Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Hadoop a jeho využití pro Data Science

Diplomová práce

Autor: Bc. Jaromír Hnik
Studijní obor: Aplikovaná informatika

Vedoucí bakalářské práce: Ing. Karel Mls, Ph. D.

Hradec Králové Srpen 2019

Prohlášení:

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

V Hradci Králové dne 12.08.2019 Bc. Jaromír Hnik

Poděkování:

Děkuji tímto vedoucímu diplomové práce, kterým byl pan Ing. Karel Mls, Ph.D. za jeho metodické směrování a užitečné rady při tvorbě této práce. Stejným dílem bych chtěl poděkovat celé své rodině, přátelům i známým za podporu a trpělivost při studiu.

Anotace

Cílem této diplomové práce je hlubší a metodická analýza problematiky zpracovávání většího množství dat v reálném čase. V práci je detailně vysvětleno, jak k takovému problému přistupovat a jak dosáhnout efektivních výsledků s konkrétními řešeními. Čtenář této práce by po prostudování popsaných postupů měl porozumět, jak k takové analýze přistupovat.

Annotation

Title: Hadoop and its potential use for Data Science

The aim of this work is deeper and methodological analysis of the processing large amounts of data in real time. In this thesis is explained how to achieve effective results with maximum efficiency. Reader of this work should understand how to approach to analysis of big data and how to use effective algorithms for development.

Obsah

1 Úvod

Tématem této diplomové práce byla vybrána analýza problematiky Hadoop a jeho využití v Data Science. Toto téma bylo vybráno především k současnému stoupajícímu trendu o atraktivitu dané problematiky. V současné době je na efektivní studii jakéhokoli problému potřeba stále více dat, a to s sebou přináší úskalí toho, jak tato data zpracovávat efektivně v co nejkratším čase. Z tohoto důvodu je čtenáři popsána tato problematika tak, aby jí co nejlépe porozuměl a byl jí následně schopen aplikovat. Je nezbytně nutné uvést čtenáře do problematiky chronologicky, tzn., že je nejdříve vysvětleno, co je dle dnešních měřítek považováno za velké množství dat, následně co si má čtenář představit pod pojmem Hadoop a až následovně se přistupuje k problematice Data Science. Poté, co čtenář pochopí danou problematiku teoreticky, je nastíněna implementace postupů, jak správně a efektivně přistupovat k samotné implementaci teoretických znalostí. Je také popsáno, jaké jsou klady, ale i zápory tohoto řešení a proč se vlastně danou problematikou zabývat v širším měřítku.

2 Problematika velkého množství dat

Představení problematiky kolektivního sběru dat a informací, využití databází, popsání a definice Big Data[1].

2.1 Historie získávání informací

Jak je známo ze života, tak pro ucelený přehled o nějakém problému je zapotřebí sběr dat. Z těchto dat pak lze posuzovat konkrétní závěry a výsledky. Kolektivní sběr informací z dat si lze představit jako např. primitivní sčítání lidu. Tato činnost je prvně datována kolem roku 1800, díky sčítání lidu se také projevilo, že je zapotřebí data určitým způsobem analyzovat. S touto problematikou si poradil *Herman Hollerith*[2], který potřebný analytický stroj sestavil, je známý jako *Hollerith Tabulating Machine.* Tento přístroj obsahoval data z minulých sčítání lidu a dokázal analyzovat roky a měsíce. Zajímavostí je, že používal děrné štítky, které předznamenaly vznik elektronických počítačů [1]. V polovině 20. století nastala otázka, jak ukládat data, když vznikali první prototypy elektronických počítačů. V té době však také vznikaly polovodičové paměťové čipy, které nahrazovali klasické elektronky používané na stroji *Holleritha*, a to otázku ukládání dat částečně vyřešilo. Díky stále většímu komerčnímu zájmu o počítače rovněž narůstal zájem o více úložného prostoru. Například bankovní sféra již v té době viděla potenciál v další analýze zpracovávaných dat. V 80. letech minulého století se tak začala data ukládat do prvotních tzv. datových skladů, které byly realizovány velkým množstvím magnetických disků. Díky potenciálu analýzy firemních informací v bankovním sektoru také *Howard Dresner* tento jev pojmenoval dodnes používaným názvem „Business Intelligence“[3].

Naprostou revolucí ve sběru dat byl vývoj internetu. V 90. letech byl internet na počátku komerčního využití a díky němu se také začalo objevovat více datových typů a s tím spojené nové způsoby shromažďování dat. Bylo jasné, že tento tok dat bude třeba nějakým způsobem zpracovávat, ukládat a analyzovat. Časem však nárok na data ve větších společnostech stále rostl a bylo otázkou času, kdy zpracovávání takového toku dat bude problémem.

[1] Technická kategorie bez překladu pro velké množství dat
[2] Americký statistik a vynálezce, jeden ze zakladatelů IBM (1860–1929)
[3] Zobrazení historických, současných a prediktivních obchodních operací

2.2 Struktura dat

Pro celkové pochopení problematiky je na místě zmínit i to, jak vlastně počítač pracuje s datovou strukturou. Datová struktura v počítačové terminologii obecně znamená konkrétní způsob toho, jak jsou data organizována v paměti počítače. Tato organizace dat zajišťuje, že mohou být data efektivně využívána. Datové struktury obecně umožňují uchování a zpracování množiny dat stejných datových typů *(datové typy jsou více popsány v podkapitole 2.2.1).* Datové struktury obecně implementují jeden nebo více ADT *(abstraktní datový typ)*. Operace, které se nad ADT provádějí se označují kontraktem. Datová struktura je tak konkrétní implementace kontraktu, což lze chápat tak, že poskytuje sadu operací pro aktualizování, mazání, vyhledávání a vkládání dat. Tento soubor operací se pak nazývá rozhraním datové struktury. Při vývoji konkrétního software také záleží na celkové rychlosti programu i správné volby datových struktur [2]. Datové struktury jsou dále obvykle založeny na principu načítání a ukládání dat na jakékoliv místo v paměti počítače. Přístup k těmto místům určuje ukazatel neboli bitový řetězec určující konkrétní adresu v paměti. Pro datové struktury je stěžejní, jaké budou mít vlastnosti, kvůli následné manipulaci. Mezi tyto vlastnosti patří především rychlost čtení, zápisu a paměťová náročnost.

2.2.1 Datové typy

Obecná definice datového typu zní, že je to druh nebo význam hodnot, kterých smí proměnná nabývat. Datový typ je definován oborem hodnot a operacemi, které nad ním lze provádět. Z hlediska složitosti je dělíme na jednoduché a složené. Jednoduché datové typy jsou většinou zabudovány v konstrukci programovacího jazyka. Základním jednoduchým datovým typem je logická hodnota *boolean*[4], která určuje pravdivost výroku. Má tedy jen 2 logické stavy – pravda a nepravda. Mezi definici data číslem lze zařadit *integer*[5]. Nutno však zmínit, že integer slouží pouze k popisu celých čísel. Pro popis znaku se používá *char*, ve vlastní paměti počítače je však samotná reprezentace provedena pomocí celého čísla, kde jsou znaky kódovány převážně pomocí ASCII tabulky. Pokud chceme uložit reálné číslo, pak použijeme datové typy s plovoucí desetinnou čárkou *double, float, nebo real.* Všechny výše zmíněné lze klasifikovat, jako jednoduché datové typy. [3]

[4] Boolean nabývá hodnot true a false, v jazyce C není definovaný
[5] Rozsah je čísel je přesně definován, př. 16b – (-32 768 až 32 767)

Zatímco jednoduchý datový typ obsahuje vždy pouze jeden prvek, složený, jak již z názvu vyplývá je složeninou několika prvků dohromady. Mezi složené datové typy se řadí např. *array*[6]. Pro pole je typická jeho indexace, tzn., že se na konkrétní prvky v poli přistupuje pod jejich pozičním indexem. Pro některé programovací jazyky je nutné, aby pole obsahovalo pouze položky stejných jednoduchých datových typů. Výhodou však je, že pole prvků je efektivně využitelné pro rychlý přístup k položkám a operacím jednoduchých algoritmů, jako je např. řazení prvků. Mezi nejčastěji viditelné i pro neznalé patří *string*. Ten umožňuje skládání jakýchkoliv prvků, které budou zakódovány jako znak. To však také v praxi často znamená, že pokud se programátor rozhodne ukládat např. číselné typy do tohoto řetězce, musí je také zpětně přeložit na požadovaný datový typ. Pokud je nutné uložit větší počet řetězců, využívá se *list*, oproti poli však neumožňuje prvky indexovat, protože tento seznam je nutné procházet postupně. Existuje mnoho různých názorů na to, která struktura je výhodnější. Pokud máme zmínit konkrétní výhodu, pak je to, že u seznamů lze jednoduše přidávat nebo mazat prvky uprostřed, kdežto u pole taková operace znamená, že je nutné vždy uvažovat tak, že je nutné prvky před požadovaným zásahem někam překopírovat. Posledním důležitým složitým datovým typem je *record* [7]. Záznam je složen z několika různých datových typů a umožňuje programátorovi snadnější přenášení a volání. Výhodou seznamů je tedy, že vždy bude dodržena jeho definice.

2.3 Databázový přístup

Když už známe možné datové typy, tak můžeme přejít k tomu, jak tato data efektivně uchovávat. Databáze je obecně definována jako systém souborů, který má pevnou strukturu záznamů. Aby celá koncepce dávala smysl, jsou mezi sebou tyto soubory propojeny pomocí klíčů. Kvůli efektivnímu přístupu ke konkrétním datům se využívají softwarové prostředky neboli SŘBD[8]. Jak taková databáze funguje se popisuje často jako přirovnání ke kartotéce. Zde se záznamy zakládají pro každého pacienta zvlášť a mají často stejnou strukturu. Pro jejich lepší editaci a vyhledávání v nich pak slouží klasifikace dle jmen, nebo rodných čísel, což si lze připodobnit k unikátním klíčům, které jsou využívané v databázových systémech. Databázové systémy obecně lze klasifikovat na mnoho modelů.

[6] Pole – Může být i vícerozměrné. Dvourozměrné pole označujeme jako matici
[7] Záznam – Tento datový typ je označován také jako *structure, tuple nebo struct*
[8] Systém Řízení Báze Dat – Tvoří rozhraní mezi programy a uloženými daty

2.3.1 Hierarchické databáze

Tento databázový model je označován za původní. Pokud si představíme stromovou strukturu, tak přesně tak byla data organizována v hierarchické databázi. V minulosti byla často využívána, ale koncem 70. let minulého století tuto architekturu překonala koncepce síťová a relační.

2.3.2 Síťové databáze

Síťová architektura databází měla zdokonalovat hierarchickou koncepci. Přišla oproti hierarchické s novým vztahem více ku více, což znamená, že jedna entita *(objekt reálného světa, např. osoba)* mohla mít více entit předka. Tento koncept se uchytil především v 80. letech, kdy jej aktivně využívaly komerční databázové systémy. Síťová struktura databází se modelovala pomocí Bachmanových[9] diagramů.

2.3.3 Relační databáze

Dostáváme se ke stěžejnímu druhu databázových systémů. Tento typ vychází z databáze založené na relacích a predikátové logice. Tuto myšlenku představil v roce 1970 *E. F. Codd*[10]. Jeho formulace byla popis relačního modelu, který je hlavním stavebním prvkem relačních databází. Znamená to, že se data shromažďují do relací *(tabulek)* a obsahují n-tice *(řádky).* Relace jsou definovány atributy *(sloupci),* kde každý atribut nese jedinečný název a má přesně daný typ a rozsah, tzv. doménu. Takto definované relace se pak zaplní daty, která jsou jasně strukturována. Kolekce takovýchto relací, vztahů mezi nimi a indexů pak tvoří jako celek relační databázi [4]. Pro detailnější pochopení toho, jak relační databáze funguje je třeba také zmínit základní funkcionalitu toho, jak fungují klíče. Primární klíč je entita, která definuje určitou entitu z tabulky unikátně. Žádný primární klíč nesmí obsahovat hodnotu *NULL* a kvůli zachování entitní integrity tabulky by měla mít každá tabulka definován právě jeden primární klíč. V reálném světě si lze primární klíč představit jako DIČ podnikatelů, kde je jasně dané, že jedna podnikatelská entita má přiděleno právě jedno unikátní daňové identifikační číslo. Dalším je kandidátní klíč, který představuje sloupec, nebo jejich kombinaci, ve kterých jsou všechny řádky tabulky unikátní. Díky tomu lze pomocí kandidátního klíče jednoznačně identifikovat každý řádek relace. Jeden z těchto

[9] *Charles Bachman* (1924–2017), tento diagram popisuje entity a jejich vzájemné vztahy
[10] *Edgar Frank Codd* (1923–2003), v roce 1981 získal Turingovu cenu

kandidátních klíčů pak slouží jako klíč primární. Cizí klíč obecně ošetřuje integritní omezení, kdy do konkrétní položky v tabulce umožňuje vkládat pouze povolené hodnoty. Defacto tak udává vztah mezi dvěma tabulkami, kde hodnota v určitém sloupci jedné tabulky musí existovat v jiném klíči, jinými slovy popisuje referenci. Díky omezení pomocí cizích klíčů lze udržet referenční integritu databáze. Relační databáze jsou důležité pro datové sklady, které budou popsány dále. Dalším důležitým faktorem pro relační databáze je integrita databáze. To v praxi znamená, že data, která jsou v ní uložená splňují určité podmínky, díky kterým jsou konzistentní a splňují tak předdefinovaná pravidla. Pro příklad si lze představit, že pokud je určitý atribut definován, jako číselný typ *(např. telefonní číslo),* pak do něj lze zapsat pouze celé číslo, nikoli řetězec[11], to je modelová situace pro vkládání hodnot. Integritní omezení však fungují i při editaci relací. Například pokud smažeme záznam, který je vázán na jiné tabulky, pak se smažou i tabulky provázané, které již ztrácí bez předchozí tabulky význam.

Integritní omezení lze rozdělit do několika klasifikací.

- ***Doménové*** – Jak již bylo zmíněno, tak každý atribut má přesně definován datový typ *(doménu).* Díky tomu není možné do tabulky, kde je striktně definován datový typ ve sloupci přidat hodnotu jinou než požadovanou.
- ***Entitní*** – Zabraňuje se tomu, aby jedné entitě nebylo možno přidávat do databáze stejné neboli duplicitní záznamy. Tím pádem se zajišťuje unikátnost identifikátorů jednotlivých objektů.
- ***Referenční*** – Kontrola vztahů tabulek mezi sebou, kde je relace definována pomocí vazeb mezi primárním a cizím klíčem.

O tom, jak tato omezení co nejefektivněji kontrolovat lze polemizovat. Pokud se integritní omezení kontroluje pomocí jednoduchých mechanismů na straně databázového serveru, pak je metoda bezpečná, ale přináší určitou odezvu[12] na straně klienta. Oproti tomu, pokud vše kontrolujeme na straně klienta, pak je odezva nezatížena, ale je nutné počítat s omezeným použitím při přenášení na jiný databázový systém. V moderních databázích se užívá tzv. triggerů, což lze chápat jako jednoduché procedury, které jsou automaticky spouštěné před i po práci s daty. Díky přizpůsobitelnosti triggerů lze provádět i složitější operace. Ideální je tedy kombinace výše zmíněného v závislosti na konkrétním případu.

[11] Samozřejmě zde záleží na definování datového typu, pokud je požadován řetězec, pak lze
[12] Často označována jako latence. Jde o čas nečinnosti, kdy běží proces na pozadí a uživatel má tak dojem nečinnosti celého systému

Relační databázové tabulky mezi sebou mají nejčastěji binární vztah, který má různé kardinality[13]. Pokud záznamu v jedné tabulce odpovídá právě jeden záznam v tabulce druhé, pak je tento vztah označován jako 1:1. Nejpoužívanějším typem vztahu je však 1:N, ten je popsán tak, že je jednomu záznamu z tabulky přiřazeno více záznamů z jiné tabulky. Nejčastější je především kvůli reflexi reálného světa. Jako příklad může posloužit studium žáka ve škole. Žák studuje právě jednu školu, kdežto škola má několik studentů. Zbývá zmínit ještě vztah M:N, který umožňuje propojit záznam z tabulky k libovolnému počtu záznamů z druhé tabulky, přičemž lze zároveň propojit tento konkrétní záznam z druhé tabulky s libovolným počtem záznamů z tabulky první. Snadno lze tento princip pochopit na edukativním příkladu vztahů mezi herci a filmy. Jeden konkrétní herec může hrát v mnoha filmech, ale zároveň film obsazuje několik různých konkrétních herců. Realizace tohoto vztahu v databázovém systému se často provádí pomocí pomocné tabulky a rozložením vztahu na 1:N a 1:M. Aby byly relační databázové systémy co nejefektivnější, provází tvorbu i proces normalizace[14], který zjednodušuje a optimalizuje navržené struktury tabulky tak, aby se zamezilo zbytečným redundancím. Proces normalizace databáze umožňuje v celkovém důsledku s daty efektivněji pracovat. Celý proces je rozdělen do několika kroků. Po dokončení každého z nich se databáze nachází v tzv. normální formě. Normální formy lze rozdělit dle následujícího schématu.

- ***0NF*** – Tabulka musí obsahovat alespoň jeden atribut.
- ***1NF*** – Prvky v atributech musí být atomické[15].
- ***2NF*** – Každý neklíčový[16] atribut je plně závislý na kandidátním klíči.
- ***3NF*** – Mezi neklíčovými atributy relace neexistují žádné vztahy.
- ***4NF*** – Každý atribut popisuje pouze jednu souvislost, nebo fakt.
- ***5NF*** – Relace je dále nerozložitelná, což znamená, že přidáním libovolného atributu by se rozpadla na více tabulek.
- ***6NF*** – Řádek tabulky obsahuje primární klíč a pouze jeden další atribut[17].

[13] Kardinalita je popsání počtu vzájemných vztahů mezi tabulkami
[14] *E. F. Codd* definoval, že až normalizovaná databáze může skutečně těžit z výhod relačního modelu
[15] Jinými slovy dále nedělitelné, což znamená, že atributy neobsahují složené hodnoty
[16] Sloupec není součástí žádného kandidátního klíče
[17] Šestá normální forma byla představena v roce 2003 a jejím cílem je dekompozice relací na již dále neredukovatelné komponenty. Tato myšlenka přišla s využíváním datových úložišť a datových skladů

2.3.4 Objektově – relační databáze

Jak je jistě patrné, tak nároky na ukládání dat do databázových systémů stále rostou. Relační databáze nutí uživatele štěpit objekty na dílčí části, a tak přišla myšlenka ukládání celých objektů. V reálném světě vnímáme entity kolem nás jako objekty, díky tomu přišel požadavek s obdobným zpracováváním dat. ORDB *(Objektově – Relační databáze)* tak ukládají data přímo jako objekty. Při nástupu ORDB se vycházelo z objektově orientovaného programování, známé jako OOP. Ukazovalo se však, že přístup k objektům je složitější, proto bylo zapotřebí využívat ORM *(Objektově – Relační mapování)*. Objektově – relační mapování zajišťuje automatickou soudržnost dat mezi relační databází a OOP. Hlavním cílem ORM je tedy především synchronizace mezi používanými objekty v aplikaci a jejich reprezentací v databázi tak, aby data byla perzistentní [5]. ORM se snaží obecně oddělovat uživatele od nutnosti psaní SQL[18] kódu, mezi zástupce takového frameworku patří například *Hibernate*. Znamená to tedy, že se programátor ptá na objekty v databázi pomocí funkcí, které jsou ve frameworku definovány, místo toho, aby psal SQL kód a ptal se databáze napřímo. K ORDB se přistupuje pomocí dvou přístupů. Prvním je využívání univerzální paměti, kde SŘBD řídí všechny druhy dat, nebo druhým univerzálním přístupem, kdy jsou všechna data původní a přistupuje se k nim pomocí tzv. middleware. V ORDB jsou zachovávány relační základy tabulek a k datům lze stále přistupovat deklarativním způsobem. Atributy n-tic jsou však složité díky hnízděným relacím *(ve sloupcích mohou být další tabulky)*. Mezi hlavní ORDB řadíme např. Oracle[19] který má své vlastní abstraktní datové typy, kolekce, metody, nebo nabízí možnost ukládání velkých datových objektů.

2.4 Datové sklady

Nyní můžeme přejít k tomu, jak je definován datový sklad a proč se používá. Datové sklady jsou speciální typy relačních databází, které umožňují uživateli vykonávat úlohy zaměřené převážně na strukturované analytické dotazy nad větším množstvím dat. Aby bylo možno pochopit rozdíl mezi „obyčejnou“ relační databází a datovým skladem, definoval *W.H.Inmon*[20] čtyři charakteristiky, díky kterým lze dané rozdíly rozeznat.

[18] Structured Query Language – Strukturovaný dotazovací jazyk pro práci s relační databází
[19] Moderní multiplatformní databázový systém s širokou možností zpracování dat, vysokým výkonem nebo škálovatelností
[20] *William H. (Bill) Inmon* (1945), považován za otce teorií o datových skladech

- ***Subjektová orientace*** – U relačních databází je obvykle cílem co nejefektivnější uložení dat bez zbytečné redundance. Toho lze dosáhnout pomocí 3NF a vnitřním provazováním funkčních celků. Oproti tomu u DWH[21] je vždy cílem vytvořit strukturu, která bude čitelnější díky separaci vnitřních celků za cenu vyšší paměťové náročnosti.

- ***Integrita*** – Aplikace se u relačních databází soustředí na svůj konkrétní okruh úloh, kterými obsluhuje pouze svá specifická data. Oproti tomu u DWH je cílem seskupit informace z mnoha zdrojů podle jejich významu, nikoli podle původu. V praxi si lze tuto definici představit tak, že potřebujeme mít data, která spolu logicky souvisí pohromadě bez ohledu na to, odkud pocházejí.

- ***Proměnlivost*** – Zatímco u relačních databází se data mění často *(samozřejmě záleží na konkrétních případech užívání),* tak u DWH se data nahrávají v časových intervalech, většinou jedenkrát denně. K datům, které se na DWH uloží se poté pouze přistupuje, ale už se nijak neupravují.

- ***Historizace*** – V relačních databázích se data často přemazávají aktuálními, kdežto u DWH se shromažďují i data historická. Díky tomu lze analyzovat údaje v časovém měřítku, na což je datový sklad převážně zaměřen.

Díky výše popsaným znakům lze rozeznat odlišné nároky na DWH a relační databázi. Datové sklady mají rovněž několik základních charakteristik. Musí obsahovat nástroje pro nahrávání a ukládání dat z různých zdrojů v různých formátech a lze je ukládat na různá fyzická umístění. Datový sklad data obecně ukládá tak, aby bylo co nejefektivnější v nich vyhledávat, nikoliv je editovat [6]. Díky tomu lze provádět složitější dotazy rychleji, proto se používá pro ukládání dat technologie OLAP, více v kapitole 2.4.1. Při architektuře datových skladů se počítá pouze s typem úloh, které nad daty budou vykonávány, neví se však přesná struktura dotazů, proto je potřeba při tvorbě myslet na to, že je zapotřebí dostatek analytických nástrojů, se kterými bude uživatel později pracovat. Data jsou tak z pohledu uživatele členěná do schémat, kde každé schéma odpovídá konkrétní logické funkční specifikaci. Tato schémata jsou tvořena faktovými tabulkami a dimenzemi.

[21] Data Ware House – Angl. zkratka pro označení datového skladu

Faktové tabulky si lze představit jako hlavní prvek schématu, neboť jsou v nich uložena dále zkoumaná data. Tato data jsou následně využívána k analytickým výpočtům. Faktové tabulky jsou spojené s dimenzemi pomocí cizích klíčů *(viz. relační databáze),* a obsahují seznam dat, které slouží ke klasifikaci[22], nebo třídění dat ve faktových tabulkách. DWH je z paměťového hlediska obsazen převážně faktovými tabulkami, neboť jsou v nich uchovávána detailní data. Pro představu si lze uvést příklad toho, že je třeba evidovat údaje o prodeji vozidel. Pro schéma prodeje bude vytvořena faktová tabulka „Prodej“, kde bude pro každou položku ukládán záznam o značce vozidla, roku výroby, ceně atd. Pokud jde o globálního prodejce vozidel po ČR, pak lze vytvořit dimenzi „Pobočka“, a z hlediska časové analýzy lze vytvořit dimenzi „Datum prodeje“. Nezapomeňme však, že faktová tabulka „Prodej“ musí být pomocí cizího klíče spojena s oběma dimenzemi. Pokud by však bylo zapotřebí tvořit dimenze, které budou nějakým způsob kategorizovat konkrétní prodej, dostáváme se tím do problému toho, že lze dané odvětví dále štěpit. Vytvoříme-li dimenzi „Typ vozu“, lze jí kategorizovat do dalších odvětví (např. do 3.5t, nad 3.5t, nákladní) a vnořená kategorie „do 3.5t“ je možná dále dělit (např. osobní, VAN, kombi) atd., tím pádem zde mluvíme o hierarchické dimenzi.

Abychom byly schopni taková data zpracovávat, máme na výběr dvě možnosti, které odráží faktory reálného světa.

- ***Hvězdicové schéma*** – Dimenze, která se dále rozpadá se přetvoří na jednu dimenzní tabulku, kde budou kategorizační údaje ukládány redundantně. Díky tomu vznikne schéma, kde je každá dimenze vázána pomocí cizího klíče na faktovou tabulku.

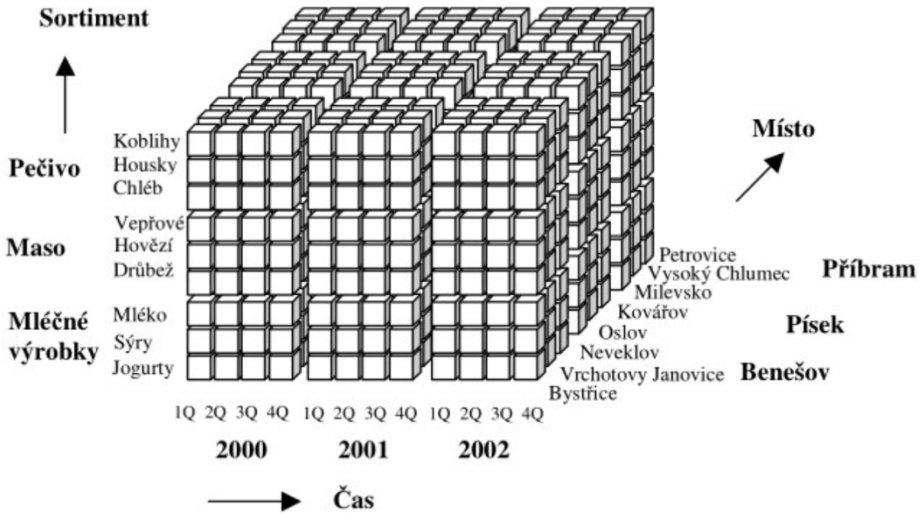
- ***Vločkové schéma*** – Na dimenzi, která se dá dále hierarchicky rozdělit se aplikuje 3NF. Tím pádem bude na faktovou tabulku napojena pouze dimenze na nejnižším stupni hierarchie. Zbylé dimenze pak budou napojené na některou dimenzi, která je v hierarchickém stromu níže [7].

Data uložená v datových skladech mají své využití především pro Business Intelligence, tvoření historických analýz, průzkum trhu a podporu při rozhodování *(Decision Support Systems).* K celkovému použití se však dostaneme v dalších částech této práce.

[22] Díky klasifikaci lze rozeznávat logické začlenění dat. Klasifikace je detailněji popsána v kapitole strojového učení

2.4.1 OLAP a OLTP

OLAP *(Online Analytical Processing)* technologie ukládání dat v databázi umožňuje uspořádat velký objem dat tak, aby byl přístupný a uživatelsky srozumitelný pro uživatele. Cílem OLAP je rychlé zpracování hledacích dotazů. Díky tomu se tvoří tzv. manažerský systém relace, kde je možná duplicita dat kvůli rychlejšímu přístupu[23]. OLAP je datový sklad, a tak využívá faktových tabulek a dimenzí, celková koncepce je však tvořena pomocí datové krychle. Tento přístup se používá především kvůli efektivitě, neboť v ní lze tzv. kostkovat, což znamená, že se omezují atributy dimenzí na množiny hodnot. Dále se lze v datové krychli pohybovat nahoru a dolu, což si lze např. v časovém měřítku představit jako zanořování, kdy z roku přejdeme na jednotlivé kvartály zkoumaného období. Pro detailnější analýzu se využívá také pivotování[24], což je efektivní při hledání závislostí, nebo trendů díky změně pořadí atributů v dimenzi [7].



Obr. 1 - Znázornění OLAP datové kostky, *převzato z [7]*

Jako další lze užívat OLTP, což je technologie, která není zaměřená na analýzu, ale na editaci dat pro mnoho uživatelů zároveň. Tato technologie je používána v mnoha komerčních sférách, neboť je třeba obsluhovat tisíce uživatelů najednou. Celá koncepce tak klade důraz na bezchybné zpracovávání transakcí, a až poté na vyhledávací a čtecí funkce.

[23] Celková relace tak nemusí být ve 3NF
[24] Pivot obecně znamená první nenulový prvek v seznamu

2.5 Big Data

Po vysvětlení souvisejících témat se dostáváme na problematiku big data[25]. Jako definice toho, co jsou to big data by mělo postačit, že to je takové množství dat, které nelze zpracovávat běžnými softwarovými prostředky v reálném čase. Taková data jsou často ukládána v datových skladech kvůli zefektivnění možnosti jejich zpracovávání. Lze si tento datový sklad představit, jako centrálu, kam se všechna data ukládají z různých datových zdrojů a systémů pomocí procedur. Tyto procedury fungují tak, že se data extrahují z relačních databází, tabulek a dalších zdrojů, kde se data následně transformují do požadovaného formátu. Tato data se v časových intervalech nahrávají do datových skladů a jsou přístupná i v historické podobě. Těmto procedurám se říká ETL[26]. Big data jsou obecně označována, jako data, která jsou ve velikosti petabytů, a více. Zde nastává otázka, proč pouze neupravit přístup a nezvolit výkonnější procedury k analýze takového množství dat, ale důvody, proč to nejde jsou obecně známé jako 4V[27], které big data charakterizují [8].

- ***Volume*** – Jde o množství vygenerovaných a uložených dat. Důležitý náhled na to, jestli lze data považovat za big data však spočívá v tom, zdali s množstvím přidávaných dat jejich objem narůstá exponenciálně.

- ***Variety*** – Popisuje typ dat. Slouží pro analýzu toho, jak z dat efektivně čerpat důležité vlastnosti. Počet těchto vlastností však vzrůstá díky digitalizaci většiny firemních potřeb (text < obrázek < video, …). V tomto kontextu jde především o různorodost struktury dat, která stále stoupá.

- ***Velocity*** – Důležitým faktorem je také rychlost zpracovávání požadavků vzhledem k velikosti a struktuře dat. Rychlost, jakou dat přibývá je v dnešní době digitalizace vysoká, ale potřeba zpracovávat tato data v reálném čase zůstává.

- ***Veracity*** – Vzhledem ke zmíněné digitalizaci lze pokládat otázku, jak poznat, která data jsou věrohodná. Hodnotí se proto index toho, jak jsou data označována za relevantní, a jak ovlivní následnou analýzu.[28]

[25] Big Data je pojem, který se nemusí překládat, v češtině se však zřídka používají i „veledata“
[26] Extract, Transform, Load – Extrakce, transformace a nahrání dat do datového skladu
[27] Mnoho zdrojů se rozchází v názoru, jestli je čtvrté V – Veracity (věrohodnost) rozhodující
[28] Příkladem může být sociální síť, kde se ukládá konverzace. Zde lze těžko posuzovat věrohodnost informací

Big data zahrnují strukturovaná, polo-strukturovaná a nestrukturovaná data. Na tento druh dat je obecně zatíženo největší množství prostředků, neboť je s nimi práce nejsložitější. Jednou z velkých výhod využití big data je machine learning[29], který bude popsán v kapitole 4.3. Velkým specifikem big data je, že vztahy mezi jednotlivými daty je nutno odhalovat. Big data je obecně velice těžké zpracovávat především kvůli jejich složitosti, a klasické datové sklady nestačí na analýzu takto velkých dat v přijatelném krátkém čase, případně real-time. Díky tomu vzniklo framework řešení Hadoop, které je detailně vysvětleno v kapitole 3. Dá se říci, že big data přišla masivně s digitalizací všeho kolem nás. Typickým příkladem, kde lze zachytit takový proces ukládání velkého množství dat je například v automobilovém průmyslu. Formule 1 testují své jízdní vlastnosti a potřebují zaznamenávat a selektovat obrovské množství informací, které jsou sbírány prostřednictvím senzorů. S postupem času a vývojem informačních systémů se nároky na paměťová úložiště stupňovaly. Například, když v roce 1992 firma *Teradata Corporation* jako první analyzovala 1TB dat, mluvilo se o této události jako průlomové a 1TB dat byl v té době považován za *enormní (pro představu se v té době do PC běžně dodávaly HDD[30] s pamětí 1 GB).* S postupem času také vyspěl názor na to, jak big data zpracovávat. Zprvu se jevilo, že jde pouze o větší množství dat a vyšší matematický výkon by mohl postačit, používaly se proto superpočítače, které byly schopné provádět mnohonásobně více matematických operací v reálném čase, než běžné PC. V roce 2004 však Google publikoval teorii o architektuře MapReduce, která využívá paralelního zpracování dat [9]. O tom, jak přesně MapReduce funguje je však popsáno více v sekci Hadoop. Ve zkratce však lze říci, že tento typ zpracování funguje tak, že se distribuce dotazů dělí a posílá paralelně přes uzly v clusteru[31]. Výsledky dotazů se pak shromáždí a doručí. Vícevrstvá architektura se tak jevila jediným možný řešením, jak big data efektivně zpracovávat. Paralelní architektura totiž zasílá data mnoha serverům naráz, a to tak výrazně zvyšuje rychlost výpočtů. Tento typ architektury zasílá data do paralelních SŘBD, kde se využívá Hadoop. Celý tento proces se děje v pozadí a uživatel jej může kontrolovat přes UI[32]. Big data mají základní vlastnosti a charakteristiky, ty v roce 2011 definoval *McKinsey Global Institute* třemi hlavními body, které stručně big data ekosystém popisují [10].

[29] Strojové učení z analýzy dat. Díky velkému množství dat lze jednodušeji predikovat a poznat vzory
[30] Hard Disk Drive – Paměťové médium obsahující pohyblivé mechanické části, které se užívá v PC, dnes vytlačováno pomocí disků využívajících paměti flash *(SSD)*
[31] Jde o sdílení paměťových prostředků, které vede k vyššímu toku dat
[32] User Interface – Prostředí aplikace, v tomto případě Ambari UI

- Mají techniky pro analýzu dat, jako A/B testování[33], strojové učení a v neposlední řadě NLP[34].
- Využívají se techniky a principy z databází, business intelligence nebo cloud computingu[35].
- Data se vizualizují, což znamená, že lze provádět základní analýzy přímo v UI.

V současné době je mnoho odvětví, kde big data efektivně využívat. Vznikl tak defacto nový trh. Dodavatelé se snaží cílit na jednotlivá pokrytí segmentů trhu a tento trh se dělí na čtyři základní segmenty. Nejprve potřebujeme hardware k tomu, aby byla celá síť vybudována *(úložiště, servery atd..),* následně potřebujeme data distribuovat po celé síti, k čemuž slouží již zmíněný Hadoop, dále je třeba vyřešit management ukládaných dat *(NoSQL DB, integrace dat),* a nakonec potřebujeme data analyzovat pomocí vizualizace a analytických platforem. Samotná reprezentace big data je pak realizována několika způsoby, např. pomocí datových kostek nebo tenzorů[36]. Další technologie se snaží o zdokonalení toho, jak budou big data využívána, jako např. MPP[37] databáze, data mining[38], nebo distribuovaný prostor *(souborový systém, databáze).*

S nástupem big data a teoretických poznatků toho, jak data analyzovat byla tendence tyto principy přenést i na menší sdílená úložiště, jako jsou NAS[39]. Tento koncept se však ukázal, jako neefektivní, protože NAS úložiště jsou pomalá a drahá, což je v rozporu s definicí toho, co big data jsou *(je kladen důraz na rychlost zpracování, infrastrukturu a nízké pořizovací náklady).* Trend dnešní doby zkrátka nutí k využití výše zmíněných principů. Pro představu bylo v roce 2017 přes internet přeneseno 1,5 ZB[40]. Predikce Cisco však odhadují na rok 2022 až 4,8 ZB ročně, což je ohromný nárůst [11].

[33] Testování dvou verzí *např. webových aplikací* naráz. Sleduje se výkon, který je pak dále využitý
[34] Natural Language Processing – Počítačová věda, kde se program snaží zpracovávat a porozumět pomocí AI lidskému jazyku *(rozpoznávání řeči)*
[35] Využívání výpočetního prostoru skrze internet, nejde jen o zálohování dat na cloud
[36] Obdoba datové kostky, která pracuje ve vektorovém prostoru a fungují v ní tak matematické operace, které jsou s vektory spojené
[37] Massively Parallel – Využívá velkého množství procesorů, které provádějí výpočty paralelně
[38] Proces, který se snaží rozpoznat vzory v dané struktuře. Doslovným překladem je těžení dat
[39] Network Attached Storage – Jedná se o vybudování sítě, které umožňuje sdílet fyzické úložiště s klienty
[40] Zettabyte – 1 ZB = 10^{21} B, což je přibližně velikost jedné miliardy TB

3 Apache™ Hadoop®

Tato kapitola představuje detailní popis toho, co vlastně framework Hadoop je, jak v celkovém měřítku funguje, z čeho se skládá a jaké je jeho využití.

3.1 Základní specifikace

Jak již bylo zmíněno v kapitole 2.5, tak Hadoop je stěžejním prvkem při zpracovávání big data. Tento framework je open-source licencovaný produkt firmy Apache. Je to framework, který obsahuje SW komponenty, které umožňují zpracovávat a analyzovat velké množství nestrukturovaných distribuovaných dat téměř v reálném čase, a to až v řádech EB[41]. Myšlenka vychází z výše zmíněného definování MapReduce firmou Google, kde se tento princip využívá k indexaci webu. Hlavní charakteristika Hadoopu je, že pracuje s daty, která jsou distribuovány přes více uzlů současně. Tyto uzly se nazývají ***nodes*** a dohromady tvoří ***cluster***. Základní funkcionalitu lze popsat tak, že MapReduce *(je popsáno v kapitole 4.2)* přistupuje k datům, která jsou distribuována různě po datových centrech, nebo internetu, a rozdělí je do replikací, které pak paralelně zpracovávají jednotlivé uzly. Nad jednotlivými uzly lze také vykonávat dotazy. Všechny uzly pracují paralelně, což logicky zvyšuje rychlost zpracování. Poté, co jsou odpovědi poskytnuty se výsledky shromažďují do tzv. HDFS (*Hadoop Distributed File System)*, což je úložná vrstva Hadoopu. Na HDFS jsou napojená analytická prostředí, což znamená, že zde lze provádět analýzu zpracovaných výsledků k dalšímu použití. Jak lze z textu odvodit, Hadoop přináší odlišný přístup pro zpracovávání nestrukturovaných, nebo částečně strukturovaných dat nejen z hlediska rychlosti, ale především nákladů. Pro celkové nasazení ekosystému a složitější analýzy dat však potřebuje uživatel vysokou znalost celého prostředí. Tohoto faktu se ujalo pár velkých počítačových firem *(Oracle, IBM, ...),* které danou situaci využily a nabízejí celý ekosystém přímo koncovým zákazníkům. Vždy se však tento ekosystém skládá z výše popsaných čtyř odvětví *(HW, big data, management dat, vizualizace).* Díky tomu, že je Hadoop open-source[42] se tak nabízí v současné době mnoho možných řešení při celkové tvorbě komerčních ekosystémů, ale vždy se vychází z principů, které jsou závislé na Hadoopu.

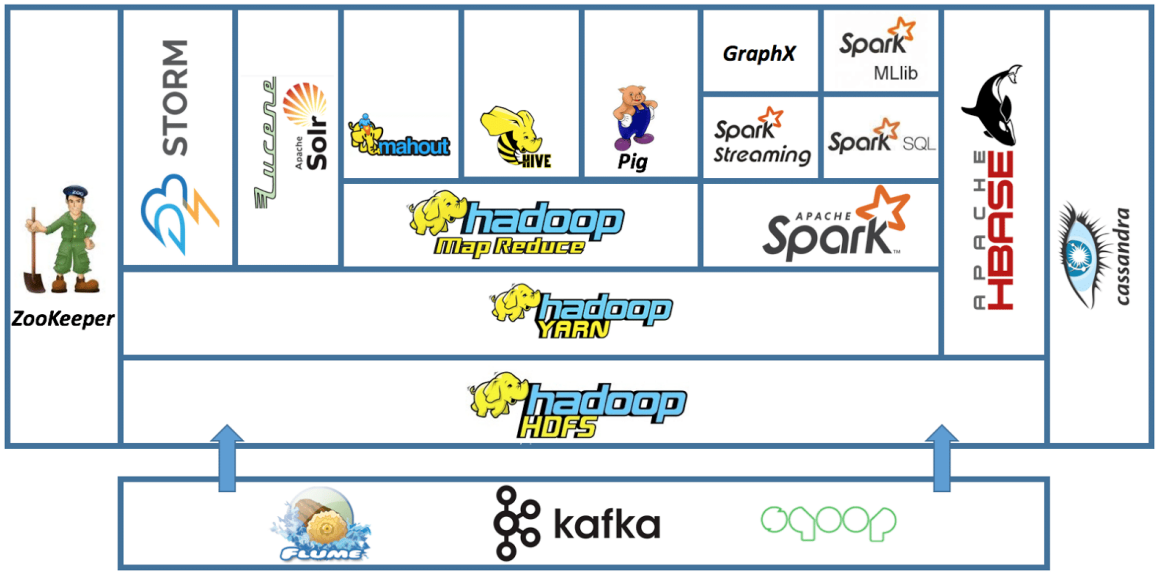
[41] Exabyte - 1 EB = 10^{18} B, což je přibližně velikost jednoho milionu TB
[42] Open-source znamená, že zdrojový kód je veřejný. Lze tak SW s licencí od Apache šířit dále

„The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.“ [12]

3.1.1 Pochopení clusteru, nodes a utility machine

Pro další postup je nutné chápat rozdíly, mezi tím, co je node a cluster. ***Cluster*** je rozšiřovatelný, což si lze v praxi představit, že může být velikosti jednoho až tisíce počítačů připojených do jednoho ekosystému. Pomocí redundantních informací lze rozeznávat chyby v systému, které se zobrazují v UI. Clustery mají obrovskou výhodu toho, že je lze realizovat pomocí obyčejných HW komponent, které jsou v dnešní době snadno dostupné *(není potřeba superpočítačů).* Tím se rapidně snižují pořizovací náklady spolu s náklady na opravy. Jak již bylo zmíněno, tak Hadoop umožňuje vysoký stupeň škálovatelnosti[43]. Díky distribuované architektuře lze nejen rozdělovat data, ale také výkon. To má za následek, že zpracovávání požadavků je rovněž distribuované, tím lze docílit výkonu se stovkami CPU a GB paměti RAM. Pokud se přesuneme k uzlům, pak je nutné rozlišovat hlavní a vedlejší *(MasterNode, WorkerNode).* ***MasterNode*** má za úkol zpracovávat úkoly, které se týkají celého clusteru. Běží na nich hlavní procesy celého Hadoopu *(např. má na starost spouštění NameNode, který koordinuje ukládání dat)*. Lze postavit ekosystém, který využívá pouze jen MasterNode, avšak kvůli škálovatelnosti se procesy Hadoop často rozkládají mezi více nodes. ***WorkerNode*** má oproti tomu na starost zajištění a poskytování výpočetního výkonu *(RAM, CPU)* spolu s poskytováním lokálního úložiště k ukládání a zpracovávání dat. Na WorkerNodes zpravidla běží pracovní procesy Hadoopu *(např. má na starost spouštění DataNode, který pracuje pod vedením MasterNode).* Zmíněný ***DataNode*** má na starost samotné čtení a zápis datových bloků do úložiště. Protože Hadoop je framework, je nainstalován na straně klienta. Tento počítač se obecně označuje ***Utility Machine***. Znamená to, že tento PC je bránou k přístupu do celého systému a je na něm spuštěno Ambari, nebo Knox *(více v kapitole 3.3)* ke správě ekosystému a zabezpečení přístupu do clusterů [13].

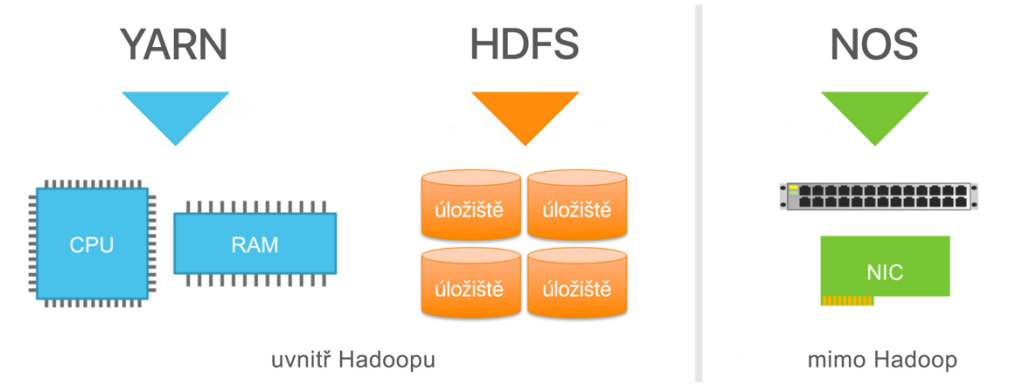
[43] Možnost rozšiřování o další datová úložiště



Obr. 2 - Schéma frameworků v Apache Hadoop, *převzato z [14]*

3.2 Resource management v Hadoop clusteru

Při zpracovávání dat hrají důležitou roli zmíněné WorkerNodes, které zajišťují potřebný výpočetní výkon *(CPU, RAM),* úložný prostor a síťové prostředky pro celý Hadoop cluster v globálním měřítku. Pro správu paměti RAM a výpočetního výkonu jednotlivých WorkerNodes slouží služba ***YARN***[44] a pro správu úložné paměti clusteru slouží ***HDFS***. Síťové prostředky na jednotlivých WorkerNodes jsou řízeny jejich operačním systémem, nebo přímo je řeší přímo switch síťovým operačním systémem, zkráceně NOS.



Obr. 3 - Resource management v Hadoop clusteru, *vlastní tvorba dle [13]*

[44] Yet Another Resource Negotiator – Oficiální název. Jeho funkce je detailněji popsána v kapitole 3.6

3.3 Dílčí frameworky celého ekosystému Hadoop

Hadoop funguje tak, že se skládá z několika různých softwarových frameworků *(většinou od Apache),* kde je každý zaměřen na určitou funkcionalitu. Pro celkové pochopení ekosystému je dobré ty nejdůležitější alespoň abstraktně vysvětlit [15].

3.3.1 Data management

Prvním okruhem je Data Management, sem lze zařadit již zmiňovaný HDFS a YARN. Tyto frameworky se primárně starají o správu dat.

- ***HDFS*** – Jak již bylo nastíněno, tak se jedná o souborový systém. HDFS je distribuovaný souborový systém, postavený na Java jazyce. Je škálovatelný, nabízí spolehlivý přístup k aplikačním datům a je podobný mnohým komerčním souborovým systémům. To v praxi znamená, že nabízí funkce, jako např. R/W[45] nebo práci s adresáři.

- ***YARN*** – Framework, který je zodpovědný za Resource management a job scheduling *(plánování úloh)* celého clusteru. Lze si YARN představit, jako samotné jádro systému, které má na starost správu výkonu. Díky němu lze provádět operace, jako jsou např. real-time streaming, data science atd.

3.3.2 Správa a kontrola úkolů

Aby byla zajištěna obsluha operací, existují frameworky, které mají na starost kontrolu operací v celém clusteru, mezi nejdůležitější patří Ambari a ZooKeeper.

- ***Ambari*** – Jedná se o UI framework, se kterým administrátor reálně pracuje. Aplikační prostředí nabízí nástroje pro řízení a kontrolu Hadoop clusterů. Zároveň obsahuje mnoho REST API[46], díky nimž se pro uživatele zjednodušuje složitost celého Hadoopu a systém se stává intuitivnější.

- ***ZooKeeper*** – Vzhledem k celkové složitosti celého systému je zapotřebí tento ekosystém monitorovat, neboť je pravděpodobnost, že celá síť bude náchylná k poruchám. Je potřeba monitorovat problémy se synchronizací,

[45] Operace Read, Write, Delete – Možnosti čtení, zápisu a mazání dat jako u jiných systémů *(např. Linux)*
[46] RESTful API – Využívá metod GET, POST, PUT, DELETE aj. na datových souborech

škálovatelností, nebo výpadky jednotlivých zařízení, což způsobuje nekonzistentnost. ZooKeeper je framework, který obsahuje sadu nástrojů na opravu těchto běžných provozních chyb.

3.3.3 Přístup k datům

Nyní je na místě zmínit frameworky, které k samotným datům přistupují. Na rozdíl od YARN a HDFS jsou přímo navrženy tak, aby byly schopné s daty pracovat.

- ***Hive*** – Apache Hive je framework, který v Hadoopu tvoří strukturu datového skladu. Hlavní předností je, že umožňuje zkušeným uživatelům databázových systémů pracovat nad big data stejným způsobem, tzn., pomocí dotazů SQL.

- ***Pig*** – Framework Apache Pig obsahuje skriptovací procedurální jazyk, který umožňuje uživateli analýzu, nebo transformaci velkých datasetů.

- ***HBase*** – Jedná se o NoSQL[47] databázi uvnitř celého ekosystému. HBase je zaměřena především na velmi velké relace, obsahující miliony sloupců a miliardy řádků. I přes takto velké rozměry tabulky je schopen tento framework přistupovat k datům v reálném čase. Zároveň se skrze něj provádějí příkazy jako např. *update, insert* a *delete* v celém Hadoopu.

- ***Spark*** – Apache Spark je open-source framework, který umožňuje psát vývojářům aplikace k využití data science na Hadoopu. Framework Spark poskytuje řadu API, se kterými lze psát aplikace např. v jazyce Scala[48], zároveň podporuje dotazy SQL, aplikace pro streaming dat, a především machine learning.

- ***Storm*** – Apache Storm je distribuovaný výpočetní systém pro zpracovávání nepřetržitého proudu dat v reálném čase. Rozšiřuje možnosti MapReduce a Tez[49], čímž poskytuje tento výkon celému clusteru v Hadoopu.

[47] Databáze, která na rozdíl od relačních databází využívá jiných principů, než tabulek kvůli jednoduchosti
[48] Jde o hybridní funkční a OOP programovací jazyk
[49] V podstatě framework, který zvyšuje výkonnost MapReduce tím, že snižuje nároky na jednotlivé frameworky (Hive, Pig, atd.) a celý systém se tak stává rychlejším

- ***Solr*** – Jedná se o distribuovanou vyhledávací platformu. Tento framework poskytuje celému ekosystému možnost komfortnějšího vyhledávání požadovaných dat *(vzory chování, korelace, vztahy mezi jednotlivými daty atd.)* neboť dokáže indexovat petabyty dat celého clusteru.

Existují i další frameworky, které určitým způsobem přistupují k datům (*HCatalog, Cascading, Phoenix, aj.)*, výše zmíněných se však nejvíce využívá ke zkoumání data science.

3.3.4 Správa řízení Hadoopu

Výše popsané frameworky řeší přístup k datům na technické úrovni. Je však nutné znát i frameworky, které umožní přístup datům přímo uživateli. Jde tedy o to, jak s daty manipulovat.

- ***HDFS NFS*** – Pro celkovou správu dat je HDFS NFS zřejmě nejzásadnějším z pohledu uživatele. V principu jde o to, že se HDFS může napojit na lokální NFS souborový systém klienta. Lze tedy přes NFS připojit *root* adresář HDFS jako svazek, a následně využívat skriptů, aplikací nebo například průzkumníka souborů k manipulaci s adresáři/soubory v HDFS.

- ***WebHDFS*** – Pro lepší obsluhu dat a následnou manipulaci s nimi přišla myšlenka napojení Hadoopu na webové API. WebHDFS využívá známých metod *GET, POST, PUT* a *DELETE*, díky kterým lze přistupovat[50] k datům. Tímto frameworkem lze také ručně měnit oprávnění souborů a složek.

- ***Falcon*** – Apache Falcon je nástroj pro správu dat, který monitoruje data workflow, umožňuje konfiguraci lifecycle[51] dat, a především umožňuje uživatelům Hadoopu napojování nových dat a konfiguraci data managementu.

- ***Sqoop*** – Za zmínku ještě stojí framework Apache Sqoop, který obsahuje sadu nástrojů pro import a export datových souborů mezi Hadoopem a relačními databázovými systémy. Celkově tak tento framework ulehčuje využívání skriptů nebo principů MapReduce pro přenos souborů.

[50] Myslí se tím zobrazení, vytváření, mazání, zapisování a čtení adresářů a souborů
[51] Konfigurace životnosti dat, což si lze představit jako časový úsek, po jakou dobu mají být data aktivní

3.3.5 Zabezpečení

Důležitým faktorem celého ekosystému je rovněž řešení zabezpečení. Z hlediska funkcí si zabezpečení mnoho frameworků řeší sama. HDFS řeší oprávnění přístupu ke složkám a souborům. Řeší také to, že jsou data šifrována. YARN obsahuje seznam přístupů k využívání výpočetního výkonu celého clusteru. Díky tomu tak může určitý framework využívat pouze to, co mu YARN přidělí. Z hlediska dílčích frameworků tedy existují defacto dva hlavní, kterých se využívá individuálně.

- ***Knox*** – Jedná se o gateway[52], která chrání přístup do Hadoop clusteru. Veškerý přístup do clusteru tak musí jít přes tuto bránu a tím pádem lze filtrovat, co je povoleno.

- ***Ranger*** – Apache Ranger je bezpečnostní framework, který má v sobě obsaženo předdefinované chování pro dílčí frameworky celého ekosystému, čímž se ošetřuje nebezpečné chování, které by mohlo rozladit celý Hadoop. Zároveň lze skrze Ranger konzoli nastavovat zásady přístupu k souborům, adresářům nebo celým relacím pro jednotlivé uživatele.

3.3.6 Cluster management

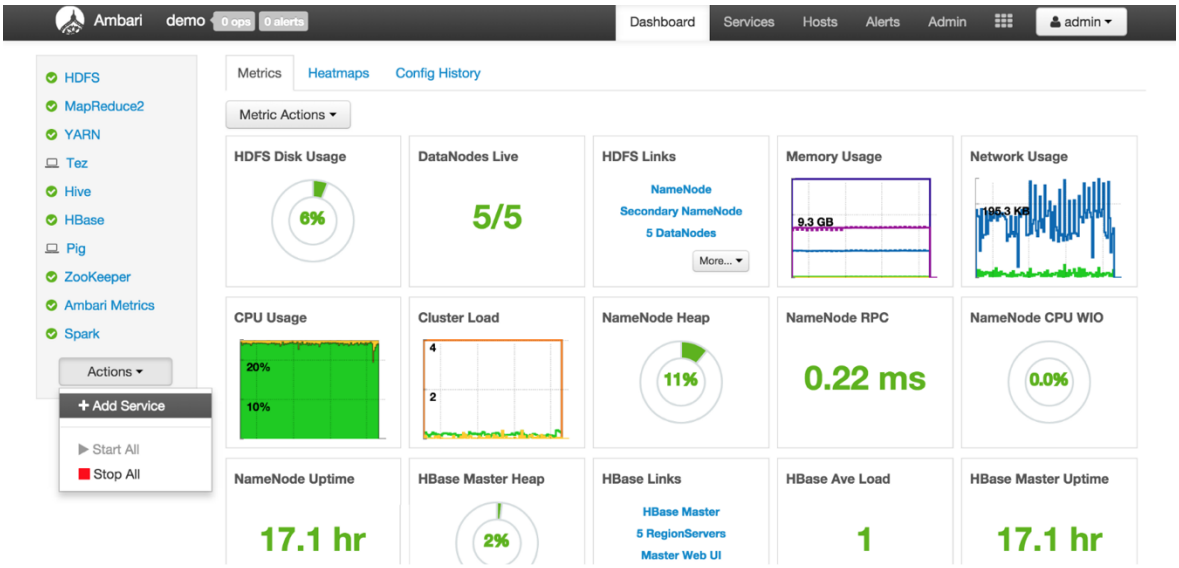
Pro správu celého clusteru se dá využít mnoho možností. Nejefektivnějším je však Ambari agent, který pomocí webového rozhraní dokáže kontrolovat celý ekosystém. Kvůli komerčnímu užití je tento způsob rovněž nejefektivnějším z hlediska škálovatelnosti[53]. Pro správu celého Hadoop clusteru se využívá výše zmíněných frameworků, kde má každý na starost svou danou úlohu. Pro celkové fungování v business odvětví je ale rovněž důležité využívat frameworků třetích stran, neboť každý use-case[54] je jiný a mnohdy jsou potřeba odlišné cíle. Nejdůležitějším faktorem v celém fungování clusteru je však jednoznačně automatizace. Systém by měl být navržen tak, aby uživatel nemusel zasahovat při činnostech, které se mnohou provádět autonomně, ale aby se systémem pracoval na abstraktní vrstvě *(snaha ulehčit práci pro uživatele bez nutnosti konfigurace všeho).* Je však nutné zmínit, že i tak se jedná o těžký proces, kdy je nutná hluboká znalost všech komponent.

[52] Uzel, skrze který jde veškerá komunikace z vnější do interní sítě
[53] Myšlena přizpůsobitelnost pro jednotlivá užití
[54] Konkrétní model využití. Většinou je use-case obdobný, neboť je třeba dělat analýzu a predikci s datovými soubory

3.4 Ambari

Je na místě přiblížit si celou funkci Apache Ambari. Bylo již řečeno, že Ambari funguje jako server, který sbírá data z celého Hadoop clusteru a je přístupný přes REST API. Stěžejním prvkem Ambari je webové UI, ale uvnitř jde o kolekci celé topologie a konfiguračních souborů do svojí relace. Aby na Ambari Serveru mohl fungovat celý ekosystém, je nezbytně nutné mít nainstalovaného agenta Ambari na každém node clusteru, díky čemuž lze ovládat každý tento uzel individuálně, a ne jako celek. K webovému rozhraní (*viz. obr. 4)* se přistupuje pomocí zadání adresy agenta obyčejně ve webovém prohlížeči. Autentizace přístupu je založena na obyčejném přihlašování pod uživatelským jménem a heslem. Pro různé uživatelské operace nám pak slouží Views[55], pomocí kterých lze monitorovat různé sekce celého clusteru. Slouží také k upozorňování na chyby, které se běžným provozem zkrátka dějí. Toto upozornění na chyby je způsobeno monitoringem neobvyklého chování a je zobrazeno v Ambari Serveru, kde se pomocí notifikace zobrazí detail chyby a nabídne se možnost automatické opravy. Víme však z předchozí kapitoly, že tuto opravu řídí ZooKeeper a většinou se povede předdefinovanou funkcí chyba opravit. Zde je jasně viditelné, proč je nutné znát celou problematiku alespoň abstraktně před tím, než se do analýz a monitoringu dat administrátor pustí. V každém clusteru je rovněž s Ambari agentem nainstalován i monitor metrik, který slouží k zasílání aktuálních informací do serveru, kde se pomocí webového UI zobrazují. Poslední důležitou zmínku si zaslouží možnost konfigurace uživatelů a jejich skupin. Uživatelé se ve výchozím nastavení konfigurují do lokální databáze, ale v praxi se využívá přístup LDAP[56], nebo Active Directory[57]. To znamená, že lze jednoduše vytvořit celou hierarchickou strukturu oprávnění například podle pracovních pozic ve firmě *(samozřejmě uživatelům, kteří potřebují s Hadoopem pracovat)*. Databáze samotného Ambari *(výchozí se využívá Derby, lze překonfigurovat na Oracle, MySQL nebo PostgreSQL)* si uchovává pouze data o konfiguracích clusteru a celkové topologii spolu s údaji o uživatelích. Je však důležité rozlišovat uživatele Ambari a uživatele celého Hadoopu. Celý server Ambari je napsán v jazyce Java, přičemž samotné rozhraní napsáno v Javascriptu[58]. K následné komunikaci mezi serverem a agenty se využívá skriptů v jazyce Python[59].

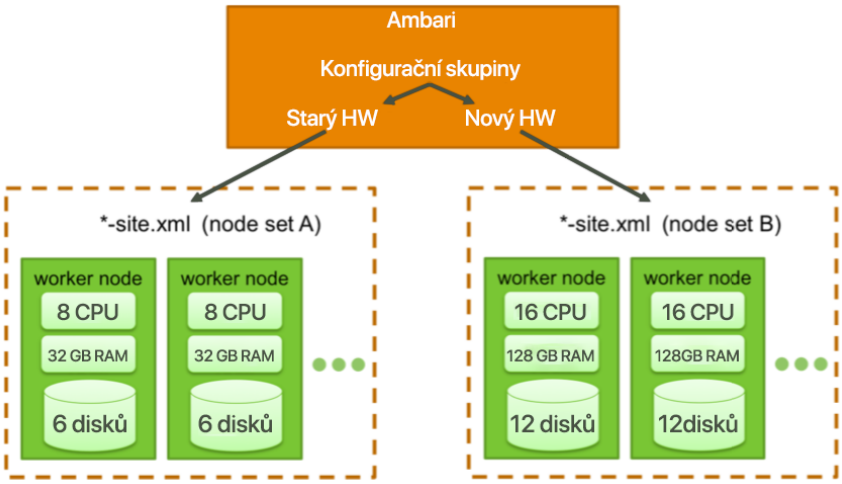
[55] Sekce v Ambari s předdefinovanými funkcemi, grafy a analýzou dat
[56] Lightweight Directory Access Protocol – Ukládání dat o uživatelích do stromové struktury na server
[57] Název adresářových služeb v LDAP – Použito ve Windows 2000
[58] Multiplatformní, objektově-orientovaný jazyk, patřící do rodiny C/C++/Java
[59] Skriptovací jazyk podporující přístupy OOP, procedurálního nebo funkcionálního programování



Obr. 4 - Ambari UI, kde lze vidět monitoring služeb, *převzato z [16]*

3.4.1 Nastavování konfiguračních skupin

V konfiguraci Ambari je mnoho možností, jak nastavovat požadované chování a přizpůsobovat celý systém dle jednotlivých use-case[60]. V praxi se většinou řeší koncová zařízení stejnými typy počítačových zařízení, tudíž lze vytvářet stejné konfigurace systému. Stává se však, že časem je třeba upgrade a namísto nahrazení celé sítě se pouze přidávají nová zařízení [17]. Konfiguračními skupinami však lze oddělit konfigurační zásady chování pro „starý“ hardware a přiřadit odlišné konfigurace pouze nově přidaným.



Obr. 5 - Nastavení konfiguračních skupin dle HW, *vlastní tvorba dle [17]*

[60] Na to, jak správně nastavit celý Hadoop jsou vedena několikadenní školení pro administrátory dat. Často jsou principy podobné, ale musí se dbát na cíl analýzy a další využití

3.5 Hadoop Distributed File System

Hadoop je obecně navržen pro práci s různými typy souborových systémů, ke kterým mohou uživatelé přistupovat. Přístup k různým souborům je zajištěn pomocí příkazového řádku s podobnými příkazy, jako v Linuxu, avšak s jasně daným prefixem.

```
hdfs dfs -ls file:///slozka
hdfs dfs -ls hdfs:///slozka
```

Obr. 6 - *Kód*: Příkaz pro výpis složky na lokálním úložišti a HDFS

Je na místě zmínit i to, že celý Hadoop je nakonfigurován pomocí vlastností v souboru *core-site.xml*, který je v rootu[61] celého systému. HDFS je zde nastaven pomocí vlastnosti *fs.defaultFS = hdfs://<nazev_NameNode>:8020*. Toto nastavení je platné pro každou NameNode[62]. Souborový systém HDFS má hierarchickou strukturu, obdobně jako NTFS[63] nebo ext4, což znamená, že existuje tzv. top-level adresář *(root)*, který může mít podřízené adresáře atd. HDFS je však důležitý především kvůli tomu, že jde o distribuovaný souborový systém. Díky tomu je škálovatelný a vhodný pro R/W operace s extrémně velkými daty. Velká data jsou díky HDFS automaticky distribuována mezi více diskových úložišť, a díky tomu se zvyšuje i rychlost operací R/W.

3.5.1 Replikace dat

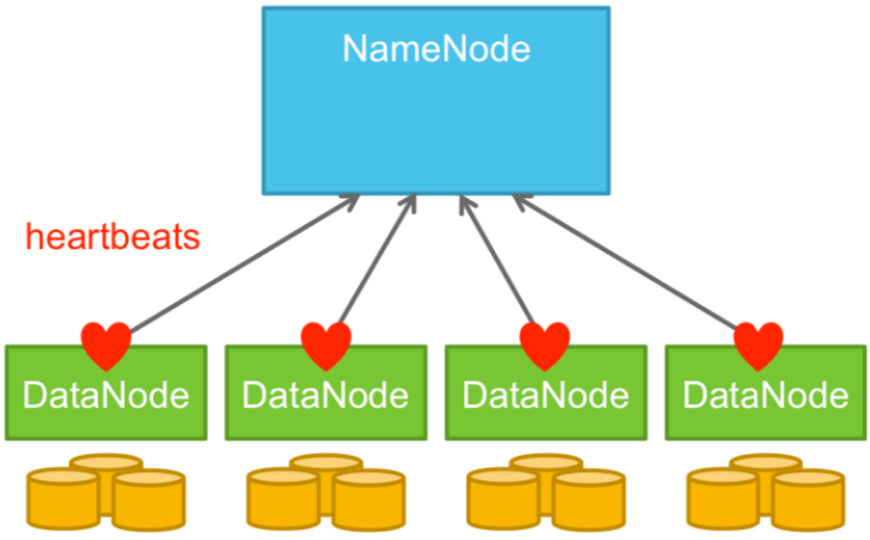
Díky škálovatelnosti se zajišťuje i ochrana dat pomocí replikací. Každý datový blok je replikován automaticky. Standardně se data replikují třikrát, což znamená, že HDFS zajišťuje, aby se každý datový segment uchovával nezávisle na sobě ve třech různých discích v samostatných systémech. Jak již bylo zmíněno na začátku práce, tak se předpokládá, že big data jsou zapisována jednou a několikrát dokola čtena, nebo dále využívána. S touto myšlenkou pracuje i replikace. Kvůli tomu, že jsou data rozmístěna na třech různých úložištích HDFS podporuje i více uživatelů přistupujících k jednomu souboru najednou, kde sám systém určí, ke které replice konkrétní uživatel zrovna přistupuje. Kvůli ochraně integrity dat jsou však tyto replikované soubory zamčené, aby nebylo možné, že budou do jednoho souboru zapisovat dva uživatelé najednou [18].

[61] Kořenový (nejvyšší) adresář celé struktury
[62] Co je to NameNode je přesněji vysvětleno na další straně v kapitole 3.5.2
[63] New Technology File System – Souborový systém využívaný ve Windows, navržen již koncem 80. let

3.5.2 NameNode a DataNode

HDFS má dva hlavní komponenty, které se starají o celkovou logiku funkčnosti. Prvním z nich je NameNode, který je hlavní a udržuje v sobě informace o prostoru a metadatech v HDFS. Tím se myslí např. *názvy souborů, názvy adresářů, oprávnění přístupu k souborům, celkovou hierarchii složek nebo například timestamp*[64] *poslední modifikace.* Data jsou ukládána redundantně, neboť pokud by došlo ke ztrátě vazeb na NameNode, ztratil by se taky přístup k samotným datům. Tím se dostáváme k DataNode.

Jak z názvu vyplývá, tak DataNode se starají o samotná data, což znamená, že obsahují pouze samotné datové bloky[65]. Každému datovému bloku je přiřazeno jedinečné ID a název. Pomocí NameNode se tak celkový soubor propojí a stane se přístupným, neboť samotný DataNode nelze bez této vazby na NameNode namapovat. Jak již bylo zmíněno, tak se zde využívá replikace. Pro upřesnění se replikují pouze datové bloky napříč různými DataNodes. Samotná funkcionalita je pak závislá na kontrolování dostupnosti, která probíhá tak, že si NameNode vysílá signál, na který mu DataNode musí odpovědět. V Hadoop terminologii se těmto signálům říká *heartbeats*, díky čemuž se ošetří tzv. *fault tolerance*[66]. Tento signál je vysílán každé tři sekundy a je vyžadována odpověď. Pokud však DataNode do třiceti sekund neodpoví, považuje se za poruchový. Do takového DataNode se pak data dále nezapisují[67] a je nutné chybu opravit.

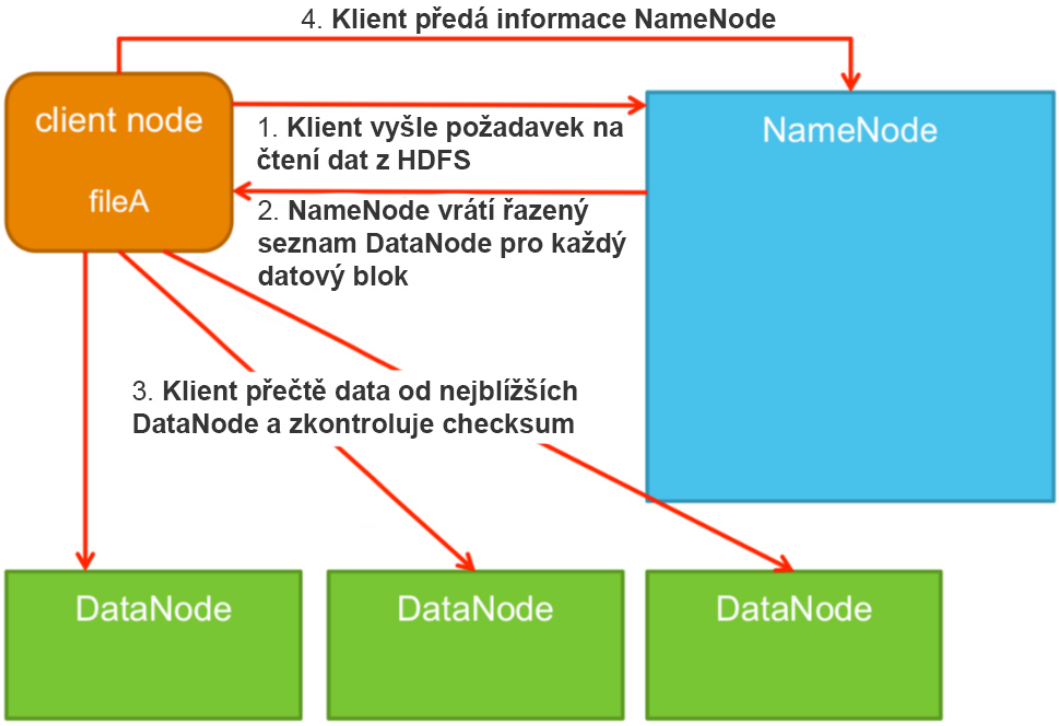


Obr. 7 - Komunikace mezi NameNode a DataNode, *vlastní tvorba dle [19]*

[64] Tzv. časové razítko. Je to časový údaj, který je vlastností souboru
[65] V Hadoopu je výchozí velikost bloku maximálně 128 MB
[66] Zjištění nestandardního chování, což znamená poruchu systému
[67] Je výjimka, kdy lze do porušeného DataNode ukládat, a sice, když je konfigurace předělána manuálně

3.5.3 Princip čtení dat

Bylo zmíněno, že v Hadoopu se klade velký důraz na efektivitu a rychlost práce s daty. S tím souvisí i to, že se data nikdy nepřesouvají skrze NameNode, ale vše je čteno napřímo z DataNode. Celý princip se tak dá popsat následujícím způsobem. Pokud klient vyšle příkaz ke čtení souboru, pak HDFS vyšle příkaz NameNode k určení umístění datových bloků konkrétního souboru. Pro každý blok pak vrátí NameNode klientovi seřazený seznam DataNodes s jeho replikami v závislosti na vzdálenosti od klienta[68] pomocí kterého se k datům následně přistupuje. Toto řazení je důležité, pokud je klient HDFS součástí clusteru, neboť se pak načítají datové bloky z lokálního DataNode. V závislosti na tomto řazení také funguje prevence chybné komunikace. Pokud klient obdrží chybnou komunikaci s DataNode, zkusí přistoupit k dalšímu DataNode v pořadí a zapisuje si chybný DataNode, aby k němu při dalším čtení nepřistupoval. Klient zároveň ověřuje *checksum*[69] každého datového bloku a pokud nalezne chybný datový blok, pak to dá vědět NameNode, který automaticky přepíše chybný datový blok z vytvořené repliky [20].



Obr. 8 - Komunikace při čtení dat, *vlastní tvorba dle [20]*

[68] Využití nejrychlejšího přístupu, protože klient následně bude přistupovat k latentně nejnižšímu DataNode
[69] Kontrolní součet – Slouží k tomu, aby se porovnávala pouze datová informace a ne obsah

3.6 YARN

Dalším důležitým prvkem fungování Hadoopu je YARN. Jde o výpočetní framework pro Hadoop. YARN se považuje za clusterový operační systém[70], který je centrem celého Hadoopu. Hlavní důvod využívání YARN je to, že funguje obdobně jako operační systémy, které známe. Má tedy na starost přidělování prostředků pro výkon *(CPU, RAM, paměťové úložiště)* pro celý cluster. Pokud je zkrátka třeba, aby nějaký framework přistupoval k HDFS, musí se tak stát skrze přidělení prostředků od YARN. Ten zároveň zajišťuje tzv. *queries*, což znamená, že jsou některé úlohy upřednostňovány v přístupu k jednotlivým prostředkům. YARN má na starost také job scheduling, což znamená, že se skrze YARN dají tyto úlohy plánovat. Tento framework využívá své vlastní vícevrstvé architektury[71] *(multi-tenant architecture)*. Funguje tak, že se rozpozná, pokud jedna aplikace funguje pro více uživatelů najednou a každý uživatel sdílí jeden společný přístup k aplikaci, HW a datovým souborům. Každý uživatel má však svůj speciální identifikátor, na základě, kterého YARN udělí konkrétnímu uživateli požadovaná oprávnění[72]. Dá se to tedy přirovnat ke službě SaaS *(Software as a Service)*.

3.6.1 Architektura YARN

YARN se skládá ze dvou hlavních komponent ResourceManager a NodeManager. ***ResourceManager*** je spuštěn na hlavním uzlu clusteru *(master node)* a má na starost správu prostředků pro všechny aplikace, které na YARN běží. Jeho hlavní činnost by se dala rozdělit do třech hlavních částí:

- ***Plánování*** – Aby byla zvýšena celková efektivita využívání zdrojů, lze plánovat, jaké skupiny uživatelů budou mít přístup ke konkrétním zdrojům pod hlavičkou sdruženého clusteru. Dá se tak nastavit, že pokud jedna uživatelská skupina potřebuje více zdrojů, může si je vypůjčit od skupiny jiné, čímž se efektivita využívaných zdrojů zvyšuje. Je také možné kontrolovat, který uživatel ke konkrétním zdrojům přistupuje. O tom, jak se plánování využívá rozhoduje administrátor celého Hadoopu.

[70] Pro připodobnění je oproti tomu HDFS souborovým systémem
[71] U YARN se tento pojem označuje jako „multitenancy“
[72] U klasické vícevrstvé architektury *(multi-instance architecture)* se o požadované zdroje „soutěží“

- ***Správa uzlů*** – Správa jednotlivých uzlů a aplikací se provádí pomocí podpůrných služeb. Obdobně jako u HDFS se využívá *heartbeats* pro kontrolu systému, což zajišťuje NodeManager monitoring. Zde se odesílá *heartbeat* každou sekundu a pokud se NodeManager *(resp. přímo ApplicationMaster)* do 10 min neozve zpět, pak se pokusí o restart.

- ***Bezpečnost*** – ResourceManager řídí přístup ke zdrojům pomocí ACL[73]. Zabezpečení se provádí pomocí předávaných tokenů, které ověřují platnost požadavků. ApplicationMaster musí předat informaci o prostředcích, které by měli být přiděleny na NodeManager. Skrytě se tak kontroluje, zdali ApplicationMaster nepožaduje více lokálních prostředků, než povolil ResourceManager.

NodeManager se spouští na každém uzlu clusteru *(worker node)* a provádí řízení místních zdrojů dle toho, co mu ResourceManager přikáže. Pokud se zašle požadavek na ResourceManager o přidělení zdrojů skrze ApplicationMaster, pak NodeManager vytvoří kontejner, kterému dané prostředky přidělí. Kontejner si lze představit jako místo, které má na starost řízení aplikace. Tento kontejner pak spolupracuje s danou aplikací a ta tak může svou činnost vykonat. Kontejnery se vytváří s každým požadavkem na ResourceManager skrze ApplicationMaster [21].

ApplicationMaster je spouštěcí proces požadavku na aplikaci. Každý framework má odlišný ApplicationMaster *(Hive se liší od Storm atd.)* ApplicationMaster vyjednává s ResourceManagerem prostředky, které budou v kontejneru sloužit pro konkrétní aplikaci. Zároveň je zodpovědný za monitorování spotřeby zdrojů daných kontejnerů, přičemž spolupracuje s NodeManagerem, který dané kontejnery spouští. ResourceManager tedy v globálním měřítku monitoruje a schvaluje požadavky na úrovni clusteru, kdežto ApplicationMaster má na starost samotné řízení v konkrétním uzlu spolu s NodeManagerem. ApplicationMaster je rovněž zodpovědný za fault tolerance. Tím se zvyšuje škálovatelnost na tisíce pracovních uzlů v clusteru, neboť za chyby v aplikacích není zodpovědný ResourceManager.

[73] Access Control List – Definovaný seznam uživatelů nebo skupin, které mohou přistupovat k určitému elementu (v tomto případě ke zdrojům). Definuje se buď v Ambari nebo pomocí příkazů

Tomu, jak takové plánování aplikací probíhá se říká *job scheduling*[74]. Jde o to, že ApplicationMaster prohledá výkonnostní možnosti celého clusteru a poté se spojí s ResourceManagerem. O požadované zdroje si zažádá, a od ResourceManageru obdrží token, který mu umožní použít zdroje, které jsou mu přidělené. ApplicationMaster se pak spojí s NodeManagerem, díky kterému se vytvoří kontejner s alokovanými zdroji. Poté, co se kontejner vytvoří, je ApplicationMaster sám zodpovědný za požadovanou konfiguraci tak, aby se úkol dokázal realizovat. Tento proces se opakuje v závislosti na náročnosti daného úkolu, dokud kontejner nevyčerpá všechny zdroje, nebo se úkol nesplní celý.

Obr. 9 - Představa NodeManageru a ApplicationMasteru, *vlastní tvorba dle [21]*

3.7 Dodatečné informace k Hadoopu

V této kapitole je kladen důraz na to, aby byla vysvětlena celková problematika stěžejních prvků a principů celého Hadoopu. Existuje mnoho dalších věcí, které lze zmínit, ale z hlediska dalšího zkoumání v data science to není nezbytně důležité. Pro kompletní správu Hadoop ekosystému je důležité umět abstraktně všechny okruhy, protože jak je patrné z výše uvedeného, tak jsou komponenty navzájem provázané svou funkcionalitou. I v dnešní době je lidí, kteří by uměli Hadoop a znali danou problematiku alespoň abstraktně velmi málo. Když už se podnikové řešení rozhodne využívat celý Hadoop, pak se administrátoři zabývají spíše tím, jak obsluhovat Ambari a nastavit určitá specifika pro konkrétní sledovaný segment trhu. Jak již bylo zmíněno, tak na to, aby se člověk stal Hadoop administrátorem je třeba absolvovat několikadenní školení, které mu znalost prohloubí, ale vše ho naučí až jedině praxe a „osahání“ systému.

[74] V Hadoopu jde o specifické označení konkrétního postupu mezi ApplicationMaster a NodeManager

4 Data Science

Po probrání problematiky Hadoopu lze přejít k tomu, co to vlastně data science je a jaký je přínos tohoto vědního oboru.

4.1 Popis problematiky

Data science je vědní obor, který se zabývá objevováním a zkoumáním informací z širokého spektra dat. Často využívá matematických a algoritmických operací k tomu, aby spolehlivě určil požadované hodnoty. Často se zkoumá obrovské množství dat, aby byly nalezeny vnitřní vztahy k tomu, aby bylo dále možné nalezení určitých souvislostí, které mohou vést k prediktivnímu určení dalších dat.

4.1.1 Rozšíření kolem nás

Principů data science se využívá v dnešní době všude kolem nás, ale lidé to často nevnímají, a ani o tom neví. Historie využívání poznatků analýzy dat sahá do 80.let minulého století, kde se objevili kvantitativní analytici na Wall Street, kteří pomocí analýzy dat zkoumali, jestli jsou akcie nadhodnocené, nebo podhodnocené. Pro ilustraci efektivity data science může sloužit i kniha *Moneyball*[75] [22]. Tato kniha pojednává o tom, jak sportovní manažer baseballového týmu pomocí matematické analýzy získal mladé levné hráče s atributy, které potřeboval. Díky tomu, že si sestavil dokonalý model úspěšného týmu a pomocí jednotlivých aspektů hráčů požadovaný tým sestavil. Zatímco špičkové týmy té doby vsázely na drahé hvězdy, tak tento tým s levnými hráči vyhrál celou baseballovou divizi. V dnešní době je však používáno obdobných přístupů téměř všude kolem nás. Placené reklamy od firmy Google *(Google Ads)* využívají strategie CPC[76] nabídek na základě prediktivních modelů chování daného publika, kdy dokáží odhadnout skutečný zájem o dané téma a na základě toho se stanoví maximální nabídka pro koupi prostoru na reklamu. U sociálních sítí *(Facebook, LinkedIn atd.)* se například využívá společenských grafů, kdy se tvoří množiny společných přátel a na základě toho se uživateli doporučí osoba, kterou by mohl znát. Stejného přístupu využívá například i Netflix[77], který analyzuje potřebné

[75] Autorem knihy je *Michael Lewis* a byla vydána v roce 2004
[76] Cost Per Click – Cena za proklik u placené reklamy. Efektivní je, pokud není cena příliš vysoká
[77] Americký poskytovatel filmů a pořadů online. Je dostupný ve 190 zemích světa v několika lokalizacích

vlastnosti u pořadů, které uživatel sleduje a na základě toho vnitřní algoritmus doporučí, který další pořad by se mu mohl líbit. Principů data science využívá rovněž data mining. Konkrétně se těchto principů uplatňuje v analýze a statistice prodejů *(MBA[78])* určitého produktu. Obchodní firma tak dokáže sledovat chování zákazníků a na základě získaných dat např. poskytovat promo akce na zboží, o které je nejvyšší zájem.

4.1.2 Proč využít Hadoop

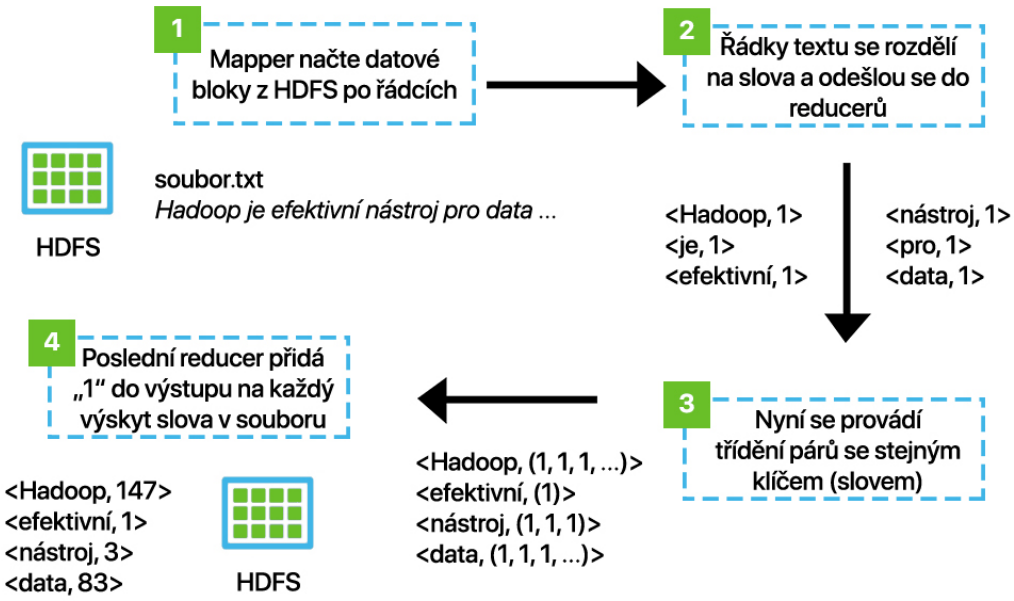
Je evidentní, že pro využití data science je třeba velké množství dat. Jak je popsáno v předchozích kapitolách, tak je HDFS tím správným řešením pro distribuci big data tak, aby se k nim přistupovalo s největší efektivitou. Principů data science by samozřejmě šlo uplatnit i s využitím relačních databází, ale použitelnost v reálném prostředí by byla téměř nulová. Vždy závisí rychlost operace především na zatíženosti disků. Relační databáze využívají indexace, čímž náklady na diskové zatížení snižují. Indexy lze načíst do paměti RAM a díky nim lze rychle prohledávat určité množství dat. Problém s velkým množstvím dat však nastává, když se indexace už nevejde do operační paměti, nebo výsledek operace vyžaduje nadměrnou alokaci diskového úložiště. Hadoop však předpokládá, že k operaci bude potřeba načíst velké množství dat a přečte rovnou celý soubor. Uveďme si výhodu tohoto zpracování na příkladu.

Pokud máme 500 GB datový soubor, který je třeba pro další analýzu, pak by se v relační databázi zpracovával při 1030[79] MB/s necelých 62 minut *(což, jak tušíme není dostatečné, když víme, že big data jsou i v řádech petabytů)*. V Hadoopu však tento 500 GB datový soubor můžeme rozdělit na dva tisíce bloků s velikostí 256 MB[80], což znamená, že pokud by uživatel zadal např. vyhledávací dotaz, tak by se zpracovávalo 2000 operací. Pro přečtení celého disku jsou zapotřebí přibližně dvě sekundy. Pokud poskládáme cluster tak, že má např. 40 DataNodes *(v každé DataNode osm disků)*, pak bude stačit cca 16 sekund ke zpracování celého datového souboru. Samozřejmě se zde bavíme o pouhém přečtení dat, pokud by byla operace spjata s MapReduce, trvala by déle. Bavíme se zde však stále o řádech sekund a nikoliv minut. Z toho je jasně dokazatelné, že využití Hadoopu k dalším analýzám, nebo jen vyhledávání dat posouvá efektivitu na zcela odlišnou úroveň.

[78] Market basket analysis – Analýza chování nákupu zákazníka. Sleduje se, jaké položky zákazník nakupuje
[79] Standardní maximální přenosová rychlost disků je 7200 otáček za minutu
[80] Zvýšená velikost datového bloku. Výchozí je 128 MB, jak již bylo zmíněno v kapitole 3.5.2

4.2 Využití MapReduce přístupu

Jedná se o software framework, díky kterému lze zpracovávat velké množství dat napříč paralelní distribucí souborů v Hadoopu. Celý přístup je založen na mapování a redukci fragmentů. Tento přístup se dá jednoduše popsat tak, že se data vkládají do ***Mapper*** komponenty, kde se transformují a následně jsou předány do ***Reduceru***, který si transformovaná data načte a provádí na nich různé výpočty, nebo analýzy [23]. MapReduce úlohy probíhají ve vlastních JVM *(Java Virtual Machine)* na DataNodes. Mapper zpracovává hodnoty z HDFS a tvoří strukturu *klíč/hodnota*. Po dokončení mapování se data seřadí / roztřídí dle klasifikace pomocí stejných klíčů a data, která mají stejný klíč se posílají do Reduceru. Reducer pak přidává své vlastní hodnoty, a až ty se zapisují do HDFS. Pro lepší ilustraci je princip vysvětlen na následujícím obrázku, kde se MapReduce využívá k počítání výskytu slov v dokumentu.



Obr. 10 - Princip funkce MapReduce v počítání slov, *vlastní tvorba dle [23]*

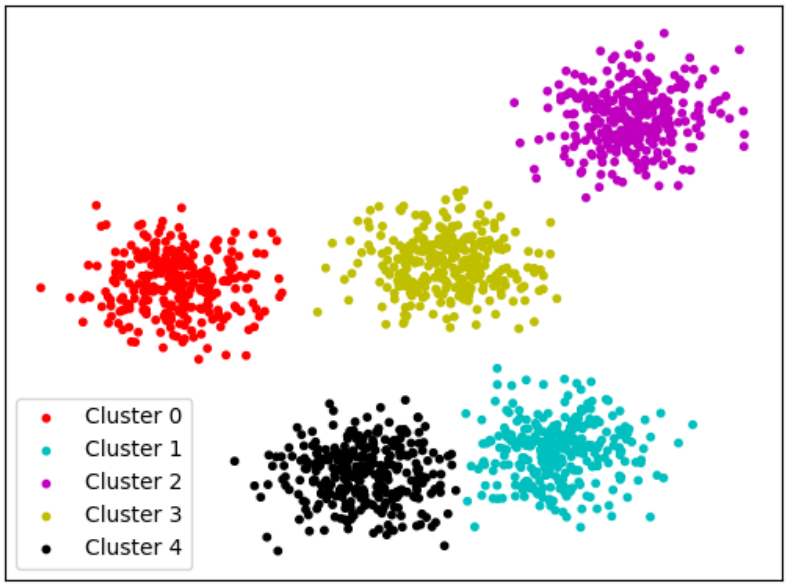
Jak je z obrázku patrné, tak mapování probíhá automaticky z HDFS, ale o tom, jak se data budou redukovat rozhoduje uživatel. Od toho se také odvíjí, že počet Mapperů se určuje sám v závislosti na struktuře datového souboru, ale počet Reducerů si uživatel volí sám dle požadovaných operací. Při mapování se využívá bufferu, ve kterém se průběh operace shromažďuje. Při překročení vyrovnávací paměti bufferu se data předají na dílčí disky v uzlu a následně se spojí do jednoho souboru, který je vstupem pro Reducer.

4.3 Strojové učení

Hlavním oborem data science je rozhodně strojové učení *(angl. machine learning)*. V základním principu jde o to, jak počítač naučit, aby se z dat, která zpracovává učil. Velkou roli v tomto oboru hrají i popisovaná big data. Ta jsou vhodná pro statistické analýzy a predikce. Stále se vytvářejí nové metody, jak přistupovat k distribuovanému prostředí, jako je Hadoop. Mnoho obecných principů využívá dvou základních charakteristik *(řízené a neřízené učení)*[81]. Řízené učení využívá tzv. tréninkových dat *(také známo jako označená data),* na základě, kterých probíhá klasifikace nových dat. Neřízené učení se však snaží o nalezení souvislostí, nebo struktur v neoznačených datech [24]. Algoritmy samotné využívající tento princip se tak snaží objevit informace, které by vedly k úspěšné klasifikaci prvku.

4.3.1 Neřízené strojové učení

Mezi první techniku neřízeného strojového učení patří ***shluková analýza*** *(angl. clustering)*. Funguje na principu seskupování dat, která sdílejí nějaké podobné vlastnosti. Je to určitý systém toho, jak data shlukovat. Většinou je model clusteringu N-dimenzionální graf na základě N vlastností daného objektu. Jednotlivě se pak data promítají na grafu dle zadaných vlastností. Tím pádem dochází, k již zmíněnému shlukování, kde je patrné, že se data dělí do určitých skupin.

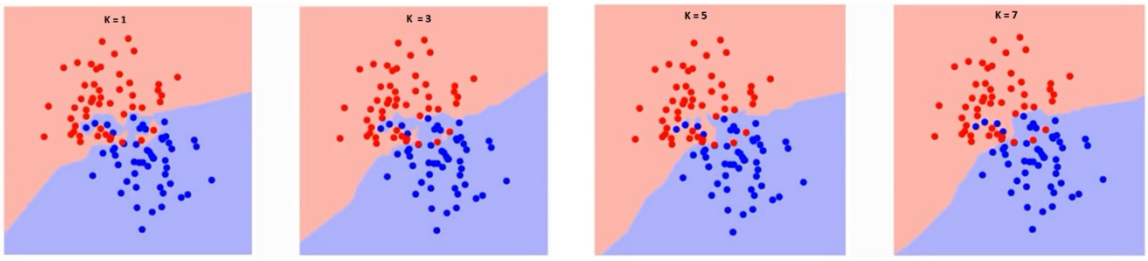


Obr. 11 - Ukázka dvourozměrné shlukové analýzy, *převzato z [25]*

[81] V angličtině jsou tyto pojmy označeny, jako supervised a unsupervised machine learning

Jak lze vidět na obr.11, tak se data shlukují do skupin, které mají velice podobné, nebo stejné vlastnosti. Pro využití s big data se samozřejmě bavíme o relacích s miliony řádků. Pokud je tedy třeba určit klasifikaci nového objektu, pak se provede analýza vzhledem k míře podobnosti s danou skupinou je přímo úměrná vzdálenosti mezi novým bodem a danou skupinou.

Další technikou je ***analýza odlehlých hodnot*** *(angl. outlier detection).* Touto metodou se snažíme klasifikovat objekt, který se nachází zcela mimo shluky, které byly clusteringem určeny. Samozřejmě je tu myšlenka, že daný bod je chybou a do datového souboru nepatří. Pomocí takových bodů lze tuto hranici určit, díky čemuž lze „očistit" data od chyb před další analýzou. Existují různé možnosti, jak určit, která data jsou chybová a která jsou pouze neobvyklá. Patří k nim *určení lokálního faktoru odlehlého bodu*, které porovnává lokální hustotu sousedství bodu s lokální hustotou sousedství. Touto metodou se graf rozdělí na clusterové regiony s podobnou hustotou a body, které mají hustotu nižší se považují za extrémy hranice mezi chybou a anomálií. Využívá se také techniky k-NN[82] algoritmů, kde se bere průměrná vzdálenost nejbližšího okolí „K" vůči novému prvku [26].



Obr. 12 - Správné určení „k" v k-NN algoritmech, *převzato z [26]*

Těchto principů se využívá například v bankovnictví. Banky v dnešní době využívají algoritmy, které detekují odlehlé body a soustředí se na možné zneužití platebních karet. Pokud například s kartou platíte několikrát měsíčně pouze potraviny a drobné zboží v hypermarketu, tak se vytváří cluster, do kterého se obdobné platby zařazují. Jakmile však např. náhle odcestujete do ciziny a koupíte si zcela odlišné zboží, které je několikanásobně dražší *(např. drahý šperk),* tak banka velmi pravděpodobně takovou kartu zablokuje kvůli potencionálnímu zneužití, neboť se vyskytl bod, který nespadá do žádné klasifikace a považuje se za chybný.

[82] K-Nearest Neighbor princip. Pro malou chybovost se doporučuje „K" volit jako 1, 3,5 nebo 7

Poslední technikou, která spadá výhradně do kategorie unsupervised je ***afinitní analýza*** *(angl. affinity analysis)*. Tento druh analýzy se snaží odhalovat vzájemné vztahy mezi akcemi, které jsou ve skupinách *(clusteru)* prováděny. Obecně lze takový postup aplikovat na jakýkoli model chování, kde se související data vyskytnou ve stejném kontextu. Ve své podstatě se tak snažíme identifikovat společné výskyty vzájemných položek a získat tak množinu těchto výskytů. Typickým příkladem využití této analýzy v praxi je rozložení potravin v supermarketech. Na základě affinity analýzy se sleduje, jaké položky si zákazník kupuje. Pokud si například s mlékem často zákazníci kupují i cereálie, pak se z důvodu vyššího prodeje vyplatí umístit cereálie do blízkosti mléka. Zároveň se tak dá sledovat, které potraviny jsou nejvíce žádané a na ty pak může supermarket zaměřit propagaci. Tato analýza využívá dvou základních principů. Buď se generují položky, přičemž se testují, jestli se často vyskytují v množinách, nebo se využívá hierarchického přístupu k objevování těch nejčastějších položek.

4.3.2 Řízené strojové učení

Model řízeného učení je založen na učení systému, jak nová data zpracovávat na základě dat již existujících. Prvním typickým řízeným algoritmem je ***klasifikace*** *(angl. classification)***.** Vše funguje v závislosti učení pomocí trénovacích dat[83]. Testovací model dostane tato trénovací data, na kterých se naučí rozlišovat skupiny dle vlastností a jeho úkolem je každá další data správně zařadit. Ke klasifikaci se využívá různých algoritmů:

- ***Naive Bayes[84] klasifikátor*** – Založený na Bayesově větě o podmíněné pravděpodobnosti. Tento algoritmus naivně *(z toho odvozen název Naive)* předpokládá, že je každá vlastnost datového objektu nezávislá.

- ***SVM[85]* -** Jedná se o řízený model učení, kde se analyzují data a rozpoznávají vzory. Cílem je nalézt dělící „nadrovinu“, která odděluje dvě odlišné skupiny z odlehlých vektorů skupin v training datech. SVM algoritmus tak vezme soubor vstupních dat a určuje jeden ze dvou možných výsledků. Je využíván také v regresní analýze.

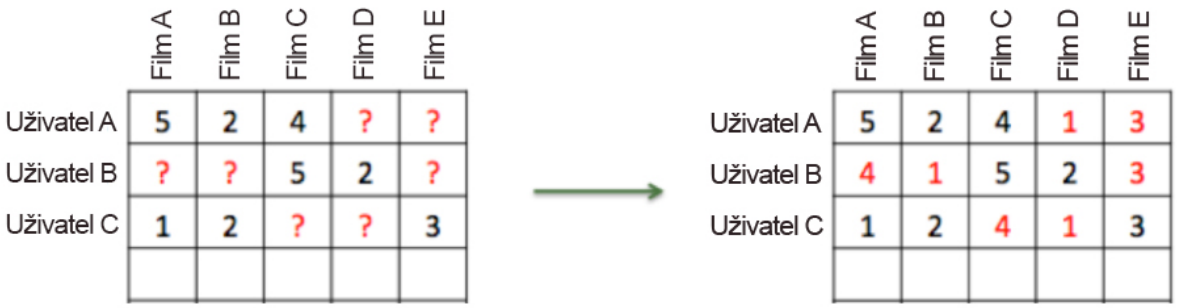
[83] Soubor dat, které jsou úspěšně klasifikovány a model se z nich může učit
[84] *Thomas Bayes* (1702–1761), statistik, formulace věty podmíněné pravděpodobnosti
[85] Support Vector Machines – Česky lze přeložit jako metoda podpůrných vektorů

- ***Logistická regrese*** – Využívá se pro predikci výsledku proměnné na základě jedné nebo více nezávisle proměnných. Tento druh algoritmu tak odhaduje hodnoty parametrů, které mohou výsledek ovlivnit.

Druhým modelem, který se učí z trénovacích dat je ***regresní analýza*** *(angl. regression analysis)*. Jde o modelaci a analýzu korelace mezi dvěma nebo více proměnnými. Hlavním cílem je výsledek predikovat většinou numerickou hodnotou. Výše šlo spíše o správnou klasifikaci do kategorie, kdežto zde se snažíme odhadnout budoucí hodnotu *(např. hodnoty akcií, nebo měny)*. Regresní analýzu tak můžeme použít k predikci jedné numerické proměnné na základě hodnot jiných proměnných. Tento druh analýzy využívá principů lineární regrese, metoda rozhodovacích stromů[86] nebo LMS[87]. V praxi se s regresní analýzou lze setkat u již zmíněných placených reklam, kde se využívá očekávané míry prokliku *(CTR)* a dle toho se reklamy uživatelům zobrazují.

4.3.3 Doporučování obsahu

Algoritmy na doporučování obsahu se využívají v dnešní době téměř všude. V principu jde o to, že se uživateli nabízejí možnosti toho, o co by mohl mít zájem. Jak bylo zmíněno na začátku kapitoly, tak typickým odvětvím, kde se s doporučovacími algoritmy setkat je Netflix. Propojuje totiž hodnocení filmů jiných uživatelů a u požadovaného uživatele jej dopočítá. Na základě zájmu a vypočítaného indexu hodnocení se následně konkrétní film / seriál doporučí. V současné době se však doporučení využívá opravdu v hojné míře i jinde, neboť se ukázalo, že jde o úspěšný obchodní model.



Obr. 13 - *Tabulka:* Dopočítávání hodnocení u filmů v Netflix

[86] Technika data miningu, kdy se tvoří rozhodovací model na základě vlastností dat
[87] Least Mean Squares – Metoda nejmenších čtverců

4.3.4 Umělá neuronová síť

Výpočetní systémy neuronových sítí *(angl. Artifical Neural Networks – zkr. ANN)* jsou používány v umělé inteligenci. Celá myšlenka principu vychází z biologického chování. Např. u rozpoznávání obrazu se systém učí z trénovacích dat, kde se popisuje výskyt nějakého elementu. Když je na obrázku člověk, pak se systém učí podle vzorů, kde se definují stavy *je člověk* a *není člověk*. Princip je však v tom, že se systém učí bez znalosti popisu člověka *(má vlasy, ruce apod.).* ANN využívá uzlů, které se nazývají umělé neurony. Jednotlivé neurony jsou mezi sebou pak spojené pomocí hran. Přes tyto hrany se předávají informace pomocí přenosových funkcí, a zároveň se tyto hrany hodnotí vahou, která se v průběhu učení přizpůsobuje. Ve spojení se strojovým učením lze zmínit *Deep Learning*, který se skládá z několika vrstev v ANN [27]. V principu jde o snahu modelovat neuronovou síť tak, jak by to dělal lidský mozek. Mezi nejvyužívanější metody deep learningu se řadí rozpoznávání řeči, nebo technika počítačového vidění.

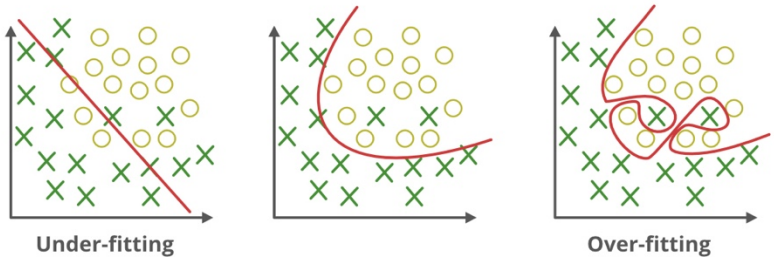
4.4 Hodnocení modelů

K tomu, aby byla popsána chyba regrese lze využít MAE[88] a MSE[89]. V obou případech se snažíme změřit průměrný rozdíl dat vypočítaných modelem a reálných dat [28].

$$MAE = \frac{\sum_{i=1}^{N} |predikovaná_i - reálná_i|}{N} \qquad MSE = \frac{\sum_{i=1}^{N} (predikovaná_i - reálná_i)^2}{N}$$

Vzorce pro MAE a MSE při hodnocení modelů

Při testování modelů se také může stát, že dochází k *under-fittingu* nebo *over-fittingu*. Znamená to, že pro popsání modelu používáme málo nebo naopak příliš mnoho predikčních proměnných a nelze nalézt optimální model k tomu, aby byla data správně popsána.



Obr. 14 - Rozpoznání optimálního modelu predikce, *převzato z [29]*

[88] Mean Absolute Error – Absolutní chyba dvou porovnávaných veličin
[89] Root Mean Squared Error – Známé jako střední kvadratická chyba

5 Hadoop® a data science

Nyní, po objasnění teoretických pojmů lze přejít k samotné implementaci získaných znalostí. V této kapitole jsou popsány algoritmy sloužící k získávání důležitých informací z dat s využitím Hadoopu.

5.1 Využití Apache Pig

V předchozí kapitole byl zmíněný přístup mapování a redukování komponent. Pro klasické programování je tento přístup velice složitý, proto je na místě zmínit komponentu Apache Pig, která umožňuje psaní programů, které fungují na stejném principu pomocí jazyka Pig Latin. Tento programovací jazyk je často připodobňován k SQL, neboť je jeho syntaxe[90] podobná. Pig je typický tím, že dokáže zpracovávat strukturovaná i nestrukturovaná data a jeho skripty mohou běžet na téměř jakémkoliv frameworku, takže jej lze aplikovat i mimo Hadoop. Samotný Pig běží nad vrstvou YARN.

5.1.1 Pig Latin

Jedná se o skriptovací jazyk pro Pig, který lze aplikovat pro zpracování big data. Příkazy lze provádět dvěma základními způsoby. Lze spouštět programy napřímo pomocí spustitelného Pig souboru, který je v tomto jazyce napsán nebo lze využívat tzv. Grunt shell *(interpret příkazů)*. Princip zpracování kódu lze shrnout tak, že je nejprve každý příkaz zpracováván interpretem Pig, který zároveň kontroluje validitu příkazu. Pokud je příkaz zadán správně, tak se přidá do fronty, která se má vykonat. Až po zavolání příkazu *DUMP*, nebo *STORE* se tato fronta skutečně spustí a příkazy se vykonají. Obdobně, jako v řadě programovacích jazyků zde musíme vytvářet proměnné, do kterých uložíme data, se kterými budeme dále pracovat.

```
proměnná = LOAD 'soubor.data' USING PigStorage(';') AS
(vlastnost1, vlastnost2, vlastnost3);

DUMP proměnná;
```

Obr. 15 - *Kód*: definice proměnné s rozdělením podle vlastností

[90] Soubor pravidel, který definuje strukturu jazyka

Pig Latin má datové typy, jako ostatní jazyky *(int, double, chararray atd.)*. Má však také speciální komplexní datové typy. Prvním z nich je ***Tuple***[91], který obsahuje vnitřní strukturu dat. Dalším je tzv. ***Bag***[92], který shromažďuje neuspořádané kolekce tuples. Posledním je pak ***Map***[93], který shromažďuje data pomocí *klíč / hodnota*, kde klíč musí být samozřejmě jedinečný [30]. Pig Latin využívá také operátory, jako jsou *GROUP*, *FILTER*, *LIMIT* nebo např. *FOREACH*. Pro komerční využití je nejčastějším operátorem např. *GROUP*, díky čemuž se lze v big data orientovat pomocí seskupení na základě určitých vlastností.

```
proměnná = LOAD 'soubor.data' USING PigStorage(';') AS
(vlastnost1:chararray, vlastnost2:int, vlastnost3:double);

proměnnáPodleVlastnosti = GROUP proměnná BY vlastnost2;
```

Obr. 16 - *Kód*: Seskupení dat dle stejné vlastnosti do tuples

Pro procházení dat se pak dají využít výše zmiňované operátory. Pomocí *FILTER* lze upřesnit vlastnosti na požadovaný rozsah. Využití je např. při číselných údajích, kdy chceme vyfiltrovat jen proměnné, které dané kritérium splňují. Mějme příklad, kdy je třeba analyzovat, kteří zaměstnanci nadnárodní korporace si zaslouží benefit, neboť sjednali více, než 100 realizací a zároveň mají plat menší, než 40 000 Kč. Představme si tedy, že v naší načtené proměnné máme strukturu, kdy se do vlastnosti 2 ukládá počet realizací a do vlastnosti 3 plat.

```
benefitZamestnanec = FILTER proměnná BY vlastnost2 > 100 AND
vlastnost3 < 40000;
```

Obr. 17 - *Kód*: Využití filtrace dat na zmíněném příkladu

FOREACH nám pak umožní provádět operace procházením celé relace. Pokud chceme např. nalézt, který zaměstnanec uskutečnil nejvíce realizací provedeme následující příkaz:

```
maxRealizaceZamestnanec = FOREACH proměnná GENERATE group,
MAX(proměnná.vlastnost2);
```

Obr. 18 - *Kód*: Využití procházení celé relace

[91] Uloženo ve tvaru *(položka1, položka2, položka3)*
[92] Uloženo ve tvaru *{(položka1, položka2, položka3), (jináPoložka1, jináPoložka2, jináPoložka3)}*
[93] Uloženo ve tvaru *[nazev1#položka1, nazev2#položka2, nazev3#položka3],* kde název reprezentuje klíč

5.2 Využití Pythonu

Python je vysokoúrovňový programovací jazyk, který je tzv. *human-readable*[94]. Je interpretovaný, interaktivní a využívá principů OOP. Využívá tedy tříd, modulů, výjimek, typování atd. Díky jeho rozhraní lze přistupovat k systémovým voláním a knihovnám, a je přenosný. Znamená to tedy, že skript napsaný v Pythonu lze spustit na mnoha variantách Unix systémů, Mac a Windows. Obdobně, jako Pig Latin využívá speciálních datových typů, jako je ***Tuple***. Tuple je jako u Pig Latin posloupnost hodnot, ale nelze ji po inicializaci změnit. Dále pak máme k dispozici ***List***[95], kde lze aplikovat funkce *insert*, *remove*, *sort*, *append* atd. Speciální funkcí je pak *range,* díky které lze definovat rozpětí generovaných hodnot. Speciálním datovým typem je také ***Dictionary***[96]. Důležitým faktorem však je, že datové typy v Pythonu nemusíme definovat, neboť je interpret sám rozpozná při spuštění programu [31]. Znamená to tedy, že pokud definujeme proměnnou *a = 10,* tak je rozpoznána jako int, oproti tomu *b = 3.14* je rozpoznáno jako float. Pomocí tohoto jazyka lze využívat i příkazů *if / else, for* atd. Poslední velice důležitou vlastností jsou lambda funkce, díky nimž lze napsat malé, často využívané podpůrné funkce. Záleží zkrátka na případu, jak má daný program fungovat. Programy, které jsou napsány v jazyce Python mají koncovku **.py*.

```
proměnná = "Diplomová práce na téma Hadoop a Data Science"
slova = proměnná.split()
délkaSlova = map(lambda slovo: len(slovo), slova)
print délkaSlova
>> [9, 5, 2, 4, 6, 1, 4, 7]
```

Obr. 19 - *Kód*: Využití funkce lambda v Python programu

Jak lze vidět na obrázku výše, tak pomocí funkce lambda lze efektivněji nahradit *for* cyklus, kterým by daný příklad šel také udělat. Kód na obrázku načte slovní řetězec do proměnné, následně jej ořeže dle mezer mezi slovy a pomocí lambda funkce vytvoří kolekci s počtem písmen, které se nachází v každém slově řetězce. Pokud bychom chtěli takový kód využívat v programu často, lze jej samozřejmě provádět jako funkci, která se v Pythonu definuje jako ***def*** *nazevFunkce (parametry)*.

[94] Programovací jazyk, který je snadno čitelný pro člověka
[95] Ukládání položek do sbírek. Funguje stejně, jako v Java
[96] Uloženo ve tvaru {'klíč1': 'hodnota1', 'klíč2': 'hodnota2'}

5.3 Práce s daty

Ke zpracování big data se využívá výše zmíněných principů. Je však klíčové využívat také již existujících balíčků, které obsahují nástroje pro komplexní analýzu dat. Výhodou využívání Pythonu pro data science je především to, že tento jazyk má širokou sbírku podpůrných knihoven, které umožňují využívat znalostí z matematiky, statistiky nebo třeba machine learningu. Celému ekosystému, který je na takovou analýzu zaměřen se říká SciPy[97]. Pro práci s knihovnami je nutné tyto knihovny importovat do systému.

```
import numpy as numpyBalicek
from numpy import *
```

Obr. 20 - *Kód*: Import balíčků pro matematické operace

5.3.1 NumPy

Numerical Python slouží především k provádění matematických operací. Jde o balíček, který obsahuje základní komponenty pro vědecké výpočty. Hlavním specifikem je, že pomocí něj lze vytvářet N-dimenzionální kontejner, zvaný ***ndarray***. Celkově poskytuje mnoho užitečných funkcí pro práci s polem a maticemi, lineární algebrou, ale také nástroje pro finanční analýzu nebo např. Fourierovy[98] transformace. Nad *ndarray* se dají provádět operace, jako s klasickým polem *(sčítání, odčítání, součin atd.)*.

```
ndarray1 = numpy.array([[1, 2, 3],[4, 5, 6]])
ndarray1 = numpy.array([[2, 4, 6],[8, 3, 5]])
numpyBalicek.add(ndarray1, ndarray2)


>>  array ([3, 6, 9],
           [12, 8, 11])
```

Obr. 21 - *Kód*: Operace součtu s ndarray

Pro využití dalších funkcí slouží balíčky *linalg*, *fft*, nebo například *matlib*.

[97] Scientific Python – Open-source ekosystém pro využívání matematických operací
[98] *Joseph Fourier* (1768–1830), transformace slouží pro převod signálu z časové do frekvenční oblasti

5.3.2 Pandas

Stejně tak, jako analýza dat je důležitá i jejich modelace. Pandas[99] je open-source knihovna pro modelování dat. Obsahuje datové struktury a funkce, které zjednodušují práci s daty v jazyce Python. Tato knihovna obsahuje datové struktury *Series* a *DataFrame,* funkce pro čtení a zápis komerčních tabulkových souborů *(xls, csv)* a umožňuje seskupovat datové soubory na základě určitých vlastností.

```
popisky = ['Pardubice', 'Praha', 'Hradec K.']
obsah = {'sloupec1':Series((range(3)), index = popisky),
'sloupec2':Series((range(3,6)), index = popisky)}
dataFrame = DataFrame(obsah)
>>          sloupec1    sloupec2
Pardubice   0           3
Praha       1           4
Hradec K.   2           5
```

Obr. 22 - *Kód*: Vytvoření DataFrame s využitím range

DataFrame je nejvíce využívaný prvek v pandas. Lze ho připodobnit také k tabulkovému editoru ve kterém lze shromažďovat data. Dalším důležitým prvkem je *TimeSeries*, díky kterému lze zaznamenávat časové údaje [32]. Balíček *datetime* obsahuje sadu nástrojů pro ukládání času, data, časových razítek nebo časových zón. To je velice efektivní při práci s big data, kde je modelem analyzovat data napříč různými státy.

5.3.3 Matplotlib

Poslední důležitou knihovnou ve SciPy je *matplotlib*. Jak název napovídá, tato knihovna obsahuje nástroje k vizualizaci grafů, schémat apod. V praxi však povětšinou stačí importovat pouze 2D zobrazování pro analýzu závislostí, k čemuž slouží přímo třída *matlotlib.pyplot.* Tato knihovna obsahuje také efektivní nástroje pro vizualizaci clusteringu dat, 3D zobrazování komplexnějších dat apod. Více lze nalézt v oficiální dokumentaci na internetových stránkách *matplotlib.org*.

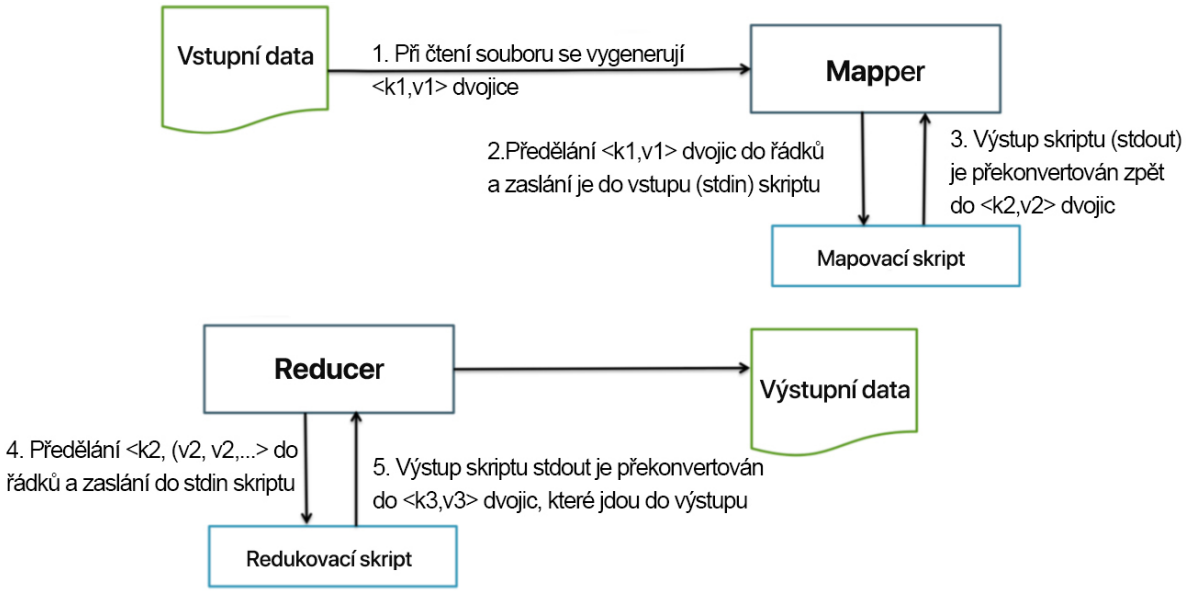
[99] Název pochází ze statistického pojmu Panel Data, který se užívá v ekonomii. Jde o popsání vícerozměrných dat, která jsou měřena v různých časových intervalech

5.4 Využití znalostí Pythonu v Hadoopu

Pro využití data science řešení pomocí jazyku Python v Hadoopu lze užívat obecně tři postupy.

5.4.1 Hadoop streaming

Prvním řešením je Hadoop streaming. Toto řešení je implementováno přímo v Hadoopu. Díky tomuto mechanismu lze spouštět MapReduce pomocí skriptů napsaných v jakémkoliv jazyce *(v tomto případě zmiňovaný Python)*. V podstatě jde o to, že se na straně mapování a redukování předá kód ze skriptu, který je přeložen[100] a vykoná se. Využívá se tak Hadoop Streaming API, které čte vstup na základě vzorců *key/value* [33]. Při zpracovávání je však počet *key/value* hodnot omezen a musí se zpracovávat přes standardizovaný vstup – *stdin* a výstup – *stdout*.



Obr. 23 - Princip funkce Hadoop streaming API, *vlastní tvorba dle [33]*

Tím pádem dokážeme přenést proces MapReduce z teoretické roviny do praktické, neboť jsme si zmínili, že pomocí Pythonu lze psát skripty, které budou provádět úkoly k analýze dat a pomocí Hadoop Streamingu je lze využít přímo na datech v Hadoopu. Tento proces je uložen ve spustitelném souboru *hadoop-streaming.jar* v domovském adresáři Hadoopu a lze jej spustit příkazem[101] ***`yarn jar $HADOOP_HOME/lib/hadoop-streaming.jar`***

[100] Záleží na jazyku skriptu, proto musí být na každém uzlu nainstalován Python
[101] Příkaz má prefix *yarn*, neboť běží na YARN vrstvě

5.4.2 Pig Streaming

Oproti Hadoop Streamingu nabízí Pig streaming pár vylepšení. Zde se Pig sám rozhoduje o tom, jaké hodnoty převést do *key/value* vzorců, jaké algoritmy spustit na mapovací části a jaké na redukovací části. V Hadoop Streamingu o tom, kde a jaké algoritmy spouštět rozhoduje uživatel. Důležitou funkcí je ***STREAM***, díky čemuž lze poslat data skrze skript do programu. Zároveň je však nutné rozlišit, jaká data pomocí *STREAM* příkazu zasíláme. Pokud se jedná o neřazená data, není nikde psáno, jak budou data před streamingem seřazena. Pokud jsou data uspořádána ve skupinách dle klíčů, pak se budou tato data streamovat souvisle podle daných klíčů. A poslední možností je, že jsou data seskupena dle klíčů a zároveň seřazena. To je pak streaming dat prováděn tak, že se streamují seskupená data souvisle vzhledem k seřazení ve skupině dle sekundárního klíče.

5.4.3 User Defined Functions

Jak z názvu vyplývá, jde o uživatelem definované funkce. Toto rozšíření má na starost komponenta Pig. UDFs jsou realizovány pomocí tzv. Jythonu *(slovní kombinace Java a Python)*, což je implementace Pythonu, která běží na JVM, neboť UDFs psané pouze v Pythonu na Hadoopu nefungují. Obecně lze využívat několika základních UDFs, které jsou open-source a volně použitelné. V praxi se velice využívá níže zmíněných.

- ***DataFu*** – Kolekce Pig UDFs k analýze dat na Hadoopu. Tuto kolekci vytvořila společnost LinkedIn.
- ***Piggy Bank*** – Jak název napovídá, tak se jedná o kolekci často využívaných funkcí psaná pro Pig.
- ***ElephantBird*** – Knihovna od Twitteru, která poskytuje standardizované funkce pro přehlednost, jako *OutputFormats*, *Writables* a další.

Pro psaní UDFs v jazyce Python je nezbytně nutné anotovat funkci pomocí *outputSchema*, kvůli tomu, aby Pig rozpoznal datový typ návratové hodnoty. Níže je ukázáno, jak takovou UDF napsat. Mějme příklad, kdy chceme zjistit, jestli je v instagramových příspěvcích zmíněno *UHK* nebo *FIM*.

```
IG_POST = set((\'uhk','fim'\))
@outputSchema('prohledavane:chararray, jeZminka:int')
def jeZminka(celyText):
    lowText = set(celyText.lower().split())
    for prohledavane in lowText :
        if prohledavane in IG_POST :
            return prohledavane, 1
    return '', 0
```

Obr. 24 - *Kód*: Využití UDF k napsání jednoduchého skriptu pro analýzu

Na příkladu výše vidíme, že musíme nejprve definovat množinu výrazů, pro které odpovídá náš program. Dále je nutné definovat výstupní formát *(v tomto případě příspěvek, ve kterém je zmínka #uhk nebo #fim a pomocnou indikaci pomocí čísla).* Pomocí příkazu def lze vytvořit funkci *jeZminka*, která má za úkol vstupní text příspěvků osekat a rozřezat kvůli snazšímu vyhledávání řetězců. Pokud se prohledávaný řetězec shoduje s naší množinou, pak se řetězec vrátí s příslušným označením pomocí *1*. Při dalším zpracování tak lze lehce odfiltrovat, kde byla zmínka naší školy a kde nikoliv.

Aby bylo možné s UDF dále pracovat pomocí Pig, je nutné jej zaregistrovat přes Java spustitelný soubor. Např. pro registraci kolekci funkcí Pig `register 'datafu-1.X.X.jar'`. Po této registraci lze vytvořit[102] alias, díky kterému k danému UDF budeme přistupovat, pak jen zbývá probrat data pomocí *STREAM* přes vytvořený skript. Aby bylo možné využívat skripty napříč uzly v Hadoopu, je také nutné je do všech uzlů poslat pomocí příkazu *SHIP* [34].

```
DEFINE mujSkript `mujskript.py` SHIP ('mujSkript.py')
testData = LOAD 'testovanaData' USING PigStorage(';')
    AS (text: chararray, pocet: int);
vystupPresSkript = STREAM testData THROUGH mujSkript;
DUMP vystupPresSkript;
```

Obr. 25 - *Kód*: Distribuce skriptu na uzly a následný Pig Streaming

[102] Skriptu přiřadíme alias pomocí příkazu *DEFINE*

5.5 Využití machine learning principů

Pro využití strojového učení v Hadoopu existuje mnoho nástrojů. Mezi nejdůležitější dva se řadí *Mahout* a *SciKit-Learn*. Apache Mahout je zaměřen na rychlou tvorbu dále rozšiřitelných programů pro machine learning a využívá mnohých algoritmů Spark a Scala, díky nimž dokáže pracovat na bázi MapReduce. SciKit-Learn poskytuje spíše nástroje na analýzu dat a data mining, přičemž se drží využití zmíněných NumPy, SciPy apod.

5.5.1 Systém doporučování obsahu

Vždy je nutné k problematice přistupovat tak, že je třeba zjistit pár faktorů před tím, než začneme tvořit systém, který bude uživateli doporučovat obsah na základě určitých atributů. Je nutné analyzovat data tak, aby bylo evidentní, co vše lze použít a tím se také řídit. Prvotním předpokladem je vytvoření tzv. matice preferencí. Tu lze realizovat pomocí zmíněných *ndarray*, nebo *DataFrame,* přičemž se využívá kolaborací *(CF[103]).* Cílem kooperativního filtrování je úspěšné doporučení na základě vybraných vlastností.

Doporučení na základě uživatele je zaměřeno na hledání podobných uživatelů ve vytvořené matici preferencí. Podobný uživatel má tendenci ke stejnému chování, což v systému doporučování lze sledovat např. u hodnocení filmů. Pokud nalezneme podobného uživatele, tak s vysokou pravděpodobností bude hodnotit filmy, které se mu líbí stejně, nebo podobně jako první uživatel. Z toho také vyplývá, že lze odhadnout chybějící hodnoty. Pokud první uživatel něco neohodnotil, pak lze předpokládat[104], že by se jeho hodnocení blížilo podobnému uživateli. Pro výpočet si lze představit uživatele, jako vektory a hodnocení jako souřadnice. Pokud porovnáme dva uživatele navzájem a představíme si jejich hodnocení jako vektory, pak lze dopočítat podobnost vektorů pomocí *cosSim[105]* [35].

$$cosSim(X,Y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Vzorec pro výpočet kosinové podobnosti

[103] Collaborative Filtering – Kooperativní filtrování, kde se sleduje provázanost prvků k dalšímu postupu
[104] Tato idea je založena na homogenním chování lidí. Například, pokud uživatel nemá rád sci-fi, pak se předpokládá, že své názory nezmění kvůli jednomu filmu
[105] Cosine Similarity – Určení faktoru podobnosti dvou veličin X a Y

Pro lepší ilustraci si daný postup vysvětleme na příkladu. *Mějme dva uživatele Jaromíra a Patrika. Oba hodnotili několik filmů nezávisle na sobě a potřebujeme zjistit, jestli by Patrik hodnotil seriál Walking Dead podobně, jako Jaromír.*

	Griffinovi	Walking Dead	Hra o Trůny	Chernobyl	Stranger Things
Jaromír	4	4	5	4	4
Patrik	3	Chceme zjistit	5	3	5

Obr. 26 - *Tabulka*: Data pro doporučení dle uživatele

Faktor podobnosti pak vypočítáme následujícím vztahem, kde dosadíme hodnoty z tabulky, která je reprezentována pomocí ndarray.

$$cosSim(J,P) = \frac{\sum_i J_i P_i}{\sqrt{\sum_i J_i^2}\sqrt{\sum_i P_i^2}} = \frac{(4*3+5*5+4*3+4*5)}{\sqrt{(4*4+5*5+4*4+4*4)}\sqrt{(3*3+5*5+3*3+5*5)}} = \mathbf{0.979}$$

Pomocí výpočtu jsme zjistili, že podobnost hodnocení seriálů uživatelů Jaromír a Patrik jsou si velice podobná, neboť se hodnota blíží „1“[106]. Díky vypočítané hodnotě lze již snadno dopočítat chybějící hodnocení u uživatele Patrika, kde je výpočet dán jako hodnocení podobného uživatele *(v tomto případě Jaromíra)* * hodnota podobnosti. Lze tak dopočítat, že hodnota, kterou by zadal Patrik je $hodPatrik = 4 * 0.979 = 3.916 \approx \mathbf{4}$ [107]

Doporučení na základě položek je obdobné. Zde však nejde o to, najít podobného uživatele, ale nalézt podobný produkt *(v tomto případě film)* dle recenzí ostatních uživatelů. Jde tedy o to, že namísto uživatelů využijeme ve vzorci hodnoty u konkrétního sloupce. Zde je tedy cílem analyzovat data tak, aby bylo možné určit hodnocení, které u daného uživatele chybí. Pro ukázku a lepší pochopení je na další straně opět sestavena tabulka, která obsahuje vzorová data. Zde je důležitá velikost sloupce, místo řádku, jako tomu bylo u předchozího příkladu. Musíme tedy uvažovat více uživatelů, kteří hodnotili nějaký film. V následujícím příkladu nás zajímá, zdali jsou seriály *Walking Dead* a *Stranger Things* podobné, a jak bude na základě podobnosti fungovat predikce.

[106] Hodnota 1 je ve velkém množství dat teoretickým pojmem. Znamenalo by to, že je v systému výskyt dvou totožně hodnotících osob, což se, když uvážíme množství možných kombinací téměř nemůže stát
[107] Při takto malém množství dat je však možno hodnoty 1 dosáhnout použitím extrémů *(např., že Jaromír vše hodnotí 5 a Patrik 1)*

	Griffinovi	Walking Dead	Hra o Trůny	Chernobyl	Stranger Things
Jaromír	4	4	5	/	4
Patrik	3	/	5	3	5
Tereza	3	3	/	4	4
Michal	4	5	4	/	3
Aneta	2	/	5	2	/

Obr. 27 - *Tabulka*: Data pro doporučení dle položek

$$cosSim(WD, ST) = \frac{\sum_j WD_j ST_j}{\sqrt{\sum_j WD_j^2}\sqrt{\sum_j ST_j^2}} = \frac{(4*4+3*4+5*3)}{\sqrt{(4*4+3*3+5*5)}\sqrt{(4*4+4*4+3*3)}} = \mathbf{0.950}$$

Zjistili jsme, že koeficient podobnosti seriálu je 0.95. Díky tomu lze určit predikci pro to, zdali se seriál *Walking Dead* bude líbit Patrikovi, jako $hodWD_{Patrik} = 5 * 0.95 = 4.75 \approx \mathbf{5}$. V praxi je samozřejmě nutné zjistit, který model bude vyhovovat nejvíce a až poté, co nalezneme nejvíce podobnostní shodu predikci aplikovat. Stejných, ba dokonce lepších výsledků lze dosáhnout pomocí *Pearsonova*[108] koeficientu. Je však nutné si stanovit, jaké jsou priority v analýze dat. Tento výše zmiňovaný postup je efektivní, přesný, ale především rychlý. Použití *Pearsonova* koeficientu by bylo na místě užít, kdyby byl kladen důraz na přesnost celkové analýzy. Často se také v praxi stává, že je třeba nad modelem přemýšlet jinak. Např. je třeba zaznamenávat, jestli byla na webu provedena konverze, nebo jestli se stal určitý požadovaný cíl, proč se vlastně samotná analýza dělá. Pro velké množství analýz stačí zaznamenávat hodnoty typu *true/false.* V takovém případě se pomocí ndarray vytváří tzv. binární preferenční matice, která může používat na analýzu jiných metod, jako je *Jaccardův*[109] koeficient podobnosti.

$$Jaccard(x, y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{\sum_i X_i \wedge Y_i}{\sum_i X_i \vee Y_i}$$

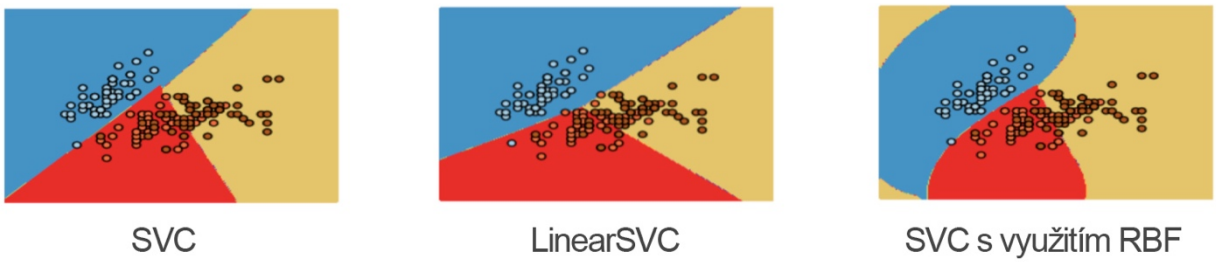
Vzorec pro Jaccardův koeficient podobnosti

[108] Tento koeficient měří sílu lineární závislosti mezi dvěma proměnnými
[109] *Paul Jaccard* (1868–1944) – Koeficient podobnosti publikoval v roce 1901

5.5.2 SVM metody

Jak bylo zmíněno v teoretické části, tyto metody podpůrných vektorů slouží pro klasifikaci dat. V celku jde především o to, že na základě známých bodů a jejich klasifikace lze určovat nové body dle znalostí o skupinách. Algoritmus tedy funguje tak, že se nejprve začne učením, jak jsou training data rozdělena do kategorií. Následně se vytvoří model pro popsání hranice mezi těmito skupinami. Zde se klade důraz na to, aby byla hranice s co největší stejnou mezerou mezi skupinami. Nově příchozí data se tak budou klasifikovat na základě toho, na jaké straně hranice se nacházejí.

Pro úspěšnou klasifikaci poskytuje SciKit-Learn tři základní knihovny, díky kterým lze úspěšně popsat, do které skupiny daný bod spadá. Jsou to **`sklearn.svm.SVC`** a **`sklearn.svm.NuSVC`**, které využívají knihovny **`libsvm`**. NuSVC navíc lze rozšířit o parametr, který určí počet podpůrných vektorů. Této klasifikaci se také říká SVC *(Support Vector Classification)*. Oproti tomu **`sklearn.svm.LinearSVC`** využívá lineární knihovnu **`liblinear`**, tudíž je přesnější, než lineární metoda SVC. U SVC se také často kvůli přesnější klasifikaci volí metoda RBF[110].



Obr. 28 - Využití SciKit knihoven k úspěšné klasifikaci, *převzato z [36]*

5.5.3 Určování klasifikací

K hledání pravděpodobnosti závislosti se využívá zmíněné Bayesovy věty, která předpokládá, že každá zkoumaná dvojice proměnných je vzájemně nezávislá. Níže zmíněné třídy nacházejí své uplatnění především při kategorizaci, např. určení, zdali je e-mail *SPAM*. SciKit-Learn poskytuje ke klasifikaci třídy **`sklearn.naive_Bayes.GaussianNB`**, **`sklearn.naive_Bayes.MultinomiaNB`** a **`sklearn.naive_Bayes.BernoulliNB`**.

[110] Radial Basis Function – Ke klasifikaci tato metoda využívá výpočet vzdálenosti bodu od počátku

5.5.4 Hledání nejbližšího souseda

Pro regresi a klasifikaci se také využívá algoritmů, které hledají tzv. nejbližšího souseda. Princip se zakládá na tom, že je zde snaha nalézt co nejvíce trénovacích bodů, které jsou nejblíže k nově přidanému a z nich určit dále jeho klasifikaci. Pro hledání se využívá dvou technik. Buď se pracuje s k-nejbližším sousedním prvkem, kde *k* je uživatelem zvolená konstanta *(viz 4.3.1),* nebo se bere v potaz poloměr od nově vzniklého bodu, kde se má klasifikace vypočítat. Velikou výhodou těchto algoritmů je, že dokáží pracovat jako *supervised*, ale i *unsupervised*. SciKit knihovna obsahuje několik tříd, které hledání nejbližšího souseda nabízejí. Pro unsupervised hledání sousedů se využívá třídy **`sklearn.neighbors.NearestNeighbors`**. Pro algoritmy založené na hledání k-nejbližšího souseda se využívá **`sklearn.neighbors.KNeighborsClassifier`** pro klasifikaci, a **`sklearn.neighbors.KNeighborsRegressor`** pro regresní analýzu. A pro hledání v poloměru slouží třída **`sklearn.neighbors.RadiusNeighborsClassifier.`**

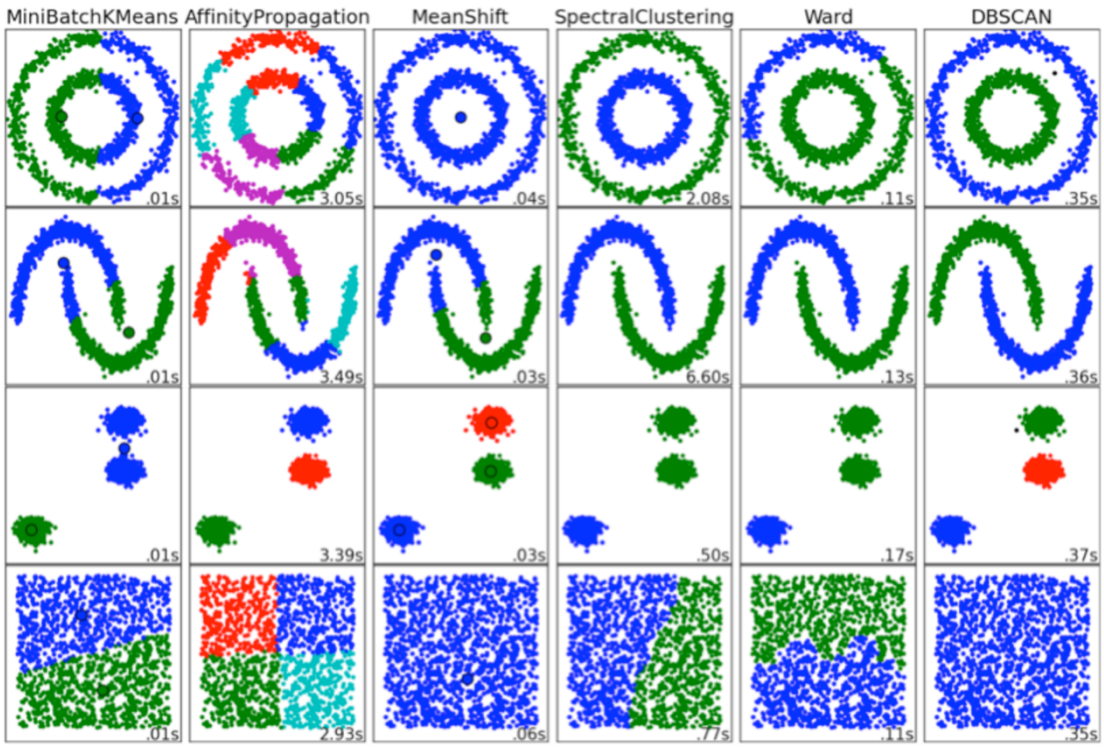
5.5.5 Zvolení optimálního přístupu

To, jak co nejefektivněji zvolit správný přístup k analýze dat není nikde dáno. Vždy je třeba na danou problematiku hledět abstraktně a volit algoritmy tak, aby byly cíle analýzy co nejvíce splněny. Základními přístupy jsou již zmiňovaná shlukování *(clustering),* regrese, ale také využití rozhodovacích stromů[111]. Níže jsou shrnuty výhody a nevýhody daných přístupů.

- ***Shlukování*** – Snadno rozpozná skupiny a lze jí dále využívat pro regresní analýzu. Je však citlivá na odlehlá data, a tak je třeba data normalizovat.
- ***Regrese*** – Dokáže spolehlivě určit hranici, kterou lze třídit data i ve velkém množství. Nevýhodou však je, že regrese je dělaná na numerická data a je také citlivá na odlehlé hodnoty.
- ***Rozhodovací stromy*** – Jsou snadno pochopitelné i pro obyčejné uživatele díky jednorázovým rozhodnutím ANO/NE. Mohou být však rozsáhle složité díky mnohým proměnným a zároveň nedokáží určit nová data.

[111] Často využívaná metoda při rozhodování *(rozhoduje se podle grafu, který zachycuje možné varianty do stromové struktury)*, ale lze jí využívat i k analýze

Pro **regresi** se využívá tříd **`sklearn.svm.SVR`** a **`sklearn.svm.NuSVR`**. V podstatě se jedná o stejný přístup, jako u SVM metod s tím rozdílem, že zde se pracuje přímo s vektory. Ke **shlukování** se využívá několika metod *(kde každá má své zastoupení v knihovně SciKit)*. Pro *KMeans*[112] se využívá **`sklearn.cluster.KMeans`** a funguje tak, že se snaží minimalizovat rozptyly mezi skupinami. Tento algoritmus je nejvíce využívaným, kvůli jeho jednoduchosti a rychlosti oproti dále zmíněným. Pokud jsou data strukturována tak, že počet skupin bude vyšší, je třeba použít **`sklearn.cluster.AffinityPropagation`**. Tento algoritmus sám určí celkový počet skupin *(shluků)* a vychází z toho, že data v něm jsou tvořena maticí podobnosti *(viz výše)*. Dalšími třídami, které se využívají jsou **`sklearn.cluster.DBSCAN`**, **`sklearn.cluster.SpectralClustering`**, nebo například **`sklearn.cluster.MeanShift`** [37]. Jak bylo zmíněno, KMeans je však nejvíce efektivní kvůli úspoře času.



Obr. 29 - Srovnání různých tříd pro shlukování, *převzato z [37]*

Technika **rozhodovacích stromů** je metoda *supervised* učení pro klasifikaci a regresi. Díky tvorbě modelu, který je strukturován pomocí hierarchicky nadřazených otázek typu *ANO/NE* lze předpovídat, jak se proměnná klasifikuje. V celkovém principu se zkrátka proměnná klasifikuje odpovídáním na tyto otázky.

[112] Nutno zadat parametr počtu skupin, které zvolí sám datový analytik

5.6 Porozumění lidskému jazyku

S přibývajícími nároky na analýzu dat se také stupňuje snaha o naučení programů porozumět přirozenému lidskému jazyku. Díky technikám machine learningu je možné se k této myšlence přiblížit. Obecně používaný přístup se nazývá NLP *(Natural Language Processing)*. Historicky se takových principů pokoušelo dosáhnout ručním programováním rozsáhlých datových sad, které se pak algoritmus snažil spárovat s porovnávanými znaky. Je evidentní, že machine learning tomuto postupu dost ulehčil, neboť je možné využívat výše zmíněných principů k automatizovanému učení. NLP lze využívat pomocí jazyků Java *(knihovny OpenNLP, MALLET)* nebo Python *(NLTK).* Vzhledem k vlastnostem big data se nabízí využívání těchto algoritmů v praxi. Big data jsou z masivní části tvořena textem, a je na místě přemýšlet nad automatizací zpracovávání. Textové záznamy big data se tak musí dávat do logických celků, které jsou následně rozdělovány a dochází ke zpracovávání pomocí NLP, aby byl pochopen jejich význam. Rozšíření těchto algoritmů je v dnešní době především kvůli sociálním sítím.

5.6.1 Rozpoznávání znaků

Nerozšířenější formou rozpoznávání znaků je OCR *(Optical Character Recognition)*. Tato metoda je hojně využívána v tiskárnách se scannerem. Základním principem je to, že se algoritmus snaží převést text na papíře[113] do digitální podoby. Před samotným rozpoznáváním znaků je však nutným předpokladem skenovaný text maximálně zaostřit a redukovat šum. V NLP se využívá k rozpoznávání znaků v řetězcích a pro základní mechanizaci umělé inteligence. Těchto principů lze využívat pomocí enginu *Tesseract*.

5.6.2 Tvoření logických celků

Při chápání znaků je také na místě zmínit, že by algoritmy měly být schopné určit i logickou strukturu textu, a to členění do vět. Většina algoritmů využívá tečky k rozeznání konce věty, ale existují i výjimky, kde je tečka uprostřed věty, například u oslovení *(pí. Nováková).* Tyto algoritmy s tím však dokáží počítat a jsou i tak velice úspěšné. Pro využití segmentace věty jsou zde k dispozici knihovny *OpenNLP, NLTK*.

[113] Text musí být rozpoznáván z digitální podoby. Ani v dnešní době neexistují spolehlivé algoritmy, které by dokázaly rozpoznat psací písmo

5.6.3 Rozpoznávání struktury vět

S výše zmíněnou segmentací rovněž souvisí princip rozpoznávání věty podle vlastností slov. Obecně se tyto algoritmy označují jako POS[114]. V praxi jde o to, rozeznat, jaké slovo ve větě je podstatné jméno, sloveso atd. Poté, co proběhne tato analýza je možné zvýšit přesnost segmentace. Ukažme si to na níže přiloženém příkladu.

```
mojeVeta = "Já píšu diplomovou práci."
rozdelenaVeta = ['Já', 'píšu', 'diplomovou', 'práci']
>> POS = [('Já', 'PRO'), ('píšu', 'VRB'), ('diplomovou',
'ADJ'), ('práci', 'NN')]
```

Obr. 30 - *Kód*: Rozdělení věty dle principu POS

Na příkladu si lze všimnout, že po rozdělení se páruje hodnota řetězce s hodnotou vlastnosti *(PRO – zájmeno, VRB – sloveso, ADJ – přídavné jméno, NN – podstatné jméno)*. K implementaci takového principu slouží *NLTK*.

5.6.4 Klasifikace textu

Vzhledem k povaze machine learningu je vhodné přenést praktické zkušenosti i do NLP. Klasifikace textu se dá využívat pro rozpoznávání spamu, ale také pro rozpoznávání kontextu, jako je např. význam přídavných jmen. Pro využití v praxi se užívá klasifikátoru **`NaiveBayesClassifier`** dle Bayesovy věty, který je obsažen v NLTK [38].

```
from nltk.classify import NaiveBayesClassifier
trenovaciData = load_train_data()
klasifikator = NaiveBayesClassifier.train(trenovaciData)
klasifikator.classify(load_test_data())
```

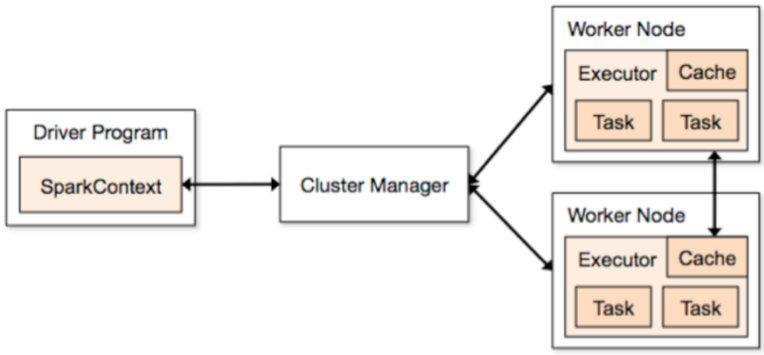
Obr. 31 - *Kód*: Využití klasifikace textu přes Bayesovu větu

Klasifikace proběhne pomocí vytvořeného klasifikátoru, který se naučí význam z training data, která jsou ve tvaru `trenovaciData = [('super', 'pozitivní'), …]`.

[114] Part of Speech – Snaha rozpoznat části textu, které se musí před analýzou rozdělit

5.7 Apache Spark

Důležitým prvkem při zpracovávání dat na Hadoopu je také Apache Spark. Jde o rychlý výpočetní systém v paměti, který není vázaný na MapReduce, podobně jako YARN. Spark lze spouštět buď samostatně, nebo jako aplikaci běžící pod YARN. Při samostatném spouštění[115] si Spark vytvoří hlavní jednotku *(Master deamon),* která koordinuje výpočetní jednotky na každém uzlu [39]. Ve Spark aplikacích se dá využívat cache[116], neboť se uchovávají data v paměti mezi jednotlivými úkoly. Aplikace fungují obdobně, jako skripty napsané v jazyce Pig Latin, což znamená, že je vytvořen seznam operací na datech, které se provádějí, až když si uživatel data vyžádá.



Obr. 32 - Princip funkce a řízení aplikací Apache Spark, *převzato z [39]*

Na obrázku je naznačen princip funkce Spark programu. Přes driver program se spustí hlavní funkce a provádějí se operace na clusteru. Worker nodes jsou pak samotné uzly, které aplikaci spouštějí pomocí procesů, které kontroluje executor. Jsou dvě možnosti, jak spouštět aplikace na YARN. Buď se Spark spouští lokálně, což znamená, že se samotný kód spouští na *klient* straně a lze jej kontrolovat pomocí Ambari, nebo jej lze spouštět přes cluster management.

5.7.1 Spark Context

Klíčovým prvkem k fungování programu je *SparkContext*. Má na starost připojení programu k ResourceManageru YARN a zažádání o prostředky v Hadoopu. Poté odesílá kód aplikace a spouští executory na uzly, které mu ResourceManager přidělil. Má na starost také maximální počet opakování[117] úkolu, pokud se nezdaří.

[115] Oficiální označení tohoto režimu zní „Standalone“
[116] Vyrovnávací paměť. Udržuje data v paměti, kvůli rychlejšímu přístupu
[117] Standardně nastavená hodnota je *spark.task.maxFailures = 4*

5.7.2 Data ve Spark

Hlavním prvkem pro správu dat ve Sparku jsou RDD[118]. Ve své podstatě jde o odolné řešení proti chybám. RDD jsou neměnné kolekce, které lze ovládat paralelně s možností preferovaných umístění datových bloků pro HDFS. Odolnost proti chybám je zajištěna tak, že pokud úloha Spark aplikace selže, pak se přepracovává pouze RDD, ve které se chyba objevila. Spark ukládá oddíly RDD do paměti RAM[119] na worker nodes a dokáže si tato data předávat mezi jednotlivými úlohami. Existuje řada způsobů, jak RDD vytvářet. Nejpoužívanějším v Hadoopu je `textFile` příkaz, který vytváří RDD z existujícího textového souboru distribuovaného přes HDFS. Pro binární soubory se využívá funkce `sequenceFile` a pro mapování RDD z existujícího kódu v Pythonu *(např. vytvořené pole)* se využívá příkazu `parallelize`. Vždy je však nutné tyto příkazy volat přes Spark Context, který musí být importován do kódu.

```
from pyspark import SparkContext as sparkContext
mojeRDD = sparkContext.textFile("/user/../data.txt")
```

Obr. 33 - *Kód*: Vytvoření RDD z textového souboru

Po vytvoření RDD je na místě s ním dále pracovat. Spark umožňuje dva základní typy operací, které jsou s RDD provádět.

- ***Transformace*** – Vytváří se nový RDD z existujících dat. Nespouští se okamžitě, ale až po zavolání. Existuje mnoho různých transformací: `map`, `filter`, `reduceByKey`, `sortByKey` atd.
- ***Akce*** – Po zahájení výpočtu se hodnoty vrací do ovladače programu. Akce tak defacto spouští transformaci. Oproti Pig zde existuje mnoho akcí, jako např. `reduce`, `foreach`, `count`, `first`, `take`, `saveAsTextFile`, `saveAsObjectFile`, `countByKey`, `collect` aj.

Lze si tento princip připodobnit ke zmiňovanému fungování Pig. Spark transformace se dají přirovnat k relacím Pig a akce k příkazu *DUMP* nebo *STORE*. Je evidentní, že lze používat široké spektrum operací k tomu, aby byla provedena analýza dat.

[118] Resilient Distributed Dataset – Distribuovaná datová sada s vysokou odolností
[119] Pokud je na worker node dostatek místa

Pro konkrétní vysvětlení akcí a transformací si představme následující příklad. Do datového souboru se ukládají data o návštěvnosti areálu tím způsobem, že ukládáme příjmení[120] a zajímá nás, kdo areál navštívil nejčastěji, aby byl vyhlášen, jako nejvěrnější zákazník. Samozřejmě se předpokládá, že máme k dispozici rozsáhlý záznam s miliony řádků.

```
from pyspark import SparkContext as sparkContext
dataRDD = sparkContext.textFile("/user/../data.txt")
kolekceRDD = dataRDD.flatMap(lambda radek: line.split())
                    .map(lambda slovo: (slovo, 1))
                    .reduceByKey(lambda a, b: a + b)
vyskytRDD = kolekceRDD.map(lambda rdd: (rdd[1], rdd[0]))
vystup = vyskytRDD.sortByKey(ascending = False).take(3)
>> [(841, 'Hnik'),
    (594, 'Novák'),
    (497, 'Novotný')]
```

Obr. 34 - *Kód*: Příklad využití Spark pro analýzu

Příklad je realizován tak, že je třeba přes Spark Context načíst data do RDD, následně textový soubor zpracovat principem algoritmu počítání slov a poté vytvořit strukturovaný výstup, který poskytne tři nejvíce frekventovaná příjmení.

5.7.3 Spark MLlib

Pro datové analytiky vznikla knihovna, které implementuje principy machine learningu. MLlib *(Machine Learning library)* je snadno škálovatelný napříč celým clusterem Hadoopu. Všechna data, která MLlib zpracovává jsou v RDD a jsou numerická *(RDD má vnitřní mechanismy k převodu textu na numerické hodnoty).* Díky numerickým hodnotám lze efektivně zpracovávat data pomocí různých datových typů *(Vectors, LabeledPoints[121], Rating atd.)*. Díky knihovně MLlib lze provádět klasifikace, regrese, clustering aj.

[120] Samozřejmě je na místě zamezit duplicitám, takže pro příklad by bylo vhodnější využít unikátní ID, ale z důvodu ilustrace předpokládejme, že nikdo nemá stejné příjmení
[121] Jedná se o vektory, které s sebou zároveň nesou informaci popisku

Knihovna MLlib obsahuje různé třídy pro konkrétní algoritmy a záleží jen na uživateli, které využije. Pro klasifikaci se využívá třídy **`pyspark.mllib.classification`**, která v sobě obsahuje konkrétní klasifikační algoritmy *(SVM metody, Naive Bayes, rozhodovací stromy atd.).* Pro regresi se pak využívá **`pyspark.mllib.regression`**, pro clustering **`pyspark.mllib.clustering`** a pro doporučování obsahu uživateli slouží **`pyspark.mllib.recommendation`** [40].

Pro pochopení si na příkladu vysvětleme např. využití metody KMeans, která je jedním z principů clusteringu.

```
from pyspark import SparkContext as sparkContext
from pyspark.mllib.clustering import KMeans
from numpy import array
mojeData = array ([[1,1], [2,2], [2,3], [1,4], [3,1], …])
dataRDD = sparkContext.parallelize(mojeData)
mlLibClustery = KMeans (dataRDD, 2, maxIterations = 10,
                runs = 10, initializationMode = "random")
```

Obr. 35 - *Kód*: Vytvoření shluků pomocí MLlib knihovny

Jak vidíme, tak zdánlivě těžké metody machine learningu lze provést pouze pomocí zavolání jednoduchých metod, které jsou obsažené v knihovně MLlib. Zde se vytvoří 2 clustery na základě definovaného pole hodnot, které se převede do RDD a pomocí deseti průchodů se určí clustery.

Spark nabízí mnoho možností, jak problematiku, kterou potřebujeme zkoumat uchopit. Rozhodně se doporučuje využít oficiální dokumentace, která je přívětivě napsaná tak, že obsahuje i vzorové příklady, podobně jako tento k tomu, aby datový analytik snadno pochopil, co chce implementovat. MLlib osahuje i machine learning algoritmy, které v této práci nejsou zmíněny[122], ale pracují na obdobném principu, jen jsou většinou rozšířeny o doplňující parametry pro upřesnění výsledků.

[122] Power interaction clustering, Streaming k-means, FP growth, Singular value decomposition aj.

5.8 Komplexní příklad doporučování obsahu

Výše bylo uvedeno mnoho principů, jak přistupovat k datům, analyzovat je atd. Uveďme si však závěrem ještě příklad toho, jak pracovat s doporučováním obsahu. V dnešní době je toto velice aktuální téma a je na místě vědět, jak celý proces funguje. Vezměme si jako vzor filmovou databázi *(kvůli současné popularitě služby Netflix)*, kde máme záznamy filmů *(ID, název, žánr)* a záznamy hodnocení *(ID uživatele, ID filmu, hodnota, timestamp).*

```
# tail filmy.dat
156;;V zajetí démonu;;Drama|Horor
168;;Avengers;;Sci-Fi
157;;Podfukáři;;Drama|Mysteriózní
…
```

Obr. 36 - *Kód:* Náhled struktury datového souboru filmů

```
# tail hodnoceni.dat
5837;;156;;3;;1555846988
8463;;168;;5;;1554142914
…
```

Obr. 37 - *Kód:* Náhled struktury datového souboru hodnocení

Ke zpracování dat a dalším krokům je třeba data umístit do Hadoop clusteru, kde se s nimi bude dále pracovat. Také je nutné data zpracovat tak, aby je Mahout[123] mohl dále využívat.

```
# hdfs dfs -put filmy.dat mojeSlozka/
# hdfs dfs -put hodnoceni.dat mojeSlozka/
```

Obr. 38 - *Kód*: Umístění dat pro doporučení do HDFS

Nyní je třeba data transformovat do požadované podoby, kde se zbavíme znaků, které oddělují řetězce *(záměrně se využívá např. ;; kvůli zamezení chyby při ořezávacích funkcích).* Definujme si proto skript, který převede “;;“ na “,“ které budou řetězce oddělovat.

[123] Apache Mahout – Systém pro doporučování položek

```
# !/bin/awk -f

BEGIN {FS = "";;""}

{print $1","$2","$3}
```

Obr. 39 - *Kód*: Skript pro transformaci souboru v *.awk[124]

Všimněme si také, že poslední údaj *timestamp* ignorujeme ve výstupu, neboť není pro další analýzu důležitý. Když už víme, jak data transformovat, můžeme přejít k samotné aplikaci skriptu na data. Pomocí Hadoop Streamingu spustíme MapReduce s naším skriptem, jako mapovací částí.

```
# yarn jar $HADOOP_HOME/slozka/hadoop-streaming.jar -files
mujSkript.awk  -input mojeSlozka/hodnoceni.dat  -output
mojeSlozka/hodnoceniFinal -mapper mujSkript.awk
```

Obr. 40 - *Kód*: Využití Hadoop Streamingu k aplikaci

Nyní je na místě provést samotný algoritmus doporučení. Využijeme zde Apache Mahout recommenderu k tomu, aby poskládal řetězce a predikoval hodnocení tak, aby z nich mohl datový analytik dále čerpat.

```
# mahout recommenditembased -i mojeSlozka/hodnoceniFinal -o
mojeSlozka/vysledek –tempDir mojeSlozka/temp
-s SIMILARITY_COSINE

>> 5837[182:5.0, 742:5.0, 153:5.0, 978:5.0, …]

   8463[173:5.0, 423:5.0, 384:5.0, 635:5.0, …]
```

Obr. 41 - *Kód*: Aplikace Apache Mahout na data hodnocení

Díky tomuto algoritmu dostáváme seznam vygenerovaných doporučení pro jednotlivé uživatele na základě položek a kosinové podobnosti. Ve výstupu můžeme vidět, že pro uživatele s *ID 5837* jsou doporučené filmy s *ID 182, 742, 153* atd. s hodnocením 5.0. Na základě toho už lze pracovat dále, neboť jsme byli schopni analyzovat obrovské množství dat a vytvořit doporučení pro každého uživatele. Takto vygenerovaný seznam lze například použít pro zobrazení v prohlížečích, nebo předat dále do jiných API.

[124] Skriptový jazyk navržený pro zpracovávání textových řetězců, pro takový úkol je efektivnější využít tento rychlý přístup než programovat Python skript, který musí soubor rozdělit po řádcích a ořezávat je

6 Využití v praxi

Je nutné přemýšlet o problematice velkých dat také z praktické stránky, tudíž kdy se vlastně společnostem vyplatí na řešení ekosystému Hadoop přejít. Je nutné zdůraznit, že Hadoop byl první, který nabídl velkým společnostem efektivní zpracovávání velkého množství nestrukturovaných dat. Jak je z práce patrné, tak celý ekosystém je provázaný a skládá se z dílčích částí, které dohromady tvoří velice funkční celek, díky kterému lze provádět téměř jakékoliv operace s big data. Výhodou celého systému je jednoduchost jeho nasazení pro velké společnosti. Zde se totiž předpokládá, že už je vytvořena struktura pro celé fungování systému neboli že je k dispozici dostatečné množství výpočetních jednotek. K nasazení systému prakticky stačí instalace pomocí skriptů a následné nastavení celého systému. Tuto věc provádí zásadně pouze proškolený personál a jak bylo zmíněno, celkové porozumění tomu, co se vlastně nastavuje je velice sofistikované. To však zůstává na straně instalace a pro společnost to není zatěžující, proto je celková migrace na ekosystém Hadoop pro koncové klienty velice jednoduchá, neboť se nic nemění, ale pouze se propojí clustery, nodes a celkové nastavení umožní využívat firemní infrastrukturu pomocí Hadoopu.

Je však pravda, že se v dnešní době stále více intenzivněji mluví o tom, zdali je Hadoop perspektivním přínosem pro nová řešení. Na trhu se objevují nové možnosti toho, jak k velkému množství dat přistupovat. Internetové zdroje často uvádějí informace o tom, že alternativou k Hadoopu je Apache Storm, nebo Apache Spark. Jak bylo v práci uvedeno, jde o frameworky, které jsou v Hadoopu již obsažené. Samozřejmě díky nim lze přistupovat k big data, ale nenabízí stále tolik možností, jako celkový Hadoop.

Opravdu perspektivním řešením však začíná být *Google BigQuery.* Vzhledem k rostoucí popularitě internetových úložišť se Google rozhodl pro řešení, které je lehce nasaditelné a díky cloud computingu je také velice rychlé. Nabízí jako Hadoop možnosti využívání machine learningu a celková analýza tak může být pro datové analytiky pohodlná, neboť nemusí řešit firemní HW infrastrukturu. Nastává tu však také zásadní zlom s jeho výhodou a nevýhodou v jednom. Mnoho velkých korporací si nepřeje, aby byla data ukládána mimo společnost a raději si připlatí, za cenu, že data zůstanou v jejich datových centrech. Do budoucna však může být tato platforma hojně využívána středními podniky, ale rozhodně to neznamená, že Hadoop už není a nebude v kurzu. Hadoop je stále nejrozšířenějším na trhu a vzhledem k tomu, že zde byl jako jeden z prvních je také mnoho

datových analytiků zvyklých na jeho prostředí, což je pro ně přínosnější než hrát o milisekundy početního výkonu.

Zde je také dobré přemýšlet nad efektivitou data science. V práci bylo mnohokrát zmíněno to, jak přínosné může být využívání technik strojového učení. V praxi se s doporučovacími mechanismy setkáváme stále častěji a v tom, jak predikovat to, co by koncový uživatel mohl chtít, nebo potřebovat je velký potenciál pro budoucí obchodní využití společností. Práce jasně popisuje modelovou situaci s doporučováním oblíbených filmů, které využívá Netflix, ale v současné době je tento přístup žádoucím již v každém odvětví. Těchto principů se nevyužívá pouze v placených reklamách, nebo Facebooku. Aby byla jakákoli větší společnost v dnešní době úspěšná, snaží se využívat zmíněných principů tak, aby dovedla potencionální zákazníky k provedení transakce. Díky tomu se tak otevírají možnosti pro datové analytiky na trhu práce, neboť je poptávka po jejich práci stále vyšší.

7 Shrnutí výsledků

Práce se v praktické části zabývala detailním vysvětlením jednotlivých mechanismů pro práci s big data. Práce objasnila, že je výhodné využívat skriptovací jazyk Pig Latin pro zjišťování důležitých informací nad velkými datovými celky pomocí různých metod. Dále byl v práci objasněn jazyk Python, který je, jak práce vysvětluje vysoce efektivní pro psaní skriptů k logickému zpracovávání úloh nad big data. Bylo vysvětleno, jak pomocí tohoto jazyka definovat funkce, nebo psát celé skripty. Pro data science analýzu dat byla práce směřována na popsání jednotlivých možností, jak teoretické znalosti implementovat. Byly popsány knihovny pro matematické operace a modelování datových struktur.

V další části této práce se přechází na samotné využití algoritmů na Hadoopu. Bylo objasněno, jak docílit psaní skriptů pro MapReduce přístup, tak, aby byl algoritmus distribuován napříč celým clusterem, a pomocí Hadoop nebo Pig streamingu byl aplikován na požadovaná data. Bylo také zmíněno, jak psát pomocí kombinace jazyků Pig Latin a Pythonu uživatelem definované funkce, které pak lze přenášet na velké datasety a provádět tak cílené analýzy.

Následně práce přechází do praktických ukázek toho, jak využívat principy strojového učení v praxi. Nejprve je vysvětleno, jak pomocí matematických operací využívat znalostí pro doporučování obsahu a poté je práce zaměřena na výklad a ukázky toho, jak pracovat se SciKit-Learn knihovnou, která je pro využití strojového učení navržena. Je zde rozebráno, jaké třídy využívat pro konkrétní případy využití klasifikace, shlukování nebo regrese. Práce se rovněž zaměřuje na stále aktuálnější téma, a sice, jak pomocí počítače rozumět lidskému jazyku. Je vysvětleno, jakých knihoven pro úspěšnou klasifikaci využívat, jak rozeznat a správně určit text, a tím tak systém „naučit“ principům klasifikace lidského jazyka.

V poslední části práce se popisuje Apache Spark, který je pro zpracování dat v Hadoopu velice důležitý. Je popsáno, jak v něm funguje distribuce dat a je názorně ukázáno, jak takové postupy využívat. Je zde také popsána knihovna MLlib, která obsahuje důležité třídy k využití strojového učení, což je také následně vysvětleno na příkladu. Závěrem je nastíněna kompletní algoritmizace pro doporučování obsahu s využitím Apache Mahout, díky kterému lze získat požadovaný výsledek, který lze dále využít.

8 Závěr práce

V této diplomové práci bylo podrobně chronologicky rozebráno, jak se v minulosti přistupovalo k datovým objektům a jak šel trend až do současné podoby. Stejně tak bylo vysvětleno, jak se charakterizují big data a proč jsou důležitá. Dále bylo abstraktně popsáno, co je to ekosystém Hadoop a z čeho se skládá tak, aby bylo možné po přečtení chápat, jak je celý systém provázaný a jak vlastně celý přístup k datům a obsluha celého Hadoopu funguje. Dalším tématem, kterým se tato práce zabývala bylo data science. Bylo vysvětleno, jak lze charakterizovat prvky, které vedou k analýze na základě strojového učení. Po celkovém popsání problematiky data science práce vede k jasnému objasnění, jak pomocí Hadoopu využívat principů data science v praxi. Je popsáno, jak konkrétně navrhovat algoritmy, tak, aby datový analytik shromažďoval důležité informace ve velkém nestrukturovaném množství dat. Rovněž je detailně popsána implementace mnohých alternativ přístupu k datům v různých programovacích jazycích a s využitím mnohých frameworků, které Hadoop nabízí. Dále je také popsáno, proč je toto téma aktuální a klíčové pro smýšlení nad zpracováváním velkého objemu dat.

Po prostudování této práce by měl být čtenář schopen porozumět a následně aplikovat mnohé popisované algoritmy v praxi pro další využití. Popsané algoritmy a problematika je stále více aktuální z hlediska shromažďování větších objemů dat a je nezbytná pro efektivní přístup v reálném čase. Je však jen na programátorských schopnostech, jak efektivní přístup bude. Celkově je tak Hadoop pro data science vysoce efektivním nástrojem, kterého lze využívat v širokém spektru funkcionalit.

9 Seznam použité literatury

[1] A History Of Data Collection. Gutcheckit.com [online]. Dostupné z: https://www.gutcheckit.com/blog/a-history-of-data/

[2] AHO, Alfred V., John E. HOPCROFT a Jeffrey D. ULLMAN. Data structures and algorithms. Reading, Mass.: Addison-Wesley, c1983. ISBN 978-0201000238.

[3] McGraw-Hill dictionary of scientific and technical terms. 6th ed. New York: McGraw-Hill, c2003. ISBN 978-0070423138.

[4] CODD, E. F. The relational model for database management: version 2. Reading, Mass.: Addison-Wesley, c1990. ISBN 978-0-201-14192-4.

[5] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. ISBN isbn0-321-12742-0.

[6] INMON, William H. Data warehouse performance. New York: John Wiley, 1999. ISBN 0471298085.

[7] Datové sklady. ČVUT [online]. Praha. Dostupné z: https://cw.fel.cvut.cz/wiki/_media/courses/a5m33izs/datove-sklady.pdf

[8] MAYER-SCHÖNBERGER, Viktor a Kenneth CUKIER. Big Data. Brno: Computer Press, 2014. ISBN isbn978-80-251-4119-9.

[9] DEAN, Jeffrey a Sanjay GHEMAWAT. MapReduce: Simplified Data Processing on Large Clusters [online]. 2004, 13. Dostupné z: https://static.googleusercontent.com/media/research.google.com/cs//archive/mapreduce-osdi04.pdf

[10] Big Data: The next frontier for innovation [online]. 2016. Dostupné také z: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation

[11] Forecast and Trends, 2017–2022 [online]. Cisco, 2018. Dostupné také z: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[12] Apache Hadoop [online]. Dostupné také z: http://hadoop.apache.org

[13] ALAPATI, Sam R. Expert hadoop administration: managing, tuning, and securing spark, yarn, and hdfs. Boston, MA: Addison-Wesley, 2016. ISBN 9780134597195.

[14] Hadoop Tutorial [online]. Dostupné z: https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/

[15] VOHRA, Deepak. Practical Hadoop ecosystem: a definitive guide to Hadoop-related frameworks and tools. New York, NY: Apress, [2016]. ISBN 9781484221983.

[16] Hadoop Admin Manual – Ambari [online]. Dostupné z: https://docs.cask.co/cdap/4.2.0/en/admin-manual/installation/ambari.html

[17] Ambari User Guide [online]. Dostupné také z: ftp://ftp.newtek.com/pub/reads/bk_ambari-user-guide-20161128.pdf

[18] HDFS Architecture [online]. Apache. Dostupné také z: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[19] ALVARO, Peter, Tyson CONDIE a Neil CONWAY. BOOM Analytics: Exploring Data-Centric, Declarative Programming for the Cloud [online]. 2010, , 223-236. Dostupné z: http://www.neilconway.org/docs/booma_eurosys2010.pdf

[20] HOLMES, Alex. Hadoop in practice. Shelter Island, NY: Manning, c2012. ISBN 1617290238.

[21] Apache Hadoop Yarn [online]. Hortonworks. Dostupné také z: https://hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/

[22] LEWIS, Michael. Moneyball: the art of winning an unfair game. New York: W. W. Norton, c2003. ISBN 0393057658.

[23] LEE, Kyong-Ha, Hyunsik CHOI a Bongki MOON. Parallel Data Processing with MapReduce: A Survey [online]. 2011, 11-20. Dostupné z: https://sigmodrecord.org/publications/sigmodRecord/1112/pdfs/04.surveys.lee.pdf

[24] PROVOST, Foster; FAWCETT, Tom. Data Science for Business: What you need to know about data mining and data-analytic thinking. " O'Reilly Media, Inc.", 2013.

[25] CHABLANI, Manish. Clustering based on similarity [online]. Dostupné z: https://towardsdatascience.com/semantic-similarity-classifier-and-clustering-sentences-based-on-semantic-similarity-a5a564e22304

[26] K-Algorithm Clustering [online]. Dostupné také z: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[27] LECUN, Yann, Yoshua BENGIO a Geoffrey HINTON. Deep learning. Nature [online]. 2015, 521(7553), 436-444. DOI: 10.1038/nature14539. ISSN 0028-0836. Dostupné z: http://www.nature.com/articles/nature14539

[28] Human in a machine world: MAE and RMSE [online]. Dostupné také z: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

[29] Underfitting and Overfitting in Machine Learning [online]. Dostupné z: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

[30] Pig Latin: A Not-So-Foreign Language for Data Processing [online]. Dostupné z: http://infolab.stanford.edu/~olston/publications/sigmod08.pdf

[31] HARMS, Daryl D. a Kenneth MCDONALD. Začínáme programovat v jazyce Python. 2., opr. vyd. Přeložil Ivo FOŘT, přeložil Lubomír ŠKAPA. Brno: Computer Press, 2008. ISBN 978-80-251-2161-0.

[32] Pandas: Python Data Analysis [online]. Dostupné také z: https://pandas.pydata.org

[33] DING, Mengwei, Long ZHENG a Yanchao LU. More Convenient More Overhead: The Performance Evaluation of Hadoop Streaming [online]. , 307-313. Dostupné z: https://www.researchgate.net/profile/Song_Guo6/publication/239761247_More_convenient_more_overhead_The_performance_evaluation_of_Hadoop_streaming/links/56b932d708ae9d9ac67dcf16.pdf

[34] BENGFORT, Benjamin a Jenny KIM. Data analytics with Hadoop: an introduction for data scientists. Sebastopol, CA: O'Reilly, [2016]. ISBN 1491913703.

[35] Cosine Similarity. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation. Dostupné z: https://en.wikipedia.org/wiki/Cosine_similarity

[36] SciKit-Learn SVM [online]. Dostupné z: https://www.bogotobogo.com/python/scikit-learn/images/svm

[37] SciKit-Learn: Dokumentace [online]. Dostupné také z: https://scikit-learn.org/

[38] ARUMUGAM, Rajesh a Rajalingappaa SHANMUGAMANI. Hands-on natural language processing with Python: a practical guide to applying deep learning architectures to your NLP applications. Birmingham: Packt Publishing, 2018. ISBN 978-1-78913-949-5.

[39] How to run an application on Standalone cluster in Spark? [online]. Dostupné také z: https://blog.knoldus.com/how-to-run-an-application-on-standalone-cluster-in-spark/

[40] Apache Spark: MLlib [online]. 2019. Dostupné také z: https://spark.apache.org/mllib/

10 Seznam obrázků

11 Rejstřík použitých názvů

12 Zadání diplomové práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Hnik Jaromír	Veská 114, Sezemice - Veská	I1600869

TÉMA ČESKY:

Hadoop a jeho využití pro Data Science

TÉMA ANGLICKY:

Hadoop and its potential use for Data Science

VEDOUCÍ PRÁCE:

Ing. Karel Mls, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Cílem práce bude analýza problematiky a návrh řešení implementace pro analýzu dat za použití strojového učení a vybraných knihoven.

Osnova práce:
Úvod
Seznamení s Hadoop problematikou
Seznámení s Data Science problematikou
Návrh řešení implementace pro analýzu dat
Shrnutí výsledků
Závěr

SEZNAM DOPORUČENÉ LITERATURY:

ZIKOPOULOS, Paul, et al. Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, 2011.
HOLMES, Alex. Hadoop in practice. Manning Publications Co., 2012.
SAGIROGLU, Seref; SINANC, Duygu. Big data: A review. In: Collaboration Technologies and Systems (CTS), 2013 International Conference on. IEEE, 2013. p. 42-47.
PROVOST, Foster; FAWCETT, Tom. Data Science for Business: What you need to know about data mining and data-analytic thinking. " O'Reilly Media, Inc.", 2013.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....