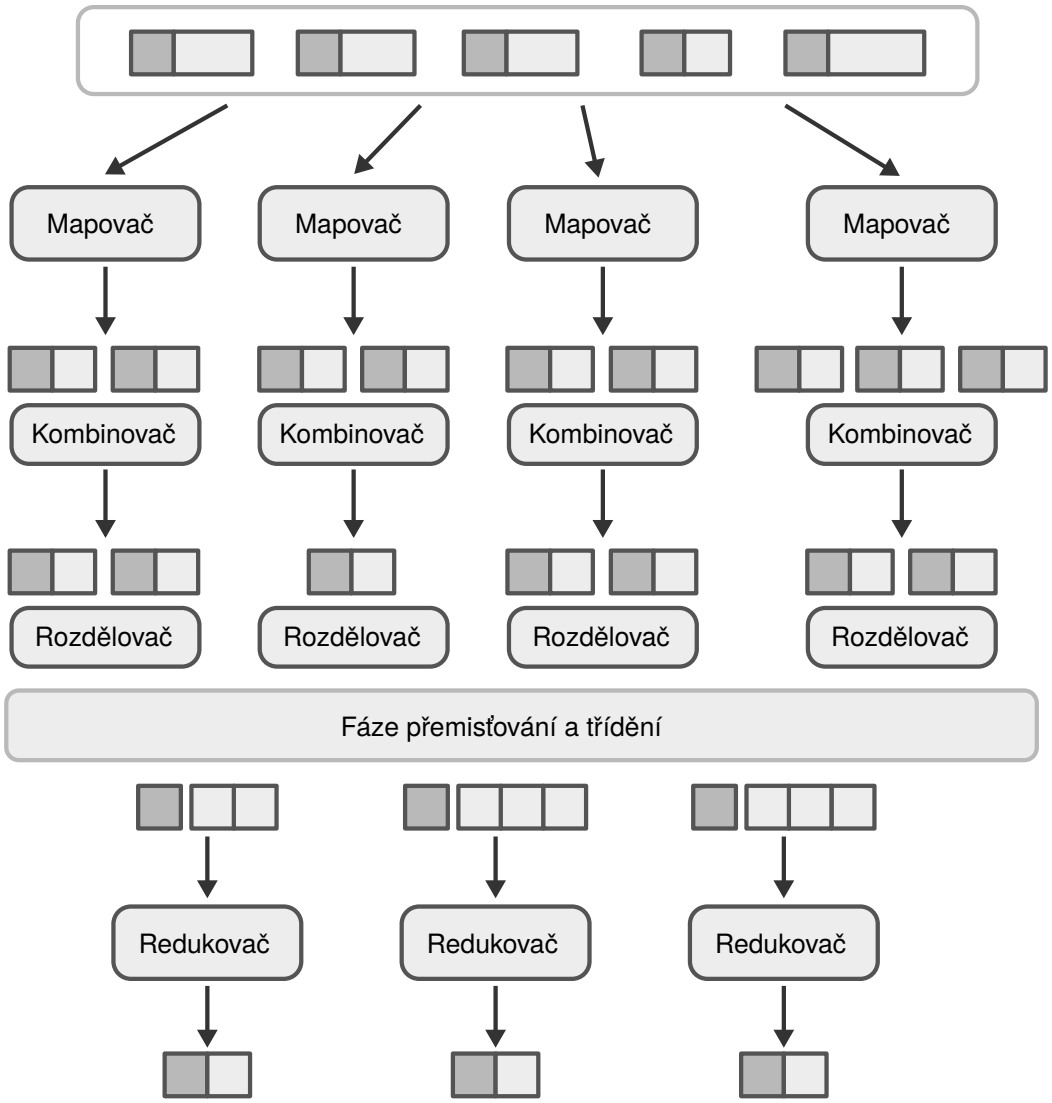
2.4 Běhové prostředí MapReduce

MapReduce framework bývá označován jako *běhové prostředí* modelu MapReduce. Je to implementace programovacího modelu nasazená na počítačovém klastru. Programátor do systému nahraje *úlohu* a běhové prostředí se postará o celý průběh výpočtu, ať běží na jednom nebo na tisíci počítačích v klastru [5].

Běhové prostředí se stará o *plánování* výpočtů. Úlohy rozděluje na jednotlivé úkoly, např. jednotlivá provádění mapovací funkce. Framework hlídá, zda vykonávání nějakého úkolu netrvá příliš dlouho a v případě potřeby nechá na stejném úkolu pracovat více uzlů klastru, čímž zvyšuje pravděpodobnost co nejdřívějšího dokončení celé úlohy.

Další odpovědností běhového prostředí je maximalizovat *lokálnost* provádění úkolů. Každý

[2] U pojmů psaných kurzívou jsem zvolil vlastní překlad. Anglicky: *mapovač* – mapper, *kombinovač* – combiner, *rozdělovač* – partitioner, *redukovač* – reducer, *přemisťování* – shuffling.



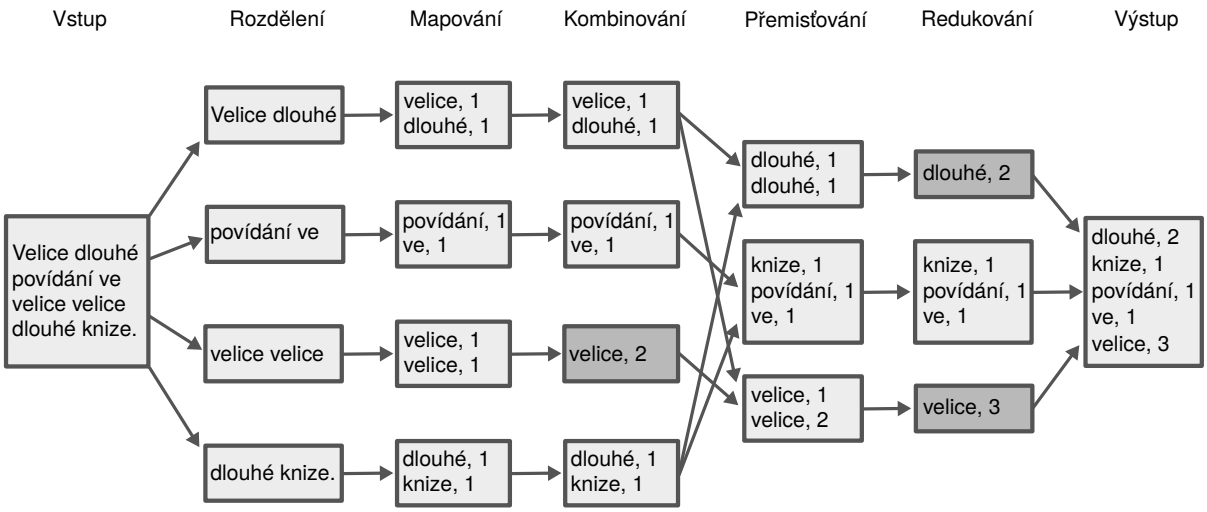
Obrázek 2: Schéma procesů v modelu MapReduce. Zdroj: Převzato z knihy Data-intensive Text Processing with MapReduce [5] a pozměněno.

mapovací i redukční úkol proběhne výrazně rychleji, pokud je prováděn s daty přítomnými na daném počítači. Přesuny dat po síti mezi počítači v klastru nejvíce zpomalují celý výpočet [9].

Přechodné výsledky, které jsou výstupem mapovací fáze, musí být distribuovány do uzlů klastru, kde bude probíhat redukční fáze. Tato distribuce se nazývá *přemisťování a třídění* a seskupují se při ní hodnoty podle klíčů. Redukční fáze nemůže začít, dokud není přemisťování a třídění dokončeno. O tuto *synchronizaci* se stará běhové prostředí.

Jedním z nejdůležitějších aspektů celého konceptu MapReduce je *odolnost proti chybám* hardwaru i softwaru. MapReduce je určený k nasazení na levnější hardware nižší třídy, kde se se selháními musí počítat.[3]

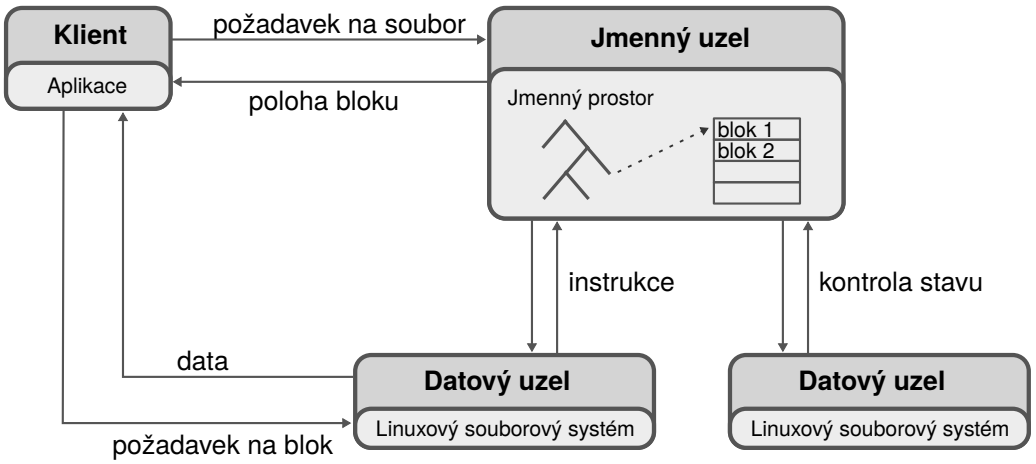
[3] Anglický překlad uvedených pojmů: *běhové prostředí* – runtime, *MapReduce úloha* – MapReduce job, *fáze přemisťování a třídění* – shuffle and sort phase.



Obrázek 3: Znázornění průběhu výpočtu četnosti slov. Tmavěji jsou vyznačená místa, kde došlo k redukci dat. Zdroj: Převzato od Martinja van Groningena [8] a upraveno.

2.5 Distribuovaný souborový systém

Aby se projevily přednosti MapReduce modelu, je využíván ve spojitosti s distribuovaným souborovým systémem. V takovém systému jsou soubory rozděleny na bloky, které jsou uložené na datových uzlech. Každý blok je replikován na více uzlů pro zajištění spolehlivosti. Replikace také napomáhá MapReduce frameworku s plánováním úloh, protože umožňuje více využít lokálnosti dat. Všechny informace (metadata) o souborech jsou uloženy v centrálním jmenném uzlu. Každý datový uzel je v neustálém spojení se jmenným uzlem, který kontroluje, že bloky dat jsou dostatečně replikovány. Při výpadku nějakého uzlu tak může ihned zařídit nápravu. Na obrázku 4 jsou schematicky znázorněny základní součásti distribuovaného souborového systému.



Obrázek 4: Schéma distribuovaného souborového systému. Zdroj: Převzato z knihy Data-intensive Text Processing with MapReduce [5] a upraveno.

Distribuovaný datový systém, který podporuje činnost MapReduce modelu, splňuje něko-

lik předpokladů [5]:

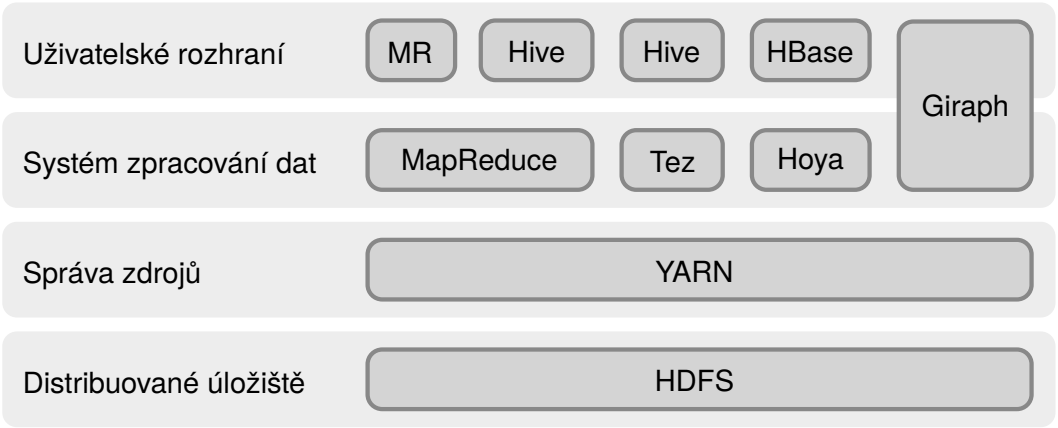
- Systém je optimalizovaný na skladování menšího množství větších souborů. Velikost souborů se může pohybovat v gigabytech, což je vzhledem k dávkové povaze MapReduce modelu lepší, velké množství menších souborů. Režie spojená s vytvářením úkolů pro jednotlivé soubory by znatelně zpomalila celý výpočet.
- Důležitější je vysoká přenosová rychlost udržitelná po delší dobu, než nízká odezva mezi uzly klastru. Opět to souvisí s dávkovou povahou MapReduce úloh.
- Bezpečnost souborového systému nemusí být řešena, protože se předpokládá nasazení v prostředí, kam mají přístup jen oprávnění uživatelé.
- I při chybách a výpadcích některých uzlů klastru je systém schopen poskytovat svoje služby spolehlivě.

Distribuovaný souborový systém tvoří společně s programovacím modelem MapReduce systém, který umožňuje zpracovat velké množství dat bez nutnosti investice do vysoce výkonného a specializovaného počítače.

3 Hadoop framework

Hadoop je open-source framework vybudovaný na principech MapReduce modelu. Hadoop začal vznikat v reakci na zveřejnění technologických postupů využívaných ve společnosti Google [10, 1] v letech 2003 a 2004. Zpočátku byl vyvíjen pro účely projektu Apache Nutch, později se z něj stal samostatný projekt organizace Apache s podporou velkých firem jako Yahoo nebo Facebook [11]. Hadoop je framework je napsaný v programovacím jazyce Java, uživatelské programy umožňuje vytvářet i v jiných jazycích díky nástroji *Hadoop Streaming* [7].

Hadoop dnes zastřešuje celý ekosystém souvisejících projektů. Na obrázku 5 je znázorněna struktura těchto projektů. Základem je distribuovaný souborový systém HDFS. Ke správě výpočetních zdrojů a pro plánování úloh je určen YARN[4]. O samotné paralelní zpracování dat se stará Hadoop MapReduce.



Obrázek 5: Hadoop a ekosystém souvisejících projektů. Zdroj: Převzato z kurzu Hadoop Fundamentals I [12] a upraveno.

Hadoop doprovází velké množství souvisejících projektů. Pro základní orientaci uvedu jejich přehled.

Ambari	Systém pro monitorování, správu a nasazení Hadoop klastru.
HBase	Sloupcově orientovaná databáze určená pro velká množství dat.
Hive	Systém umožňující dotazování pomocí jazyka podobného SQL nad daty v HDFS.
Hue	Webové rozhraní pro ovládání Hadoop klastru a nasazených systémů.
Chukwa	Systém pro monitorování klastrů a sbírání dat.
Mahout	Systém pro strojové učení využívající MapReduce a Spark.
Oozie	Nástroj pro plánování úloh. Umí spravovat úlohy systémů MapReduce, Pig, Hive.

[4] HDFS = Hadoop Distributed File System, YARN = Yet Another Resource Negotiator

Pig	Skriptovací jazyk a běhové prostředí pro paralelní zpracování dat využívající HDFS a MapReduce.
Spark	Systém pro distribuované výpočty v paměti uzlů klastru, bez využití zápisů na disk. Spark podporuje HDFS a YARN ale může běžet i nad jinými systémy.
Sqoop	Nástroj pro přesouvání dat mezi relačními databázemi a HDFS.
Tez	Obecný framework pro zpracování toků dat, vystavěný na Hadoop YARN.
ZooKeeper	Systém pro koordinaci distribuovaných aplikací.

3.1 Architektura Hadoop frameworku

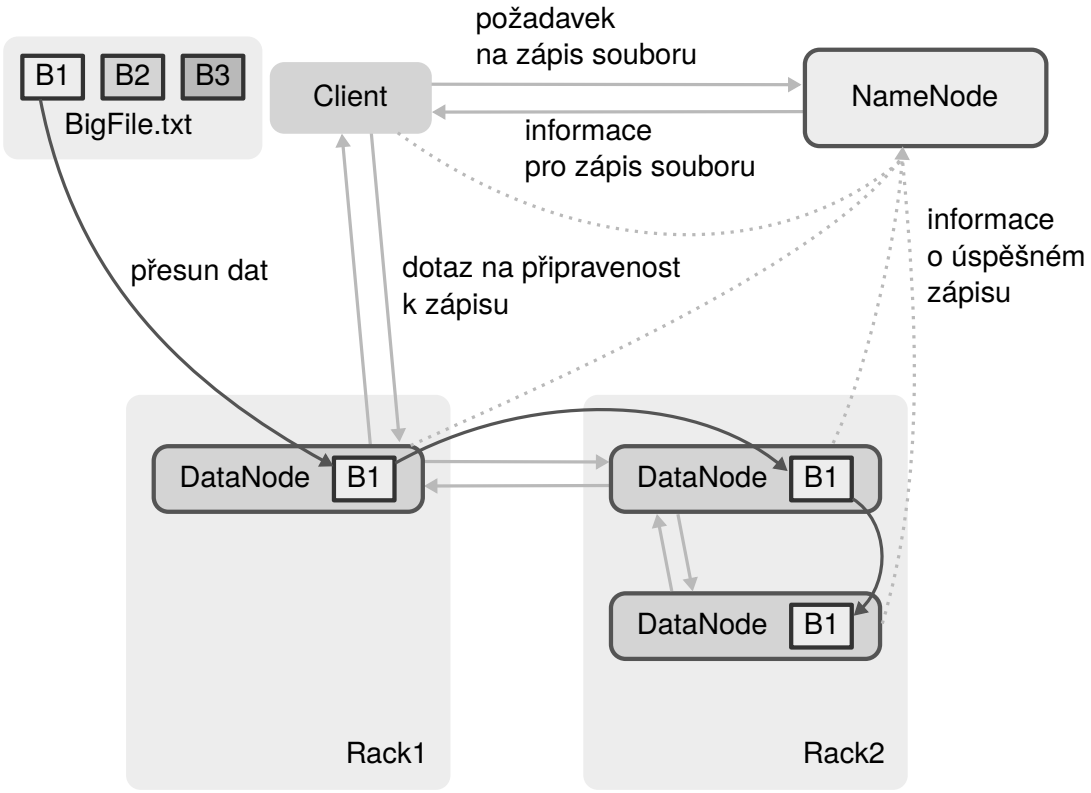
3.1.1 HDFS

Distribuovaný souborový systém HDFS běží nad souborovým systémem každého datového uzlu klastru. Je stavěný na práci s velkým objemem dat, velkými soubory a díky replikaci umožňuje zajistit spolehlivost a toleranci k selhání počítačů v klastru.

HDFS je určen k hromadnému čtení a zápisu velkého množství dat. Naopak pro mnoho přístupů na náhodná místa vhodný není. Tomu odpovídá architektura systému znázorněná na obrázku 6 na následující straně. Základem zajištění spolehlivosti je replikace. Soubory jsou před zapsáním do systému nejprve rozděleny na bloky o velikosti 64 MB nebo větší. Klientská aplikace zjistí, kam má být daný blok zapsán a zkontroluje, že přidělené datové uzly jsou připraveny přijmout data. Po potvrzení je datový blok zapsán na všechny určené datové uzly najednou. Tzv. replikační faktor určuje, na kolik uzlů bude datový blok zapsán. Standardní hodnota je 3. Každý datový blok je zapsán na 3 různé uzly, jeden v jednom racku a další dva v jiném. Tato strategie vytváří dobrý poměr mezi spolehlivostí a výkonem při zápisu a čtení dat [11, str. 74].

Využití bloků namísto samotných souborů přináší několik výhod. Systém může díky rozdělení pracovat i se soubory většími než je velikost disků na počítačích v klastru. Díky fixní velikosti bloků je zjednodušena práce s daty a informacemi o nich (metadata uložená na Name node).

HDFS není vhodný pro aplikace, které vyžadují krátkou dobu odezvy. V systému je dána přednost vysoké rychlosti při sekvenčním čtení dat před rychlostí okamžité odezvy.



Obrázek 6: Architektura distribuovaného systému HDFS se znázorněním replikace. Zdroj: Vlastní zpracování.

3.1.2 Interakce s HDFS

HDFS neposkytuje tolik možností interakce jako například souborový systém kompatibilní se standardem POSIX. Pro základní úkony však poskytuje obdobné rozhraní. Interakci s HDFS pomocí příkazové řádky zprostředkovává stejnojmenný program `hdfs`.
Příkaz

```
hdfs fsck /data -files -blocks
```

vypíše všechny soubory z adresáře `data` v HDFS a informace o jejich rozdělení do bloků.
Ke kopírování a procházení distribuovaným souborovým systémem slouží příkaz `hdfs dfs`.

```
hdfs dfs -mkdir /data
```

vytvoří složku data v kořenovém adresáři.

```
hdfs dfs -put ~/soubor.txt /data
```

zkopíruje soubor.txt do adresáře data v HDFS.

```
hdfs dfs -get /data/output ~/output
```

zkopíruje obsah složky `/data/output` do domovského adresáře do složky `output`.

Příkaz `-getmerge` také kopíruje data z HDFS na lokální disk, nejdříve ale sloučí všechny soubory ve zdrojové složce do jednoho.

```
hdfs dfs -ls /data
```

vypíše podrobný obsah obsah složky `/data` podobně jako unixový příkaz `ls -l`.

```
hdfs dfs -cat /data/soubor.txt
```

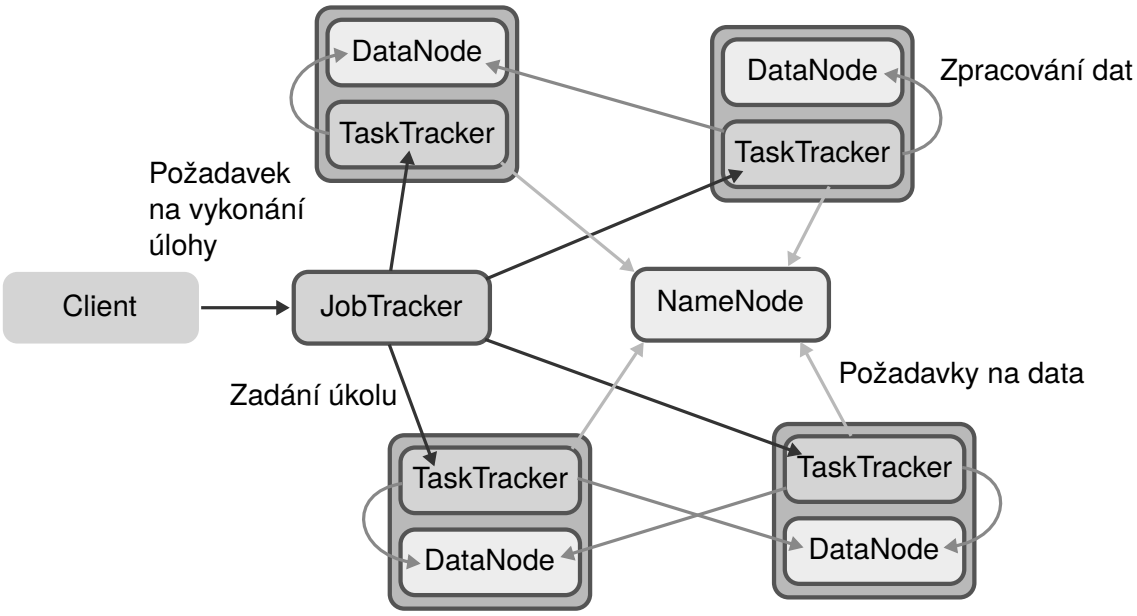
vypíše přímo obsah souboru `soubor.txt`.

Kromě HDFS podporuje Hadoop řadu dalších souborových systémů, ke kterým lze díky vrstvě abstrakce přistupovat přes společné rozhraní. Podporováno je například úložiště S3 společnosti Amazon, nebo úložiště dostupné pomocí protokolu FTP [11].

3.1.3 Hadoop verze 1

Systém Hadoop MapReduce původně řešil nejen proces zpracování dat podle MapReduce modelu ale i správu zdrojů a rozdělování úkolů. Na obrázku 7 je naznačena spolupráce mezi komponentami HDFS a MapReduce.

JobTracker je řídící uzel, který rozděluje úkoly mezi uzly typu TaskTracker. Ty provádějí zadané mapovací a redukovací úkoly a využívají při tom dat z HDFS prostřednictvím příslušného DataNode.



Obrázek 7: Architektura systému Hadoop verze 1. Zdroj: Převzato z kurzu Hadoop Fundamentals I [12] a upraveno.

3.1.4 Hadoop verze 2

V systému Hadoop verze 2 došlo k rozdělení odpovědností v systému tak, aby řízení a správa zdrojů byla oddělena od způsobu jakým jsou prováděny výpočty. Díky tomu je možné na Hadoop klastru spouštět i jiné aplikace než MapReduce a klastry mohou být tvořeny více uzly.

Hlavním řídícím uzlem systému YARN je *Resource Manager* (obrázek 8 na následující straně). Jeho úkolem je přidělovat zdroje aplikacím v klastru. Na každém podřízeném uzlu klastru běží *Node Manager*, který komunikuje s Resource Managerem. Na jeho žádost vytváří Node Manager tzv. *kontejnery*, které alokují určité zdroje (paměť, CPU) na daném uzlu. V kontejnerech běží aplikace. Chod každé aplikace řídí *Application Master*. Ten si vyjednává přidělení dalších kontejnerů pro spuštění svých úkolů. Na *History server* se ukládají informace o zpracovávaných úlohách.

Celý systém využívá HDFS jako zdroj dat. V druhé verzi je zvýšena spolehlivost souborového systému přidáním záložní NameNode, který v případě poruchy primárního NameNode převezme jeho úlohu. Výše zmíněný nástroj ZooKeeper může být použit pro koordinaci hlavního a záložního NameNode uzlu.

V případě velkého množství dat v systému hrozí, že na NameNode dojde paměť. To je od Hadoop verze 2 možné řešit pomocí tzv. HDFS Federation. Při využití federace je v systému více uzlů NameNode, každý z nich spravuje jinou část souborového systému.

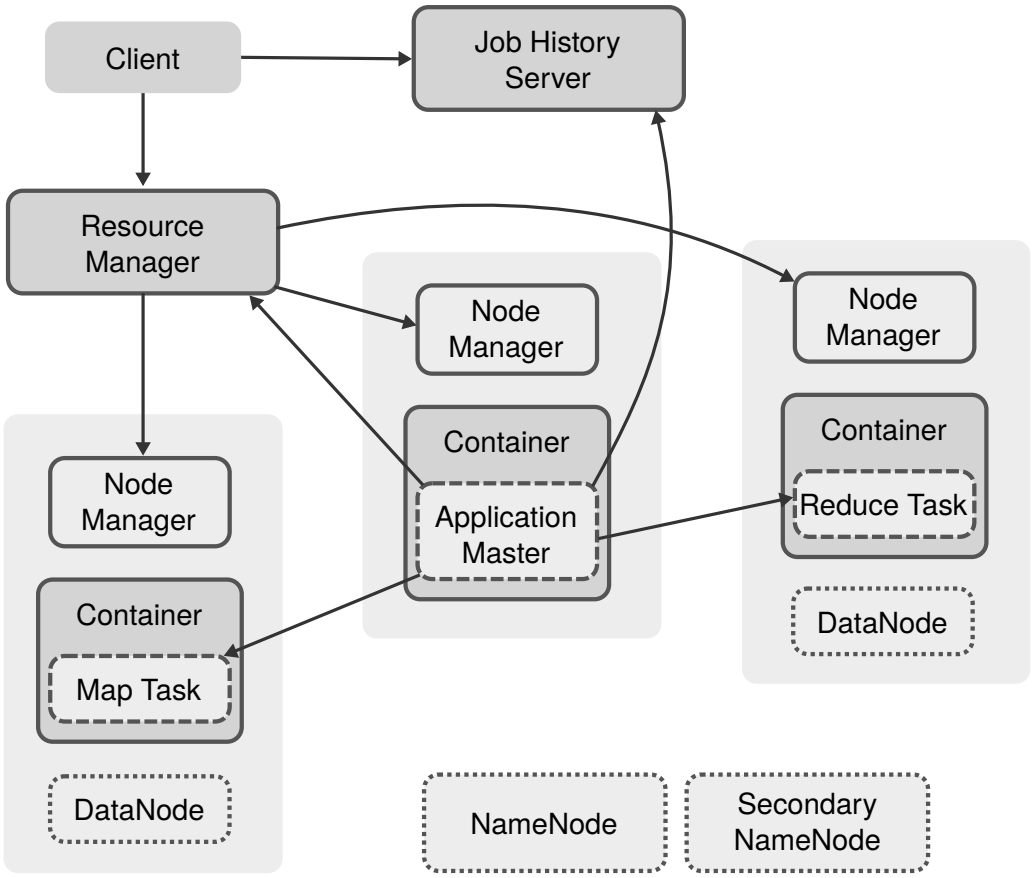
3.2 Běh systému Hadoop

Hadoop je open-source projekt, je k dispozici ke stažení na oficiálních stránkách projektu, kde se také nachází obsáhlá dokumentace. Hadoop je primárně určen pro běh na operačním systému Linux. Pro chod je nezbytný *Java SDK* s *ssh* [13].

Hadoop může být spuštěn ve třech různých módech:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

První je určen ke spuštění na jednom počítači, jako jeden Java proces. V pseudo-distribuovaném módu je pro každou komponentu systému spuštěna samostatná JVM. To je vhodné pro testovací účely, jelikož celý klastr se skládá z jediného uzlu a zároveň se komponenty chovají jako v distribuovaném prostředí (komunikují prostřednictvím soketů atp.). Jedině plně distribuovaný mód je použitelný pro klastry o více uzlech.



Obrázek 8: Architektura Hadoop verze 2 se systémem YARN. Zdroj: Vlastní zpracování.

3.2.1 Pseudo-distribuovaný mód

Pro účely vývoje a testování je doporučené [7] mít na lokálním počítači nainstalovaný Hadoop v pseudo-distribuovaném módu. V tomto módu je chování systému podobné, jako při běhu v klastru, ale zároveň je možné celý systém ovládat lokálně, protože běží jen na jednom počítači. Vyžaduje přípravu prostředí a konfiguraci jako v distribuovaném prostředí.
Instalace probíhá ze staženého instalačního balíčku:

```
tar -xf /vagrant/installators/hadoop-2.7.1.tar.gz
```

Hadoop vyžaduje nastavení několika proměnných prostředí:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_HOME=/opt/hadoop/hadoop-2.7.1
export PATH=$HADOOP_HOME/bin:$PATH
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
```

A explicitní nastavení cesty k instalaci Javy v konfiguračním souboru `hadoop-env.sh`:

```
vi $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Dále je třeba nastavit protokol ssh, přes který Hadoop procesy komunikují. Následující série příkazů vytvoří veřejný klíč pro komunikaci pomocí protokolu ssh a umístí ho mezi důvěryhodné klíče. Procesy tak budou moci komunikovat aniž by vyžadovali heslo pro připojení.

```
mkdir ~/.ssh; cd ~/.ssh
ssh-keygen
cp id_rsa.pub authorized_keys
```

Hadoop obsahuje v základní instalaci příklady MapReduce programů. Pro ověření, že je Hadoop správně nastaven je vhodné použít příklad, který nevyžaduje žádné vstupy [7]. Takovým příkladem je přibližný výpočet čísla π:

```
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce
   -examples-2.7.1.jar pi 4 100
```

Standardní výstup spuštěného příkladu zakončený textem `Job Finished in ... seconds` signalizuje úspěšné dokončení úlohy.

Hadoop poskytuje široké možnosti nastavení prostřednictvím konfiguračních dokumentů. Nacházejí se v adresáři `$HADOOP_HOME/etc/hadoop`. V souboru `core-site.xml` jsou umístěny konfigurační parametry společné pro HDFS a MapReduce. V souboru `hdfs-site.xml` jsou parametry týkající se HDFS, NameNode a DataNodes. V souboru `mapred-site.xml` se nastavuje MapReduce a v souboru `yarn-site.xml` YARN.

Základní nastavení pro pseudo-distribuovaný mód obsahuje konfigurační parametry uvedené ve výpisu 2. Parametr `fs.defaultFS` určuje adresu NameNode a typ souborového systému. Parametr `hadoop.tmp.dir` nastavený na jiný adresář než standardní linuxový `/tmp` zajistí, aby data v HDFS nebyla nechtěně smazána systémem. Protože při běhu na lokálním počítači není užitečná replikace bloků dat, je třeba jí omezit parametrem `dfs.replication`.

```
# core-site.xml
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/opt/hadoop/data</value>
  </property>

# hdfs-site.xml
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
```

Algoritmus 2: Základní konfigurační parametry pro pseudo-distribuovaný mód. Konfigurační parametry se aplikují přidáním tagu `<property>` s příslušným jménem a hodnotou do konfiguračního souboru. Zdroj: Převzato z dokumentace k Hadoop MapReduce [13].

Před prvním spuštěním HDFS je třeba naformátovat NameNode:

```
hdfs namenode -format
```

Po předchozích krocích lze spustit HDFS a YARN:

```
cd $HADOOP_HOME
./sbin/start-dfs.sh
./sbin/start-yarn.sh
```

Pokud proběhlo spuštění úspěšně, vypíše příkaz `jps` následující Java procesy:

```
24050 NameNode
24410 SecondaryNameNode
24185 DataNode
24590 ResourceManager
24725 NodeManager
```

Správný chod systému je možné opět ověřit spuštěním příkladu. Soubor `/data/text.txt` obsahuje libovolný text.

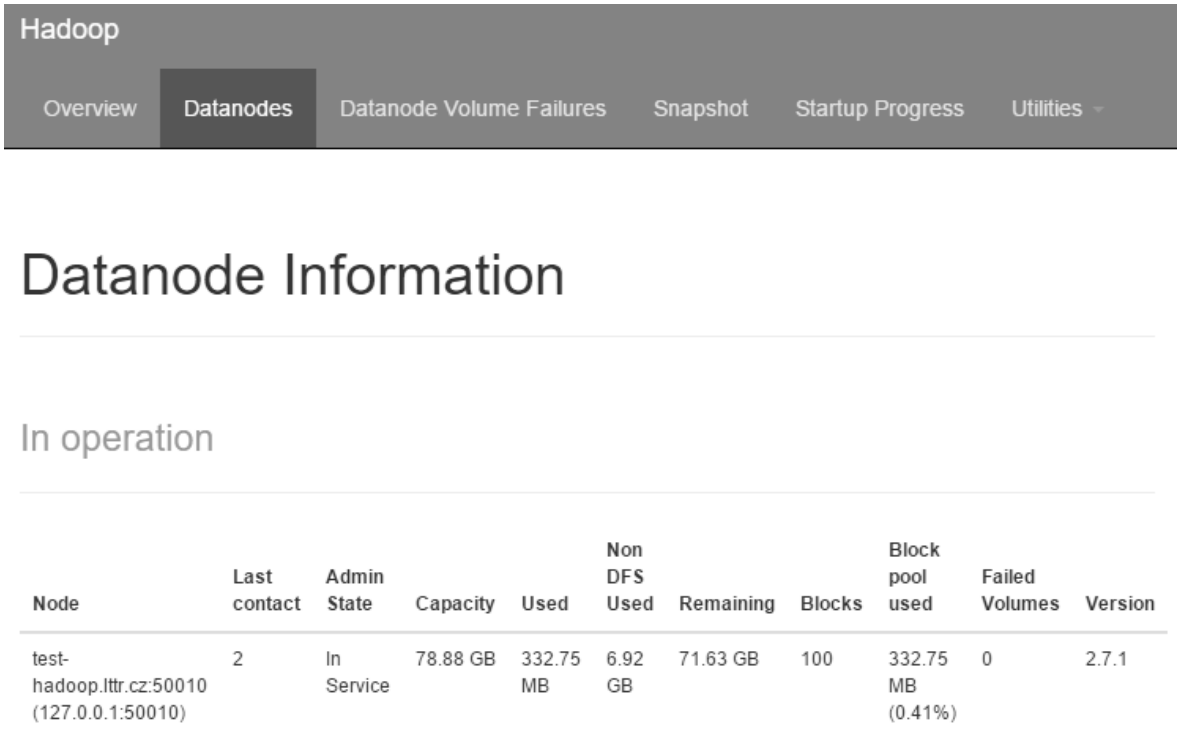
```
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce
   -examples-2.7.1.jar wordcount /data /vystup
```

Do adresáře `/vystup` jsou uloženy výsledky. Pro každý *Reducer* se vytvoří soubor s názvem ve tvaru `part-r-00000`. Pro tento příklad vypíše příkaz

```
hdfs dfs -cat /vystup/part-r-00000
```

slova, která se vyskytují v souboru `text.txt` a jejich četnosti výskytu.

Hadoop poskytuje webové rozhraní se základními informacemi o běžícím prostředí. Na portu 50070 a URL adrese NameNode je přístupné rozhraní zobrazené na obrázku 9 s přehledem běžících datových uzlů. Obrázek je z lokálního prostředí, kde běží jen jeden uzel. Přes webové rozhraní je také možné prohlížet soubory a provádět s nimi základní operace, viz obrázek 10 na následující straně.



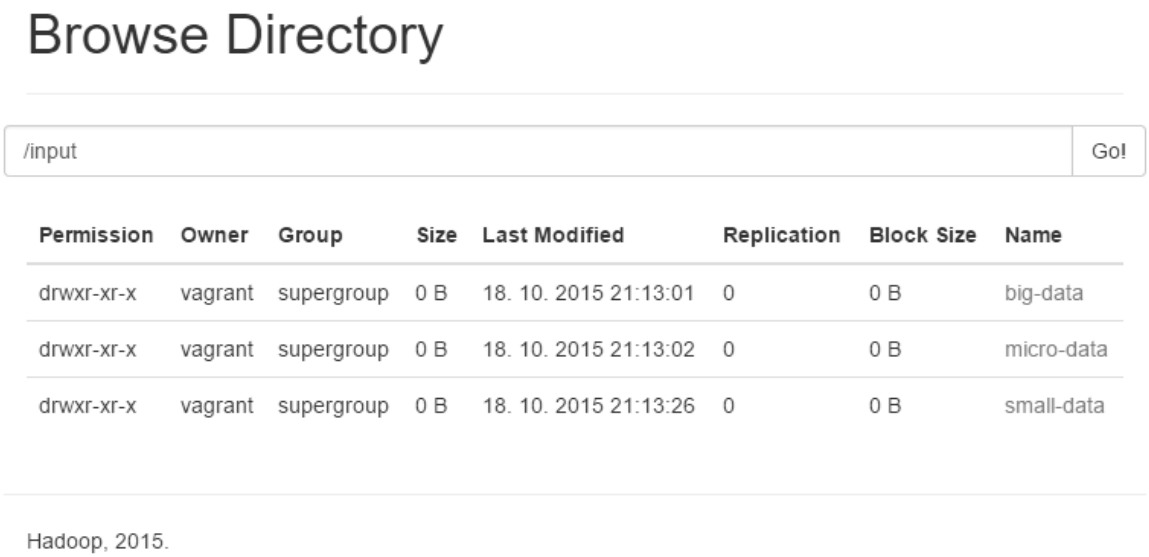
Obrázek 9: Informace o DataNode ve webovém rozhraní. Zdroj: Webové rozhraní Hadoop frameworku.

3.3 Implementace MapReduce úlohy

Jedním z cílů programovacího modelu MapReduce je poskytnout uživateli-programátorovi jednoduché rozhraní pro tvorbu spustitelných úloh. V jazyce Java to v nejjednodušším provedení znamená implementaci tří tříd, a vytvoření *jar* balíčku, který bude předán MapReduce frameworku ke spuštění v klastru.

3.3.1 Základní kostra MapReduce úlohy

V hlavní třídě je nutné nastavit základní konfiguraci úlohy, např. cestu ke vstupním datům a k adresáři pro výstup. Metoda `main` spouští úlohu a čeká na její dokončení. Jednoduchou



Obrázek 10: Výpis souborů na datovém uzlu. Zdroj: Webové rozhraní Hadoop frameworku.

implementaci ilustruje výpis 3 na následující straně, jak je uveden v Hadoop MapReduce dokumentaci [13].

```
public class WordCount {
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");

    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Algoritmus 3: Příklad implementace hlavní třídy MapReduce úlohy. Zdroj: Převzato z dokumentace k Hadoop MapReduce [13].

Základní příklad uváděný v dokumentaci je `WordCount`, program, který spočítá četnost výskytu slov ve zdrojovém textu. K implementaci je kromě hlavní třídy potřeba jen metoda `map` a metoda `reduce`. `Map` musí být metoda přepsaná v potomku třídy `Mapper` a metoda

reduce v potomku třídy Reducer. Princip algoritmu je vysvětlen v kapitole 2.3 na straně 4. Třídy s mapovací a redukční funkcí tvoří společně s hlavní řídící třídou základní kostru programu, který je interpretován v běhovém prostředí MapReduce frameworku. Na výpise 4 je zdrojový kód obou tříd.

```
public class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
      ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString())
       ;
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}

public class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
      Context context
      ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Algoritmus 4: Implementace mapovací a redukční funkce programu WordCount. Zdroj: Převzato z dokumentace k Hadoop MapReduce [13].

3.3.2 Datové typy v Hadoop frameworku

Jedním ze základních konceptů Hadoop frameworku je vzdálené volání metod a z toho vyplývající posílání serializovaných objektů mezi uzly klastru. Serializace je převedení strukturované podoby objektu na posloupnost bitů za účelem odeslání po síti nebo perzistentního uložení. Protože si Hadoop klade na serializaci zvláštní podmínky (např. kompaktnost, in-

teroperabilita s jinými programovacími jazyky apod.), byly vytvořeny datové typy speciálně pro tento účel.

Datové typy v Hadoop frameworku implementují rozhraní `Writable`. Např. `IntWritable` je datový typ pro přirozená čísla. Obaluje základní datový typ `int`. Dalšími příklady mohou být `LongWritable`, `DoubleWritable` nebo `Text`, který obaluje typ `String`. Typ `NullWritable` reprezentuje hodnotu `null`.

Dalším z ústředních konceptů MapReduce modelu je porovnávání datových typů. Porovnatelné datové typy musí implementovat rozhraní `Comparable` nebo odvozené rozhraní `WritableComparable`.

Na výpise 5 na straně 24 je příklad vlastního datového typu, který implementuje rozhraní `WritableComparable`. Metoda `compareTo` určuje vlastnosti porovnání dvou objektů typu `TextIntPair`. Metoda `toString` ovlivňuje, v jaké podobě bude datový typ zapsán např. do výstupního souboru.

3.3.3 Vstup a výstup metod map a reduce

Vstup i výstup do obou metod `map` a `reduce` je definován pomocí dvojice klíč-hodnota. Oba prvky této dvojice mají svůj datový typ. Ten je definován v hlavičce dané třídy, např. Reducer

```
public class IntSumReducer extends Reducer<Text, IntWritable,
    Text, IntWritable>
```

svůj vstup i výstup definuje jako dvojici `<Text, IntWritable>`. Výstupní datové typy celé úlohy jsou nastaveny v hlavní třídě voláním metod `setOutputKeyClass` a `setOutputValueClass`.

3.3.4 Combiner a Partitioner

Třídy `Combiner` a `Partitioner` řídí fázi *kombinování* a *rozdělování* (viz kapitola 2.3 na straně 4). Nastavují se v hlavní třídě pomocí metod `setCombinerClass`, `setPartitionerClass`.

Pro roli `Combiner` se hodí samotný `Reducer`. Musí však být implementovaný tak, aby i při opakovaném spuštění vždy vrátil požadovaný agregovaný výsledek.

`Partitioner` obsahuje metodu `getPartition`, která vrací přirozené číslo označující určitou část. Defaultní implementace se chová jako hašovací tabulka.

Příklad třídy `Partitioner` je uveden v algoritmu 6 na straně 25. Metoda `getPartition` vrací číslo dne – data z map fáze jsou tak rozdělena do skupin podle dne, ke kterému záznam patří. Z čísla dne je operací modulo získán zbytek po dělení proměnnou `numPartitions`. Tím je zajištěno, že počet dílů nebude větší než nejvyšší možný počet.

3.4 Amazon Web Services

Provozování vlastního počítačového klastru je nákladná a náročná činnost. Řada firem proto začala nabízet software nebo platformy jako služby provozované v cloudu, tedy v datovém centru firmy. Ke službě má uživatel vzdálený přístup přes webové nebo programové rozhraní.

Jednou ze společností, které takové služby nabízejí je Amazon. V rámci Amazon Web Services (AWS) existují tři vzájemně se doplňující služby, které umožňují spouštění MapReduce úloh v klastru, který je na vyžádání uživatelem vytvořen v cloudu.

Elastic Compute Cloud (EC2) je služba spočívající v poskytnutí virtuálního serveru s požadovanými vlastnostmi. Poskytnutí může být na vyžádání nebo na předem dané období. EC2 nabízí velkou flexibilitu v možnostech zvýšení výkonu, přidání disků a v doplňkových službách.

Pro stálé, spolehlivé a zároveň snadno dostupné uložení dat slouží Simple Storage Service (S3). Je to jednoduchá služba na ukládání dat do hierarchické struktury.

Elastic MapReduce (EMR) je služba, která umožňuje relativně jednoduše nakonfigurovat parametry MapReduce úlohy a spustit jí v cloudu. Na pozadí jsou v rámci EC2 vytvořeny virtuální servery, na které je nasazen Hadoop a nakonfigurován podle požadavků uživatele. Z S3 jsou stažena vstupní data a zdrojový kód MapReduce úlohy, na virtuálním klastru je úloha spuštěna a výsledky jsou nahrány zpět na S3. Po dokončení úlohy je tak možné nechat celý klastr zrušit. [14]

Díky tomuto principu mohou i jednotlivci a malé společnosti zpracovat velká data během krátké doby a za nízkou cenu.

3.4.1 Typy instancí na AWS

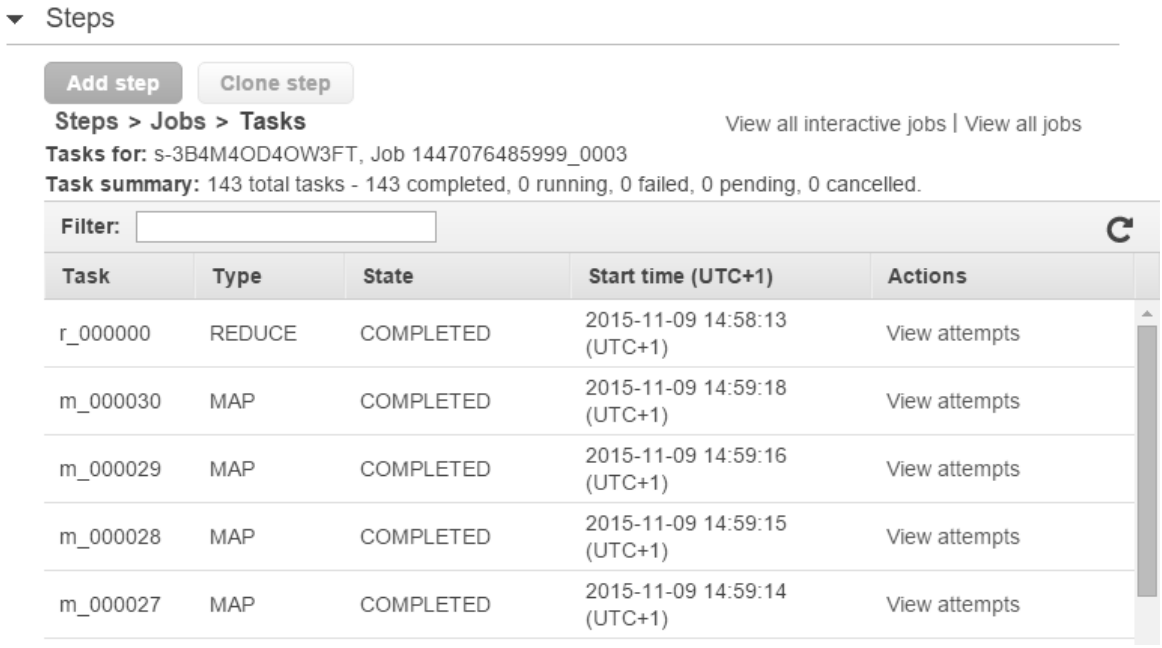
AWS poskytují širokou škálu typových počítačů (instance, uzly klastru). Sestavení instancí je zaměřeno na různé potřeby – vysoký výkon procesoru, velké nároky na paměť nebo prostor na pevných discích. Instance jsou označovány ve tvaru $aX.b$, kde a je typ instance, X označuje pořadové číslo generace instancí a b je velikost nebo výkon instance.

Instance typu t a m jsou určené k obecnému použití, paměť, procesor a síťové připojení mají nastavené vyváženě. Instance typu c jsou optimalizované pro výpočty, obsahují nejsilnější procesory. Instance typu r jsou určené pro paměťově náročné úlohy. Pro aplikace náročné na počty zápisů a čtení z pevných disků jsou optimalizované instance typu i [15].

3.4.2 Webové rozhraní AWS

Každá služba v rámci AWS lze ovládat přes webové rozhraní. Ve službě S3 je možné nahrávat soubory a organizovat je do hierarchické struktury. V rámci služby EC2 je k dispozici průvodce s nastavením typů a počtů požadovaných instancí, operačního systému, který má být na ně nainstalován, dále bezpečnostní a síťová nastavení a volba velikosti a typu úložných prostorů.

Ve webovém rozhraní služby EMR lze prohlížet běžící a ukončené klastry, nebo vytvořit klastr nový. Na obrázku 12 na straně 23 je pohled na základní parametry klastru, jehož běh byl ukončen. V tomto klastru byla nastavena jedna instance typu *m1.medium* jako řídící (*Master*) a čtyři jako výpočetní (*Core*). Z hlediska kompatibility je důležitý parametr *Hadoop Distribution* určující, jaká verze Hadoop frameworku bude nasazena na instancích klastru.



Obrázek 11: Obrazovka s přehledem úkolů MapReduce úlohy. Zdroj: Webové rozhraní AWS EMR.

Uživatelem přidané programy určené ke spuštění na klastru se označují jako kroky (*Steps*). Krokem může být MapReduce úloha předaná do AWS EMR v `jar` archivu. Na obrázku 11 je přehled úkolů a jejich stavů v rámci takto spuštěné MapReduce úlohy.

3.4.3 AWS CLI

Pro opakování obdobných činností je užitečné využít rozhraní příkazové řádky[5] pro přístup ke službám AWS.
Pro nahrání souboru do služby S3 slouží příkaz

```
aws s3 cp mapreduce-job.jar s3://my-bucket/jar/
```

Bucket je základní adresář, který musí mít jedinečný název v rámci celé služby S3. Uživatel ho musí mít předem vytvořený.
Stažení všech souborů určitého adresáře do aktuální složky je možné použitím příkazu

```
aws s3 cp s3://my-bucket/output/ . --recursive
```

Pro vytvoření klastru slouží příkaz

[5] https://aws.amazon.com/cli/

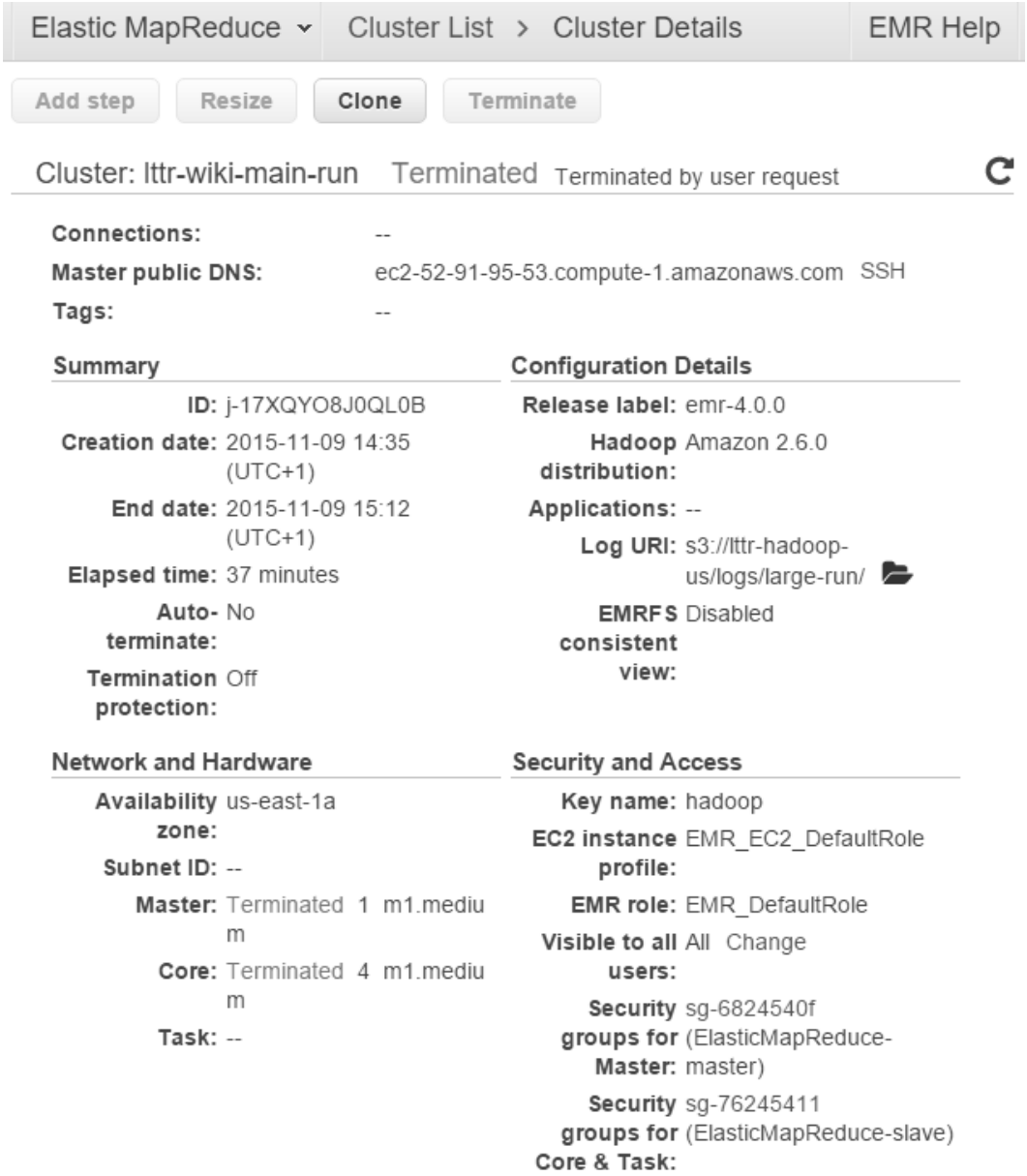
```
aws emr create-cluster
```

který umožňuje přidat velké množství parametrů. Ty budou blíže popsány v následující kapitole.
S vytvořeným klastrem je možné provádět další úkony, např. vypsat jeho stav:

```
aws emr describe-cluster --cluster-id "<id-klastru>"
```

Nebo ho ukončit:

```
aws emr terminate-clusters --cluster-id "<id-klastru>"
```



Obrázek 12: Obrazovka s pohledem na nastavení ukončeného EMR klastru. Zdroj: Webové rozhraní AWS EMR.

```
public class TextIntPair implements WritableComparable<
   TextIntPair> {

  private Text text = new Text();
  private IntWritable integer = new IntWritable();

  public TextIntPair(String text, int integer) {
    this.text.set(text);
    this.integer.set(integer);
  }

  public TextIntPair() {
  }

  public void write(DataOutput out) throws IOException {
    text.write(out);
    integer.write(out);
  }

  public void readFields(DataInput in) throws IOException {
    text.readFields(in);
    integer.readFields(in);
  }

  public int compareTo(TextIntPair textIntPair) {
    int compareValue = this.text.toString().compareTo(
       textIntPair.getText());
    if (compareValue == 0) {
      compareValue = Integer.compare(this.integer.get(),
         textIntPair.getInt());
    }
    return compareValue;
  }

  public String toString() {
    return text + "\t" + integer;
  }
}
```

Algoritmus 5: Příklad implementace vlastního datového typu, který spojuje dva základní datové typy Hadoop frameworku do jednoho objektu. Z výpisu jsou vynechány metody pro přístup k atributům (*get*, *set*) a metody *equals* a *hashCode*. Zdroj: Vlastní zpracování.

```
public class SumByDayPartitioner extends Partitioner<
   NameDayPair, IntWritable> {

  @Override
  public int getPartition(NameDayPair key, IntWritable value,
     int numPartitions) {
    String dayString = key.getDay();
    int dayInt = Integer.parseInt(dayString);
    return (dayInt - 1) % numPartitions;
  }

}
```

Algoritmus 6: Příklad třídy implementující vlastní rozdělovací funkci. Zdroj: Vlastní zpracování.

4 Statistika přístupů na Wikipedii

4.1 Popis řešeného problému

Jako problém k řešení užitím principů MapReduce modelu jsem si zvolil statistiku návštěvnosti Wikipedie. Organizace Wikimedia Foundation průběžně zveřejňuje data [16], která obsahují informace o návštěvách všech stránek Wikipedie a souvisejících projektů. Tato data jsou pro každou hodinu za posledních několik let. Jejich velikost je značná – jeden zkomprimovaný (ve formátu *gz*) soubor má za měsíc červenec 2015 průměrně 90 MB, což je přes 2 GB dat pro každý den, za celý měsíc 65 GB. Data jsou uložena v textových souborech v následujícím formátu:

```
<identifikátor_jazyka>:<projektu> <název_stránky>
  <počet_požadavků_na_stránku> <množství_přenesených_dat>
```

Díky tomuto textovému formátu a charakteru dat[6] je tento problém vhodný řešit pomocí MapReduce úlohy. Data sice neobsahují počty unikátních návštěvníků, i tak ale mohou výsledné statistiky poskytnout zajímavý náhled na využívání Wikipedie.

Za cíl jsem si dal zjistit, kterých 10 stránek česky psané Wikipedie bylo nejvíce navštěvováno v červenci 2015. Pro těchto 10 stránek jsem sestavil grafy jejich návštěvnosti v průběhu celého měsíce.

Pro tvorbu výstupu jsem potřeboval data rozdělená do souborů pro jednotlivé nejnavštěvovanější stránky Wikipedie. Každý výstupní soubor by měl být pojmenovaný podle dané stránky a obsahovat až 31 řádků seřazených podle pořadového čísla dní, ve kterých byla stránka alespoň jednou navštívena.

```
Složka výstupních dat
+-- Soubor dat jedné stránky.txt
|   +-- <název stránky> <číslo dne> <počet požadavků>
|   +-- ...
+-- Soubor dat jiné stránky.txt
|   +-- <název stránky> <číslo dne> <počet požadavků>
|   +-- ...
+-- ...
```

Pro řešení problému jsem zvolil následující postup:

1. Navrhnout strategii, jak data získat a jakým postupem je zpracovávat
2. Získat data a připravit testovací vzorky dat
3. Implementovat úlohu pomocí skriptu pro `bash` a změřit rychlost zpracování dat

[6] Soubory jsou komprimované ve formátu *gz*. Hadoop framework vstupní soubory v tomto komprimačním formátu automaticky dekomprimuje.

4. Připravit vývojové prostředí pro Hadoop na lokálním počítači
5. Implementovat úlohu pomocí Hadoop MapReduce
6. Nakonfigurovat a spustit úlohu na klastru (AWS EMR), změřit rychlost zpracování testovacího vzorku a kompletních dat
7. Porovnat doby trvání běhu MapReduce úlohy a skriptu a porovnat přístupy k práci s daty
8. Zpracovat data a vytvořit grafy

4.2 Příprava vstupních dat

Jako referenční počítač pro provedení testů a hlavního běhu MapReduce úlohy, jsem zvolil EC2 instanci *m1.medium*. Je to jedna ze základních instancí pro všeobecné použití, dostupná jako samostatný server na AWS EC2 nebo jako uzel klastru v EMR. Obsahuje jedno jádro o frekvenci 2 GHz a 4 GB paměti.

Vstupní data jsem potřeboval dostat do AWS S3 (odkud jsou dostupná úloze běžící na EMR) a na samostatnou instanci *m1.medium*, na které jsem chtěl otestovat zpracování dat pomocí skriptu.

Data jsem stáhl pomocí programu `wget`. První příkaz v algoritmu 7 stáhne html stránku s absolutními odkazy. Druhý příkaz extrahuje URL adresy a uloží je do jiného souboru. Třetí příkaz stáhne data z uložených adres.

```
wget --convert-links http://dumps.wikimedia.org/other/
   pagecounts-raw/2015/2015-07 -O index.html
cat index.html \
  | sed -n "/href/ s/.*href=\"\(.*\)\".*/\1/gp" \
  | grep "gz$" > urls.txt
wget -i urls.txt
```

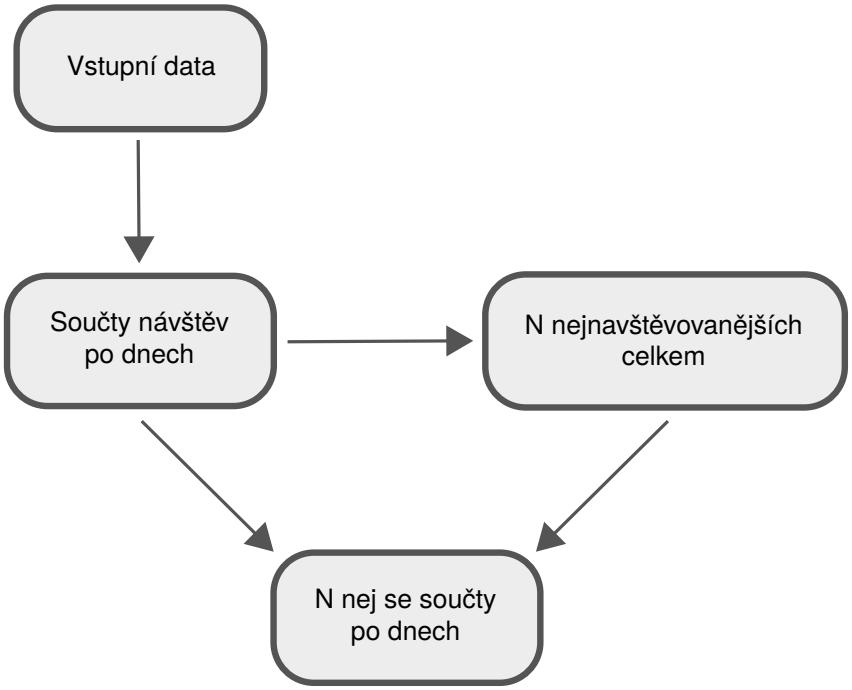
Algoritmus 7: Příkazy pro automatizované stažení vstupních dat. Zdroj: Vlastní zpracování.

Abych mohl porovnávat vliv dat na běh úloh připravil jsem si několik testovacích sad. Všechny velikosti jsou udávané pro data zkomprimovaná v *gz* formátu.

- **micro-data** 4 soubory ze 2 dní jen s několika řádky (8 kB)
- **small-data** 4 kompletní soubory ze 2 dní (320 MB)
- **medium-data** 16 souborů ze 4 dní (1,1 GB)
- **double-medium-data** zduplikovaná medium-data (2,2 GB)
- **large-data** 48 souborů ze 2 dní – všechny hodiny (3,8 GB)
- **big-data** Všechny soubory (744) z jednoho měsíce – 7/2015 (65 GB)

Skriptem i pomocí MapReduce úlohy jsem problém řešil ve třech krocích (obrázek 13):

1. Sečíst počty návštěv jednotlivých stránek v každém dni (*sumy po dnech*)
2. Sečíst počty návštěv jednotlivých stránek za celý měsíc a vybrat z nich 20 nejnavštěvovanějších (*celková suma*)
3. Pro celkově nejnavštěvovanější stránky nalézt návštěvnost těchto stránek v každém dni a seřadit data chronologicky (*nej stránky po dnech*)



Obrázek 13: Schéma postupu zpracování dat. Zdroj: Vlastní zpracování.

4.3 Řešení pomocí skriptu

Výpočty pomocí distribuovaného systému jsem chtěl porovnat s jiným, pokud možno jednoduchým přístupem. Zvolil jsem řešení pomocí skriptu pro `bash`, který je spustitelný na každém linuxovém počítači.

Skript implementuje tři kroky zmíněné výše. Klíčové části skriptu doprovázené komentáři jsou v algoritmu 8 na následující straně. Celý skript je v elektronické příloze této práce.

```
# 1. klíče v~asociativním poli reprezentují dny
for key in ${!input_files[@]}
do
  # všechny soubory pro daný den se rozbalí na stdout
  gunzip -c ${input_files[$key]} |
    # pro řádky, které vyhovují kritériím, spočítá sumu návštěv
    # pro každou stránku
  awk '$1=="cs" && length($2)<120 && NF>3 { a[$2] += $3 }
    END { for (i in a) print i,a[i] }' |
    # setřídí a uloží do souborů po dnech
  sort > $SUM_DIR/$key
done


# 2. z~výsledků vytvoří celkovou sumu pro každou stránku
cat $SUM_DIR/* |
awk ' { a[$1] += $2 } END { for (i in a) print i,a[i] } ' |
# setřídí podle hodnot ve druhém sloupci a uloží jen 20 záznamů
# s nejvyšší hodnotou
sort -r -n -k 2 |
head -20 > $TOTAL_DIR/total


# 3. setřídí nejnavštěvovanější stránky, budou tak ve stejném
# pořadí, jako záznamy po první části
declare -a top_pages=($(awk '{print $1}' $TOTAL_DIR/total |
   sort))
for day_file in $SUM_DIR/*
do
  top_pages_counter=1
  day_number=${day_file: -2}
  while read line # pro každou stránku a sumu jejích návštěv
  do
    line_array=($line)
    page_name=${line_array[0]}
    page_count=${line_array[1]}
    top_page=${top_pages[$top_pages_counter]}
    # otestuje, jestli je stránka mezi nejnavštěvovanějšími
    if [ "$top_page" = "$page_name" ]
    then
      # pokud ano přidá stránku a~návštěvnost do souboru
      echo $day_number" "$page_count >> $PAGES_DIR/$page_name
      # posune se k~další nejnavštěvovanější stránce
      ((top_pages_counter+=1))
    fi
  done < $day_file
done
```

Algoritmus 8: Klíčové části skriptu pro výpočet návštěvnosti Wikipedie. Zdroj: Vlastní zpracování.

Skript jsem spustil pro každou datovou sadu na instanci *m1.medium.* Výsledné časy jsou uvedené v tabulce 1. Z tabulky je patrné, že všechny fáze se s velikostí dat čím dál více zpomalují. Pro menší vzorky dat je sice růst pomalejší než lineární, pro kompletní sadu dat ale roste mnohem rychleji. U sady dat *double-medium* (ve které jsou obsažena *medium* data dvakrát) je dobře vidět, že data jsou po první fázi stejná jako v případě *medium* sady.

	small	**medium**	**double-medium**	**large**	**big**
rel. velikost		*4x small*	*2x medium*	*4x medium*	*16x large*
sumy po dnech	35 s	133 s	272 s	443 s	12 728 s (03:32:08)
celková suma	2 s	3 s	3 s	5 s	38 s (00:00:38)
nej stránky	13 s	30 s	31 s	60 s	1 009 s (00:16:49)
	50 s	166 s	306 s	508 s	13 775 s (3:49:35)

Tabulka 1: Výsledky běhu skriptu pro různé sady dat. Zdroj: Vlastní zpracování.

4.4 Implementace Hadoop MapReduce úlohy

Pro spouštění MapReduce úlohy na lokálním počítači jsem využil postupu, který jsem popsal v kapitole 3.2.1 na straně 14. Pro implementaci jsem použil vývojové prostředí Eclipse a pro vytváření *jar* archivu nástroj Maven.

4.4.1 Využití Mavenu pro sestavení jar archivu

Maven je řízen obsahem souboru `pom.xml` umístěném v kořenovém adresáři projektu. Na výpise 9 na následující straně jsou uvedené závislosti, které je třeba přidat do `pom.xml` aby se *jar* archiv správně sestavil.
Příkazem

```
mvn install
```

spuštěným v adresáři se souborem `pom.xml` Maven vytvoří požadovaný *jar* soubor v adresáři *target*.

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-core</artifactId>
  <version>1.2.1</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>2.7.1</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-hdfs</artifactId>
  <version>2.7.1</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-common</artifactId>
  <version>2.7.1</version>
</dependency>
```

Algoritmus 9: Závislosti Hadoop frameworku uvedené v `pom.xml`. Zdroj: Vlastní zpracování.

Maven také zajistí, že *Manifest* jar archivu obsahuje definici spustitelné třídy a tu nebude nutné zadávat jako argument na příkazové řádce.
Bez Mavenu nebo jiného build nástroje je možné zkompilovat a spustit MapReduce program následovně:

```
javac -classpath `yarn classpath` WordCount.java
jar -cvf word-count.jar WordCount.class
hadoop jar word-count.jar WordCount /data/text.txt /output
```

Příkaz `yarn classpath` se expanduje na cesty k *jar* souborům Hadoop frameworku.

4.4.2 Struktura zdrojového kódu MapReduce úlohy

Obdobně jako skript i MapReduce úloha je sestavena ze tří částí. Využívá vlastnosti MapReduce modelu a Hadoop frameworku, který umožňuje snadné zřetězení několika pod-úloh.

První *map* funkce odfiltruje záznamy, které jsou pravděpodobně poškozené, nebo jinak nezajímavé a přizpůsobí záznamy tak, aby šlo snadno načíst den, ke kterému záznam patří. Funkce *getPartition* poté rozdělí záznamy po dnech – pořadové číslo dne je číslem části. Při zpracování dat za celý měsíc (červenec) tak vznikne 31 částí, které jsou zpracovány 31 funkcemi *reduce*. Ty sečtou návštěvnosti u každé stránky a uloží mezivýsledky.

Druhá pod-úloha má za úkol sečíst mezivýsledky seskupené podle jména stránky. *Map* funkce vybere z každého záznamu jméno stránky a sumu návstěv za daný den. *Reduce* funkce

spočítá celkovou sumu návštěv pro každou stránku ale do dalších mezivýsledků již uloží jen *N* stránek s nejvyšší celkovou návštěvností.

Funkce *map* ve třetí pod-úloze načte seznam N stránek s nejvyšší návštěvností a seznam stránek a jejich návštěvnosti pro každý měsíc. Stránky, které jsou mezi celkově nejnavštěvovanějšími pošle spolu s označením dne a počtem návštěvníků v daném dni *reduce* funkci. Ta již jen dekóduje název stránky, protože znaky jsou ve vstupních souborech URL-zakódované.

Tabulka 2 obsahuje přehled vstupů, výstupů a jejich datových typů v jednotlivých pod-úlohách. Sloupec M/R označuje páry, které se týkají vstupů do funkce *map* (*>M*) výstupů z *map* a zároveň vstupů do *reduce* funkce (*M>R*) a výstupů z *reduce* funkce (*R>*).

Úloha	M/R	Páry (klíč, hodnota)	Datové typy
sumy po dnech	*>M*	(soubor, obsah-souboru)	<Object, Text>
	M>R	(stránka+den, počet)	<NameDayPair, IntWritable>
	R>	(stránka+den, počet)	<NameDayPair, IntWritable>
celková suma	*>M*	(soubor, obsah-souboru)	<Object, Text>
	M>R	(stránka, počet)	<Text, IntWritable>
	R>	(stránka+pořadí, počet)	<TextIntPair, IntWritable>
nej stránky	*>M*	(soubor, obsah-souboru)	<Object, Text>
	M>R	(stránka+den, počet)	<NameDayPair, IntWritable>
	R>	(stránka+den, počet)	<NameDayPair, IntWritable>

Tabulka 2: Přehled vstupních a výstupních datových typů jednotlivých MapReduce pod-úloh. Zdroj: Vlastní zpracování.

4.5 Běh MapReduce úlohy v klastru

Pro spouštění úloh se mi osvědčila kombinace příkazové řádky s doplňkovým webovým rozhraním pro kontrolu stavu výpočetního klastru. Příkaz na příkazové řádce umožňuje jednodušší kontrolu správnosti a zjednodušuje opakování. Webové rozhraní zase poskytuje přehledný monitoring klastru.

Pro hlavní běh MapReduce úlohy nad kompletními daty jsem použil následující definici.

```
aws emr create-cluster
--name "wiki-main-run"
--instance-groups   InstanceGroupType=MASTER,InstanceCount=1,
   InstanceType=m1.medium
  InstanceGroupType=CORE,InstanceCount=16,InstanceType=m1.
     medium
```

```
--use-default-roles
--ec2-attributes KeyName=hadoop
--release-label emr-4.0.0
--applications Name=Hadoop
--steps Type=CUSTOM_JAR,
  Name="Wikipedia visitors",
    ActionOnFailure=CONTINUE,
    Jar=s3://<bucket>/jar/wikipedia-visitors.jar,
    Args=["s3://<bucket>/input/big-data", "s3://<bucket>/output
       "]
--no-auto-terminate
--log-uri s3://<bucket>/logs
--enable-debugging
```

Pro spuštění na příkazové řádce ve Windows musí být příkaz celý na jedné řádce:

```
aws emr create-cluster --name "wiki-main-run" --instance-groups
    InstanceGroupType=MASTER,InstanceCount=1,InstanceType=m1.
   medium InstanceGroupType=CORE,InstanceCount=16,InstanceType=
   m1.medium --use-default-roles --ec2-attributes KeyName=
   hadoop --release-label emr-4.0.0 --applications Name=Hadoop
   --steps Type=CUSTOM_JAR,Name="Wikipedia visitors",
   ActionOnFailure=CONTINUE,Jar=s3://<bucket>/jar/wikipedia-
   visitors.jar,Args=["s3://<bucket>/input/big-data", "s3://<
   bucket>/output"] --no-auto-terminate --log-uri s3://<bucket
   >/logs --enable-debugging
```

Příkaz vytvoří jednu instanci typu MASTER (pro Name Node, Resource Manager) a 16 výkonných instancí typu CORE, všechny *m1.medium*. Na klastr nasadí Hadoop s danou verzí a spustí *jar* soubor se zadanými argumenty pro vstupní data a výstupní složku. Díky `--no-auto-terminate` zůstane klastr i po dokončení úlohy v chodu. Logy ze všech serverů jsou po dokončení kroků staženy a uloženy na zadanou cestu v S3.
Webové rozhraní Resource Manageru a dalších komponent je dostupné i v AWS EMR. Následující příkaz vytvoří síťový tunel, díky kterému je možné se na webové rozhraní hlavního serveru klastru připojit. Soubor se soukromým klíčem *.pem* je k získání přes administrační rozhraní AWS účtu.

```
aws emr socks --cluster-id "<cluster-id>" --key-pair-file "<*.
   pem soubor>"
```

Job History server poskytuje užitečné pohledy pro zpětnou analýzu proběhlých MapReduce úloh. Na obrázku 14 na následující straně jsou vidět tři úlohy mého MapReduce programu,

spuštěné s daty z *medium* data setu, který obsahuje 16 souborů, proto bylo vytvořeno 16 *Map* úkolů v první úloze.

Obrázek 14: Obrazovka Job History serveru. Zdroj: Webové rozhraní Hadoop JobHistory serveru.

Dalším nástrojem pro analýzu běhu MapReduce úloh jsou logy. Na výpise 10 na straně 37 je část logu zaznamenaná při běhu MapReduce úlohy s datovou sadou *big* v klastru. Jedná se o druhou ze tří pod-úloh, která počítá celkovou sumu návštěv na všech stránkách. V logu je kromě záznamu o průběhu mapovací a redukční fáze několik užitečných čítačů. Například `S3 : Number of bytes read` udává množství dat načtených z úložiště S3. Číslo 296 tisíc bytů odpovídá velikosti výstupu z předchozí pod-úlohy. Počet map úkolů – 31, byl odvozen od stejného počtu vstupních souborů. Podle záznamu `Map input records` bylo v mapovací fázi zpracováno 10 080 964 řádků. Výstupních řádků bylo jen 20 (výstupem této fáze má být *N* nejnavštěvovanějších stránek) jak je uvedeno u `Reduce output records`. Těchto 20 řádků mělo velikost 599 bytů, které byly zapsány na výstupu.

Doby trvání běhu programu pro sady dat *medium*, *large* a *big* jsou uvedené v tabulce 3. První dva běhy byly spuštěny na 4 výkonných instancích, třetí na 16 instancích.

	medium	**large**	**big**
rel. velikost		*4x medium*	*16x large*
instance	4	4	16
sumy po dnech	314 s	625 s	2 077 s
celková suma	167 s	155 s	101 s
nej stránky	147 s	167 s	69 s
	628 s (10:28)	947 s (15:47)	2 247 s (37:27)

Tabulka 3: Výsledky běhu MapReduce programu pro různé sady dat. Zdroj: Vlastní zpracování.

4.6 Vyhodnocení výsledků

Provedl jsme měření doby výpočtu pomocí skriptu a MapReduce úlohy. Měl jsem k tomu připraveno několik sad testovacích dat. Všechny výpočty, paralelizované i sekvenční jsem prováděl na stejném typu počítače – EC2 instanci *m1.small* s čistou instalací Linuxu. Datová sada *medium* měla zaokrouhleně 1 GB, *large* 4 GB a *big* 64 GB.

Běh skriptu byl rychlejší pro všechny sady, kromě té největší – kompletní sady dat za jeden měsíc. Tu zpracovával skript 3 hodiny a 49 minut na jednom počítači, zatímco MapReduce úloha 37 minut na 16 počítačích. Jedna šestnáctina z doby běhu skriptu je 14,3 minuty. Paralelní zpracování je tedy pro tuto úlohu asi 2,6krát pomalejší než při ideálním rozdělení výpočetní síly.

Naměřené časy odpovídají předpokladu, že MapReduce model by měl být škálovatelný, tedy při rostoucím množství dat by měl čas růst lineárně, ne rychleji. V tomto případě dokonce čas, který by odpovídal nárůstu dat klesá. Zřejmě dochází k efektivnějšímu využití vnitřního zpracování dat v modelu, který ale zbytečně zdržuje úlohy s malým množstvím dat, nebo špatným poměrem počtů mapovacích a redukčních funkcí vzhledem k počtu uzlů klastru.

V sadě dat *big* je 16krát více dat a úloha běžela na 4krát více instancích než úloha s *large* daty. Dal by se tedy očekávat 4krát delší čas zpracování. Doba zpracování je ale jen 2,4krát pomalejší.

Naopak při sekvenčním zpracování pomocí skriptu by se dal čas jeho běhu s *big* daty odhadnout na 2 hodiny a 15 minut (16krát více) avšak reálná doba byla o hodinu a půl delší.

Úlohu, kterou jsem řešil bych charakterizoval jako náročnou na procesor, relativně méně na diskový prostor a na paměť. Žádná fáze běhu skriptu nebo MapReduce úlohy nemusela zpracovávat celý objem dat najednou, paměť tedy nebyla limitující. U úloh náročnějších na paměť by se doba běhu na jednom počítači mohla kvůli omezené paměti výrazněji prodlužovat s rostoucím množstvím dat.

Údaje jsou shrnuté v tabulce 4. Z měření vyplývá, že pro úlohy takového typu, jako jsem řešil, se nevyplatí používat paralelizované zpracování pomocí MapReduce, pokud jsou vstupní data jen v řádech jednotek GB.

	medium	**large**	**big**
skript	2:46	8:28	3:49:35
MapReduce	10:28	15:47	37:27

Tabulka 4: Přehled doby trvání běhu skriptu a MapReduce úlohy. Údaje jsou ve formátu *hodiny:minuty:vteřiny*. Zdroj: Vlastní zpracování.

4.6.1 Grafická podoba výstupu úlohy

Grafy počtu návštěv deseti nejžádanějších stránek jsou na obrázku 15 na straně 38. Na ose x jsou dny července 2015, na ose y počty návštěv stránky v daném dni. Grafy jsou zobrazeny v abecedním pořadí.

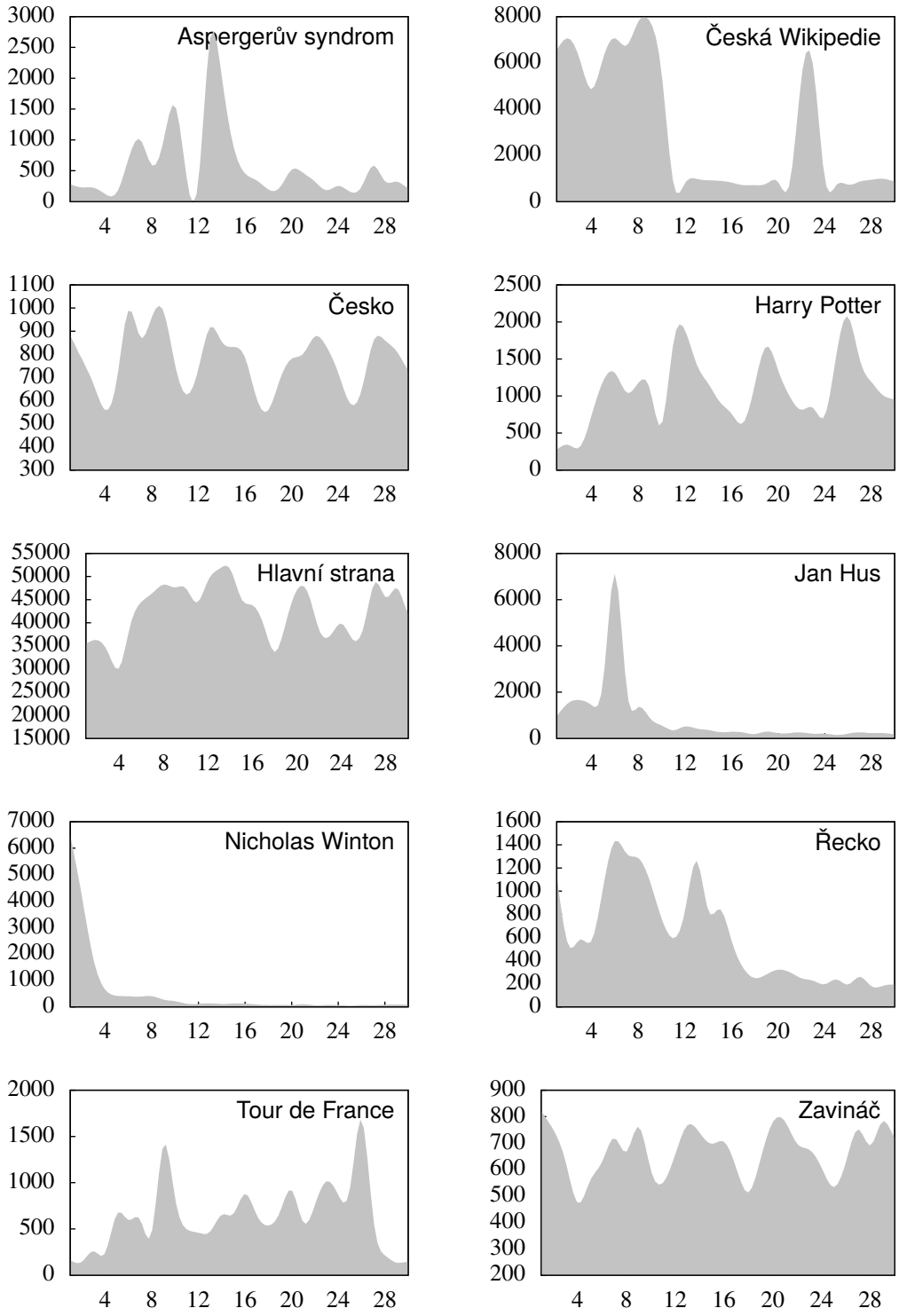
Takto reprezentovaná data o návštěvnosti přinášejí vhled do zajímavých souvislostí. Např. 1. července 2015 zemřel britský burzovní makléř a humanitární pracovník Nicholas Winton ve věku 106 let. Velká poptávka po informacích o tomto člověku se projevili jen pár dní po této události, poté zájem zcela upadl.

Zajímavý je také první vrchol grafu stránky Tour de France. Ten signalizuje nějakou událost důležitou pro Čechy – 9. července vyhrál po dvou letech první Čech etapu tohoto slavného závodu. 26. července, kde je druhý vrchol grafu se konala závěrečná etapa závodu. Charakteristický je zájem o Jana Husa, výrazně zvýšený ve státní svátek, jinak se držící na nízké úrovni.

I tato jednoduchá statistika (suma za určité období) přináší informace, které mohou přinést vhled do jinak nepoznaných souvislostí.

```
Running job: job_1447053760776_0002
 map 0% reduce 0%
 map 11% reduce 0%
 map 37% reduce 0%
 map 74% reduce 0%
 map 95% reduce 0%
 map 100% reduce 0%
 map 100% reduce 36%
 map 100% reduce 58%
 map 100% reduce 75%
 map 100% reduce 100%
Job job_1447053760776_0002 completed successfully
Counters: 55
File System Counters
    S3: Number of bytes read=296851639
    S3: Number of bytes written=599
    S3: Number of read operations=0
    S3: Number of large read operations=0
    S3: Number of write operations=0
Job Counters
    Killed map tasks=1
    Launched map tasks=32
    Launched reduce tasks=1
    Data-local map tasks=32
    Total time spent by all map tasks (ms)=1083034
    Total time spent by all reduce tasks (ms)=42086
Map-Reduce Framework
    Map input records=10080964
    Map output records=10080964
    Map output bytes=285867663
    Input split bytes=4185
    Combine input records=0
    Combine output records=0
    Reduce input groups=1409125
    Reduce shuffle bytes=152247403
    Reduce input records=10080964
    Reduce output records=20
    Spilled Records=20161928
    Shuffled Maps =31
    Failed Shuffles=0
    Merged Map outputs=31
File Input Format Counters
    Bytes Read=296851639
File Output Format Counters
    Bytes Written=599
```

Algoritmus 10: Část logu z běhu MapReduce úlohy v klastru při zpracování datové sady *big*. Zdroj: Vlastní zpracování.



Obrázek 15: Deset nejnavštěvovanějších stránek české Wikipedie v červenci 2015. Na vodorovné ose jsou dny, na svislé počet návštěv v daném dnu. Zdroj: Vlastní zpracování.

5 Závěr

Mým cílem bylo seznámit se s teoretickými i praktickými aspekty programovacího modelu MapReduce a popsat jeho principy. Na základě nabytých znalostí implementovat funkční příklad paralelizovaného výpočtu statistik přístupů na web. Pro příklad jsem si zvolil výpočet přístupů na web Wikipedie, protože jsem se domníval, že obdržím zajímavé výsledky. To se mi podařilo – z veřejně dostupných, jen částečně strukturovaných dat, jsem pomocí MapReduce získal výsledky, jejichž význam lze interpretovat na reálných událostech. Výpočet je navíc možné snadno opakovat pro obdobná data nebo principielně podobné problémy.

Výhodou použití modelu MapReduce je relativní jednoduchost řešení problému paralelizovaných výpočtů. Vstupní data není (pro určité typy úloh) nutné předem strukturovat nebo vytvářet jejich schéma. Další výhodou je rozdělení řešení problému do několika samostatných map a reduce funkcí, což zlepšuje přehlednost a čitelnost implementace. Umožňuje také efektivní pokrytí unit testy, i když ty jsem při implementaci nevyužil.

Z hlediska rychlosti zpracování dat se podle výsledků z této práce domnívám, že se použití MapReduce modelu vyplatí aplikovat až při vstupních datech v řádech desítek GB a vyšších. Nicméně myšlenkový proces, který probíhá při dekompozici problému s cílem implementovat MapReduce úlohu, je myslím užitečný pro širokou škálu úkolů, jejichž cílem je nějaká transformace dat. Funkcionální přístup MapReduce modelu může přinést lepší vhled do problému i při implementaci zcela odlišným způsobem.

Domnívám se, že jsem splnil všechny cíle určené pro tuto práci. Větší prostor by si však zasloužilo studium a popis různých rozhraní a detailů Hadoop frameworku, konfigurace klastru a implementace různých algoritmů do MapReduce modelu. Na tomto základě bych pak mohl vytvořit komplexnější aplikaci přinášející hodnotnější výsledky než jednoduchá statistika přístupů na web.

Literatura

[1] DEAN, Jeffrey a GHEMAWAT, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, 13 s, Berkeley, CA, USA, 2004. Dostupné z: `http://research.google.com/archive/mapreduce.html`.

[2] DEAN, Jeffrey a GHEMAWAT, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. *Communications of ACM*. 2008, 51, 1, s. 107–113. ISSN 0001-0782. doi: 10.1145/1327452.1327492. Dostupné z: `http://doi.acm.org/10.1145/1327452.1327492`.

[3] MONASH, Curt. *Facebook, Hadoop, and Hive* [online]. DBMS2.com. 11.5.2009. [cit. 4.8.2015]. Dostupné z: `http://www.dbms2.com/2009/05/11/facebook-hadoop-and-hive/`.

[4] TAM, Donna. *Facebook processes more than 500 TB of data daily* [online]. CNET. 22.4.2012. [cit. 4.8.2015]. Dostupné z: `http://news.cnet.com/8301-1023_3-57498531-93/facebook-processes-more-than-500-tb-of-data-daily/`.

[5] LIN, Jimmy a DYER, Chris. *Data-intensive Text Processing with MapReduce*. Morgan & Claypool. 2010. 165 s. G - Reference, Information and Interdisciplinary Subjects Series. Dostupné z: `http://lintool.github.io/MapReduceAlgorithms/index.html`. ISBN 9781608453429.

[6] BARROSO, L.A. a CLIDARAS, J. a HÖLZLE, U. *The Datacenter as a Computer : An Introduction to the Design of Warehouse-Scale Machines*. 2nd. Morgan & Claypool Publishers. 2013. 154 s. Synthesis Lectures on Computer Architecture. doi: 10.2200/S00516ED2V01Y201306CAC024. ISBN 9781627050098.

[7] TURKINGTON, G. *Hadoop Beginner's Guide*. Packt Publishing, Limited. 2013. Packt Open Source : Community Experience Distilled. Dostupné z: `http://books.google.cz/books?id=88iia_wT_NgC`. ISBN 9781849517300.

[8] GRONINGEN, Martijn. *Introduction to Hadoop* [online]. Trifork blog. 4.8.2009. [cit. 4.8.2015]. Dostupné z: `http://blog.trifork.com/2009/08/04/introduction-to-hadoop/`.

[9] Google Technology RoundTable: Map Reduce. In *Youtube* [online], 21.8.2008. [cit. 4.8.2015]. Kanál uživatele Life at Google. Dostupné z: `https://www.youtube.com/watch?v=NXCIItzkn3E`.

[10] GHEMAWAT, Sanjay a GOBIOFF, Howard a LEUNG, Shun-Tak. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 29–43 s, New York, 2003. Dostupné z: http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf.

[11] WHITE, T. *Hadoop: The Definitive Guide*. 3rd edition. O'Reilly Media. 2012. Dostupné z: http://books.google.cz/books?id=Wu_xeGdU4G8C. ISBN 9781449338770.

[12] MCDONALD, K. et al. *Hadoop Fundamentals I* [online]. Big Data University. July 2013. [cit. 4.8.2015]. Version 2. Online course, Lecture 2 - Hadoop Architecture. Dostupné z: http://bigdatauniversity.com/bdu-wp/bdu-course/hadoop-fundamentals-i-version-2/.

[13] *Apache Hadoop* [online]. Apache Software Foundation. 2015. [cit. 20.8.2015]. Project Documentation. Dostupné z: http://hadoop.apache.org/docs/current/.

[14] *What is Amazon EMR?* [online]. Amazon Web Services. 2015. [cit. 20.8.2015]. Management Guide. Dostupné z: http://docs.aws.amazon.com/ElasticMapReduce/latest/ManagementGuide/.

[15] *Amazon EC2 Instances* [online]. Amazon Web Services. 2015. [cit. 9.11.2015]. Dostupné z: https://aws.amazon.com/ec2/instance-types/.

[16] *Page view statistics for Wikimedia projects* [online]. Wikimedia Foundation. 2015. [cit. 20.8.2015]. Maintained by Wikimedia Foundation's Analytics team. Dostupné z: http://dumps.wikimedia.org/other/pagecounts-raw/.

Seznam obrázků

Seznam algoritmů

Seznam tabulek

Přílohy

Přílohou této práce jsou zdrojové kódy programů, které vznikly v průběhu tvorby práce. Přiložené elektronické materiály[7] mají následující strukturu:

- `data` (struktura složek pro uložení dat, složky obsahují URL adresy datových souborů)
- `mapreduce-job` (kořenová složka MapReduce úlohy)
  - `src/main/java/cz/lttr/hadoop` (struktura projektu vyhovující konvenci Mavenu)
    - `pairs` (složka balíčku s vlastními datovými typy – dvojicemi hodnot)
    - `sumbyday` (složka balíčku první fáze úlohy *sumy po dnech*)
    - `topnbyday` (složka balíčku třetí fáze úlohy *nejnavštěvovanější stránky po dnech*)
    - `totaltopn` (složka balíčku druhé fáze úlohy *celková suma*)
- `notes` (složka obsahující užitečné poznámky k běhu skriptu, MapReduce úlohy a k instalaci Hadoop frameworku)
- `scripts` (bash skript vykonávající stejnou práci jako MapReduce úloha)
- `Vagrantfile` (soubor s definicí virtuálního počítače využívaného pro testování)
- `Readme.md` (popis projektu)

[7] Elektronické přílohy jsou také dostupné na autorově Github účtu: https://github.com/lttr/mapreduce-wikipedia-visitors

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Lukáš Trumm**

Obor studia: Informační management (3)
Jméno a příjmení vedoucího práce: **Pavel Kříž**

Název práce:
Paralelizace výpočtů pomocí modelu MapReduce.

Název práce v AJ:
Parallel Computing Using MapReduce Model

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Navrhnout a implementovat paralelizovaný výpočet statistiky přístupů na web pomocí MapReduce. Popsat principy tohoto přístupu.

Osnova práce:

- Anotace a obsah
- Úvodní část
  - Úvod
  - Literární rešerše
- Teoretická část
  - Úvod do paralelních systémů
  - Principy modelu MapReduce
  - Využití modelu MapReduce
  - Srovnání modelu MapReduce s jinými přístupy
- Praktická část
  - Možnosti implementace modelu MapReduce
  - Popis implementovaného problému
  - Vysvětlení zvolených postupů
  - Dokumentace
- Závěrečná část
  - Shrnutí výsledků
  - Závěr
  - Seznam literatury
  - Zadání práce

Projednáno dne: 11.10.2013

Podpis studenta

Podpis vedoucího práce