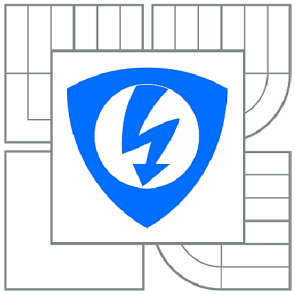
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ŠKÁLOVATELNÉ STROJOVÉ UČENÍ S VYUŽITÍM NÁSTROJŮ HADOOP A MAHOUT

SCALABLE MACHINE LEARNING USING HADOOP AND MAHOUT TOOLS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

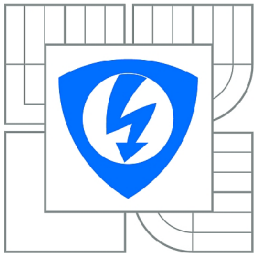
AUTOR PRÁCE — LUKÁŠ KRYŠKE

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADIM BURGET, Ph.D.

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Lukáš Kryške ***ID:*** 119311
Ročník: 3 ***Akademický rok:*** 2011/2012

NÁZEV TÉMATU:

Škálovatelné strojové učení s využitím nástrojů Hadoop a Mahout

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s nástrojem Hadoop a Mahout a stručně srovnejte s obdobnými dostupnými nástroji. Zhodnoťte jeho klady a zápory. Vytvořte demonstrační instalaci a demonstrujte funkčnost a na vybraném příkladu. Zhodnoťte výkonnost a měření zaneste do grafu.

DOPORUČENÁ LITERATURA:

[1] T. White, Hadoop: The Definitive Guide, O'Reilly Media; Original edition (June 12, 2009)
[2] Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters April 2010

Termín zadání: 6.2.2012 ***Termín odevzdání:*** 31.5.2012

Vedoucí práce: Ing. Radim Burget, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Tato bakalářská práce srovnává několik nástrojů pro realizaci škálovatelné platformy strojového učení a popisuje jejich výhody a nevýhody. Dále práce prakticky realizuje funkčnost škálovatelné platformy založené na nástroji Apache Hadoop a zabývá se měřením výkonu samoučícího algoritmu K-Means pomocí knihoven strojového učení Apache Mahout na celkem pěti výpočetních uzlech.

KLÍČOVÁ SLOVA

Hadoop, Mahout, superpočítač, paralelní zpracování dat, strojové učení

ABSTRACT

This bachelor's thesis compares several tools for building a scalable, machine learning platform and describes their advantages and disadvantages. It also practically demonstrates functionality of this scalable platform based on the Apache Hadoop and Apache Mahout tools and measures performance of the K-Means algorithm for total of five computing nodes.

KEYWORDS

Hadoop, Mahout, supercomputer, parallel supercomputing, machine learning

KRYŠKE, L. *Škálovatelné strojové učení s využitím nástrojů Hadoop a Mahout*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2012. 49 s., 2 s. příloh. Bakalářská práce. Vedoucí práce: Ing. Radim Burget, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Škálovatelné strojové učení s využitím nástrojů Hadoop a Mahout jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Tímto chci poděkovat vedoucímu mé bakalářské práce, panu Ing. Radimovi Burgetovi, PhD. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

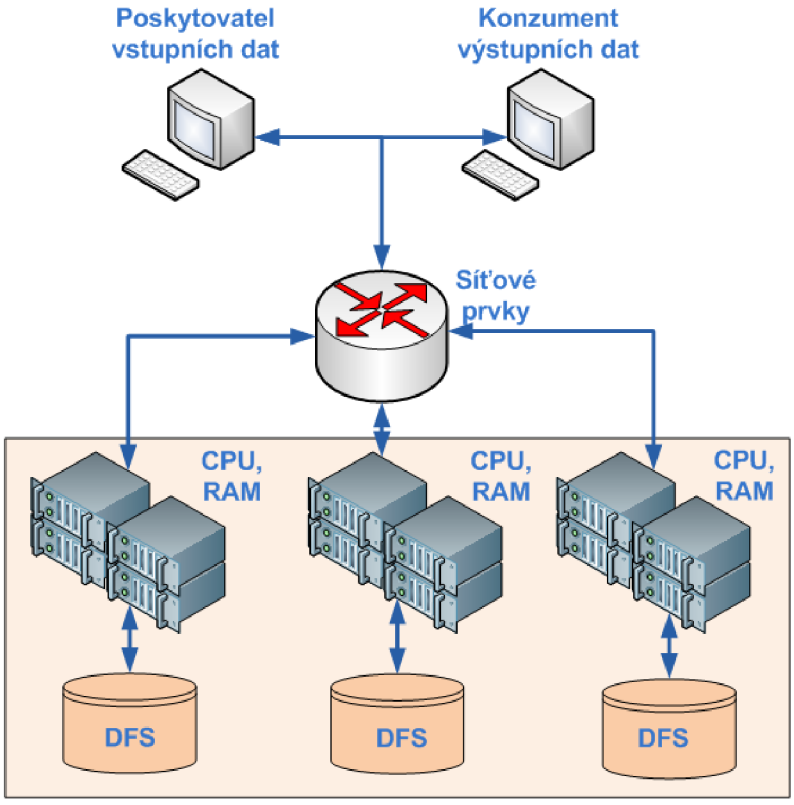
V současné době generuje celosvětová síť internet a různé počítačové systémy velké množství dat. Například New Yorská burza cenných papírů vytváří kolem 1 TB obchodních dat denně [1] a vědecké centrum velkého částicového urychlovače v Ženevě (CERN) vygeneruje za rok kolem 15 PB dat [1]. Tento neustále se zvyšující trend tvorby velkého množství dat vyžaduje vývoj a používání takových nástrojů, které dokáží s takovými objemy dat pracovat, třídit je, analyzovat a dolovat z nich další užitečné informace potřebné ke statistickým výpočtům, k porovnávání a také k aplikaci algoritmů pro strojové učení. Je zřejmé, že tyto úkony nemohou být efektivně prováděny v rámci jednotlivých počítačů a že je potřeba používat nejrůznější platformy, které tyto úkony dokáží distribuovat mezi větší množství počítačů - vykonávat tyto operace souběžně na více výpočetních strojích. Těmito postupy je pak dosaženo časové úspory ve výpočtech a data uložená v těchto distribuovaných systémech jsou také odolnější vůči výpadkům a chybám, protože obsahy jednotlivých uzlů jsou zpravidla replikovány na uzlech ostatních. Knihovny pro algoritmy strojového učení nám pak na těchto datech umožní spouštět určité procedury pro nalezení podobností dat, jejich klasifikaci atd. Cílém této bakalářské práce je seznámit se s nástroji Apache Hadoop a Apache Mahout a srovnat je s obdobnými dostupnými nástroji, včetně vytvoření demonstrační instalace a změření jejího výkonu.

Hlavním přínosem této práce je praktická realizace škálovatelné platformy pro algoritmy strojového učení, které jsou uplatnitelné v široké škále oborů napříč různými odvětvími pro automatizované klasifikování a shlukování dat a také prokázání, že počet výpočetních uzlů v takovéto platformě má významný vliv na zkrácení doby výpočtů samotných. V rámci této praktické realizace byl nástroj Hadoop zprovozněn v distribuovaném módu na pěti serverech, byly implementovány knihovny pro podporu strojového učení a nakonec byl změřen a zanesen do grafu výkon tohoto celku.

Jednotlivé kapitoly seznamují čtenáře s nástroji pro realizaci škálovatelné platformy, s instalací nástroje Hadoop a jeho konfigurací, s nástrojem pro strojové učení Mahout a s provedenými výkonnostními testy celého řešení.

1 NÁSTROJE PRO REALIZACI ŠKÁLOVATELNÉ PLATFORMY

Škálovatelnou platformu je možné realizovat několika způsoby. Jako velmi perspektivní se jeví metoda zpracování paralelních výpočtů použitím grafických karet ve standardních počítačích [10], byť se zatím jedná o relativně nákladné řešení, minimálně co se týče pořizovacích nákladů. Tato bakalářská práce se zabývá takovými platformami, které je možné provozovat na komoditním hardware. Superpočítač z komoditního hardware lze sestavit vzájemným propojením více počítačů běžnými síťovými technologiemi. Vznikne tak lehce škálovatelný celek několika výpočetních jednotek. Obrázek 1.1 demonstruje schéma, jak lze navrhnout škálovatelnou platformu pro strojové učení, která obsahuje vstupní počítač, soustavu aktivních síťových prvků pro propojení jednotlivých uzlů této platformy, samotné výpočetní jednotky propojené pomocí distribuovaného souborového systému a výstupní počítač pro čtení výsledných dat.



Obrázek 1.1: Použití několika výpočetních jednotek jako jeden superpočítač

Pro efektivní použití by měl takto vytvořený superpočítač splňovat několik podmínek, kterými jsou zejména dobrá škálovatelnost, efektivní hierarchie výpočetních jednotek a jednotný souborový systém. Schopnost škálovat udává, jak snadno lze v superpočítači provádět změny v počtu a rozložení výpočetních uzlů a to nejen změny hardwarové (připojování a odpojování dalších výpočetních jednotek) ale i změny

softwarové (automatizovaná instalace a konfigurace výpočetní jednotky pro provádění úloh v superpočítači). Dalším požadavkem by měla být efektivní hierarchie výpočetních jednotek, aby bylo dosaženo maximální efektivity a výkonu celého shluku jednotlivých počítačů (clusteru) – to zahrnuje zejména vnitřní rozdělení na samotné výpočetní jednotky a na servery, které dohlíží nad výpočty a mezi tyto výpočetní jednotky rozdělují výpočetní úlohy. Tyto řídící servery by také měly zpracovávat finální výsledky výpočtů a předávat je dále například uživatelům. Posledním z bodů na požadavky pro efektivní superpočítač by měl být jednotný souborový systém – použití jednotného souborového systému eliminuje nutnost použití určitých konverzních nástrojů a formátových adaptérů pro výměnu souborů mezi výpočetními jednotkami, popřípadě řídícími jednotkami a ušetří se tímto značný výpočetní čas.

Jedním z úkolů zadání této bakalářské práce je porovnat nástroj Apache Hadoop s ostatními konkurenčními nástroji pro realizaci superpočítače. Tabulka 1.1 tyto nástroje stručně srovnává a uvádí odkaz na webovou stránku, kde lze o projektu nalézt detailní informace. V posledním sloupci tabulky jsou shrnuty klíčové vlastnosti daného nástroje. Apache Hadoop poskytuje dynamickou a škálovatelnou platformu pro paralelizaci výpočtů, včetně výkonného a vysoce redundantního distribuovaného souborového systému HDFS (Hadoop Distributed File System) a nástroj Apache Mahout implementuje funkce a knihovny strojového učení, zejména knihovny pro klasifikaci, shlukování a doporučování. Protože jsou projekty Hadoop i Mahout vzájemně propojené a tvoří mezi sebou ekosystém, jsou dostupné zdarma včetně zdrojových kódů, dokáží vzájemně spolupracovat a jako celek implementují platformu pro škálovatelné strojové učení, jeví se jako nejvhodnější pro praktickou realizaci v rámci této bakalářské práce. Ostatní nástroje z Tabulka 1.1 poskytují podobnou funkcionalitu jako nástroj Apache Hadoop a některé jej v některých vlastnostech dokonce převyšují, jejich nevýhodou ale je, že standardně neimplementují knihovny strojového učení a jeví se tak pro účel této bakalářské práce jako méně vhodné. Další významnou výhodou nástroje Apache Hadoop je, že jej lze implementovat na komoditní hardware, tzn. na dnešní, naprosto obyčejné počítače.

Tabulka 1.1: Přehled nástrojů pro realizaci superpočítače

Název produktu	**Vývojový web**	**Klíčové vlastnosti**
Disco	http://discoproject.org	Jedná se o produkt používající paradigma MapReduce, který je vyvíjen výzkumným centrem společnosti Nokia. Funguje na principu Master – Slave (tedy pán – otrok), kdy masterovi jsou zasílány úlohy na zpracování dat, který je následně přiděluje dostupným serverům typu Slave. Na Slavech běží procesy Worker, které zpracovávají data a výsledky zasílají zpět serveru typu Master.
Misco	http://www.cs.ucr.edu/~jdou/misco/	Jedná se o MapReduce framework určený především pro mobilní aplikační využití.

Phoenix	http://mapreduce.stanford.edu/	Phoenix je systém, který podporuje používání MapReduce paradigma a je vyvíjen univerzitou Standford. Programovacím jazykem MapReduce aplikací je C++.
Cloud MapReduce	http://code.google.com/p/cloudmapreduce	Jedná se o velmi široce podporovaný MapReduce framework, využívaný firmou Amazon. Jeho vývojáři na uvedené webové stránce proklamují, že Cloud MapReduce je až 60x rychlejší než Apache Hadoop, zejména pak díky své velikosti – pouze cca 3000 řádků zdrojového kódu.
Bashreduce	http://blog.last.fm/2009/04/06/mapreduce-bash-script	Bashreduce je MapReduce implementace, která je tvořena na bázi Bash skriptů. Nemá tak široké možnosti jako Cloud MapReduce nebo Hadoop, na druhou stranu je velmi jednoduchý.
Qizmt	http://code.google.com/p/qizmt	Qizmt je další z MapReduce frameworků, které umožňují zpracovávat výpočty v distribuovaném prostředí. Jeho oficiální distribuce je určena pouze pro operační systémy Microsoft Windows.
Galago's TupleFlow	http://www.galagosearch.org/guide.html	TupleFlow je především indexovací nástroj pro velké objemy dat, který se implementuje pomocí jazyka C++ a který je šířen pod BSD licencí.
Skynet	http://skynet.rubyforge.org/	Skynet je svobodnou implementací paradigma MapReduce dle specifikací Google MapReduce. Programovacím jazykem pro Skynet je Ruby.
Sphere	http://sector.sourceforge.net/	Jedná se o robustní nástroj pro paralelní zpracování velkých dat. Vývojáři tohoto nástroje uvádějí, že je dvakrát až čtyřikrát rychlejší, než Hadoop MapReduce.
Riak	http://riak.basho.com/mapreduce.html	Riak je systém pro distribuovaný supercomputing, primárně zaměřen na zpracování multimediálních dat a tvorbu klientských aplikací pro konzumaci těchto dat.
Starfish	http://rufy.com/starfish/doc/	Starfish je systém pro distribuované programování. Na uvedených internetových stránkách je velmi dobře zdokumentován. Starfish plně ctí MapReduce paradigma. V současné době lze pro Starfish vyvíjet pouze v jazyku Ruby on Rails. Jako úložiště metadat pak slouží opensource databázový stroj MySQL.

Octopy	http://code.google.com/p/octopy/	Octopy je rychlý a snadný nástroj pro vyvíjení MapReduce aplikací po vzoru Starfishe, s tím rozdílem, že je primárně určen pro jazyk Python.
MPI-MR	http://www.sandia.gov/~sjplimp/mapreduce.html	MPI-MR je další implementací zpracování dat pomocí metod Map-Reduce. Obsahuje rozhraní pro programování MapReduce aplikací prostřednictvím programovacího jazyka C/C++ a Python. MPI-MR je dostupné zdarma i se zdrojovými kódy.
Filemap	http://mfisk.github.com/filemap/	Filemap je velmi jednoduchá implementace MapReduce pro zpracovávání velkých objemů dat a která nevyžaduje žádná další běhová prostředí. Taktéž není vyžadována znalost speciálních programovacích jazyků.
Plasma MapReduce	http://projects.camlcity.org/projects/plasma.html	Plasma je distribuovaný souborový systém, který podporuje ukládání dat stylem klíč – hodnota a také podporuje zpracování úloh pomocí MapReduce. Je distribuován pod svobodnou licencí GPL
Mapredus	http://rubygems.org/gems/mapredus	Mapredus je velmi jednoduchý MapReduce framework pro POSIXové operační systémy. Poslední verze nese označení 0.0.6 a byla uvedena v roce 2010. Dá se tedy předpokládat, že tento produkt neprochází příliš aktivním vývojem.
Mincemeat	http://remembersaurus.com/mincemeatpy/	Mincemeat je velmi lehká implementace MapReduce frameworku jazykem Python. Aplikace je celkově velmi malá (cca 13kB). Výpočetní cluster vytvořený pomocí Mincemeatu je velmi tolerantní vůči výpadkům a připojování jednotlivých klientů, ty se tak mohou připojovat a odpojovat aniž by to narušilo zpracovávání dat v clusteru. Veškerá komunikace mezi uzly je otevírána na základě autentifikací, autoři projektu plánují také implementaci šifrování TLS. Mincemeat je plně open – source projekt.

MapReduce Titan	http://www.kitware.com/InfovisWiki/index.php/MapReduce	MapReduceTitan je dalším z MapReduce frameworků, který je šířen jako svobodný software. Výhodou Titanu jsou velmi rozsáhlé funkční možnosti, včetně monitorování celého clusteru pomocí specializovaných dohledových nástrojů. Titan může běžet jak na operačním systému Linux, tak na Windows či Mac OS X. Internetová stránka projektu obsahuje velmi rozsáhlou dokumentaci, takže by nasazení systému mělo být jednoduché a dobře zdokumentované.
GPMR	http://www.idav.ucdavis.edu/research/projects/mgpu_mapreduce	GPMR je další MapReduce knihovna, která řeší zpracování velkých objemů dat. Narozdíl od svých konkurentů se však zaměřuje především na zpracování výpočtů pomocí grafických procesorů. Celý výpočetní cluster tak může tvořit jeden komoditní počítač s více grafickými kartami. Je pravděpodobné, že se jedná o velice perspektivní metodu zpracování dat, neboť zpracováním a dolováním dat pomocí grafických procesorů se už zabývají také někteří výrobce samotných grafických procesorů [10].
Elastic Phoenix	https://github.com/adamwg/elastic-phoenix	Jedná se o MapReduce framework, který je určený pouze pro 32bitové a 64bitové architektury operačního systému GNU/Linux. Jeho výhodou je výborná škálovatelnost. Jednotlivá výpočetní vlákna Elastic Phoenixu pracují v paměti jako samostatné procesy a navzájem komunikují pomocí speciálních struktur.

2 KONCEPCE NÁSTROJE APACHE HADOOP

Projekt Hadoop je v současné době zastřešován společností Apache Software Foundation a pokrývá několik subprojektů (tvoří deštníkovou architekturu), ty nejvýznamnější z nich jsou Core, Avro, MapReduce, HDFS, Pig, HBase, ZooKeeper, Hive a Chukwa [1]:

Core (jádro) je sada utilit, konfiguračních souborů a skriptů, které tvoří funkcionalitu nástroje Hadoop. V základním provedení je v jádře nástroje Hadoop zakomponován také ovladač distribuovaného systému souborů HDFS. Hadoop se dále dělí podle konfigurace na uzly (nody) a podle účelu jejich použití je rozdělujeme na jmenné uzly (namenode) a datové uzly (datanode). Namenody jsou servery, na kterých probíhá zadávání výpočetních úloh – tedy zadavetelé výpočtů a jsou také hlavním serverem distribuovaného souborového systému. Namenody jsou tak zadavateli a zároveň kontrolory výpočetních úloh. Datanody jsou pak servery, na kterých probíhají samotné výpočetní operace a které uchovávají replikace dat v souborovém systému HDFS. Na serveru nakonfigurovaném jako namenode navíc běží modul jobtracker, která má za úkol monitorovat chod MapReduce aplikací v reálném čase. Zároveň služba poskytuje informace o MapReduce aplikacích administrátorům systému prostřednictvím URL `http://jobtrackerserver:50030/`, kde jobtrackerserver nahradíme skutečným názvem serveru s aktivním jobtrackerem, popřípadě na jiné adrese, kterou může administrátor nástroje Hadoop definovat v konfiguračním souboru `conf/core-site.xml`. Na této dohledové stránce můžeme nalézt informace o aktuálním zpracování aplikací (vyjádřeno v procentech), dále pak množství zpracovaných dat (rozděleno do mapovací a redukční fáze), seznam uzlů, které data zpracovaly atd.

Avro dovoluje vytvářet rozsáhlé datové struktury a schémata pro serializaci dat. Apache Foundation zvažuje take jeho nasazení jako formát pro kapacitně velké soubory. [1]

MapReduce je paradigmou pro distribuované zpracování dat, především pak pro tvorbu rozsáhlých dotazů pro dolování dat a provádění výpočtů na datech uložených v distribuovaném souborovém systému ale i mimo něj. Detailněji je MapReduce popsáno v kapitole Struktura MapReduce této bakalářské práce.

HDFS je distribuovaný systém souborů. Detailněji je tento souborový systém popsán v kapitole Popis souborového systému HDFS této bakalářské práce.

Pig je skriptovací jazyk pro tvorbu vlastních datových struktur z výstupních dat MapReduce aplikací. Používá vysokoúrovňový, vlastní skriptovací jazyk a kompilátor.

HBase – jedná se o sloupcově orientovaný, distribuovaný databázový stroj, který jako fyzické úložiště používá HDFS. Má využití především v aplikacích, kdy je potřeba velmi často přistupovat k datům tvořícím velké datasety v reálném čase. HBase byl pro svůj vývoj inspirován specifikací produktu BigTable od společnosti Google. [1]

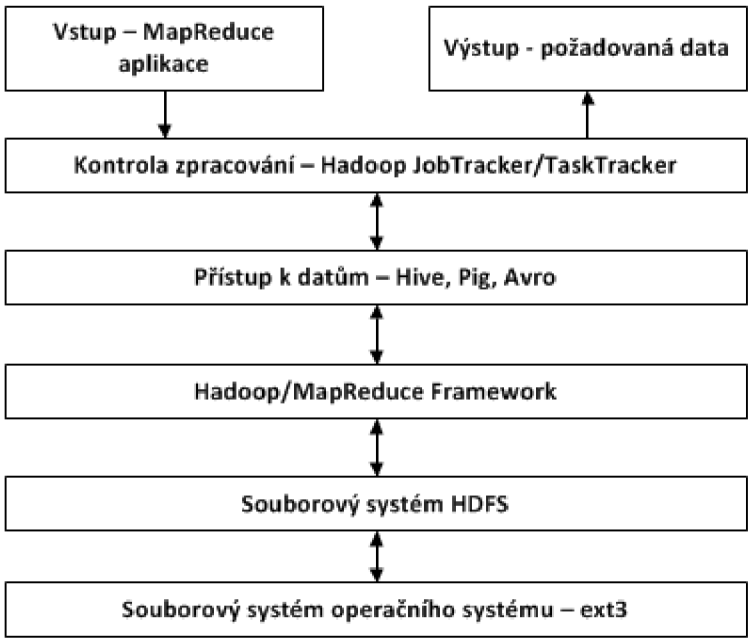
ZooKeeper je nástroj, který koordinuje distribuované prostředí. Stará se zejména

o synchronizaci uzlů v distribuovaném prostředí. [1]

Hive se stará o data v HDFS systému souborů a poskytuje vrstvu pro manipulaci s těmito daty pomocí jazyka podobnému jazyku SQL (Structured Query Language). Hive umožňuje také běžné operace s datasety, jako jsou spojování (merge), mazání atd. [1]

Chukwa – jedná se o nástroj pro sběr logů (souborů zaznamenávajími stavy různých modulů nástroje Hadoop) a jejich následnou analýzu. Tento nástroj také provádí monitorování příslušných částí systému na základě požadavků administrátorů. [1]

Obrázek 2.1 znázorňuje architekturu nástroje Apache Hadoop. Základem je komoditní hardware s operačním systémem Linux a souborovým systémem operačního systému ext3. Na něm je postavena další vrstva v podobě souborového systému HDFS k němuž přistupují MapReduce aplikace prostřednictvím Hadoop MapReduce frameworku – sadou předpřipravených knihoven a funkcí. Moduly JobTracker a TaskTracker slouží k zadávání výpočetních operací od uživatele směrem k výpočetnímu klastru a po výpočtu zpracovávají výstup MapReduce aplikací a předávají jej zpět uživateli.



Obrázek 2.1: Architektura nástroje Appache Hadoop

3 HADOOP FRAMEWORK

Hadoop Framework je soubor Java tříd, které obsahují řadu funkcí, jež jsou používány pro celý provoz nástroje Hadoop, a obsahují také funkce pro Mapper a Reducer. Dále pak obsahují takové funkce, které programátorům značně ulehčují práci tím, že se programátor nemusí zabývat sestavováním běžných primitivních ale i pokročilejších funkcí, protože jsou již zkompilovány a zahrnuty v distribuci nástroje Apache Hadoop. Tabulka 3.1 tyto funkce popisuje.

Tabulka 3.1: Java třídy Hadoop Frameworku

Java třídy nástroje Hadoop	Popis tříd a jejich metod
`org.apache.hadoop`	Balíček HadoopVersionAnnotation, který detekuje verzi nástroje Hadoop.
`org.apache.hadoop.classification`	Vrací informace o zkoumaných balíčcích, třídách a metodách v Hadoopu.
`org.apache.hadoop.conf`	Balíček, který obsahuje metody pro práci s konfiguračními soubory. Obsahuje vstupně/výstupní funkce, získávání a zápis řetězců do souborů a také práci se soubory `XML`.
`org.apache.hadoop.filecache`	Třída implementuje speciální dočasné úložiště (cache), které lze používat jak ve standalone módu (jeden server), tak i v distribuovaném prostředí Hadoop clusteru. Obsahuje taky nástroj pro správu této paměti prostřednictvím nástroje DistributedCacheManager.
`org.apache.hadoop.fs`	Obsahuje třídy pro práci s distribuovaným souborovým systémem. Mezi funkce těchto tříd patří práce s bloky a soubory. Dostupné jsou také funkce pro práci s kontrolními součty (MD5, CRC32).
`org.apache.hadoop.fs.ftp`	Tato třída obsahuje metody pro práci s FTP serverem.
`org.apache.hadoop.fs.kfs`	Obsahuje metody pro práci se souborovým systémem Kosmos File System.
`org.apache.hadoop.fs.permission`	Tato třída obsahuje metody, které dokáží parsovat příkaz `chmod` v operačních systémech standartu POSIX (příkaz slouží ke změně oprávnění v souborovém systému) a dokáží jej aplikovat na současně nainstalovaný souborový systém. Dále zde najdeme další metody pro práci s oprávněními adresářů a souborů, stejně tak jako výstupní funkce pro kontrolu oprávnění.
`org.apache.hadoop.fs.s3`	Jedná se o implementaci distribuovaného souborového systému Amazon S3.

`org.apache.hadoop.fs.s3native`	Tato třída umožňuje číst a zapisovat datové bloky do distribuovaného souborového systému Amazon S3.
`org.apache.hadoop.fs.shell`	Implementuje metody pro práci s příkazovým řádkem (shellem), dokáže také parsovat shellové příkazy, stejně jako kontrolovat jejich správný zápis.
`org.apache.hadoop.http`	Tato třída implementuje metody, které dokáží zpracovávat vstup a výstup pomocí protokolu HTTP (Hyper Text Transmission Protocol).
`org.apache.hadoop.http.lib`	Třída obsahuje metody, které slouží pro autentifikaci a zabezpečení přístupu ke clusteru prostřednictvím webového (HTTP) rozhraní.
`org.apache.hadoop.io`	Třída obsahuje základní vstupně/výstupní funkce pro práci se síťovými prostředky, databázemi a soubory.
`org.apache.hadoop.io.compress`	Třída, které umožňuje pracovat s komprimovanými datovými proudy a bloky v distribuovaném souborovém systému.
`org.apache.hadoop.io.compress.bzip2`	Třída obsahuje metody pro kompresi a dekompresi bloků nebo souborů pomocí kompresního algoritmu BZip2.
`org.apache.hadoop.io.compress.zlib`	Třída implementuje Gzip dekompresor a Zlib kompresor a dekompresor.
`org.apache.hadoop.io.file.tfile`	Třída obsahuje rozhraní, které umožňuje provádění porovnání (komparace) mezi různými objekty pomocí metody RawComparator.
`org.apache.hadoop.io.nativeio`	Třída obsahuje implementace vstuplně - výstupních rozhraní, které standardně nejsou dostupné v programovacím jazyce Java.
`org.apache.hadoop.io.retry`	Třída umožňuje definovat vlastní ošetření vyjímek (throw exception) podle určitých okolností.
`org.apache.hadoop.io.serializer`	Třída umožňuje integrování data serializátorů třetích stran do nástroje Apache Hadoop.
`org.apache.hadoop.ipc`	Třída obsahuje nástroje, které slouží pro definování klientských a serverových stanic.
`org.apache.hadoop.ipc.metrics`	Třída obsahuje metriku implementace RPC (vzdáleného volání procedur).
`org.apache.hadoop.jmx`	Třída obsahuje metody pro začleňování JMX (Java Management Extensions) do nástroje Hadoop.
`org.apache.hadoop.log`	Jedná se o třídu, která zabezpečuje správu souborů `LOG` s možností třídění upozornění a chyb do různých stavů, např. fatální chyby (`FATAL`), běžné chyby (`ERROR`) a varování (`WARNING`).

`org.apache.hadoop.mapred`	Třída obsahující metody a funkce, umožňující programování aplikací, které jsou schopny paralelně zpracovávat velké množství dat napříč servery, které tvoří výpočetní cluster.
`org.apache.hadoop.mapred.jobcontrol`	Třída obsahuje potřebné metody, které umožňují spravovat závislosti mezi jednotlivými MapReduce aplikacemi.
`org.apache.hadoop.mapred.join`	Třída umožňuje spojovat datasety ještě před průběhem funkce Map podle předem definovaného klíčování.
`org.apache.hadoop.mapred.lib`	Tato třída obsahuje již předprogramové běžné funkce pro fáze Map, Reduce a Partitioning. Dostupné jsou např. funkce: `CombineFileInputFormat`, `CombineFileRecordReader`, `CombineFileSplit`, `FieldSelectionMapReduce`, `MultipleInputs`, `MultipleSequenceFileOutputFormat` atd. Z popisů těchto funkcí je zřejmé, že jsou určeny pro analýzu vstupních souborů a bloků.
`org.apache.hadoop.mapred.lib.aggregate`	Obsahuje třídy pro práci s různými matematickými a agregačními funkcemi.
`org.apache.hadoop.mapred.lib.db`	Tato třída přidává podporu nástroje Hadoop pro přístup k SQL databázím.
`org.apache.hadoop.mapred.pipes`	Tato třída umožňuje použití jazyka C++ pro vývoj MapReduce aplikací a pro správu distribuovaného souborového systému HDFS.
`org.apache.hadoop.mapred.tools`	Třída implementuje podporu pro přístup uživatele administrator do Hadoop MapReduce.
`org.apache.hadoop.mapreduce`	Díky této třídě můžeme v reálném čase sledovat aktuální status celého MapReduce clusteru, sledovat podrobné informace o Map Reduce aplikacích (vlastník, odesílatel) atd.
`org.apache.hadoop.mapreduce.lib.input`	Tato třída obsahuje funkce pro práci se vstupními daty. Umožňuje definovat formát souboru na vstupu, rozdělování souborů, načítat jednotlivé záznamy a řádky ze souborů atd.
`org.apache.hadoop.mapreduce.lib.map`	Třída obsahuje metody pro inverzi fáze Map, tedy pro přepnutí klíčů na hodnoty a naopak.
`org.apache.hadoop.mapreduce.lib.output`	Třída obsahuje metody, které řadí soubory popsané v Map-Reduce aplikacích do příslušných adresářů v souborovém systému. Dále také umožňuje směřovat výstup do zařízení `/dev/null` nebo do souboru ve formátu plain text (běžný text).
`org.apache.hadoop.mapreduce.lib.partition`	Tato třída rozděluje klíče podle jejich hash řetězce.

`org.apache.hadoop.mapreduce.lib.reduce`	Jedná se o výchozí implementaci Reduceru, ve většině případů je přepsána vlastní aplikací pro fázi Reducer.
`org.apache.hadoop.mapreduce.security`	Třída se používá jako generátor klíčů, obsahuje také hashovací algoritmy.
`org.apache.hadoop.mapreduce.security.token`	Třída spravuje tokeny MapReduce aplikací – jedná se o unikátní identifikátor každého MapReduce aplikace.
`org.apache.hadoop.mapreduce.server.jobtracker`	Třída implementuje TaskTracker, tedy centralizovanou entitu v distribuovaném prostředí, která se stará o provádění MapReduce aplikací vykonávaných na datových uzlech (datanodes).
`org.apache.hadoop.mapreduce.server.tasktracker.userlogs`	Třída implementuje kontrolu nad stavovými a informačními soubory `LOG` tasktrackeru. Zaznamenává tyto informace: informace o smazání MapReduce aplikace, informace o úspěšném dokončení MapReduce aplikace, informace o započatém zpracování MapReduce aplikace a informace o ukončení Java Virtual Machine.
`org.apache.hadoop.metrics`	Třída implementuje metriky pro analýzu výkonnosti a spolehlivosti Hadoop clusteru. Již není dále vyvíjena a nahradila ji třída metrics2.
`org.apache.hadoop.metrics2`	Třída implementuje metriky pro analýzu výkonnosti a spolehlivosti Hadoop clusteru. Monitoruje také objem zpracovaných, přenesených a uložených dat. Její využití je čistě statistické, nicméně implementace metrik je výhodná zejména při vývoji a ladění MapReduce aplikací, kdy můžeme MapReduce aplikace ladit a zároveň pozorovat, jak se výkonnostně chová celý Hadoop cluster.
`org.apache.hadoop.net`	Jedná se o třídu pro manipulaci s konfigurací sítě v rámci Hadoop clusteru. Umožňuje konfigurovat DNS, proxy (SOCKS), dále konfiguraci jednotlivých uzlů v Hadoop clusteru, překlad jmen DNS/IP adres na identifikátory RackID atd.
`org.apache.hadoop.record`	Třída implementuje speciální jazyk RDL (Record Description Language), který je využíván pro zjednodušení serializace a deserializace záznamů dat.
`org.apache.hadoop.security`	Třída implementuje autentizační mechanizmy jako např. Kerberos, SASL atd.

`org.apache.hadoop.util`	Třída implementuje větší množství užitečných funkcí a procedur, jako např: funkce pro výpočet kontrolních součtů pro přenosy v rámci distribuovaného souborového systému, funkce pro detekci chyb disků, utility dovolující používat běžné třídy jazyka Java, čtečka řádků dat ze vstupních datových proudů, funkce pro načtení parametrů o běhovém prostředí Java, spouštěc `JAR` souborů nástroje Hadoop, nástroj pro spouštění Unixových příkazů atd.
`org.apache.hadoop.util.bloom`	Třída implementuje Bloom filtr.
`org.apache.hadoop.util.hash`	Jedná se o třídu, která obsahuje hashovací funkce – tedy generování unikátních kódů, které jednoznačným a unikátním řetězcem identifikují jednotlivé soubory nebo bloky. Tato funkcionalita může být použita pro kontrolu souborů pro možné naroušení jejich integrity.

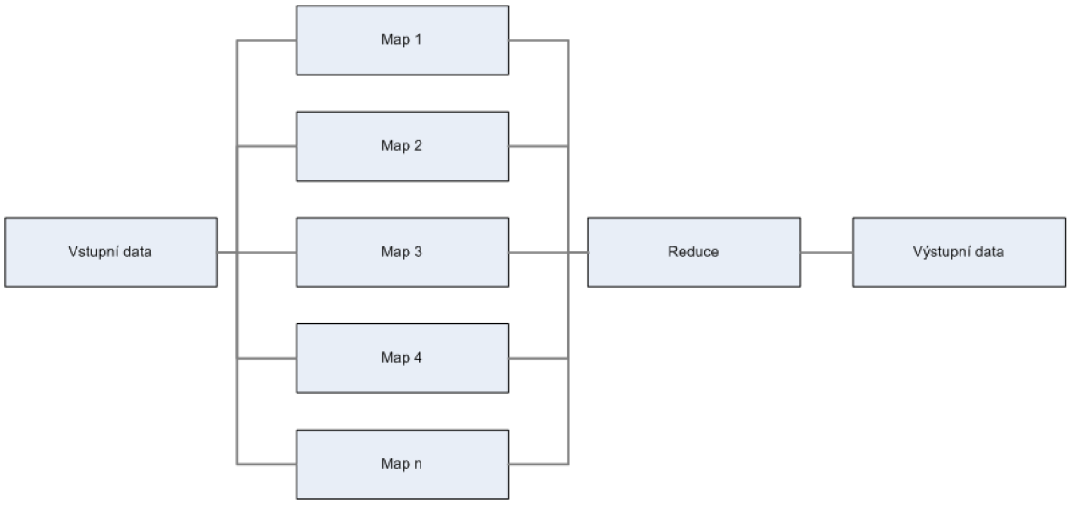
4 STRUKTURA MAPREDUCE APLIKACÍ

Základním programovacím jazykem pro programování MapReduce aplikací v Hadoop Frameworku je Java. Neoficiálními postupy lze tyto aplikace programovat i pomocí jazyků Ruby, Python a C++. V této bakalářské práce byl pro teoretické i praktické ukázky v této práci využíván programovací jazyk Java.

Metoda MapReduce je v superpočítačích používána pro zefektivnění výběru dat z lokálního souborového systému nebo z distribuovaného souborového systému (např. HDFS), který je tvořen větším či menším počtem fyzických výpočetních jednotek. Tento distribuovaný souborový systém pak tvoří na takovém počtu fyzických počítačů klastr, do kterého se posílají požadavky na poskytnutí dat právě pomocí metody MapReduce. Metodu MapReduce můžeme už podle názvu rozdělit na dvě specifické úlohy, Map a Reduce.

4.1 Princip fungování MapReduce aplikací

Paradigma MapReduce v nástroji Apache Hadoop poskytuje vývojářům takový přístup k programování, že významným způsobem ulehčuje adaptaci programu pro paralelní (distribuované) zpracování dat a provádění výpočtů. Každá MapReduce aplikace obsahuje určité konfigurační informace, funkci Mapovače (mapper) a funkci pro redukování, neboli agregační funkci (reducer). Nejefektivněji pracují MapReduce aplikace s daty ve formátu klíč – hodnota, kdy jakákoliv část této dvojice může být reprezentována např. polem hodnot nebo jiným složitějším datovým typem. Zadání výpočtů ve funkci Map je možno distribuovat démonem jobtracker do několika démonů tasktracker, které mohou běžet na separátních strojích – toto rozdělení úloh je základem pro paralelní zpracování dat v nástroji Hadoop. Obrázek 4.1 znázorňuje blokový diagram provádění MapReduce aplikace, kdy každá funkce Map běží na separátním stroji.



Obrázek 4.1: Blokový diagram provádění MapReduce aplikace

4.2 Teoretický příklad zpracování pomocí MapReduce

Dejme tomu, že potřebujeme analyzovat data v určitém tvaru, která reprezentují hodnoty teplot vzduchu dne 1.10. v několika letech jdoucích po sobě a potřebujeme vypsat, jaká teplota v onom roce a dni byla nejvyšší. Protože je soubor s daty velmi objemný, je účinnější jej zkopírovat do distribuovaného souborového systému HDFS a tuto analýzu provést pomocí MapReduce aplikace. Vstupní data v souboru mají následující tvar (bez uvozovek): „`DD.MM.RRRR HH:MM T`“, kde `DD` znamená den pořízení vzorku, `MM` znamená měsíc pořízení vzorku, `RRRR` znamená rok, `HH` znamená hodinu a `MM` minutu pořízení. Hodnota `T` pak reprezentuje samotný číselný údaj naměřené teploty vzduchu. Několik záznamů v analyzovaném souboru tak bude vypadat následovně:

```
01.10.1991 08:00 6
01.10.1991 09:00 8
01.10.1991 12:00 12
01.10.1991 16:00 10
01.10.1992 08:00 5
01.10.1992 09:00 7
01.10.1992 12:00 13
01.10.1992 16:00 9
01.10.1993 08:00 5
01.10.1993 09:00 8
01.10.1993 12:00 10
01.10.1993 16:00 2
01.10.1994 08:00 1
01.10.1994 09:00 3
01.10.1994 12:00 9
01.10.1994 16:00 5
```

Ve funkci mapovače potřebujeme napsat takový program, který pro následnou analýzu převede vstup do podoby klíč – hodnota, se kterou umí nástroj Hadoop nativně pracovat. Z teoretického zadání je zřejmé, že nás zajímají jen hodnoty o daném roku a teplotě. Funkce mapovače tak bude provádět separaci řetězců se vstupu takovým způsobem aby vytvořila následující dvojici: (rok, teplota). Nástroj Hadoop – přesněji jménný nod (namenode) po spuštění MapReduce aplikace distribuuje funkci mapovače všem strojům, které jsou do Hadoop klastru nakonfigurované a čeká na jejich odpověď. Protože funkci mapovače zpracovává více fyzických strojů, výsledek bude navrácen dříve než kdyby funkci mapovače zpracovával pouze jeden fyzický stroj. Protože je však nástroj Hadoop koncipován pro zpracování velkých dat o velikostech v řádech stovek MB (megabajt) dat až jednotek PB (petabajt), výkon nástroje Hadoop může být v případě použití malých dat (jednotek až desítek MB) značně omezený kvůli časové režii řízení celého Hadoop klastru a mohlo by se tak stát, že zpracování takto malých dat bude efektivnější na jednom fyzickém stroji než na klastru více fyzických strojů.

Po zpracování této úlohy mapovači jsou výsledky na jejich výstupu v následujícím tvaru:

```
(1991, 6)
(1991, 8)
(1991, 12)
(1991, 10)
(1992, 5)
(1992, 7)
(1992, 13)
(1992, 9)
(1993, 5)
(1993, 8)
(1993, 10)
(1993, 2)
(1994, 1)
(1994, 3)
(1994, 9)
(1994, 5)
```

Protože je nutné, aby si výpočetní uzly mezi sebou výsledky mapovačů vyměnili z hlediska správného provedení redukční fáze, spouští se mezi uzly tzv. míchací proces (shuffle), kdy si uzly tyto výsledky mezi sebou navzájem předají. Výsledkem míchacího procesu a zároveň vstupem reduktoru (entity provádějící agregaci výsledků z mapovačů) pak bude toto datové uskupení:

```
(1991, [6, 8, 12, 10])
(1992, [5, 7, 13, 9])
(1993, [5, 8, 10, 2])
(1994, [1, 3, 9, 5])
```

Ve funkci reduktoru pak mějme takový program, který pro každý klíč (rok) vybere maximální teplotu z pole umístěného v poli hodnota z dvojice klíč – hodnota. Výstupem reduktoru pak bude tento soubor dat:

```
(1991, 12)
(1992, 13)
(1993, 10)
(1994, 9)
```

Výsledkem MapReduce úlohy tedy budou maxima teplot pro 1. říjen následující: 12 stupňů pro rok 1991, 13 stupňů pro rok 1992, 10 stupňů pro rok 1993 a 9 stupňů pro rok 1994.

4.3 Praktický příklad zpracování pomocí MapReduce

4.3.1 Zadaná analýza souboru pomocí nástroje Hadoop

Zadáním praktického úkolu byl soubor `data.csv` o velikosti cca 850MB, který obsahoval několik sloupců atributů, kdy v posledním sloupci byla definována třída, do které zadané data patří. Jelikož se jedná o relativně objemný soubor, jeho zpracování na jednom počítači není efektivní, nicméně pro demonstraci v této bakalářské práci postačí. Vzorek má proměnlivý počet sloupců, všechny jsou odděleny čárkami a obsahují různé hodnoty v intervalu <-2;2>. Poslední sloupec obsahuje celočíselnou hodnotu v intervalu <1;3> a jeho hodnota je odvozena (rozhodnuta) na základě dat v předchozích sloupcích. Poslední sloupec je zakončen tečkou, která udává, že zde řádek s daty končí. Vzorek dat z první řádku vypadá následovně (pro ukázku v této práci byl první řádek dodaného vzorku dat zmenšen).

```
1.1449,1.5513,-1.1233,-0.56087,-0.74961,-1.5955,0.50984,0.19163,-
0.23135,-0.88847,-0.56066,0.13973,0.22743,0.3515,0.8746,0.075916,-
0.37896,0.96112,-0.30192,0.40254,0.13268,1.8184,-0.94241,-0.88631,-
0.71622,2.0018,-0.32812,-1.0164,0.9002,0.94681,0.85004,-
0.29373,1.1504,0.86859,0.81014,0.54307,-0.75383,-0.7932,-0.6829,-
0.76128,-0.71764,-0.74027,-0.62633,-0.80217,-0.87816,-
0.68056,0.015193,0.52201,0.94967,-0.097897,-0.4088,-0.74014,-1.3519,-
0.79554,-0.97286,0.48587,-0.70772,-1.2442,-1.0105,-0.69689,0.65226,-
0.93199,-0.48584,-1.1276,-1.7192,-0.9257,0.85216,-0.93397,0.2568,-
1.0158,-0.36392,-0.59145,-1.0458,-0.36104,-0.75234,1.0574,-0.41828,-
1.2921,0.14791,1.3729,-1.2296,-0.57539,0.3382,0.9294,0.98539,-
0.55731,0.852,1.9048,1.3846,1.3987,1.6919,0.88527,0.60637,0.48524,0.66
879,0.62859,0.50376,0.26037,0.62038,1.1827,1.0551,1.2877,1.383,0.48168
```

Každý řádek v dodaném souboru je ukončen značkou `{EOL}` - end of line (konec řádku) a posledním znakem záznamu je klasifikátor, který vychází z ostatních dat daného záznamu. Pro demonstraci MapReduce aplikace jsem sestavil program, který v zadaném souboru vypočítá celkový počet řádků s daty a výsledek uloží do výstupního souboru.

4.3.2 Příprava MapReduce aplikace

Budeme tedy vycházet z toho, že zadáním je výpočet řádků v dodaném souboru. Z toho můžeme usuzovat, že budeme potřebovat mít k programu připojeny Java třídy pro práci s textem, se soubory a v neposlední řadě také třídy pro ovládání zpracování pomocí MapReduce (`mapred`). Namapujeme tedy do kódu aplikace třídu `java.io.*` pro ošetření vyjímek typu `IOException`, `java.util.*` pro použití některých funkcí jazyka Java, `org.apache.hadoop.fs.Path` pro práci a navigaci v adresářích a souborech, `org.apache.hadoop.conf` pro práci s konfigurací nástroje Hadoop, `org.apache.hadoop.io.*` pro práci se vstupy a výstupy souborů, `org.apache.hadoop.mapred.*` pro práci s MapReduce aplikacemi a `org.apache.hadoop.util.*`, což je obdoba klasické Java knihovny `util`. Knihovna `util` implementuje metody pro práci s iteriačními funkcemi, které budeme pro výpočet počtu řádků v souboru potřebovat. Hlavička programu pak bude vypadat následovně:

```
package org.myorg;

import java.io.*;
```

```
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
```

V dalším kroku musíme připravit funkci `main`. Funkce `main` je hlavní inicializační funkcí každého programu v jazyce Java a provádí se jako první. V tomto případě navíc požadujeme, aby funkce `main` zachytávala vstupní argumenty programu – v našem případě to budou definice dvou souborů: vstupní soubor `data.csv` a výstupní soubor `output`. S oběma soubory je manipulováno nástrojem Hadoop výhradně v rámci souborového systému HDFS, proto na něm musí být vstupní soubor přítomen před samotným spuštěním MapReduce aplikace (viz. kapitola Příkazy pro základní souborové operace v HDFS). Funkce `main` pak bude vypadat následovně:

```
// Hlavni funkce programu / zachytavani argumentu (vstup, vystup)
// Nastaveni Hadoop jobu
public static void main(String[] args) throws IOException {
     JobConf conf = new JobConf(PocetRadku.class);
     conf.setJobName("jobPocetRadku");
     conf.setOutputKeyClass(Text.class);
     conf.setOutputValueClass(IntWritable.class);
     conf.setMapperClass(Map.class);
     conf.setCombinerClass(Reduce.class);
     conf.setReducerClass(Reduce.class);
     conf.setInputFormat(TextInputFormat.class);
     conf.setOutputFormat(TextOutputFormat.class);
     FileInputFormat.setInputPaths(conf, new Path(args[0]));
     FileOutputFormat.setOutputPath(conf, new Path(args[1]));
     JobClient.runJob(conf);
   }
```

Do programu dále dosadíme funkci Map:

```
public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
                private final static IntWritable one = new
IntWritable(1);
                private Text word = new Text("Pocet radku vstupniho
souboru:");
                public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
             output.collect(word, one);   // vystup Mapperu
```

```
        }
    }
```

A nakonec dosadíme funkci Reduce:

```
// Funkce Reduceru:
public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
     public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
       int sum = 0;
       while (values.hasNext()) {
         sum += values.next().get();      // inkrementace citace radku
       }
       output.collect(key, new IntWritable(sum));     // vystup
Reduceru
     }
}
```

Všechny tři funkce (tedy `main`, funkci Mapperu a Reduceru) je nutné ještě ohraničit uvozením, které označuje použití programu jako Java třídy. Všechny tři funkce tedy vnoříme do třídy PocetRadku:

```
public class PocetRadku {
.
.
.
. ZDE SE NACHÁZÍ FUNKCE MAIN, MAP A REDUCE
.
.
}
```

Soubor s programem nakonec uložíme do souboru s příponou `.java`. Soubor musí být pojmenován stejně, jako v něm vytvořená Java třída, proto soubor pojmenujeme `PocetRadku.java` a uložíme. Soubor je nyní připraven pro kompilaci.

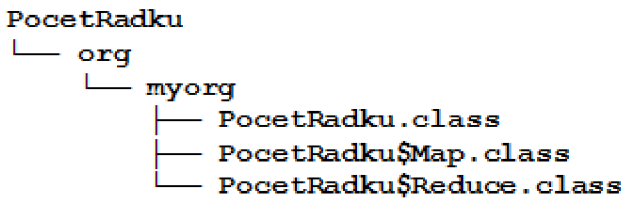
4.3.3 Kompilování MapReduce aplikace

Kompilování MapReduce aplikace se provádí podobně jako kompilování jakéhokoliv jiného Java programu s tím rozdílem, že kompilátoru ukážeme, kde se nachází třídy Mapperu a Reduceru nástroje Hadoop. Samotný binární soubor kompilátoru se nazývá `javac`. Kompilátoru je nutno předat několik argumentů:

```
./javac -classpath /usr/local/hadoop/hadoop-core-0.20.203.0.jar -d
PocetRadku PocetRadku.java
```

Kde `./javac` je binární soubor kompilátoru jazyka Java, přepínač `-classpath` a následná cesta definují cestu k Java třídám nástroje Hadoop (Hadoop Framework),

přepínač `-d` a název adresáře definuje, kam se uloží zkompilovaný program jazyka Java (soubor(y) s příponou `.class`) a posledním argumentem je vstupní kód do kompilátoru z kapitoly Příprava MapReduce aplikace. Spuštěním tohoto příkazu se nám tedy vytvoří ve výstupním adresáři (`PocetRadku`) struktura zkompilovaného programu – MapReduce aplikace, kterou znázorňuje Obrázek 4.2:



Obrázek 4.2: Souborová struktura zkompilované MapReduce aplikace

Celý adresář PocetRadku nyní zkomprimujeme programem jar:

```
./jar -cvf PocetRadku.jar -C PocetRadku .
```

Kde `./jar` je binární soubor komprimátoru jazyka Java, přepínač `-cvf` definuje, že chceme výstup komprimace odeslat do nového `JAR` archivu se specifickým názvem a že zkompilované soubory jazyka Java se nachází v adresáři `PocetRadku` (přepínač `-C`). Výsledný archív `PocetRadku.jar` si můžeme zkopírovat do složky s binárními soubory nástroje Hadoop pro jeho snazší spouštění.

4.3.4 Spuštění MapReduce aplikace v nástroji Hadoop

Předpokládejme, že již máme ve složce s binárními soubory nástroje Hadoop nachystán soubor `PocetRadku.jar`, který je nyní zkompilovanou MapReduce aplikací. Jak také bylo uvedeno dříve, aplikaci je nutno spustit s dvěma argumenty: vstupní data, výstupní data. Oba soubory budou hledány/vytvářeny v souborovém systému HDFS. Před spuštěním MapReduce aplikace je nutné se přesvědčit, že nástroj Hadoop běží, učiníme tak spuštěním následujícího příkazu v adresáři `/usr/local/hadoop/bin`:

```
./start-all.sh
```

Pokud doposud nástroj Hadoop neběžel, spustí se. Pokud již byl spuštěn, oznámí uživateli pouze informaci, že jej není potřeba znovu spouštět, neboť běží. Nyní tedy můžeme přistoupit ke spuštění dané MapReduce aplikace (soubor `PocetRadku.jar`) pomocí nástroje Hadoop:

```
./hadoop jar PocetRadku.jar org.myorg.PocetRadku data.csv vystup
```

Kde `./hadoop` je binární soubor spuštění nástroje Hadoop, přepínač jar definuje, že aplikace bude ve formátu `JAR`, dále definujeme název souboru s aplikací (`PocetRadku.jar`), následuje celý tvar třídy, která je ve zdrojovém kódu využita (`org.myorg.PocetRadku`). Poslední dva přepínače jsou pak atributy funkce `main` samotné MapReduce aplikace – prvním z nich je vstupní soubor a druhým je výstupní soubor. Vstupní soubor bude hledán v souborovém systému HDFS současného uživatele (`hadoop`) a výstupní soubor s výsledkem do něj bude zapsán.

4.3.5 Výstup z aplikace

Velikost výstupního souboru `LOG` spuštěné MapReduce aplikace `PocetRadku` nedovoluje jeho celé publikování v této bakalářské práci, proto zde vypíši pouze několik posledních řádků průběhu aplikace s výsledkem výpočtu počtu řádků:

```
11/12/13 21:57:08 INFO mapred.JobClient: Job complete: job_local_0001
11/12/13 21:57:08 INFO mapred.JobClient: Counters: 19
11/12/13 21:57:08 INFO mapred.JobClient:   File Input Format Counters
11/12/13 21:57:08 INFO mapred.JobClient:     Bytes Read=861439696
11/12/13 21:57:08 INFO mapred.JobClient:   File Output Format Counters
11/12/13 21:57:08 INFO mapred.JobClient:     Bytes Written=36
11/12/13 21:57:08 INFO mapred.JobClient:   FileSystemCounters
11/12/13 21:57:08 INFO mapred.JobClient:     FILE_BYTES_READ=160139
11/12/13 21:57:08 INFO mapred.JobClient:
HDFS_BYTES_READ=6961499618
11/12/13 21:57:08 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=628602
11/12/13 21:57:08 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=36
11/12/13 21:57:08 INFO mapred.JobClient:   Map-Reduce Framework
11/12/13 21:57:08 INFO mapred.JobClient:     Map output materialized
bytes=559
11/12/13 21:57:08 INFO mapred.JobClient:     Map input records=9860
11/12/13 21:57:08 INFO mapred.JobClient:     Reduce shuffle bytes=0
11/12/13 21:57:08 INFO mapred.JobClient:     Spilled Records=30
11/12/13 21:57:08 INFO mapred.JobClient:     Map output bytes=345100
11/12/13 21:57:08 INFO mapred.JobClient:     Map input bytes=860866244
11/12/13 21:57:08 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1248
11/12/13 21:57:08 INFO mapred.JobClient:
records=9860
11/12/13 21:57:08 INFO mapred.JobClient:     Reduce input records=13
11/12/13 21:57:08 INFO mapred.JobClient:     Reduce input groups=1
11/12/13 21:57:08 INFO mapred.JobClient:     Combine output records=13
11/12/13 21:57:08 INFO mapred.JobClient:     Reduce output records=1
11/12/13 21:57:08 INFO mapred.JobClient:     Map output records=9860
```

Z výsledku MapReduce aplikace tedy vyplývá, že zadaný soubor `data.csv` o velikosti cca 850MB má 9860 řádků. Provedl jsem také kontrolu tohoto výsledku pomocí jednoduchého příkazu v příkazovém řádku operačního systému GNU/Linux.

```
find data.csv -print0 | xargs -0 wc -l
```

a výstupem byla stejná hodnota počtu řádků:

```
9860 data.csv
```

Můžeme tedy konstatovat, že výsledek je správný, neboť jej potvrdily dva rozdílné způsoby výpočtu.

5 POPIS SOUBOROVÉHO SYSTÉMU HDFS

5.1 Základní popis HDFS

HDFS (Hadoop Distributed File System) je distribuovaný souborový systém, který umožňuje spravovat velké množství dat – řádově gigabajty až terabajty. Byl navržen pro co možná nejvíce efektivní skladování a načítání dat – systém operuje tak, že se snaží vždy o co nejmenší počet zápisů do datasetu a o co nejvyšší počet čtení z něj. Jeho nevýhodou je však pomalejší přístup k datům, proto není vhodný pro použití s aplikacemi, které vyžadují velmi rychlé nalezení potřebných informací, je to daň za vysokou propustnost tohoto souborového systému. HDFS také není vhodný pro ukládání velkého počtu malých souborů, protože každý soubor, adresář nebo blok používá přibližně 150 bajtovou režii, tak v případě použití tisíců až statisíců malých souborů by stála samotná režie souborového systému značnou část diskové kapacity. Je třeba si také uvědomit, že HDFS nepodporuje vícenásobný přístup k souborům najednou a že přidávání dat do jednotlivých souborů se vždy řídí takovým způsobem, kdy nová data jsou zapsána vždy na konec tohoto souboru. [1]

5.2 Bloky v souborovém systému HDFS

Bloky v souborovém systému HDFS rozumíme nejmenší jednotky, do nichž lze uložit data. Zatímco data na klasických pevných discích se obvykle ukládají do clusterů o velikosti kolem 512kB (záleží na použitém souborovém systému operačního systému), v souborovém systému HDFS má blok velikost 64MB. U souborových systémů operačního systému však soubor zabírá celý diskový cluster i pokud má menší velikost než samotný cluster, u HDFS tomu tak není. Použití bloků v distribuovaném souborovém systému má mnoho výhod, např. jediný soubor může mít větší velikost, než je kapacita největšího z pevných disků (nebo oddílů) v celém clusteru. Dále můžeme jmenovat sníženou režii vyhledávání v datech, protože se předpokládá, že každý blok dat je stejně velký. Toho se dá využít také při vytváření nového Hadoop nodu, protože víme (díky známé kapacitě pevných disků) kolik bloků na něj můžeme uložit. Bloky HDFS vypíšeme následujícím příkazem:

```
./hadoop fsck -blocks
```

```
hadoop@thinkpad-ubuntu:/usr/local/hadoop/bin$ ./hadoop fsck -blocks
FSCK started by hadoop from /127.0.1.1 for path / at Thu Nov 17 19:03:41 CET 2011
Status: HEALTHY
 Total size:      0 B
 Total dirs:      1
 Total files:     0
 Total blocks (validated):    0
 Minimally replicated blocks: 0
```

```
 Over-replicated blocks:      0
 Under-replicated blocks:     0
 Mis-replicated blocks:       0hadoop@thinkpad-
ubuntu:/usr/local/hadoop/bin$ ./hadoop fsck -blocks
FSCK started by hadoop from /127.0.1.1 for path / at Thu Nov 17
19:03:41 CET 2011
Status: HEALTHY
 Total size:      0 B
 Total dirs:      1
 Total files:     0
 Total blocks (validated):    0
 Minimally replicated blocks: 0
 Over-replicated blocks:      0
 Under-replicated blocks:     0
 Mis-replicated blocks:       0
 Default replication factor:  3
 Average block replication:   0.0
 Corrupt blocks:        0
 Missing replicas:            0
 Number of data-nodes:        1
 Number of racks:       1
FSCK ended at Thu Nov 17 19:03:41 CET 2011 in 19 milliseconds

The filesystem under path '/' is HEALTHY
 Default replication factor:  3
 Average block replication:   0.0
 Corrupt blocks:        0
 Missing replicas:            0
 Number of data-nodes:        1
 Number of racks:       1
FSCK ended at Thu Nov 17 19:03:41 CET 2011 in 19 milliseconds

The filesystem under path '/' is HEALTHY
```

5.3 Ochrana dat v HDFS

Souborový systém HDFS poskytuje velkou míru zabezpečení dat proti jejich ztrátě nebo poškození. Každý z bloků může být replikován na fyzicky oddělený server (nebo více serverů). Jestliže dojde k porušení dat v nějakém bloku na produkčním serveru, poškozený blok je nahrazen z těchto záložních míst, aniž by si toho uživatel všiml. Díky rozložení bloků na několik záložních serverů můžeme také HDFS nakonfigurovat pro vyšší výkon – paralelní čtení dat. Definování, kolik replikací bloků dat v HDFS má nástroj Hadoop provést se nastavuje v konfiguračním souboru `conf/hdfs-site.xml`. Počet replikací je uveden v části `dfs.replication`. Typicky se nastavuje počet replikací bloků na stejnou hodnotu, kolik uzlů má daný Hadoop klastr. Příklad souboru `hdfs-site.xml` je uveden zde:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
  <name>dfs.replication</name>
  <value>4</value>
  <description>Pocet replikaci v ramci HDFS.</description>
</property>
<property>
  <name>dfs.permissions</name>
  <value>false</value>
</property>
</configuration>
```

5.4 Datové a jmenné uzly (datanode, namenode)

HDFS cluster pracuje na principu pán – otrok (master – slave), u názvosloví HDFS přesněji master – worker (pán – pracovník). Namenode je použit jako master (řídící uzel), zatímco datanode je použit jako pracovník (datový uzel). Řídící uzel se stará o strom souborového systému a o metadata, které nesou informace o adresářích a souborech. Tyto informace jsou uloženy ve dvou souborech. Prvním z nich je obraz metadat a druhým je soubor, který zaznamenává jeho změny (edit log). Řídící uzel má také přehled o tom, na kterých datových uzlech se nacházejí bloky jednotlivých souborů. Hadoop cluster nelze provozovat bez řídícího uzlu a to ani v případě, kdy by byl z Hadoop clusteru odstraněn později, protože by se ztratily informace o tom, jak bloky uložené v datových uzlech rekonstruovat na soubory. Z toho vyplývá, že řídící uzel musí být velmi odolný vůči výpadkům a ztrátě dat. Toho se dá docílit např. redundancí řídícího uzlu, kdy se replikují metadata a záznamník změn (edit log) na

separátní server, který se sice nechová jako řídící uzel ale v případě potřeby může metadata a edit log rekonstruovat.

5.5 Příkazy pro základní souborové operace v HDFS

Příkazy pro práci se soubory a adresáři v HDFS se velmi podobají příkazům pro práci se soubory v operačním systému Linux. [11] Kompletní nabídku příkazů, které má administrátor s nástrojem Hadoop k dispozici můžeme vyvolat následujícím příkazem:

```
./hadoop fs -help
```

Jako příklad můžeme uvést kopírování z lokálního diskového souborového systému do HDFS – tento příklad také později využijeme, protože budeme tato data pomocí nástroje Hadoop zpracovávat – MapReduce aplikace nehledá data v souborovém systému operačního systému ale právě v HDFS. Vstupní data se nacházají v cestě `/home/lukas/Plocha/data.csv`, chceme je zkopírovat do domovského adresáře uživatele `hadoop` v HDFS:

```
./hadoop fs -copyFromLocal /home/lukas/Plocha/data.csv data.csv
```

Přo ověření výsledku kopírování dat můžeme zaslat do HDS příkaz `ls` abychom se ujistili, že zadaný soubor se do souborového systému HDFS zkopíroval:

```
hadoop@thinkpad-ubuntu:/usr/local/hadoop/bin$ ./hadoop fs -ls
Found 1 items
-rw-r--r--      3 hadoop supergroup    860866244 2011-11-17 19:45
/user/hadoop/data.csv
```

5.6 Přístup k HDFS prostřednictvím grafického rozhraní

Pro snazší procházení obsahu distribuovaného souborového systému HDFS je možné prohlížet jeho obsah prostřednictvím grafického rozhraní. Toto rozhraní je dostupné pomocí webového prohlížeče na adrese http://HOSTNAME:50070, kde HOSTNAME znamená doménový název serveru, na kterém běží démon jmenného uzlu nástroje Hadoop. Obrázek 5.1: aObrázek 5.2: znázorňují, jak takové grafické prostředí HDFS vypadá.

NameNode 'localhost:54310'

Started: Wed May 23 17:36:30 CEST 2012
Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

182 files and directories, 86 blocks = 268 total. Heap Size is 15.5 MB / 966.69 MB (1%)

Configured Capacity	:	59.63 GB
DFS Used	:	292.89 MB
Non DFS Used	:	55.43 GB
DFS Remaining	:	3.91 GB
DFS Used%	:	0.48 %
DFS Remaining%	:	6.56 %
Live Nodes	:	1
Dead Nodes	:	0

NameNode Storage:

Storage Directory	Type	State
/hdptmp/hadoop/tmp/dfs/name	IMAGE_AND_EDITS	Active

Hadoop, 2012.

Obrázek 5.1: Obecné informace o nainstalovaném HDFS

Contents of directory /user/hadoop

Goto : /user/hadoop go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
easy.txt	file	0.03 KB	1	64 MB	2012-04-18 20:25	rw-r--r--	hadoop	supergroup
output	dir				2012-04-16 19:06	rwxr-xr-x	hadoop	supergroup
reuters-initial-clusters	dir				2012-05-06 17:49	rwxr-xr-x	hadoop	supergroup
reuters-kmeans-clusters	dir				2012-05-06 17:51	rwxr-xr-x	hadoop	supergroup
reuters-seqfiles	dir				2012-05-01 20:34	rwxr-xr-x	hadoop	supergroup
reuters-vectors	dir				2012-05-01 20:39	rwxr-xr-x	hadoop	supergroup
seqdata1	dir				2012-04-28 19:13	rwxr-xr-x	hadoop	supergroup
seqdata2	dir				2012-04-28 19:16	rwxr-xr-x	lukas	supergroup
sequencedata	dir				2012-04-28 15:02	rwxr-xr-x	hadoop	supergroup
testdata	dir				2012-04-16 19:04	rwxr-xr-x	hadoop	supergroup
testdata.txt	file	38.25 MB	1	64 MB	2012-04-16 19:03	rw-r--r--	hadoop	supergroup
vecdata2	dir				2012-05-02 12:00	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2012.

Obrázek 5.2: Grafické rozhraní adresářové struktury HDFS

6 INSTALACE NÁSTROJE HADOOP V OS UBUNTU GNU/LINUX

6.1 Softwarové požadavky nástroje Hadoop

Instalace nástroje Hadoop byla realizována na operačním systému Ubuntu GNU/Linux verze 10.04 (kódové značení „Lucid Lynx“), který byl plně aktualizován k datu 04.12.2011. Nástroj Hadoop ke své činnosti dále potřebuje běhové prostředí Java a v případě konfigurace v distribuovaném módu musí být také funkční všechny síťové cesty mezi jednotlivými výpočetními uzly.

6.2 Instalace běhového prostředí Java

Protože je nástroj Hadoop napsán v jazyce Java, tak jeho binární soubory potřebují ke svému běhu knihovny Java – takzvaný Java runtime environment (JRE), neboli běhové prostředí Javy. [5] V případě, že navíc budeme vyvíjet aplikace pro Hadoop, neobejdeme se bez Java Development Kitu (JDK), neboli nástrojů pro vývoj v Javě. Běhové prostředí Java lze v OS Ubuntu Linux nainstalovat následovně:

```
sudo apt-get install openjdk-6-jre
```

Po zadání hesla `sudo` vyčkáme, než se potřebné balíčky stáhnou a nainstalují – počítač musí být připojen k internetu. Stejný postup instalace provedeme také v případě instalace vývojového prostředí Java:

```
sudo apt-get install openjdk-6-jdk
```

Opět vyčkáme, než se potřebné balíčky stáhnou a nainstalují. Nyní tedy máme nainstalovanou běhovou a vývojovou podporu pro programovací jazyk Java.

6.3 Příprava uživatele „hadoop“

Nástroj Hadoop vyžaduje v OS GNU/Linux separátní uživatelský účet a to z důvodu, že na počítači, kde hostujeme HDFS nod, tento nástroj vytváří vlastní adresářovou strukturu, ke které potřebuje patřičná práva. Nástroj Hadoop tak může být zcela oddělen od administrátorského účtu. Účet `hadoop`, včetně skupiny `hadoop` tedy vytvoříme následovně [5]:

Nejdříve vytvoříme skupinu uživatelů `hadoop`:

```
sudo addgroup hadoop
```

Poté můžeme vytvořit samotného uživatele `hadoop` a ihned jej přiřadit do skupiny „`hadoop`“:

```
sudo adduser --ingroup hadoop hadoop
```

6.4 Instalace OpenSSH serveru pro vzdálené ovládání

Abychom náš Hadoop nod mohli vzdáleně ovládat, je dobré na něm nainstalovat OpenSSH server. Jedná se o službu, která nám umožní komunikovat s lokálním nebo vzdálenějším serverem na úrovní příkazové řádky. OpenSSH server nainstalujeme následujícím příkazem:

```
sudo apt-get install openssh-server
```

Po stažení a instalaci potřebných balíčků můžeme ověřit, zda-li nám funguje také OpenSSH klient – v základní instalaci Ubuntu GNU/Linux by však měl být standardně obsažen:

```
ssh localhost
```

Pokud příkaz vrátí chybu, že balíček neexistuje, můžeme OpenSSH klient doinstalovat příkazem:

```
sudo apt-get install openssh-client
```

Jelikož budeme potřebovat provádět s účtem uživatele hadoop některé triviální operace, které vyžadují práva administrátora, musíme povolit používání udělení oprávnění administrátora pro účet `hadoop`. Přihlásíme se tedy jako administrátor, v případě projektu na experimentálním PC tedy:

```
su - lukas
```

A provedeme úpravu konfiguračního souboru `sudoers`, ve kterém jsou definovány uživatelské účty, kterým je přiděleno oprávnění používat příkaz `sudo` k získání administrátorských práv:

```
sudo nano /etc/sudoers
```

V textovém editoru nano přidáme pod řádek

```
root ALL=(ALL) ALL
```

následující konfiguraci:

```
hadoop ALL=(ALL) ALL
```

Pomocí klávesové zkratky Ctrl+O změny v souboru sudoers uložíme a pomocí zkratky Ctrl+X uzavřeme textový editor nano. Nyní se můžeme přepnout na uživatele hadoop a to příkazem:

```
su - hadoop
```

Nyní je ještě potřeba OpenSSH server správně nakonfigurovat pro naše použití. Konfigurační soubor OpenSSH serveru má následující cestu:

```
/etc/ssh/sshd_config
```

Konfigurační soubor tedy otevřeme pro editaci příkazem:

```
sudo nano /etc/ssh/sshd_config
```

Ujistíme se, že následující řádky nejsou zakomentovány (nemají na začátku znak mřížky – „#“):

```
Port 22
Protocol 2
```

```
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
UsePrivilegeSeparation yes
KeyRegenerationInterval 3600
ServerKeyBits 768
SysLogFacility AUTH
LogLevel INFO
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
IgnoreRhosts yes
RhostsRSAAuthentication no
HostbasedAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no
X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
TCPKeepAlive yes
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server
UsePAM yes
```

Po kontrole konfiguračního souboru OpenSSH serveru můžeme uplatnit tato nastavení restartováním démona sshd:

```
sudo /etc/init.d/ssh restart
```

Pokud potřebujeme provést velmi detailní nastavení OpenSSH serveru na základě specifických požadavků daných rozsahem řešení, můžeme použít příručku OpenSSH Manual [7].

6.5 Instalace samotného nástroje Hadoop

Nástroj Hadoop můžeme získat z webové stránky projektu Hadoop. V této bakalářské práci je použita stabilní verze nástroje Hadoop – 0.20.203.0rc1. Poslední stabilní verze nástroje Hadoop je dostupná z http://apache.mirror.dkm.cz/pub/apache//hadoop/core/stable/.

Výsledkem stahování je soubor s tímto názvem: `hadoop-0.20.203.0rc1.tar.gz`. Jedná se o komprimovaný archiv s GZip kompresí. Zkopírujme tento soubor

do adresáře `/usr/local` příkazem:

```
sudo cp hadoop-0.20.203.0rc1.tar.gz /usr/local
```

Zde soubor rozbalíme příkazem `sudo tar xzf hadoop-0.20.203.0rc1.tar.gz` a rozbalenou složku přejmenuje na hadoop:

```
sudo mv hadoop-0.20.203.0rc1 hadoop
```

Posledním krokem bude převzetí vlastnictví takto vytvořeného adresáře, aby mohl být tento adresář plně ovladatelný uživatelským účtem hadoop:

```
sudo chown -R hadoop:hadoop hadoop
```

7 KONFIGURACE NÁSTROJE HADOOP PRO STANDALONE INSTALACI

7.1 Nastavení konfiguračních souborů nástroje Hadoop

Podle prozatímních kroků jsme nainstalovali Hadoop do cesty `/usr/local/hadoop` a předali jsme vlastnictví tohoto adresáře uživateli hadoop. V kapitole Instalace běhového prostředí Java, ve které je instalace popsána pomocí balíčku, se toto prostředí nainstaluje automaticky do cesty `/usr/lib/jvm/java-6-openjdk`. Musíme tedy nyní definovat, kde má nástroj Hadoop hledat běhové prostředí Java.

Otevřeme soubor `/usr/local/hadoop/conf/hadoop-env.sh` pro editaci:

```
sudo nano /usr/local/hadoop/conf/hadoop-env.sh
```

a v konfiguračním souboru najdeme část s nadpisem „`# The Java implementation to use. Required`." V této části pak nastavíme následující definici:

```
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
export JDK_HOME=$JAVA_HOME
export PATH=$PATH:$JAVA:$JAVA_HOME/bin
```

Naše nastavení uložíme klávesovou zkratkou Ctrl+O a editor zavřeme kombinací kláves Ctrl+X. Zbytek nastavení v tomto konfiguračním souboru můžeme ponechat ve výchozím nastavení. Další nastavení se týkají distribuovaného souborového systému HDFS a metody MapReduce, jeho konfigurační soubor najdeme také v adresáři `/usr/local/hadoop/conf` a má jméno `core-site.xml`. Soubor tedy otevřeme editorem nano pro úpravy:

```
sudo nano /usr/local/hadoop/conf/core-site.xml
```

Protože má soubor XML strukturu [8][8], je potřeba zápis nastavení v něm správně ohraničit patřičnými XML značkami:

Jako příklad jsou zde uvedeny nastavení, použitá na experimentálním počítači této bakalářské práce.

```
core-site.xml:
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
      <name>hadoop.tmp.dir</name>
      <value>/hdfs/hadoop-${user.name}</value>
      <description>Korenovy adresar pro docasne soubory</description>
</property>
<property>
```

```
      <name>fs.default.name</name>
      <value>hdfs://localhost:54310</value>
      <description>URI identifikator HDFS filesystemu</description>
</property>
<property>
      <name>mapred.job.tracker</name>
      <value>localhost:54311</value>
      <description>Zde bezi MapReduce Tracker</description>
</property>
<property>
      <name>dfs.replication</name>
      <value>1</value>
      <description>Vychozi nastaveni replikace HDFS bloku</description>
</property>
</configuration>
```

Nyní je zapotřebí nástroji Hadoop definovat, které adresáře souborového systému mají být zahrnuty do logického distribuovaného souborového systému HDFS. Konfigurační soubor, kde tato nastavení můžeme provést, se nachází také v adresáři `conf`, konkrétně se jedná o tento soubor: `/usr/local/hadoop/conf/hdfs-site.xml`. Soubor otevřeme pro úpravy:

```
sudo nano /usr/local/hadoop/conf/hdfs-site.xml
```

a v souboru provedem zápis následujících parametrů – tyto značky přidáme mezi XML značky `<configuration>` a `</configuration>`:

```
<property>
      <name>dfs.name.dir</name>
      <value>/hdfs/disk1</value>
      <final>true</final>
</property>
<property>
      <name>dfs.data.dir</name>
      <value>/hdfs/disk1/data</value>
      <final>true</final>
</property>
<property>
      <name>fs.checkpoint.dir</name>
      <value>/hdfs/disk1/chkdir</value>
      <final>true</final>
</property>
```

Parametrem `dfs.name.dir` jsme definovali složku, která bude použita jako kořenová pro HDFS souborový systém na samostatné a distribuované instalaci. Parametr `dfs.data.dir` vyjadřuje samotný adresář pro datové bloky souborového systému HDFS. Parametr `fs.checkpoint.dir` následně definuje umístění adresáře pro ukládání tzv. checkpointů – ověřovacích bodů, které slouží pro zajištění redundance souborového systému – odolnost vůči chybám pomocí dvojího umístění. V produkčním prostředí se proto vyplatí tyto ověřovací body ukládat na jiném hardwarovém zařízení, než kde je nainstalován Hadoop/MapReduce uzel – může to být například síťové úložiště.

7.2 Formátování souborového systému HDFS

Po uložení XML souboru (v editoru nano klávesovou zkratkou Ctrl+O) můžeme přistoupit k formátování souborového systému. HDFS souborový systém formátujeme proto, aby se v něm vytvořila potřebná struktura adresářů a souborů, ve které budou uložené data sloužící pro zpracování dat metodou MapReduce, zejména metadat, dočasných dat, zdrojových dat a dat výsledků. Samotné formátování souborového systému HDFS nijak nenaruší souborový systém operačního systému (v našem případě se jedná o souborový systém ext3), protože je postaven nad ním - jedná se jen o logický souborový systém na aplikační úrovni, nikoliv na úrovní operačního systému nebo hardwaru. Binární soubor, který umožní formátování souborového systému HDFS se nachází ve adresáři bin, konkrétně celá cesta k adresáři je pak: `/usr/local/hadoop/bin`. Příkaz pro formátování pak zní:

```
./hadoop namenode -format
```

Pokud program hadoop nelze spustit z důvodů chybějících oprávnění, spustíme v adresáři `/usr/local/hadoop/bin` příkaz pro povolení spouštění tohoto souboru jako programu:

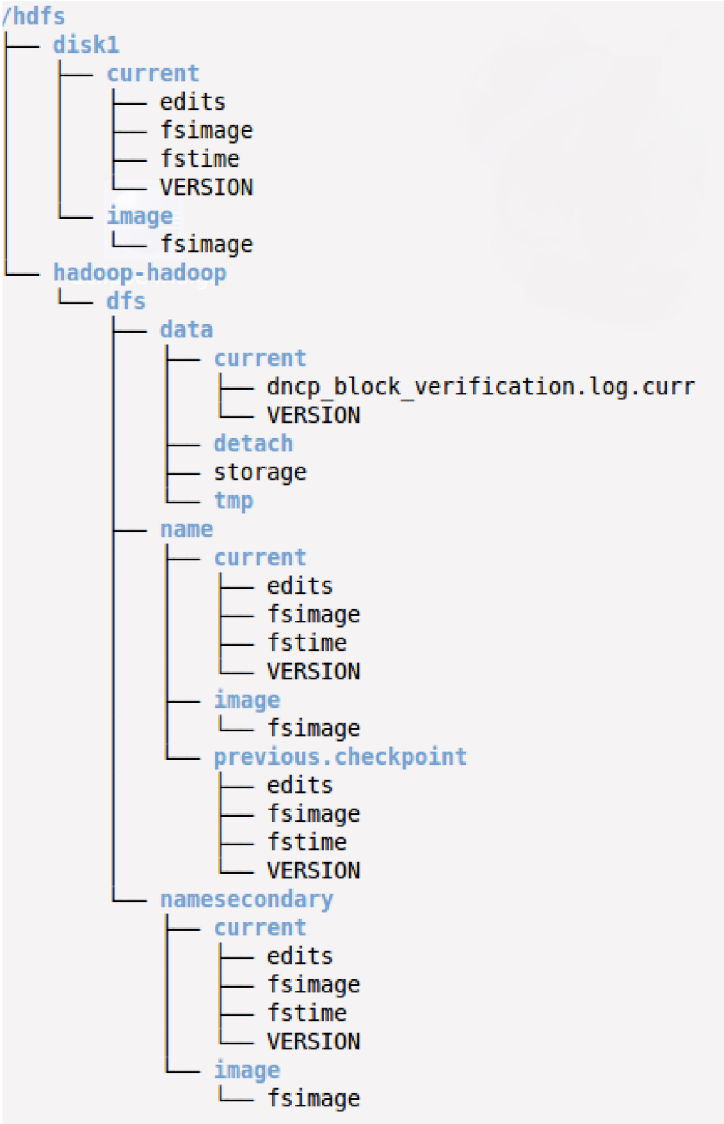
```
chmod a+x hadoop
```

Souborový systém HDFS je nyní naformátován, výsledkem spuštění formátovacího příkazu pak bude tento sled informací:

```
11/11/01 14:13:07 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = thinkpad-ubuntu/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.203.0
STARTUP_MSG:   build =
http://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20-
security-203 -r 1099333; compiled by 'oom' on Wed May  4 07:57:50 PDT
2011
************************************************************/
11/11/01 14:13:08 INFO util.GSet: VM type       = 32-bit
11/11/01 14:13:08 INFO util.GSet: 2% max memory = 19.33375 MB
```

```
11/11/01 14:13:08 INFO util.GSet: capacity      = 2^22 = 4194304
entries
11/11/01 14:13:08 INFO util.GSet: recommended=4194304, actual=4194304
11/11/01 14:13:08 INFO namenode.FSNamesystem: fsOwner=hadoop
11/11/01 14:13:08 INFO namenode.FSNamesystem: supergroup=supergroup
11/11/01 14:13:08 INFO namenode.FSNamesystem: isPermissionEnabled=true
11/11/01 14:13:08 INFO namenode.FSNamesystem:
dfs.block.invalidate.limit=100
11/11/01 14:13:08 INFO namenode.FSNamesystem:
isAccessTokenEnabled=false accessKeyUpdateInterval=0 min(s),
accessTokenLifetime=0 min(s)
11/11/01 14:13:08 INFO namenode.NameNode: Caching file names occuring
more than 10 times
11/11/01 14:13:08 INFO common.Storage: Image file of size 112 saved in
0 seconds.
11/11/01 14:13:08 INFO common.Storage: Storage directory /hdfs/disk1
has been successfully formatted.
11/11/01 14:13:09 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at thinkpad-ubuntu/127.0.1.1
************************************************************/
```

Po formátování se vytvoří v adresáři `/hdfs` adresářová/souborová struktura. Obrázek 7.1 tuto strukturu znázorňuje.



Obrázek 7.1: Výchozí stromová struktura souborového systému HDFS

7.3 Nastavení nástroje Hadoop pro distribuované prostředí

Toto nastavení se provádí v případě, že má mít nástroj Hadoop více výpočetních uzlů. Nastavení se provede stejně jako v případě standalone konfigurace nástroje Hadoop, na všech uzlech clusteru se provede stejná procedura, která je popsána v kapitole Konfigurace nástroje Hadoop pro standalone instalaci s tím rozdílem, že se pro distribuované prostředí nastaví navíc soubor `/usr/local/hadoop/conf/slaves` tak, kde každý řádek bude reprezentovat jednu IP adresu každého z uzlů v clusteru. Pro vzorový příklad na fakultních serverech bude tedy obsah souboru `slaves` vypadat následovně:

```
147.229.148.18
147.229.149.5
147.229.149.6
```

```
147.229.149.7
147.229.149.8
```

Také musí být na všech uzlech nastaven soubor `conf/core-site.xml` tak, aby parametr `fs.default.name` ukazoval URI HDFS na adresu jmenného uzlu (namenode):

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/hdptmp/hadoop/tmp</value>
  <description>Adresar pro potreby HDFS a MapReduce.</description>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://grid.utko.feec.vutbr.cz:54310</value>
  <description>HDFS na NameNodu.</description>
</property>
</configuration>
```

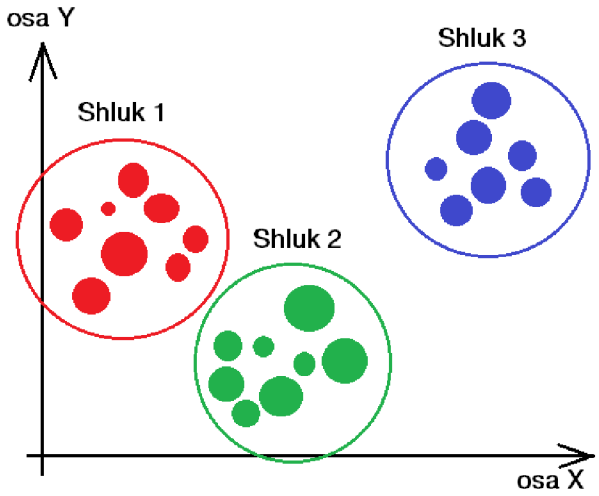
8 IMPLEMENTACE STROJOVÉHO UČENÍ DO NÁSTROJE HADOOP

8.1 Apache Mahout

Knihovny Apache Mahout obsahují několik desítek algoritmů pro strojové učení. Všechny jsou napsány v jazyce Java a jsou dostupné pod svobodnou licencí, včetně zdrojových kódů a dokumentace. Projekt implementuje tři hlavní oblasti strojově učících se algoritmů: doporučovací algoritmy, algoritmy shlukové analýzy a klasifikační algoritmy. Pro ukázku v této bakalářské práci byla vybrána oblast shlukové analýzy K-průměrů (K-Means) na ukázkových datech mediální společnosti Reuters.[12] Ukázková data společnosti společnosti Reuters obsahují několik tisíc článků uspořádaných do značkovacího formátu SGML.

8.2 Popis shlukovacího algoritmu K-Means

Shlukovací algoritmus K-Means byl objeven roku 1957 Stuartem Lloydem a původně byl používán pro výpočty s pulzně – kódovou modulací. V roce 1982 byl princip algoritmu veřejně představen. Tento algoritmus provádí objevení shluků dat na základě několika vstupů – počtu shluků, které má nalézt, souřadnic vektorů na kterých má být aplikována shluková analýza, prvotní souřadnice centroidů shluků a počet iterací, kterých má algoritmus opakovat. Algoritmus K-Means po spuštění měří vzdálenosti od centroidů shluků ke všem vektorům a následně každý vektor přiřadí do patřičného shluku podle toho, ke kterému centroidu je daný vektor nejblíže. Zároveň se vypočítají nové souřadnice centroidů shluků a to dle průměrů všech souřadnic vektorů patřících do příslušných shluků. Celý tento proces se opakuje tak dlouho, dokud se nevyčerpá stanovený limit iterací nebo dokud se nedosáhne patřičného konvergenčního kritéria. Pro měření vzdáleností vektorů od centroidů shluků se používají nejrůznější metriky, jako je Euklideanovská metrika nebo čtvercová Euklideanovská metrika. Obrázek 8.1 graficky znázorňuje rozmístění třech nalezených shluků na souboru vektorů v dvourozměrném souřadnicovém systému. [2]



Obrázek 8.1: Princip shlukové analýzy – tvorba shluků vektorů

8.3 Vybrané metriky pro výpočet vzdáleností

Tabulka 8.1: Metriky pro měření vzdáleností bodů od středů shluků algoritmu K-Means

Název metriky	Popis metriky
Euklideanovská vzdálenost	Jedna z nejjednodušších metrik pro výpočet vzdáleností dvou bodů, výpočet probíhá podle následujícího vzorce: [2] $d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_n - b_n)^2}$
Čtvrecová euklideanovská vzdálenost	Čtvercová euklideanovská vzdálenost se počítá podobně jako ryze euklideanovská vzdálenost, výpočet se však liší tím, že na daný kvadrát není aplikována odmocnina: [2] $d = (a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_n - b_n)^2$
Manhattanovská vzdálenost	U této metriky je vzdálenost mezi dvěma body dána součtem absolutních hodnot rozdílů hodnot jejich souřadnic. [2] $d = \lvert a_1 - b_1 \rvert + \lvert a_2 - b_2 \rvert + \lvert a_n - b_n \rvert$

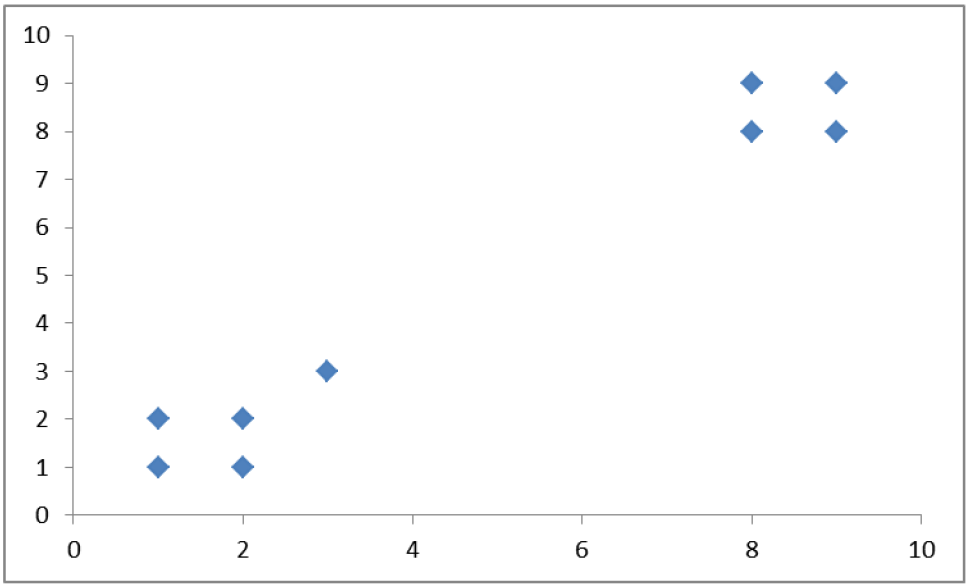
8.4 Ukázkový program škálovatelného shlukování vektorů

Tento ukázkový program provádí shlukovou analýzu celkem devíti vektorů (bodů) do celkem dvou shluků. [2] Ke shlukování používá algoritmus K-průměrů (K-Means) a shluková analýza je vypočítávána pomocí metriky čtvercové Euklideanovské vzdálenosti. Vstupními vektory jsou body uvedené v Tabulka 8.2.

Tabulka 8.2: Souřadnice vstupních vektorů algoritmu K-Means

Vektor	Souřadnice X	Souřadnice Y
1	1	1
2	2	1
3	1	2
4	2	2
5	3	3
6	8	8
7	8	9
8	9	8
9	9	9

Graficky mohou být tyto vektory vyjádřeny tak, jak je ukazuje Obrázek 8.2.



Obrázek 8.2: Grafické znázornění vektorů z Tabulka 8.2

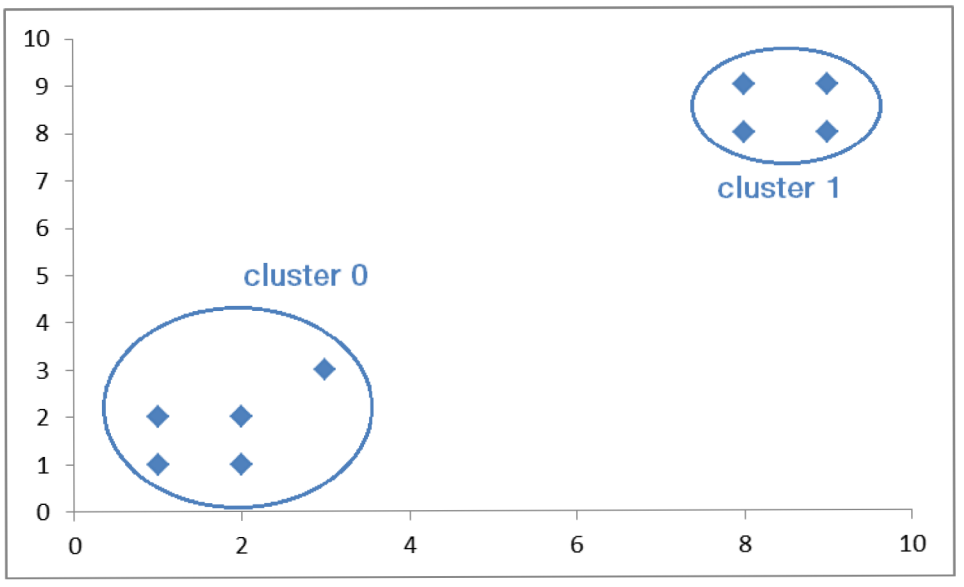
V CD příloze je uveden program [2], který tuto shlukovou analýzu provádí. V první části programu (`points`) jsou uvedeny hodnoty souřadnic jednotlivých vektorů, pro které hledáme shluky. Po spuštění funkce `main` nejprve převádí pole polí s proměnnýma na vektorový datový formát, kterému rozumí nástroj Hadoop – děje se tak pomocí funkce `getPoints`. Posléze jsou data v tomto vektorovém formátu zapsána do HDFS pomocí procedury `writePointsToFile` – vstupy této procedury jsou samotné vektory, jméno souboru do kterého se v HDFS vektory uloží a nastavení souborového systému a nástroje Hadoop. Spuštění shlukové analýzy K-Means v programu reprezentuje funkce `KmeansDriver.run`, která je prováděna na vstupních vektorech uložených již HDFS. Výsledek této shlukové analýzy probíhá pomocí metody `SequenceFile.Reader` a je vyobrazen po spuštění programu v konzolovém okně.

8.5 Výsledky z ukázkového programu

Výsledkem spuštěného programu je přiřazení vstupních vektorů do dvou shluků nalezených pomocí shlukovacího algoritmu K-Means. Následující výstup z programu z kapitoly Ukázkový program škálovatelného shlukování vektorů uvádí nalezené shluky pro dané vektory:

```
1.0: [1.000, 1.000] belongs to cluster 0
1.0: [2.000, 1.000] belongs to cluster 0
1.0: [1.000, 2.000] belongs to cluster 0
1.0: [2.000, 2.000] belongs to cluster 0
1.0: [3.000, 3.000] belongs to cluster 0
1.0: [8.000, 8.000] belongs to cluster 1
1.0: [9.000, 8.000] belongs to cluster 1
1.0: [8.000, 9.000] belongs to cluster 1
1.0: [9.000, 9.000] belongs to cluster 1
```

Obrázek 8.3 znázorňuje, jak je možno graficky nalezené shluky zadaných vektorů vyjádřit.

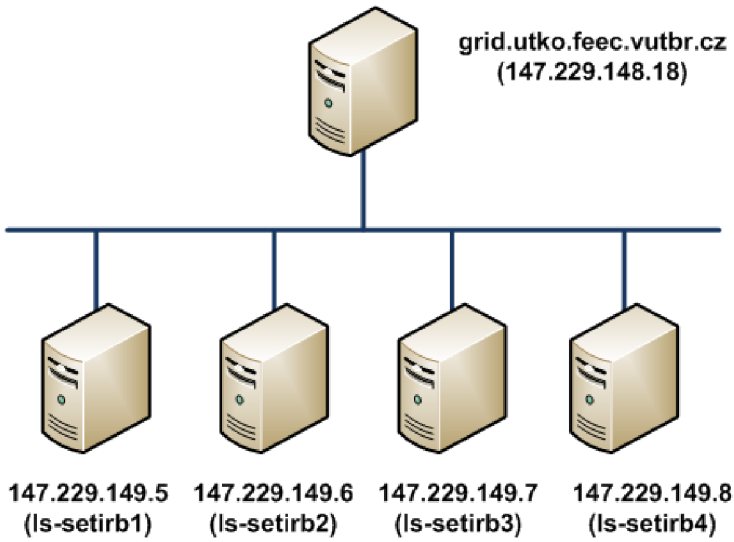


Obrázek 8.3: Grafické znázornění nalezení shluků na zadaných vektorech

9 VYHODNOCENÍ VÝKONU

9.1 Distribuované prostředí pro hodnocení výkonu

Měření bylo prováděno na shlukové analýze K-průměrů (K-Means) pro 2, 5, 20 a 30 shluků na vzorcích dat mediální společnosti Reuters [2], s použitím čtvercové Euklideanovské metriky pro změření vzdáleností středů shluků od jednotlivých vektorů. Soubor s těmito daty obsahuje celkem 22 souborů s krátkými články ve značkovacím formátu SGML, každý soubor obsahuje kolem 1,2 milionů řádku textu, které v jednom souboru tvoří cca 46 tisíc článků a celá data mají nekomprimovanou velikost 27MB. Nástroj Hadoop byl nainstalován a nakonfigurován na celkem 5 serverech, které k tomuto experimentu poskytl Ústav telekomunikací FEKT, VUT Brno a v průběhu hodnocení výkonu byly konfigurační soubory nástroje Hadoop upravovány tak, aby bylo možné měření provést také pro dvou, tří a čtyřuzlovou konfiguraci. Obrázek 9.1 znázorňuje síťovou architekturu serverů nástroje Hadoop, jak byly tyto servery zapojeny. Tabulka 9.1: Hardwarová konfigurace testovacích serverůpopisuje hardwarové konfigurace těchto serverů.



Obrázek 9.1: Schéma zapojení serverů pro měření výkonnosti

Tabulka 9.1: Hardwarová konfigurace testovacích serverů

Název serveru	Typ a frekvence procesoru	Počet jader CPU	Velikost paměti RAM
grid.utko.feec.vutbr.cz	Intel Xeon 3065, 2.33GHz	2	6GB
ls-setirb1.utko.feec.vutbr.cz	Intel Xeon E7440, 2.4GHz	4	16GB
ls-setirb2.utko.feec.vutbr.cz	Intel Xeon E7440, 2.4GHz	4	16GB
ls-setirb3.utko.feec.vutbr.cz	Intel Xeon E7440, 2.4GHz	4	16GB
ls-setirb4.utko.feec.vutbr.cz	Intel Xeon E7440, 2.4GHz	4	16GB

9.2 Metodika měření

Jako metodika měření výkonu škálovetelné platformy se samoučícím algoritmem bylo vybráno odečítání času z LOG souborů, které generuje samotná MapReduce aplikace. Před samotným výkonnostním testem bylo potřeba převést články na sekvenční binární formát srozumitelný nástroji Hadoop a z tohoto formátu převést data na binární vektory – algoritmus K-Means, který je obsažen v nástroji Mahout umí pracovat pouze s tímto formátem dat. Převedení adresáře s články na binární sekvenční formát lze provést pomocí funkce seqdirectory jako parametr spouštěcího souboru nástroje Mahout. Příkaz pak vypadá takto:

```
./mahout seqdirectory -c UTF-8 -i reuters-extracted -o reuters-seqfiles
```

kde parametr `seqdirectory` říká, že chceme převést do binárního sekvenčního formátu celou složku, parametr `-c` definuje, kterou znakovou sadu vstupní data obsahují, parametr `-i` uvádí, v které cestě se nachází vstupní data a parametr `-o` definuje, kam se uloží výstupní data, tedy sekvenční soubory. Jak již bylo uvedeno, nyní je ještě potřeba tyto sekvenční soubory převést na vektory, aby je bylo možné zpracovat nástrojem Mahout. K této vektorizaci se používá funkce `seq2sparse`, kterou opět voláme jako parametr spustitelného souboru nástroje Mahout:

```
./mahout seq2sparse -i reuters-seqfiles -o reuters-vectors
```

kde parametr `-i` definuje vstup se sekvenčními soubory a parametr `-o` definuje výstup, kam se uloží binární soubor s vektory.[2]

Časy byly odečítány po milisekundách (ms). Spouštění měření bylo vykonáváno v adresáři `/usr/local/mahout/bin` a to následujícím příkazem:

```
./mahout kmeans -i reuters-vectors/tfidf-vectors/ -c reuters-initial-clusters -o reuters-kmeans-clusters -dm org.apache.mahout.common.distance.SquaredEuclideanDistanceMeasure -cd 1.0 -k 30 -x 20 -cl
```

kde přepínač `-i` definuje, který sekvenční soubor s vektory má pro nalezení shluků použít, přepínač `-c` definuje sekvenční soubor, ve kterém se nachází vstupní středy shluků (jelikož jsou nyní neznámé, nástroj Mahout si je generuje sám), přepínač `-o` definuje, do kterého souboru se zapíše výstup ze shlukové analýzy (nalezené shluky), přepínač `-dm` definuje, kterou metodiku shlukové analýzy K-Means má program použít, přepínač `-cd` definuje přesnost metodiky shlukové analýzy, přepínač `-k` definuje počet shluků, které je potřeba nalézt a přepínač `-x` definuje maximální počet iterací algoritmu K-Means k nalezení požadovaných shluků.

9.3 Časové výsledky měření

Tato podkapitola obsahuje samotné výsledky měření. Levý sloupec obsahuje počet aktivních Hadoop uzlů v daném pokusu výpočtu. Pravý sloupec pak obsahuje časovou hodnotu, za jakou dobu byl výpočet dokončen.

Tabulka 9.2: Výkon shlukové analýzy pro 2 shluky

Počet výpočetních uzlů	Čas (ms)
1	131735
2	100230
3	103932
4	224812
5	194429

Tabulka 9.3: Výkon shlukové analýzy pro 5 shluků

Počet výpočetních uzlů	Čas (ms)
1	126957
2	164208
3	126391
4	167726
5	132324

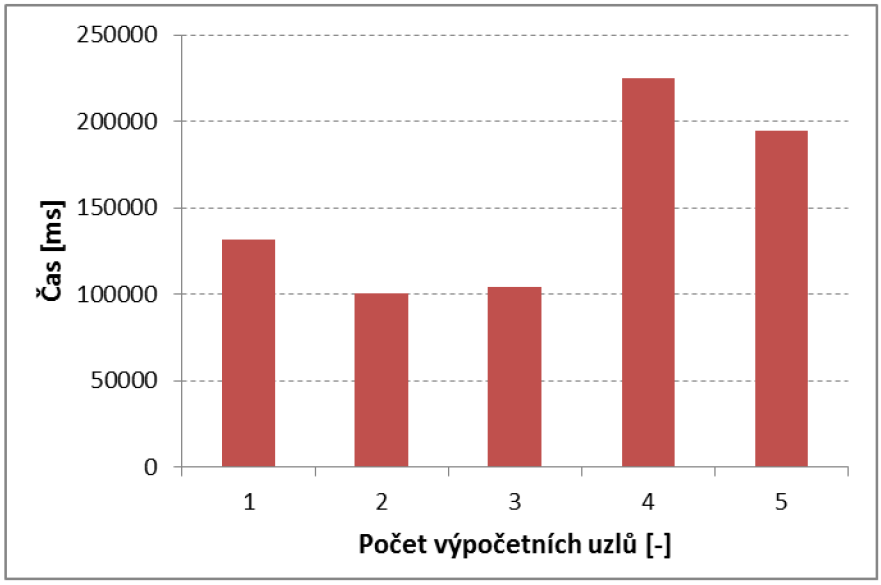
Tabulka 9.4: Výkon shlukové analýzy pro 20 shluků

Počet výpočetních uzlů	Čas (ms)
1	142582
2	180491
3	175378
4	171817
5	173178

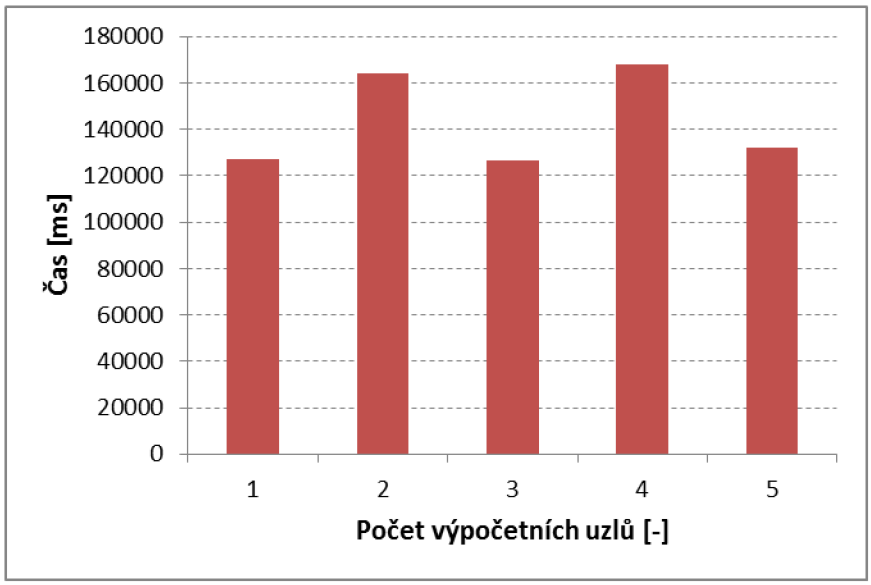
Tabulka 9.5: Výkon shlukové analýzy pro 30 shluků

Počet výpočetních uzlů	Čas (ms)
1	153593
2	104095
3	113913
4	106142
5	106528

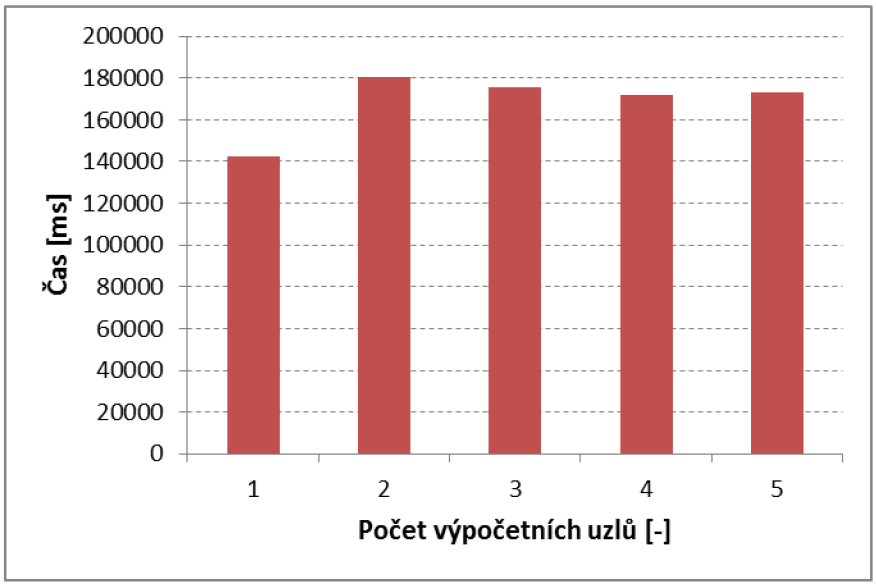
Graficky můžeme výsledky měření reprezentovat tak, jak ukazují následující obrázky.



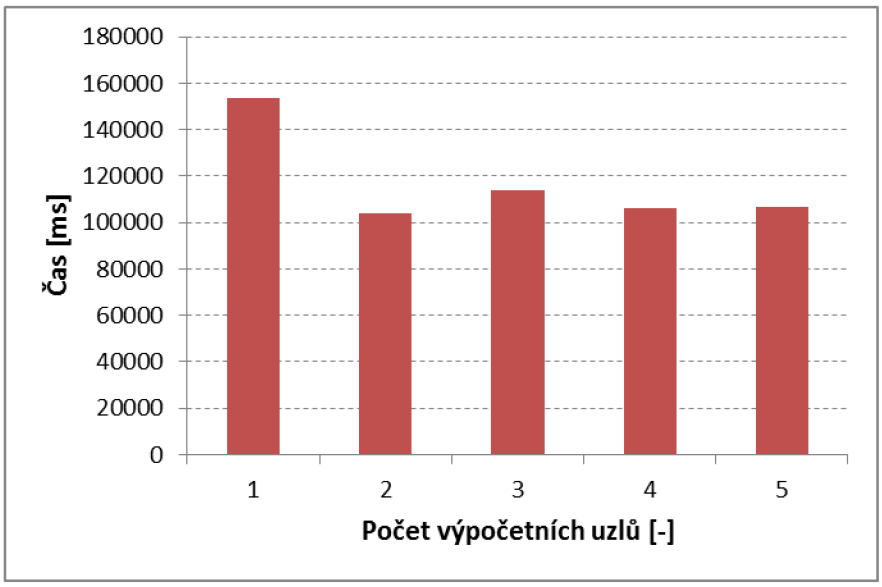
Obrázek 9.2: Výkonnost shlukové analýzy pro 2 shluky



Obrázek 9.3: Výkonnost shlukové analýzy pro 5 shluků



Obrázek 9.4: Výkonnost shlukové analýzy pro 20 shluků



Obrázek 9.5: Výkonnost shlukové analýzy pro 30 shluků

10 ZÁVĚR

V této bakalářské práci bylo navrženo prostředí pro škálovatelné strojové učení pomocí nástrojů Apache Hadoop a Apache Mahout a byla změřena jeho výkonnost. Dále byl nástroj Apache Hadoop porovnán s konkurenčními nástroji, kde ve srovnávací Tabulka 1.1 v kapitole Nástroje pro realizaci škálovatelné platformy byly vypsány základní rysy těchto konkurenčních nástrojů. Hlavním přínosem této práce byla praktická realizace škálovatelného prostředí pro algoritmy strojového učení a ověření jeho funkčnosti na zátěžovém testu pomocí shlukovacího algoritmu K-Means. Měřením výkonnosti bylo zjištěno, že škálování v nástroji Hadoop funguje dle teoretických předpokladů a z měření také vyplynulo, že je tato platforma tím více účinnější, čím více dat a složitějších funkcí musí zpracovávat. Nejvíce pozorovatelný rozdíl výkonnosti shlukové analýzy můžeme pozorovat u shlukování vstupních dat do 30 shluků, kdy je časový rozdíl mezi zpracováváním na jednom serveru a na pěti serverech téměř 1 minuta, tj. 5 serverů dokončilo výpočet téměř o minutu dříve, než 1 server. S malým množstvím dat a výpočetní zátěže tyto nástroje spotřebovávají relativně vysokou časovou režii pro řízení celého výpočtu na více serverech a tato režie může časově převýšit čas výpočtu samotného. V těchto případech malých výpočtů je tedy vhodnější použití nástrojů Hadoop a Mahout na jednouzlové architektuře, jak můžeme pozorovat u výkonnostního testu pro nalezení 2 shluků, kdy 1 server výpočet provedl o 1 minutu dříve, než jej provádělo 5 serverů. Mezi hlavní výhody nástrojů Apache Hadoop a Apache Mahout patří jejich svobodné licencování, relativně dobrý stav online dokumentace a možnost jejich implementace jak na svobodných operačních systémech, tak na OS Windows [4]. Jako další výhodu lze vyzdvihnout také jejich rychlý vývoj, kdy přibývá velké množství nových funkcí a zvyšuje se spolehlivost těchto nástrojů. Na druhou stranu ale nejčastěji používaná literatura [1],[2] nedokáže tyto změny dostatečně rychle reflektovat, proto se lze setkat v literatuře s principy, které už na současných stabilních verzích nástrojů Hadoop a Mahout nefungují. Důležitým aspektem výběru těchto nástrojů pro řešení problematiky škálovatelného strojového učení také může být možnost si oba nástroje přizpůsobit svým potřebám díky otevřenému zdrojovému kódu.

LITERATURA

[1] WHITE, T. *Hadoop, The Definitive Guide*. 1. vydání. Yahoo PRESS, 2009. 528 s. ISBN: 978-0-596-52197-4.

[2] OWEN, S., ANIL, R., DUNNING, T., FRIEDMAN, E. *Mahout In Action*. Manning Publications Co., 2012. 387 s. ISBN: 978-1-935-18268-9

[3] HAMMERBACHER, J. *What are some promising open-source alternatives to Hadoop MapReduce for map/reduce?* [online]. 2010, poslední aktualizace 20. 1. 2010. [cit. 2. 10. 2011]. Dostupné z URL: < http://www.quora.com/What-are-some-promising-open-source-alternatives-to-Hadoop-MapReduce-for-map-reduce>

[4] KOROLEV, V. *Hadoop on Windows with Eclipse* [online]. 2008. [cit. 2. 10. 2011]. Dostupné z URL: <http://ebiquity.umbc.edu/Tutorials/Hadoop/00%20-%20Intro.html>

[5] BIESACK, D. *Getting started with Hadoop* [online]. 2011, poslední aktualizace 15. 2. 2011. [cit. 2. 10. 2011]. Dostupné z URL: <http://wiki.apache.org/hadoop/GettingStartedWithHadoop>

[6] *Running Hadoop on Ubuntu Linux (Single node cluster)* [online]. 2009, poslední aktualizace 20. 9. 2009. [cit. 2. 10. 2011]. Dostupné z URL: <http://wiki.apache.org/hadoop/Running_Hadoop_On_Ubuntu_Linux_%28Single-Node_Cluster%29>

[7] *OpenSSH Manual Pages* [online]. 2008, poslední aktualizace 12. 7. 2008. [cit. 2. 10. 2011]. Dostupné z URL: <http://www.openssh.org/manual.html>

[8] GRUSOVÁ, L. *XML pro úplné začátečníky*. 1. vydání. COMPUTER PRESS, 2003. 208 s. ISBN 80-7226-697-7.

[9] *Oggdec* [online]. 2010, poslední aktualizace 29. 3. 2010. [cit. 2. 10. 2011]. Dostupné z URL: <http://rarewares.org/ogg-oggdec.php>

[10] WILDSTROM, S. *The World is parallel: Mining data on GPUs* [online]. 2010, poslední aktualizace 7. 5. 2010. [cit. 2. 10. 2011]. Dostupné z URL: <http://blogs.nvidia.com/2010/05/the-world-is-parallel-mining-data-on-gpus/>

[11] *HDFS Shell Guide* [online]. 2008, poslední aktualizace 20. 8. 2008. [cit. 2. 10. 2011]. Dostupné z URL: <http://hadoop.apache.org/common/docs/r0.17.2/hdfs_shell.html>

[12] *Reuters Test Collection* [online]. [cit. 29.5.2012]. Dostupné z URL: <http://www.daviddlewis.com/resources/testcollections/reuters21578/>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BZip2	Komprimační algoritmus s otevřeným zdrojovým kódem
C++	Objektově orientovaný programovací jazyk založený na C
CERN	Organisation européenne pour la recherche nucléaire, Evropská organizace pro jaderný výzkum, která provozuje u města Ženeva velký částicový urychlovač
CPU	Central Processing Unit, procesor počítače
CRC32	Cyclic Redundancy Check, algoritmus kontrolních součtů
Ctrl+O	klávesová zkratka
Ctrl+X	klávesová zkratka
DFS	Distributed File System, distribuovaný systém souborů
DNS	Domain Name Systém, systém doménových jmen
DNS/IP	Domain Name System/Internet Protocol, systém doménových jmen/internetový protokol
DTD	Definice Typu Dokumentu
FTP	File Transfer Protocol, protokol pro přenos dat po počítačové síti
GMPR	Graphics Procesing Unit MapReduce, zpracovávání MapReduce aplikací prostřednictvím grafických procesorů
GNU	GNU's Not Unix, rekurzivní zkratka pro označení operačního systému UNIXového typu distribuovaného pod svobodnou licencí
GZip	*GNU Zip (volně šířitelný komprimační program)*
HDFS	Hadoop Distributed File System, distribuovaný souborový systém nástroje Hadoop
HTTP	Hypertext Transfer Protocol, protokol pro přenos WWW stránek
JAR	Java Archive, komprimovaný program jazyka Java
JDK	Java Development Kit, knihovny pro vývoj v jazyce Java
JMX	Java Management Extensions, nástroje pro tvorbu Java aplikací
JRE	Java Runtime Environment, běhové prostředí programů Java
LOG	Soubor neurčitých stavových záznamů
MB	Megabyte, megabajt, 1048576 bajtů
MD5	Message Digest Algorithm, algoritmus kontrolních součtů
ms	Milisekunda
OGG	Formát pro kódování audia a videa s otevřenou specifikací

POSIX	Portable Operating Systém Interface, standard unixových OS
PB	Petabyte, petabajt, 1,12589991 . 10^{15} bajtů
RackID	Identifikátory serverových skříní
RAM	Radnom Access Memory, operační paměť počítače s náhodným přístupem k datům
RDL	Report Definition Language, speciální jazyk pro tvorbu reportů
RPC	Remote Procedure Call, vzdálené volání procedur
SASL	Simple Authentication and Security Layer, jednoduchá autentizační a zabezpečovací vrstva
SGML	Standard Generalized Markup Language, značkovací jazyk s podporou DTD
SOCKS	SOCKet Secure, internetový protokol který směruje pakety mezi klientem a serverem přes proxy server
SQL	Structured Query Language, strukturovaný jazyk pro vytváření databázových dotazů
TB	Terabyte, terabajt, 1099511627776 bajtů
URI	Uniform Resource Identifier, jednotný identifikátor zdroje
URL	Uniform Resource Locator, jednotné vyhledávání zdrojů
WAV	Waveform audio file, soubor pro ukládání zvukových dat kódovaných pomocí pulzně-kódové modulace
XML	eXtensible Markup Language, rozšířitelný značkovací jazyk
Zlib	Komprimační algoritmus s otevřeným zdrojovým kódem

SEZNAM PŘÍLOH

A POPIS OBSAHU PŘILOŽENÉHO DVD

Název souboru	**Popis souboru**
HelloWorldClustering.java	Ukázkový program shlukové analýzy K-Means z kapitoly Ukázkový program škálovatelného shlukování vektorů
LK_BP_2012.pdf	Elektronická verze této bakalářské práce