

České vysoké učení technické v Praze  
Fakulta elektrotechnická

katedra počítačů

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jakub Moravec**

Studijní program: Otevřená informatika  
Obor: Softwarové systémy

Název tématu: **Transformace uživatelských dotazů pro analýzu dat v průmyslové automatizaci**

Pokyny pro vypracování:

Cílem práce je navrhnout způsob transformace dotazů pro analýzu velkého objemu sémanticky popsaných sensorických dat s využitím nástrojů KNIME a platformy pro Big Data. Data jsou popsána pomocí existující ontologie SHS. Práce bude zahrnovat implementaci komponenty pro analýzu dat a je plánována jako součást existujícího Semantic Big Data Historianu.

Seznam odborné literatury:


- [1] Data Smart: Using Data Science to Transform Information into Insight (ISBN 978-1118661468)
- [2] Handbook on Ontologies SE (ISBN 978-3-540-92673-3)
- [3] Hadoop: The Definitive Guide (ISBN 978-1449311520)

Vedoucí: Ing. Václav Jirkovský

Platnost zadání: do konce letního semestru 2016/2017

  
prof. Ing. Filip Železný, Ph.D.  
vedoucí katedry



  
prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 13. 1. 2016



České vysoké učení technické v Praze  
Fakulta elektrotechnická  
Katedra počítačů



Bakalářská práce

**Transformace uživatelských dotazů pro analýzu dat v  
průmyslové automatizaci**

*Jakub Moravec*

Vedoucí práce: Ing. Václav Jirkovský

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové systémy

23. května 2016



## Poděkování

Rád bych poděkoval Ing. Václavu Jirkovskému za vedení této práce a ochotu při konzultacích práce, které byly důležité pro správné směřování a úspěšné dokončení práce. Za cenné postřehy pro prezentaci práce bych rád poděkoval Ing. Marku Obitkovi Ph.D. Závěrem bych rád poděkoval své rodině a přátelům za podporu a rady při psaní práce.



## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2016

.....





# Abstract

There are big goals for data science today as the importance of data grows. High volumes of heterogeneous data are generated in many fields including industry. A lot of knowledge can be gained out of this data, although it can hardly be accomplished using standard RDBMS tools. Big Data and Ontologies are technologies that help to attain these goals. These technologies together with suitable analytic tools and framework Semantic Big Data Historian are described and used in this thesis, which goal is to devise and implement a way of transformation of queries for analyzing high volumes of semantically described data. This objective is met by the implementation of an analytic component, that generates queries for transformation of semantically described data and enables defining preprocessing part of the analysis of this data. These functions distinctly simplify the analysis.

# Abstrakt

Spolu s nárůstem důležitosti dat jsou dnes kladeny i velké požadavky na jejich analýzu. V mnoha oblastech, včetně průmyslu, jsou generovány velké objemy heterogenních dat. Analýza těchto dat může vést k získání mnoha informací, je ale složité tato data analyzovat pomocí standardních RDBMS nástrojů. Koncepty Big Data a ontologie pomáhají dosažení těchto výsledků. Právě tyto technologie, spolu s vhodnými analytickými nástroji a frameworkem Semantic Big Data Historian, jsou popsány a použity v této práci, jejímž cílem je navrhnout a implementovat způsob transformace dotazů pro analýzu velkého objemu sémanticky popsaných dat. Tohoto cíle je dosaženo pomocí implementace analytické komponenty, která generuje dotazy pro transformaci sémanticky popsaných dat a umožňuje definování prvotní části analýzy. Tyto funkce analýzu dat výrazně zjednodušují.



# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Zpracování dat v průmyslu . . . . .	1
1.2	Semantic Big Data Historian . . . . .	1
1.3	Definice problému . . . . .	2
1.4	Struktura bakalářské práce . . . . .	2
<b>2</b>	<b>Teoretická část</b>	<b>5</b>
2.1	Big Data . . . . .	5
2.1.1	Koncepty Big Data . . . . .	6
2.1.2	Hadoop . . . . .	8
2.1.2.1	HDFS . . . . .	8
2.1.2.2	HBase . . . . .	11
2.2	Analytické nástroje . . . . .	11
2.2.1	MapReduce . . . . .	12
2.2.1.1	Výhody a nevýhody MapReduce . . . . .	14
2.2.2	Mahout . . . . .	14
2.2.2.1	Výhody a nevýhody Mahout . . . . .	15
2.2.3	Hive . . . . .	16
2.2.3.1	Výhody a nevýhody Hive . . . . .	18
2.2.4	KNIME . . . . .	18
2.2.4.1	Výhody a nevýhody KNIME . . . . .	20
2.3	Ontologie . . . . .	20
2.3.1	Formální reprezentace . . . . .	22
2.3.2	Sémantický web . . . . .	23
<b>3</b>	<b>Experimentální část</b>	<b>27</b>
3.1	Analyzovaná data . . . . .	27
3.2	Popis architektury . . . . .	28
3.3	Popis funkcí analytické komponenty . . . . .	30
3.3.1	Načtení metadat z Hivu . . . . .	31
3.3.2	Načtení KNIME workflow . . . . .	31
3.3.3	Vygenerování transformačního dotazu . . . . .	32
3.3.4	Úprava KNIME workflow . . . . .	33
3.3.5	Spuštění KNIME workflow . . . . .	33
3.3.6	Uložení výsledků analýzy . . . . .	33

3.3.7	Rozpad časové náročnosti analýzy . . . . .	34
3.4	Implementace . . . . .	34
3.5	Návrh budoucího rozšíření . . . . .	36
<b>4</b>	<b>Závěr</b>	<b>37</b>
<b>A</b>	<b>Obsah přiloženého CD</b>	<b>45</b>

# Seznam obrázků

2.1	Vizualizace průběhu MapReduce jobu s jedním reduce taskem [59]	13
2.2	Vizualizace architektury aplikace s využitím Mahout recommender [41]	15
2.3	Vizualizace připojení Hive klientů na Hive server [59]	18
2.4	Schéma datového toku KNIME workflow [4]	19
2.5	Příklad XML kódu RDF grafu ve formátu N-Trippl [52]	25
2.6	RDF reprezentovaný graf [52]	25
3.1	Zjednodušená vizualizace reprezentace sémanticky uložených dat	28
3.2	Průběh analýzy bez použití analytické komponenty	29
3.3	Průběh analýzy s použitím analytické komponenty	30
3.4	Popis hlavní stránky grafického uživatelského rozhraní	35



# Kapitola 1

## Úvod

### 1.1 Zpracování dat v průmyslu

Odvětví průmyslové automatizace se dnes potýká s potřebou zpracování a analýzy velkého objemu různorodých dat. Těmito daty mohou být například sensoricky generované datové řady, informace z vyšší úrovně řízení (MES/ERP systémy<sup>1</sup>) nebo data z externích zdrojů<sup>2</sup>. Tato data jsou různorodá nejen z pohledu informací, které přinášejí, ale také z pohledu své reprezentace a svého významu, který může být chápán odlišně v různých kontextech. Zároveň se zvyšuje náročnost na analýzu dat z hlediska časových omezení. Analýzy, které dříve byly prováděny jednou týdně či měsíčně je dnes zapotřebí vykonávat v reálném nebo téměř reálném čase, protože jsou často základem pro automatické řízení a rozhodování systémů.

Konvenčními přístupy ke zpracování dat bylo donedávna možné naplnit pouze některé z těchto požadavků. Změnu v tomto ohledu přinesly koncepty a nástroje obecně označované jako Big Data, které umožňují zpracování takovýchto dat s ohledem na vysoké požadavky, které má průmyslová automatizace.

### 1.2 Semantic Big Data Historian

Semantic Big Data Historian[19, 29] je stejně jako běžné historiany zaměřen na zpracování a ukládání dat v podobě časových řad produkovaných zpravidla sensorickými měřeními. Jeho hlavním cílem je rozšířit možnosti běžných historianů, a to především umožněním pokročilých analýz nad různorodými daty v měřítku, které doposud nebylo možné. Záměrem je rozšířit možnosti analýzy pomocí integrace různorodých datových zdrojů - kromě sensoricky generovaných dat se může jednat například o data z vyšší úrovně řízení nebo o externí data. Integrace těchto datových zdrojů a jejich následná analýza může přinést nový pohled na získaná data v kontextech, ve kterých zkoumání v tomto měřítku doposud nebylo možné. Právě zvětšení měřítka analýzy, tedy objemu analyzovaných dat, je dalším z cílů Semantic Big Data Historianu.

---

<sup>1</sup>Enterprise Resource Planning/ Manufacturing Execution Systems - integrované systémy průmyslových organizací řídící velkou část provozu organizace, například plánování, zásoby, finance.

<sup>2</sup>Příkladem mohou být například data o počasí nebo kurzech měn.

Schopnost integrace heterogenních dat z různých datových zdrojů je základním předpokladem pro naplnění definovaných cílů. Pro tento účel je zvolen přístup sémantického zpracování a ukládání dat. Data jsou integrována pomocí sdílené SHS ontologie<sup>3</sup>. Ontologie popisuje data generovaná jednotlivými senzory i data získaná z dalších zdrojů a slouží jako doménový model pro sémantickou integraci dat. Integrovaná data jsou ukládána ve formě trojic {subjekt, predikát, objekt} sémanticky odpovídajících ontologii, která uložená data popisuje.

Data jsou zpracovávána pomocí frameworku Apache Hadoop, díky čemuž je umožněno ukládání a analyzování velkého objemu dat. Jako datové úložiště slouží distribuovaný souborový systém HDFS, který umožňuje téměř neomezené škálování řešení z pohledu objemu dat. Obdobně je zajištěna také škálovatelnost analytických dotazů, které jsou vykonávány pomocí frameworku MapReduce. Data nejsou dotazována přímo pomocí MapReduce dotazů - pro přístup k datům je použit framework Apache Hive, který umožňuje dotazovat uložená data pomocí vlastní implementace dotazovacího jazyka SQL.

Semantic Big Data Historian tedy definuje způsoby sémantické integrace datových zdrojů a zpracování dat pomocí Big Data konceptů a nástrojů.

### 1.3 Definice problému

Pro koncovou analýzu dat uložených pomocí Semantic Big Data Historianu je používán analytický nástroj KNIME. Analytik (koncový uživatel) má možnost dotazovat uložená data pomocí Hive dotazů a získaná data poté analyzovat s využitím širokých možností analytického nástroje. Tento relativně přímočarý přístup se ovšem komplikuje faktem, že data jsou uložena v sémantické podobě po trojicích a KNIME nemá nástroje pro analýzu dat v takovémto formátu. Uživatel tedy musí uložená data před jejich analýzou nejdříve transformovat pomocí Hive dotazu, transformovaná data uložit na datové úložiště (HDFS) a až poté může přistoupit k samotné analýze. Další možnou komplikací může být objem analyzovaných dat. Semantic Big Data Historian je koncipovaný pro efektivní zpracování velkého objemu dat, což umožňují použité Big Data nástroje. Charakteristikou těchto nástrojů je, že jsou data zpracována paralelně za využití výkonu mnoha fyzických zařízení. Naproti tomu KNIME, který není primárně určený pro paralelní zpracování dat, může mít s analýzou velkého objemu dat výkonový problém - může využívat výkon pouze jednoho fyzického zařízení.

Cílem této práce je navrhnout způsob transformace dat zpracovávaných Semantic Big Data Historianem pro efektivní analýzu těchto dat pomocí definovaného analytického nástroje KNIME a tuto transformaci následně implementovat pomocí analytické komponenty. Úkolem analytické komponenty je kromě samotné transformace sémanticky uložených dat také umožnění předzpracování transformovaných dat za pomoci Big Data nástrojů, a tím snížení výpočetní náročnosti koncové analýzy prováděné nástrojem KNIME.

### 1.4 Struktura bakalářské práce

Práce je rozdělena na teoretickou a experimentální část.

---

<sup>3</sup>SHS ontologie je rozšířením SSN ontologie[60] - základní ontologie pro zpracování senzorických dat.



Cílem teoretické části práce je popsat koncepty, technologie a nástroje, které jsou využívány frameworkem Semantic Big Data Historian<sup>4</sup> nebo jsou používány v práci samotné, respektive v její experimentální části. Teoretická část slouží také jako znalostní základ pro experimentální část práce, z toho důvodu je zaměřena především na reprezentaci a způsob zpracování dat jednotlivými nástroji.

Kapitola **Big Data** popisuje nástroje, pomocí kterých framework Semantic Big Data Historian zpracovává, ukládá a dotazuje data.

Následující kapitola **Analytické nástroje** popisuje možné způsoby dotazování a analýzy dat uložených Semantic Big Data Historianem.

Poslední teoretická kapitola **Ontologie** popisuje koncepty a nástroje používané při integraci datových zdrojů a reprezentaci sémanticky uložených dat.

Experimentální část práce se zabývá samotným řešením definovaného problému. Součástí experimentální části je implementace analytické komponenty a popis řešení problému včetně popisu samotné komponenty.

Výsledky bakalářské práce jsou zhodnoceny v závěru práce.

---

<sup>4</sup>Popsány jsou i některé nástroje, které sice v současné době frameworkem používány nejsou, jejich použití je ale do budoucna uvažováno (HBase, Mahout).



# Kapitola 2

## Teoretická část

### 2.1 Big Data

Termín Big Data je dnes zmiňován v mnoha kontextech a popisován různými definicemi. Základní a zřejmě nejrozšířenější definice, která popisuje Big Data pomocí tří hlavních charakteristik<sup>1</sup> má původ ve výzkumné zprávě Douglase Laney [23] z roku 2001. Tato definice byla roku 2012 upravena do následující podoby<sup>2</sup>: "Big Data jsou soubory dat velkého objemu, velké rychlosti a/nebo velké různorodosti, která vyžadují nové formy zpracování pro umožnění lepšího rozhodování, většího porozumění domény a optimalizace procesů.". Definice popisuje Big Data z pohledu tří hlavních dimenzí - objemu, různorodosti a rychlosti.

**Objem** dat virtuálního světa je dnes obrovský a rychle narůstá. Se vznikem internetu se změnil způsob, jakým data vznikají - dříve byla data zadávána do systémů zaměstnanci společností, zatímco dnes generují data především sami uživatelé internetu. Podle statistik dnes sociální síť Facebook spravuje více než 30 petabytů uživatelských dat. Americký obchodní řetězec Walmart zpracovává milion zákaznických transakcí za hodinu a velikost jeho databází je odhadována na 2,5 petabytu. V oblasti průmyslu je generováno stále více senzorových dat potřebných ke zdokonalování automatizace procesů a tento trend má předpoklad rapidně narůstat s rozvojem oblasti IoT<sup>3</sup> [27], která přináší automatizaci i do každodenního života. Na celkovém objemu dat virtuálního světa se nepodílejí pouze velké společnosti, ale i uživatelé běžným používáním internetu, tvorbou fotografií, telefonními hovory, emailovou komunikací atd. Podle statistiky IDC<sup>4</sup> [16] se celkový objem dat virtuálního světa zdvojnásobuje každé dva roky. IDC odhaduje, že v roce 2020 bude tento objem 44 zettabytů, což je v bitech číslo blízké se počtu hvězd ve vesmíru.

Podstatné z pohledu konkrétního případu je, že soubor dat roste natolik rychle, že je obtížné s ním pracovat pomocí běžných databázových konceptů a nástrojů.

Možnost efektivního zpracování velkého objemu dat přináší nové možnosti pro analýzu dat. Je možné analyzovat data, která by za použití standardních metod analyzována být

---

<sup>1</sup>Definice Big Data pomocí "3Vs"

<sup>2</sup>V originálním znění: "Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization".

<sup>3</sup>Internet of Things

<sup>4</sup>International Data Corporation

nemohla (náročnost analýzy by byla příliš vysoká) a hledat v datech souvislosti, které by zůstaly utajeny.

**Rychlost** v pojetí definice Big Data má několik rozměrů - jak rychle jsou data vytvářena, ukládána a zpracovávána (analyzována a vizualizována). V minulosti, především v oblasti datových skladů a Business Intelligence [31], se stalo standardem dávkové ukládání, zpracovávání a analyzování dat. Následkem toho byly analýzy potřebné pro rozhodování napočítané nad daty den, týden či měsíc starými. Přestože jsou tato řešení stále hojně používána, narůstá potřeba data ukládat, zpracovávat a analyzovat v reálném nebo téměř reálném čase. Systémy a aplikace v oblasti automatizace (ať už jde o průmyslovou automatizaci, nebo například chytrá zařízení denního použití spadající pod IoT) jsou kriticky závislé na zpracování dat v reálném čase.

**Různorodost** dat, která potřebujeme zpracovat, stále narůstá. Velká část dat generovaných společnostmi je nestrukturovaná, což přináší problémy s jejich ukládáním a zpracováním. Data jsou často získávána z různých zdrojů a jejich reprezentace je rozdílná - aplikační databáze, streamy senzorových dat, logovací soubory, nestrukturované textové soubory. Důležitým zdrojem dat mohou být ale i fotografie nebo videa. Při snaze zpracovávat, ukládat a analyzovat takto různorodá data pomocí relačních databází nastává řada problémů - například pevně definované schéma databáze neumožňuje flexibilní práci s daty. Právě možnost analýzy a tvorby prognóz nad integrovanými daty je považována za jeden ze zásadních přínosů konceptů Big Data.[26]

Big Data koncepty nabízejí řešení těchto problémů, na druhé straně ale často postrádají vlastnosti, díky kterým jsou konvenční RDBMS<sup>5</sup> řešení dnes tak masivně používána.

### 2.1.1 Koncepty Big Data

Nejpodstatnějším rozdílem proti konvenčním technologiím je distribuovanost datového uložení a distribuované zpracování dat.[15] Není snaha ukládat data na jeden centralizovaný server, jako je tomu u RDBMS, naopak, data jsou distribuována přes mnoho serverů pomocí distribuovaného souborového systému (HDFS, GFS, CassandraFS, Amazon S3). Efektivita ukládání a dotazování dat úzce souvisí s počtem uzlů distribuovaného uložení a rozprostřením dat mezi uzly. Pokud je rozprostření dat rovnoměrné, potom je škálovatelnost distribuovaného uložení téměř lineární.

Masivní paralelizaci je také docíleno zvýšení rychlosti ukládání a čtení dat. Zatímco navyšovat kapacitu centralizovaného datového uložení dnes není velký problém, udržet při zvyšující se kapacitě konstantní přístupovou rychlost k datům (čas, za který mohou být přečtena všechna data v uložení) je komplikované. Nárůst kapacity datových uložení mezi lety 1990 až 2010 byl více než 700násobný, rychlost čtení dat se ale v průměru zvýšila "pouze" 22krát.[59] Je tedy zřejmé, že centralizovaná datová uložení zásadní vývoj v rychlosti zpracování dat přinést nemohou. Nabízí se tedy řešení pomocí paralelizace - data je potřeba nejen paralelně ukládat, ale také paralelně číst a zpracovávat. Ve světě relačních databází se touto cestou vydala například Teradata. Tento databázový systém užívaný v oblasti data warehousingu zaručuje lineární škálovatelnost nejen z hlediska kapacity, ale také z hlediska zpracování dat. Řešení je ovšem mnohonásobně dražší než jsou řešení používající Big Data koncepty (především kvůli nutnosti zakoupit od Teradaty servery, které mají proti běžným databázovým

---

<sup>5</sup>Relational Database Management System

serverům hardwarová specifika). Big Data technologie oproti tomu využívají běžně dostupný hardware. Jednotlivé uzly mohou používat zařízení různých parametrů od různých výrobců - není nutné, aby se jednalo o specifické databázové servery. Dotazování dat distribuovaných databází je často realizováno pomocí MapReduce algoritmu [9]. Principem je zpracování dat na jednotlivých uzlech. Pokud je tedy spolu s objemem uložených dat navyšován i počet uzlů databáze, rychlost komplexního dotazu napříč uzly v zásadě nenarůstá.

Konvenční RDBMS technologie jsou postaveny na jasně definovaném databázovém schématu, vazbách mezi tabulkami, primárních klíčích a indexovaných sloupcích, což vede k velké efektivitě při dotazování strukturovaných dat. Jak již bylo naznačeno, spolu s definicí Big Data se ale rozšířila i definice dat samotných - nyní chápeme data v mnohem širším kontextu než před deseti či patnácti lety. Statické datové schéma v chápání RDBMS systémů již není dostatečně flexibilní pro různorodá data, která chceme zpracovávat. Stejně tak indexované tabulky jsou pro Big Data přežitkem (minimálně na úrovni uložení dat, existují analytické nástroje, například Solr, které indexování používají), indexování je časově velmi nákladná činnost, zvláště v kontextu objemu dat, o kterém se bavíme v rámci Big Data. Distribuované databáze používané v Big Data řešeních jsou nazývány NoSQL (můžeme chápat jako "non SQL", "not relational" nebo "not only SQL"). Data v nich jsou ukládána následujícími způsoby:

- klíč-hodnota struktura (Membase, Voldemort, Riak),
- sloupcová struktura (HBase, BigTable),
- dokumentová struktura (MongoDb),
- grafová struktura (Neo4j).

Kvůli snaze docílit vyšší rychlosti a škálovatelnosti NoSQL databáze (až na výjimky) porušují transakční model ACID<sup>6</sup>. Vlastnosti transakcí popsané modelem ACID jsou zásadní pro RDBMS systémy, které kladou velký důraz na korektní vykonání všech operací - pro aplikace ve finanční oblasti může i jedna nekorektně provedená transakce znamenat velký problém (například ztráta informace o bankovním převodu). Big Data řešení tuto problematiku zcela neopomíjí, dávají ale větší důraz na jiné vlastnosti, v tomto případě rychlost.

Ze silného zaměření NoSQL databází na paralelní zpracování dat vyplývá, že se dle CAP teorému<sup>7</sup> dělí mezi skupiny AP<sup>8</sup> a CP<sup>9</sup>. NoSQL databáze vždy zajišťuje správnou funkčnost databáze napříč jejími oddíly. Skupina CA, tedy databáze konzistentní a stále přístupné, definuje konvenční RDBMS systémy a není v souladu se základním Big Data principem - paralelizace.

---

<sup>6</sup>ACID = Atomicity, Consistency, Isolation, Durability. Akronym popisuje čtyři základní vlastnosti transakcí v databázových systémech.

<sup>7</sup>CAP = Consistency, Availability, Partition Tolerance. Teorém říká, že databáze může splnit pouze dvě ze tří těchto základních vlastností databázových systémů.

<sup>8</sup>Tyto databáze upřednostňují přístupnost databáze před konzistencí dat, do skupiny patří mimo jiné databáze Apache Cassandra, Riak a Voldemort.

<sup>9</sup>Tyto databáze upřednostňují konzistenci dat před přístupností databáze, do této skupiny patří mimo jiné databáze BigTable, HBase a MongoDB.

Výše popsané koncepty (paralelní ukládání a zpracování dat, použití běžně dostupného hardwaru, flexibilní schéma uložení) umožňují Big Data datovým uložištím efektivně pracovat s velkým objemem různých dat, na druhou stranu ale také ukazují jejich nevýhody oproti konvenčním řešením.

### 2.1.2 Hadoop

Jak bylo řečeno v úvodu, Semantic Big Data Historian využívá (v současné době) pro ukládání a zpracování dat framework Apache Hadoop, a proto mu je v této kapitole věnována důkladnější pozornost.

Hadoop - Big Data framework, který poskytuje velice stručně řečeno distribuované uložení a nástroje pro paralelní analýzu dat, vytvořil Doug Cutting, tvůrce Apache Lucene<sup>10</sup>. Hadoop vznikl jako součást projektu Apache Nutch, open source webového vyhledávače, který je sám součástí Apache Lucene. Nutch byl ambiciózní a nákladný projekt, odhadovaná cena pro vyhledávač s indexem čítajícím bilion stránek byla milion dolarů za hardware a 30 tisíc dolarů za měsíc provozu. Cílem projektu bylo vytvořit otevřený vyhledávač a vyhledávací algoritmy. Projekt se po svém odstartování v roce 2002 začal rychle rozrůstat a Doug Cutting si uvědomil, že současná architektura nedovolí potřebné škálování projektu. V roce 2003 publikoval Google zprávu o architektuře sdíleného souborového systému GFS, který využíval.[33] Cutting se rozhodl pro vytvoření open source implementace GFS s názvem Nutch Distributed Filesystem (NDFS), která vyřešila jeho problémy se škálovatelností projektu Nutch z hlediska objemu dat. Krátce poté, co Google publikoval zprávu, ve které světu představil MapReduce[9], implementovali vývojáři MapReduce pro projekt Nutch a během půl roku byly hlavní vyhledávací algoritmy přesunuty na NDFS a MapReduce. NDFS a MapReduce byly postupně vyčleněny nejdříve jako projekt Apache Lucene s názvem Hadoop a následně jako samostatný projekt Apache Hadoop. Ve stejné době také začal Cutting spolupracovat na vývoji Hadoopu s firmou Yahoo!, která v roce 2008 deklarovalo, že Yahoo vyhledávací index je generovaný Hadoopovým clusterem s deseti tisíci uzly. Hadoop následně začaly využívat další společnosti jako Last.fm, Facebook nebo New Yourk Times. Hadoop také získal světový rekord za seřazení terabytu dat (cluster s 910 uzly, výsledný čas 209 sekund).[59]

Hadoop dnes zahrnuje celou rodinu produktů. V této práci jsou popsány HDFS (kapitola 2.1.2.1), MapReduce (kapitola 2.2.1), distribuovaná databáze HBase (kapitola 2.1.2.2) a analytické nástroje Mahout (kapitola 2.2.2) a Hive (kapitola 2.2.3).

#### 2.1.2.1 HDFS

Hadoop Distributed Filesystem je distribuovaný souborový systém, jehož účelem je nahradit kapacitu jednoho fyzického zařízení rozdělením dat napříč mnoha zařízeními (uzly) komunikujícími po počítačové síti. Jeho architektura je navržena pro ukládání souborů obrovské velikosti (stovky megabytů, gigabyty, terabyty), streamovaných, strukturovaných i nestrukturovaných dat za využití běžného hardwaru. Architektura HDFS je optimalizována pro jednorázové nahrání a časté čtení dat. Velký důraz je tedy kladen na rychlost čtení dat (analýzy nad daty vyžadují přečtení velké části, ne-li celého souboru dat) a dávkové zpracování dat. Uživatel může číst a zapisovat data na HDFS pomocí několika rozhraní, typicky

---

<sup>10</sup>Knihovna pro textové vyhledávání, v rámci projektu byl vytvořen i další analytický nástroj - Solr

pomocí rozhraní příkazové řádky (příkaz "hadoop fs cmd", kde "cmd" je POSIXový příkaz) nebo rozhraní pro Javu.[59]

V HDFS jsou data uložena po blocích o velikosti 64 MB, což je řádově více než u běžných souborových systémů. Důvodem pro tuto velikost bloků je snaha minimalizovat čas hledání souborů (nebo částí souborů). Zavedení abstraktních bloků má také další výhody:

- jeden soubor může být větší než je kapacita kteréhokoliv pevného disku v clusteru,
- správa bloků je jednodušší než správa jednotlivých souborů - je možné oddělit metadata souborů od dat a spravovat je separátně,
- bloky mohou být jednoduše replikovány napříč několika jinými fyzickými disky pro případ selhání disku.

Již bylo uvedeno, že HDFS cluster se skládá z uzlů. Tyto uzly se dělí na dva typy - *namenody* a *datanody* - podle vzoru master-worker, přičemž namenode je master, zatímco datanode je worker. Namenody mají na starosti adresářovou strukturu, udržují metadata jednotlivých souborů a informaci o tom, mezi které datanody (a jejich bloky) jsou rozděleny soubory, jejichž metadata udržují. Datanody ukládají a čtou data dle pokynů namenodů nebo klientských požadavků a periodicky informují datanode o svém stavu.

HDFS uzly jsou běžnými hardwarovými zařízeními, s jejich vzrůstajícím počtem se sice zvyšuje efektivita clusteru, ale také pravděpodobnost selhání namenodu nebo datanodů. Vzhledem k tomu, že bez znalosti metadat, která udržuje namenode nelze s daty uloženými na datanodech pracovat, je důležité, aby byla patřičně ošetřena možnost selhání namenodu. Toho může být dosaženo buďto zálohováním dat namenodu nebo přidáním sekundárního namenodu (běžícího na jiném fyzickém zařízení), který je schopen data primárního namenodu zrekonstruovat. Přesto může být v případě výpadku namenodu čas obnovení, respektive uvedení v provoz nového namenodu, relativně dlouhý - cca 30 minut. HDFS proto umožňuje High-Availability mód, který za cenu přidání záložního namenodu snižuje čas výpadku na cca 1 minutu. Pokud aktivní namenode selže, záložní převezme jeho povinnosti. Doba výpadku je potom určena především časem potřebným k detekci selhání aktivního namenodu. Způsob ošetření selhání datanodů je odlišný. HDFS cluster má nastavený replikační faktor (výchozí replikační faktor je tři), který určuje kolikrát je blok replikován na různých nodech. V případě selhání datanodu jsou "ztracené" bloky rekonstruovány z ostatních replik. Replikační faktor může ale sloužit ještě jinému účelu. HDFS umožňuje čtení dat bloku z různých replik, vyšší replikační faktor tedy může navýšit míru paralelizace čtení dat a tedy i rychlost clusteru.[22]

Čtení dat z HDFS klientem (klient může být například uživatelská aplikace nebo jeden z datanodů) probíhá následovně:

1. namenode vrátí pro každý blok souboru adresy všech datanodů, které mají repliku bloku, seznam adres je seřazený podle vzdálenosti mezi datanodem a klientem,
2. bloky souboru jsou čteny vždy z nejbližšího datanodu, pokud při čtení bloku dojde k chybě, je blok čten z dalšího nejbližšího datanodu a datanode, u kterého došlo k chybě čtení je uložen do paměti a další bloky z něj již čteny nejsou.

Klient při čtení komunikuje přímo s jednotlivými datanody, přičemž namenode klientovi nabízí ty nejbližší. Vzdálenost je v tomto případě definována šířkou pásma síťového spojení mezi klientem a datanodem, pořadí od nejbližšího po nejbližší datanode je určeno následovně:

1. čtení dat uvnitř datanodu (datanode je zde v roli klienta i v roli zdroje dat),
2. čtení dat jiného datanodu ve stejném racku,
3. čtení dat datanodu v jiném racku ve stejném datovém centru,
4. čtení dat datanodu z jiného datového centra.

Celý proces čtení je z pohledu klienta nepřerušované čtení jednoho streamu dat.

Zápis dat na HDFS je komplikovanější než čtení (HDFS je optimalizován pro čtení, koncept write-once, read-many-times).

1. Namenode zkontroluje, zda soubor již neexistuje a zda má klient oprávnění nový soubor vytvořit,
2. klient začne ukládat data pomocí streamu,
3. namenode rozhoduje, které datanody budou využity pro uložení replik bloků,
4. zápis bloků na datanody je zřetězený, při replikačním faktoru  $n$  je tedy blok zapsán nejdříve na první datanode, na druhý ... až na  $n$ tý.

Pokud některý z datanodů při zápisu dat selže, je vyřazen z řetězce a nahrazen jiným vhodným datanodem. Zápis dat je úspěšný, pokud jsou bloky zapsány na minimální definovaný počet datanodů (výchozí nastavení je jeden datanode) a po dokončení zápisu jsou bloky asynchronně replikovány na další datanody, dokud není dosažen požadovaný replikační faktor. Vhodné datanody pro ukládání replik bloků jsou vybírány namenodem tak, aby byla vyvážena spolehlivost, rychlost zápisu a rychlost čtení. První replika je uložena na náhodný datanode (je snaha vybírat méně naplněné datanody), druhá a třetí replika jsou uloženy na jiný rack, než ta první, každá na jiný datanode. Další repliky jsou ukládány na náhodné datanody.

Pro efektivní čtení dat je podstatné, aby byly jednotlivé uzly dobře vyvážené. Pokud by byla většina replik uložena jen na několika datanodech, byly by tyto datanody při čtení dat přetížené, zatímco ostatní by zůstaly nevyužívané.

I přes vysokou efektivitu a škálovatelnost HDFS jsou problémy, pro které se použití HDFS příliš nehodí. Obecně je HDFS nevhodný pro:

- aplikace s požadavkem na nízkou latenci (v řádu desítek milisekund). HDFS je optimalizovaný na velkou propustnost dat, tím se ale také zvyšuje latence. V případě potřeby nízké latence je lépe využít jiné uložení, například HBase,
- ukládání velkého množství malých souborů. Omezením je zde především operační paměť namenodů, které udržují v paměti všechna metadata souborů.



### 2.1.2.2 HBase

HBase je distribuovaná sloupcově orientovaná databáze využívající HDFS jako souborový systém pro ukládání dat.<sup>11</sup> Patří do skupiny NoSQL databází a je vhodná především pro real-timeové zpracování dat a umožňuje náhodný přístup při zápisu a čtení velkých objemů dat. [39]

Data jsou v HBase ukládána do tabulek, které jsou na úrovni buněk verzovány podle časového otisku v době vytvoření záznamu. Obsahem buněk i id řádků tabulek je pole bytů - může jít v podstatě o jakákoliv data. Řádky tabulek jsou řazeny pomocí id řádků, které jsou zároveň primárním klíčem tabulky a všechny přístupy do tabulek jsou právě přes id řádků. I transakce jsou v HBase atomické na úrovni řádků.

Sloupce tabulek jsou rozděleny do *column families*, které jsou definovány ve schématu tabulky a kde všechny sloupce mají stejný prefix. Fyzicky jsou potom všechny sloupce jedné column family uloženy společně v rámci souborového systému. I nároky na uložení dat a další parametry jsou definovány na úrovni column family. Je tedy vhodné, aby ke všem sloupcům column family bylo přistupováno stejným způsobem a z hlediska objemu měly podobnou charakteristiku. Column families mohou být tedy v jistém smyslu chápány jako vertikální oddíly tabulky.

Tabulky jsou podle rozsahu id řádků horizontálně rozděleny do oddílů zvaných *regions*. Region je základní distribuovaná entita (stejně jako u HDFS je to blok). Z toho vyplývá, že výhody paralelního přístupu k datům v HBase se neprojeví u tabulek, které mají pouze jeden region - tabulky s malým počtem záznamů.

Architektura HBase je postavena na pracujících a koordinujících uzlech - můžeme chápat jako *master-slave* architekturu. *Master node* řídí cluster obsahující jeden nebo více *regionserverů*, přiděluje nové regiony regionserverům a vypořádává se s jejich výpadky. Regionservery mají přiděleno nula až n regionů, zpracovávají klientské požadavky na čtení a zápis dat a informují master node v případě, že se některý z přidělených regionů rozdělí na dva - master node potom rozhodne, kterému regionserveru bude nový region přidělen.[7]

HBase využívá aplikaci ZooKeeper<sup>12</sup> pro konfiguraci svých serverů, ZooKeeper mimo jiné udržuje adresu aktuálního mastera clusteru a všech regionserverů. Přiřazování regionů regionserverům i řešení výpadků regionserverů je také zprostředkováno pomocí ZooKeeperu.

Vzhledem k tomu, že HBase vychází z HDFS, je snaha aby stejně řešené problémy byly v HBase implementovány stejným způsobem jako je tomu v HDFS. Od tohoto modelu se HBase odklání pouze v případech, kdy je její chování specifické - rozdílné oproti HDFS. Nejdůležitější z těchto rozdílů jsou uvedeny v této kapitole.

## 2.2 Analytické nástroje

V předchozí kapitole jsou uvedené způsoby, jakými je možné efektivně ukládat data, která považujeme pro jejich velký objem, rychlost a různorodost za Big Data. Samotná datová ulo-

<sup>11</sup>HDFS není jediný souborový systém, který HBase může používat, je ale používán nejčastěji. Alternativami mohou být například Amazon S3 nebo KFS.

<sup>12</sup>ZooKeeper je taktéž podprojektem Hadoopu, slouží ve jménu zajištění konzistence konfigurace serverů napříč distribuovaným systémem

žičtě ale nejsou dostatečným nástrojem pro komplexní zpracování dat. Potřebujeme nástroje, které nám umožní uložená data analyzovat a získat tak z dat potřebné znalosti.

Nástrojů pro analýzu Big Data je mnoho a cílem této práce je popsat pouze ty z nich, které jsou využívány (nebo je jejich využití uvažováno) ve frameworku Semantic Big Data Historian.

### 2.2.1 MapReduce

MapReduce je programovací model pro zpracování dat, který může být implementován v mnoha programovacích jazycích. Stručná historie MapReduce je popsána v 2.1.2.

Cílem MapReduce je efektivní analýza dat díky paralelnímu zpracování. Vstupní množina dat je rozdělena na menší části stejné velikosti a každá část je potom zpracovávána samostatným procesem. Tím je zajištěno, že všechny procesy trvají přibližně stejně dlouho<sup>13</sup>. Po skončení procesů zpracovávajících jednotlivé části vstupní množiny dat je možné výsledky těchto procesů zkombinovat a vypočítat celkový výsledek. Zároveň musí být zajištěno, že pokud některý z procesů zpracovávajících část množiny dat selže, bude situace vyřešena - v opačném případě by celkový výsledek nemusel být správný, nebyl by totiž vypočítán z celé množiny dat, ale pouze z její části. MapReduce model tedy řeší následující problémy:

- řízení celého výpočtu a řešení případných selhání procesů,
- rozdělení vstupní množiny dat,
- zpracování jednotlivých částí množiny dat - fáze *Map*,
- zkombinování dílčích výsledků - fáze *Shuffle*,
- vypočítání celkového výsledku - fáze *Reduce*.

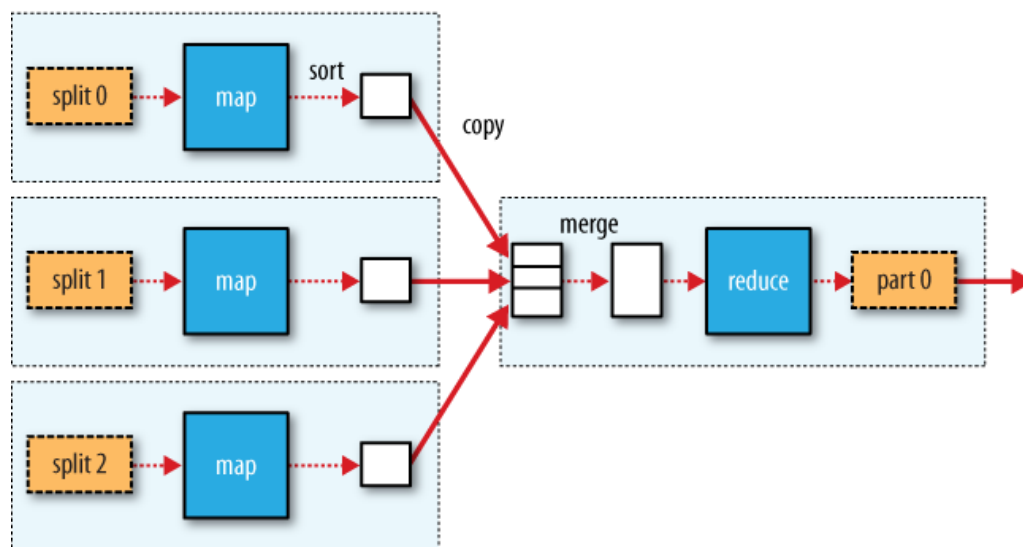
V této práci je popsána implementace MapReduce frameworkem Hadoop.

Konkrétní analýza pomocí MapReduce je nazývána *job* a skládá se ze vstupních dat jobu, samotného MapReduce programu a konfiguračních parametrů. Job je rozdělen na *tasky* dvou typů - *map* tasky a *reduce* tasky. Průběh jobu je kontrolován dvěma typy uzlů - jedním *jobtrackerem* a mnoha *tasktrackery*. Jobtracker koordinuje běh MapReduce jobu a plánuje spouštění jednotlivých tasků na *tasktrackerech*. Tasktracker spouští tasky a o jejich průběhu a výsledcích průběžně informuje jobtracker, který v případě selhání tasktrackeru přiřadí jeho tasky jinému tasktrackeru. Tím je zajištěn hladký průběh paralelního zpracování dat i v případě selhání některých jeho částí.[18]

Vstupní množinu dat MapReduce rozděluje na menší části stejné velikosti. Tyto části jsou zpracovávány samostatně pomocí *map* tasků. Velikost těchto částí bývá stejná jako je velikost bloku HDFS, nicméně je možné ji nastavit jinak. Na každou část množiny dat je vytvořen *map* task, pokud je tedy velikost určena jako příliš malá, tak je vytvořeno velké množství *map* tasků. Tím narůstá časová náročnost režie analýzy. Na druhé straně, pokud je velikost určena jako příliš velká, nemusí být zcela využit potenciál paralelního zpracování dat.

---

<sup>13</sup>Každý proces zpracovává data o stejné velikosti, případný rozdíl v rychlosti zpracování je tedy dán hardwarovou specifikací zařízení, na kterém je proces spuštěn.



Obrázek 2.1: Vizualizace průběhu MapReduce jobu s jedním reduce taskem [59]

MapReduce je navržen tak, aby většina výpočtu byla vykonána na místě, kde jsou uložena data - uplatňuje se princip datové lokality.[20] Z toho důvodu jsou map tasky přiřazeny jobtrackerem na uzly, kde je uložena replika bloku, který obsahuje část množiny dat určenou pro daný map task. Může ovšem nastat situace, že uzly, na kterých jsou všechny tři<sup>14</sup> repliky tohoto bloku, již vykonávají jiný map task. Potom se hledá jiný uzel, který může map task provést. V takovém případě je porušen princip datové lokality a task je pomalejší - nejdříve je nutné přenést data na uzel, kde bude výpočet probíhat. Tato úzká souvislost alokace map tasků s velikostí HDFS bloků je dalším důvodem, proč by měla být vstupní množina dat MapReduce jobu rozdělena na části stejně velké, jako je velikost HDFS bloku.[17]

Úkolem map tasku je zpracování části vstupní množiny dat tak, aby výsledné zpracování dat reduce taskem bylo co možná nejjednodušší. Výsledkem map tasku jsou páry klíč-hodnota. Po vykonání všech map tasků jsou tyto páry seříděny podle klíčů a předány reduce tasku. Reduce task potom může na základě výsledků všech map tasků spočítat požadovaný výsledek. Průběh MapReduce jobu s jedním reduce taskem je zobrazen na obrázku 2.1.

Reduce task nedodrží princip datové lokality. V případě užití jednoho reduce tasku jsou výsledky všech map tasků přesunuty na uzel<sup>15</sup>, na kterém běží reduce task. Výsledek reduce tasku - tedy výsledek celého MapReduce jobu je zapsán na HDFS<sup>16</sup>.

Reduce tasků nicméně může být nastaveno více. Jejich počet se neodvíjí od velikosti vstupních dat, ale je nastaven v rámci konfigurace MapReduce jobu. V případě použití více reduce tasků jsou výsledky map tasků rozděleny do tolika oddílů, kolik je definovaných

<sup>14</sup>Tři je výchozí hodnota replikačního faktoru HDFS, může být nastaveno jinak.

<sup>15</sup>Výsledky map tasků nejsou ukládány na HDFS, jak by se možná dalo očekávat, ale na běžný souborový systém. Tato data totiž není nutné udržovat v několika replikách - nepřineslo by to žádné zrychlení MapReduce jobu, spíše by to byla zbytečná zátěž pro HDFS cluster.

<sup>16</sup>Adresář pro zapsání výsledků MapReduce jobu se obvykle specifikuje při spouštění jobu

reduce tasků. Páry se stejným klíčem jsou ale vždy součástí jednoho oddílu. Při nastavení více reduce tasků je důležité brát ohled na optimalizaci rozdělení výsledku map tasků a jejich setřídění (*shuffle* fáze), protože se vzrůstajícím počtem reduce tasků narůstá složitost a časová náročnost těchto fází jobu. V případě, že je možné výsledek celého jobu určit pouze pomocí map tasků, je také možné nastavit počet reduce tasků na nula.[59]

### 2.2.1.1 Výhody a nevýhody MapReduce

MapReduce joby jsou velice efektivním způsobem analýzy velkého objemu dat. Uživatel má možnost definovat všechny atributy MapReduce jobu a tím analýzu optimalizovat z hlediska rychlosti a využití zdrojů. Psaní MapReduce programů je ale zpravidla složitější a časově náročnější než běžné způsoby analýzy dat a proto není příliš vhodný pro jednorázové analýzy. Z tohoto důvodu bývají často využívány analytické nástroje, které sice MapReduce využívají, ale nevyžadují psaní vlastních MapReduce programů - například Hive.

### 2.2.2 Mahout

Mahout je knihovna škálovatelných algoritmů z oblasti strojového učení a vytěžování dat. Algoritmy jsou implementovány nad frameworkem Apache Hadoop<sup>17</sup> a využívají MapReduce výpočetní model. Mahout poskytuje implementaci mnoha algoritmů<sup>18</sup>, přičemž některé z nich jsou určeny pro lokální použití a některé jsou určeny pro použití v distribuovaném módu - s použitím Hadoopu. Algoritmy, které implementuje Mahout, se dělí podle svého účelu do čtyř hlavních skupin:

- *Collaborative filtering* - analýza chování uživatelů a vytváření doporučení pro produkty,
- *Clustering* - rozdělení souboru dat na třídy podle vlastností jednotlivých instancí,
- *Classification* - přiřazování instancí do existujících kategorií na základě znalosti daných kategorií,
- *Frequent itemset mining* - hledání nejčastějších skupin instancí.

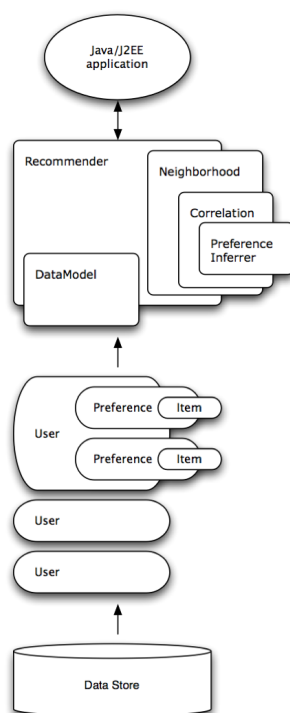
Architektura jednotlivých algoritmů je obdobná. Liší se především algoritmy samotné z pohledu disciplín vytěžování dat a strojového učení.[10] Architektura Mahout je ukázána na algoritmu *Recommender*, abstraktním jádru skupiny algoritmů *collaborative filtering*.

*Recommender* vytváří doporučení pro produkty na základě voleb uživatelů. Algoritmus tedy potřebuje dva soubory dat - uživatelské volby a produkty. Například knihkupectví může na základě předešlých transakcí analyzovat, jaké knihy je vhodné nabídnout určitým zákazníkům. Samotný *Recommender* je implementací abstraktního jádra algoritmu a jeho

---

<sup>17</sup>Pro účely této práce je Mahout popisován jako knihovna primárně využívající Hadoop - čím Mahout stále do jisté míry je. Vývoj Mahoutu se ale stále více zaměřuje směrem k využívání jiných distribuovaných frameworků pro Big Data, především Apache Spark.[43] V tomto kontextu je rozdíl především v tom, že SPARK nevyužívá Hadoop MapReduce pro analýzu dat. Na druhou stranu může využívat jiné Hadoop technologie jako například HDFS nebo YARN.[45]

<sup>18</sup>Seznam implementovaných algoritmů je dostupný na [40].



Obrázek 2.2: Vizualizace architektury aplikace s využitím Mahout recommender [41]

využití vyžaduje specifikaci algoritmu<sup>19</sup> ve smyslu implementace předdefinovaných rozhraní či abstraktních tříd.[41]

Algoritmus pracuje s datovým modelem jako rozhraním pro načtení potřebných souborů dat. Implementací datového modelu může být teoreticky jakýkoliv zdroj, nejčastěji se ale jedná o databázový systém - ať už se jedná o RDBMS nebo NoSQL databáze. Mahout vyžaduje v rámci datového modelu identifikaci uživatelů a produktů pomocí číselného primárního klíče a specifikaci vazeb mezi uživateli a produkty ve smyslu preferencí produktů uživatelem. Tato preference může být vyjádřena numericky podle míry preference (tzv. *měkký* identifikátor) nebo může být pouze specifikováno, zda uživatel daný produkt preferuje, či nikoliv (tzv. *tvrdý* identifikátor). Použití měkkých a tvrdých identifikátorů je možné zvolit téměř u všech algoritmů, které Mahout implementuje.

Výsledná doporučení jsou vytvořena na základě datového modelu a konkrétní implementace algoritmu. Tato architektura je popsána obrázkem 2.2.

### 2.2.2.1 Výhody a nevýhody Mahout

Mahout přináší zásadní výhody z hlediska škálovatelnosti analytických algoritmů vzhledem k objemu dat. Stejně jako mnoho jiných analytických nástrojů (například KNIME nebo

<sup>19</sup> Alternativně mohou být použita již vytvořená implementace. Toto řešení je jednodušší, nicméně nemusí být vyhovující z hlediska použití pro konkrétní problém.

RapidMiner[32]) nabízí možnost použití existujících implementací algoritmů z oblasti vytěžování dat a strojového učení. Navíc ale splňuje požadavky pro paralelní zpracování dat pomocí Hadoop MapReduce. Implementace algoritmů je ale vázána na konkrétní platformy a framework (Hadoop, Spark,...), což v některých případech nemusí být žádoucí.

### 2.2.3 Hive

Hive je frameworkem pro datové sklady vystavěné na základě Apache Hadoop. Cílem Hivu je umožnit analýzu dat uložených v HDFS<sup>20</sup> clusteru pomocí zavedeného nástroje typického pro analýzu dat relačních databází - SQL. Tím se výrazně rozšiřuje okruh analytiků, kteří mohou Hadoop a data uložená v HDFS využívat. Většina analytiků má dnes silné znalosti SQL, ale mnohem menší část z nich má programovací schopnosti odpovídající tomu, aby byli schopni efektivně psát pro své analýzy MapReduce programy. Hive umožňuje analyzovat data uložená v HDFS pomocí dotazů jazyka Hive QL<sup>21</sup>[49] (vlastní implementace dotazovacího jazyka SQL), a tyto dotazy následně převádí na sérii MapReduce jobů. Hive QL sice není vhodným jazykem pro psaní komplikovaných algoritmů, například z oblasti strojového učení, pro většinu běžných analýz je ale naprosto dostačujícím prostředkem.

Data jsou v Hivu uložena v tabulkách, přičemž jsou striktně oddělena metadata tabulek a data samotná. Metadata (například databázové schéma) jsou ukládána do relační databáze, typicky Derby [38] nebo MySQL, použitá ale může být téměř jakákoliv relační databáze. Metadatové uložení se nazývá *metastore* stejně jako služba, která metadata čte a zapisuje. Metastore může být používán v několika různých módech.[59]

- *Embedded metastore* je výchozím způsobem použití metastore v Hive. Služba metastore, stejně jako samotné uložení jsou spuštěny ve stejném procesu jako samotný Hive. Z tohoto důvodu může v tomto módu vzniknout pouze jedna session do metastore databáze, Hive tedy může využívat současně pouze jeden uživatel. V tomto případě je vždy použita databáze Derby.
- *Local metastore* už umožňuje vytvoření několika session do metastore databáze a tím pádem současné používání Hivu několika uživateli. Rozdíl oproti embedded metastore je v tom, že metastore databáze již není spouštěna stejným procesem jako Hive, ale samostatně - ať už na stejném zařízení, nebo vzdáleně. Metastore služba je ale stále součástí stejného procesu jako Hive. V případech, kde je metastore databáze instalována a spouštěna samostatně je často využívána MySQL databáze.
- *Remote metastore* jde ještě o krok dále, zde jsou i metastore služby spouštěny samostatně - pomocí samostatného metastore serveru a oddělené JVM. Tato architektura přináší výhody především v zabezpečení metastore databází, pro samotný Hive ale nepřináší velkou změnou oproti local metastore.

Samotné tabulky potom mohou být také dvou typů.

---

<sup>20</sup>Hive umožňuje použití většiny lokálních i distribuovaných souborových systémů pro ukládání dat, využití HDFS je ale nejčastější.

<sup>21</sup>Hive Query Language

- *Managed* tabulky jsou tabulky vytvořené Hivem a v souborovém systému jsou ukládány do adresáře *warehouse*.
- *External* tabulky jsou již existující soubory v souborovém systému, při jejich definici Hive pouze ukládá metadata tabulky.

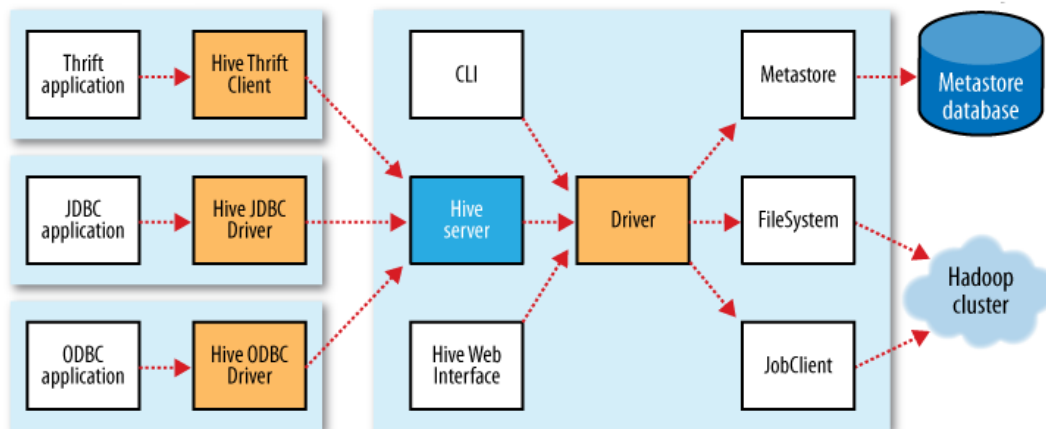
Chování obou typů tabulek je ve většině případů stejné. Rozdíl je například u operace *delete*, kdy u externí tabulky jsou mazána pouze metadata a samotný soubor v souborovém systému není operací nijak změněn. V obou případech jsou data uložena v souborovém systému (často HDFS) a dotazována jsou pomocí MapReduce jobů. MapReduce joby jsou sestaveny podle Hive QL dotazu pomocí *SQL-to-MapReduce* překladače. Překladač má zpravidla předdefinované MapReduce programy, které odpovídají jednotlivým operacím Hive QL dotazu - například *select*, *join*, *agregace*. Z těchto předdefinovaných programů poskládá sérii MapReduce jobů, které odpovídají původnímu Hive QL dotazu a vypočítají požadovaný výsledek. Nevýhodou tohoto přístupu je, že pokud je Hive QL dotaz komplikovanější, tak je vytvořena série mnoha MapReduce jobů a tím může celá analýza trvat řádově delší čas, než pokud by byly MapReduce joby psány ručně. To, co je schopn programátor vyřešit jedním MapReduce jobem, může Hive rozdělit do několika jobů, protože joby jsou přesně mapované na operace definované v Hive QL.[24]

Tabulky jsou v Hivu rozdělovány na oddíly podle hodnoty sloupců, které jsou definovány při vytvoření tabulky klauzulí *partition by*. Všechny záznamy tabulky pro danou hodnotu sloupce takto označeného jsou uloženy do stejného oddílu. Na úrovni souborového systému jsou oddíly reprezentovány jako podadresáře hlavního adresáře tabulky. Například pokud by byla tabulka do oddílů rozdělena podle datumu a Hive QL dotaz by vybíral pouze záznamy s určitým datumem (nebo s rozmezím datumů), byla by načítána pouze data z odpovídajících podadresářů. Tímto způsobem lze výrazně optimalizovat rychlost Hive QL dotazů.

Architektura Hivu je klient-server<sup>22</sup> a Hive poskytuje několik možností, jak může klient se serverem komunikovat. Komunikace Hive klientů s Hive serverem je popsána na obrázku 2.3.

- *Thrift* klient komunikuje s Hive serverem pomocí Apache Thrift frameworku.[44]
- *ODBC* ovladač umožňuje připojit se na Hive server pomocí ODBC protokolu. ODBC ovladač využívá Thrift klienta.
- *JDBC* ovladač umožňuje připojit se na Hive server pomocí JDBC protokolu. JDBC ovladač využívá Thrift klienta.
- *CLI* - rozhraní příkazové řádky operačního systému. Při využívání CLI aplikace není dodržena architektura klient-server, spouští se pouze jedna instance celé aplikace.
- *HWI* - webové rozhraní Hivu. Při využívání HWI aplikace není dodržena architektura klient-server, spustí se pouze jedna instance celé aplikace.

<sup>22</sup>Pro spuštění Hive serveru je potřeba spustit službu *hiveserver* nebo *hiveserver2*.



Obrázek 2.3: Vizualizace připojení Hive klientů na Hive server [59]

### 2.2.3.1 Výhody a nevýhody Hive

Hive je velkým přínosem pro využitelnost frameworku Apache Hadoop (nejen) v komerčním prostředí - umožňuje uživatelům analyzovat data ukládaná v HDFS pomocí všeobecně známého a široce rozšířeného nástroje - SQL. Na druhou stranu může být analýza dat pomocí Hive výrazně pomalejší než analýza pomocí uživatelsky vytvořených MapReduce programů.

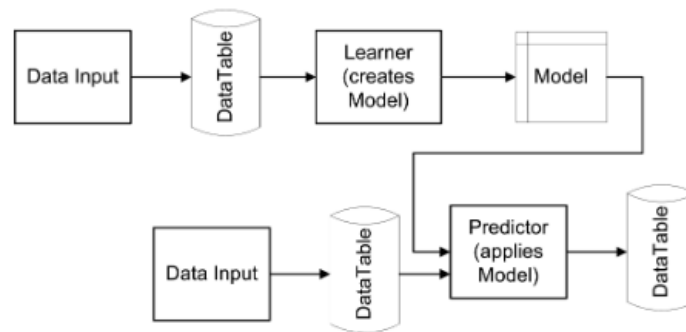
### 2.2.4 KNIME

KNIME je modulární analytický nástroj, který vznikl jako výzkumný projekt na Kostnické univerzitě. Založený je na propojování jednotlivých algoritmů pomocí *workflow*. Workflow jsou sestavena z uzlů pro čtení, zpracování, vizualizaci a ukládání dat - uzly jsou tedy základními stavebními bloky každé KNIME analýzy. Sám uzel může být (a často také je) implementací nějakého algoritmu (například uzel *Decision Tree Learner*). KNIME je koncipován primárně jako nástroj pro tvorbu analytických workflow pomocí grafického editoru. Přesto je možné definovat uzly pomocí zdrojového kódu. Cílem je, aby byly algoritmy implementovány za pomoci existujících uzlů, které jsou pomocí aplikace graficky spojeny ve workflow. KNIME workflow tedy mohou upravovat a vytvářet i analytici bez hlubokých znalostí programování a SQL. Během lét vývoje projektu<sup>23</sup> byl KNIME využíván v mnoha různých oborech a součástí KNIMU jsou tedy dnes rozsáhlé knihovny implementovaných algoritmů z oborů jako jsou například vytěžování dat, strojové učení, statistika, integrace dat, chemie a mnoha dalších. Jednou z těchto knihoven je knihovna *KNIME Big Data Extensions* [21], která umožňuje napojení KNIME na Big Data nástroje a uložště (například HBase, Hive).

Architektura aplikace KNIME je přizpůsobena třem základním principům.[4]

<sup>23</sup>Vývoj KNIME započal roku 2004.





Obrázek 2.4: Schéma datového toku KNIME workflow [4]

- Vizuální framework: je snaha, aby všechny operace při tvorbě nebo úpravě workflow bylo možné vykonat pomocí grafického editoru workflow.
- Modularita: jednotlivé uzly a datové kontejnery by na sobě měly být navzájem nezávislé tak, aby bylo možné je navzájem kombinovat při tvorbě workflow.
- Rozšiřitelnost: aplikace je rozšiřitelná o nové uzly a algoritmy pomocí pluginů. Způsob tvorby nových uzlů/pluginů je přesně specifikován.

Pro naplnění těchto principů jsou KNIME workflow implementována jako zřetězení uzlů navzájem propojených hranami, které mezi uzly přenášejí data nebo modely. Každý uzel zpracovává data a/nebo modely, které dostane přes vstupní porty, a na svůj výstupní port předává zpracovaná data nebo vytvořený model. Většina uzlů také podporuje možnost vizualizace svého výstupu. Příklad schématu KNIME workflow je na obrázku 2.4.

Data předávaná mezi uzly jsou ve formátu tabulky, která kromě dat samotných uchovává také metadata tabulky. Data mohou být čtena pouze sekvenčně - náhodný přístup pomocí primárního klíče není podporovaný z důvodu zachování rychlosti čtení tabulky i při velkém objemu dat. Pokud je tabulka příliš velká na to, aby byla zpracovávána v paměti, ukládá KNIME části tabulky na pevný disk a v případě potřeby je opět načítá do paměti. Ukládání dat na pevný disk a opětovné načítání do paměti je řízeno algoritmem pro správu mezipaměti.

Uzly mají definované vstupní a výstupní porty, přes které jsou spojené hranami s dalšími uzly a pomocí kterých si navzájem předávají data a/nebo modely. Workflow je potom acyklickým grafem tvořeným uzly a hranami. Workflow si udržuje informaci o stavu jednotlivých uzlů a o tom, kdy a s jakými vstupními daty mají být uzly spuštěny. Díky tomuto je možné spouštět uzly nebo celé cesty v grafu workflow nezávisle na zbytku grafu v samostatných vláknecích.[4, 34]

KNIME umožňuje také zanořování částí workflow pomocí takzvaných *metanodes*. Metanody se v rámci workflow chovají stejně jako běžné uzly s tou výjimkou, že sami mohou obsahovat acyklický graf uzlů a hran - jinými slovy metanody mohou obsahovat vnořené workflow. Metanody samotné je také možné zanořovat. Díky tomu je možné vytvářet komplikovanější workflow obsahující například cykly.

Ačkoliv architektura KNIME není primárně určena pro distribuované zpracování dat, je možné toho v některých situacích docílit. Již bylo uvedeno, že je možné docílit paralelního zpracování dat pomocí KNIME workflow v různých vláknech jednoho procesu, a to na následujících úrovních:

- paralelní zpracování nezávislých uzlů,
- paralelní zpracování dat jednoho uzlu,
- paralelní zpracování částí workflow - metanodů.

Metanody mohou být spouštěny nejen paralelně, ale za jistých podmínek také distribuovaně. Typickým příkladem možného využití distribuovaného spouštění metanodů mohou být například cykly, ve kterých je metanode spouštěn opakovaně (s různými vstupními daty). Distribuovaným spouštěním metanodů je možné docílit až osminásobného zrychlení při využití deseti pracovních stanic. Případů, kdy je možné a výhodné takovýto přístup zvolit ale není mnoho. Distribuované spouštění metanodů musí být konfigurováno explicitně.[34]

#### 2.2.4.1 Výhody a nevýhody KNIME

KNIME je oproti ostatním uvedeným analytickým nástrojům jediný, který může být využíván i uživateli bez znalostí programování nebo SQL. Díky otevřenosti frameworku obsahuje velké množství knihoven a implementovaných algoritmů, řádově více než například Mahout. Díky tomu je vhodný prakticky pro jakékoliv analýzy dat. Na druhou stranu je také jediným analytickým nástrojem popsaným v této práci, který není zaměřený na distribuované a paralelní zpracování dat. Je sice možné těchto přístupů v některých případech docílit, ale jedná se spíše o speciální případy.

## 2.3 Ontologie

V pojetí počítačových věd termín *ontologie* a s ním spojené koncepty definují jeden z možných pohledů přístupu k datům - pohled na data z hlediska jejich významu (sémantiky). Termín samotný se ale objevuje mnohem dříve v jiných disciplínách. Z pohledu filozofie, ze které je termín převzatý, je ontologie oborem zabývající se jsouncem a popisem reality, a jako taková je známa již od dob antiky. V počítačových vědách se termín objevuje od 70. let minulého století. Zde je definice ontologie sice rozdílná od té známé z filozofie, přesto má ale filozofické chápání ontologie přesah do počítačových věd. V roce 1993 definoval Thomas Gruber ontologii následovně<sup>24</sup>: "Ontologie je explicitní specifikace konceptualizace.".[14] Tato definice je potom rozšířena o další vlastnosti[5] a v roce 1998 shrnuta do následující formulace<sup>25</sup>: "Ontologie je formální, explicitní specifikace sdílené konceptualizace.".[35] Pro pochopení této definice je třeba rozumět jednotlivým použitým termínům.

*Konceptualizace* je abstraktním, zjednodušeným pohledem na část reálného světa (domény), který si přejeme zachytit za určitým účelem.[14] Úroveň detailu, kterou zvolíme pro

---

<sup>24</sup>V originálním znění: "Ontology is an explicit specification of conceptualization."

<sup>25</sup>V originálním znění: "An ontology is a formal, explicit specification of a shared conceptualization."

vytvoření takového pohledu, rozhoduje o tom, zda je konceptualizace obecně platná, nebo zda se mění se změnami popisované domény. Konceptualizace vždy zachycuje pouze část domény s danou úrovní detailu. Z matematického pohledu může být konceptualizace popsána jako množina objektů domény a množina relací, kde relace mohou být unární nebo binární.[13]

Tyto elementy konceptualizace musí být zachyceny jazykem, který je definuje a popisuje. Přitom ale musí být zajištěno, že výklad symbolů takového jazyka bude interpretován vždy stejným způsobem.<sup>26</sup> Z tohoto důvodu musí být všechny elementy konceptualizace (objekty a relace) *explicitně* definovány. Toho může být dosaženo dvěma způsoby. Jedním z nich je definování elementu pomocí výčtu všech možných hodnot. Tento přístup může být vhodný pro některé elementy, zatímco pro jiné může být vysoce neefektivní. Zatímco definice pohlaví člověka pomocí výčtu hodnot je velice triviální - muž/žena<sup>27</sup>, tak definice automobilu je značně problematická - bylo by zapotřebí sestavit seznam všech vyrobených automobilů. Alternativním způsobem definování elementů konceptualizace je definice pomocí axiomů - významových postulátů.[8] Budeme-li se držet příkladu s automobilem, můžeme automobil definovat jako stroj se čtyřmi koly poháněný motorem. U relací konceptualizace takto můžeme například definovat zda jsou symetrické, reflexivní a/nebo transitivní. Ontologie je souborem takovýchto axiomů a výčtových definic. Dle definice musí být jazyk popisující konceptualizace také *formální* - tedy strojově zpracovatelný. Nemůže se tedy jednat například o přirozený jazyk.[37]

Při dodržení výše zmíněných konceptů je ontologie formální, explicitní specifikací konceptualizace, což je v souladu například s definicí ontologie podle Grubera[14]. S postupem času se ale ukázalo, že sdílení ontologií (nebo alespoň jejich částí, například významových postulátů) je jedním ze základních požadavků a přínosů tohoto přístupu ke zpracování dat. Ontologie jsou sice formální a explicitní specifikací (jazyk popisující ontologii je jednoznačný a strojově čitelný), pokud jsou ale stejné objekty reálného světa definovány v různých ontologiích různě, je kombinace těchto ontologií komplikovaná. Z toho důvodu je žádoucí například používání standardizovaných významových postulátů pro základní entity. Pro pochopení konceptu sdílených ontologií je také důležité znát motivaci, která vede k jejich vytváření. Ta může být následující:[28]

- Sdílení obecného porozumění a struktury informací mezi lidmi nebo softwarovými agenty je jedním z "vyšších cílů" tvorby ontologií. Například pokud několik organizací používá a publikuje jednotnou základní ontologii, je možné znalosti těchto organizací jednoduše strojově kombinovat, analyzovat a porovnávat.
- Umožnění znovupoužitelnosti znalosti domény. Mnoho doménových modelů (v našem kontextu konceptualizací) sdílí stejné části. Díky možnosti kombinace mnoha ontologií do jedné globální je možné tento problém elegantně vyřešit. Nejen, že je ušetřen čas, který by byl stráven modelováním části domény, jejíž konceptualizace již existuje, navíc tento přístup přináší jednotný pohled na danou část domény v rámci různých modelů.

<sup>26</sup>Například při použití přirozeného jazyka může být význam jednotlivých slov interpretován různě rodilým mluvčím jazyka a cizincem se základní znalostí tohoto jazyka. Stejně tak jedna relace může být chápána různě v závislosti na doméně.

<sup>27</sup>Při výkladu dle zákonů České Republiky.

- Explicitní definice doménových znalostí a předpokladů může sloužit jako základ implementace aplikací. Pokud jsou doménové předpoklady definovány například ve zdrojovém kódu aplikace, je velice obtížné je dohledat či změnit. Především pro uživatele bez programovacích schopností. Explicitní definice těchto předpokladů může být také užitečná pro nové uživatele seznamující se s doménou.
- Oddělení doménových znalostí od operativních. Například definice parametrů produktu může být oddělená od algoritmu, který parametry produktu přiděluje. Stejně tak tento algoritmus může být nezávislý na produktu samotném. Pokud změníme ontologii, může stejný algoritmus nastavovat parametry naprosto rozdílných produktů.

### 2.3.1 Formální reprezentace

Ontologie musí být pro účely automatického zpracování definovány formálně. Již byly naznačeny požadavky na formální jazyk, kterým by měla být ontologie definována. Existuje několik (skupin) formálních jazyků, které jsou pro tyto účely více či méně vhodné, mezi nimi například:[30]

- Rámcově orientované modely. Tento koncept využívá struktury rámců jako entit domény a slotů jako jejich vlastností. Podstatnou vlastností konceptu je možnost dědičnosti rámců. Příkladem implementace rámcově orientovaného modelu je například Open Knowledge Base Connectivity.[6]
- Sémantické sítě jsou grafem, kde vrcholy reprezentují entity domény a hrany relace mezi nimi. Relace mezi entitami mohou být následující: je synonymum, je antonymum, je součástí (a inverzní relace), je typem (a inverzní relace). Sémantické sítě byly vytvořeny za záměrem překladu přirozených jazyků, typickým příkladem je WordNet.[25]
- Konceptuální grafy jsou rozšířením sémantické sítě. Skládají se z tříd, relací, individuí a kvantifikátorů. Oproti sémantickým sítím mají přímý překlad do predikátové logiky, jejíž sémantiku využívají. Díky tomu je vyjadřovací schopnost konceptuálních grafů teoreticky stejná jako vyjadřovací schopnost predikátové logiky.[36]
- Knowledge Interchange Format (KIF) je jazykem sémanticky založeným na predikátové logice a syntakticky na programovacím jazyce LISP[46]. Jeho zamýšleným cílem měly být především transformace a integrace ontologií, je ale také často využíván pro přímou specifikaci ontologií.[12]
- Common Logic je frameworkem pro rodinu ontologických jazyků založených na logice s cílem standardizace syntaxe a sémantiky ontologických jazyků. Do skupiny patří tři jazyky: CLIF (Common Logic Interchange Format), CGIF - Conceptual Graph Interchange Format a XCL /eXtended Common Logic Markup Language), všechny tři standardizované ISO normou [1]. Díky standardizaci jsou jazyky mezi sebou navzájem jednoduše přeložitelné.
- Deskriptivní logika je logika sloužící primárně pro formální popis konceptů a rolí (relací). Různé implementace deskriptivní logiky vznikly za účelem formalizace sémantických sítí a rámcově orientovaných modelů. Sémanticky jsou založeny na predikátové

logice, zatímco syntaxe je uzpůsobená pro potřeby praktického modelování a výpočetní efektivitu.[3]

Tato kapitola dále popisuje právě formální specifikaci ontologie pomocí deskriptivní logiky, protože právě ta je využívána frameworkem Semantic Big Data Historian.

Jazyky založené na deskriptivní logice se skládají ze dvou komponent - *TBox*ů a *ABox*ů. *TBoxy* popisují terminologii - tedy koncepty a role (relace) ontologie, zatímco *ABoxy* definují přiřazení jednotlivých instancí. Koncepty popisují množinu instancí a role popisují vztahy mezi těmito instancemi. Sémantika těchto jazyků odpovídá sémantice predikátové logiky, zatímco syntaxe je odlišná. Je nicméně přímo mapovatelná na syntaxi predikátové logiky - koncepty odpovídají unárním predikátům a role binárním predikátům.[11]

Jedním z hlavních důvodů pro nutnost formální specifikace ontologie je možnost dedukce znalostí - tedy vyvozování informací, které nejsou ontologií explicitně definovány. Formální reprezentace ontologie pomocí deskriptivní logiky je koncipována s ohledem na možnost automatického zpracování. Z důvodů výpočetní náročnosti nebo například úrovně detailu ontologie ovšem není dedukce možná vždy. Příkladem dedukce znalostí mohou být:[30]

- určení splnitelnosti konceptu (Může existovat individuum, které může být instancí daného konceptu?),
- určení podmnožin konceptů (Je jeden koncept podmnožinou druhého?),
- určení konzistence ontologie (Neporušují instance *ABox* specifikaci definovanou v *TBox*?),
- určení všech konceptů individua (Kterých všech konceptů je individuum instancí?),
- určení všech instancí konceptu.

### 2.3.2 Sémantický web

Internet (World Wide Web) prochází od svého vzniku neustálým vývojem. Původní koncept nazývaný Web 1.0 je založený na výměně dokumentů propojených odkazy. Tento koncept se rozšířil s přístupem webových aplikací umožňujících sdílení obsahu. Zatímco u Webu 1.0<sup>28</sup> můžeme poměrně jednoduše rozlišit, kdo je producentem a kdo konzumentem dokumentů, u Webu 2.0<sup>29</sup> už to tak jednoznačné není. Uživatel si sice zobrazuje někým vytvořený obsah, často ho ale komentuje, doplňuje a publikuje vlastní obsah. Typickými webovými aplikacemi Webu 2.0 jsou například Facebook, Wikipedie, Twitter, LinkedIn a mnoho dalších. Stále se ovšem jedná o sdílení obsahu ve formě dokumentů. Uživatelé internetu mají sice k dispozici téměř neomezené množství zdrojů (dokumentů), jejich porovnání a vyhledání požadovaných informací už ale musí vykonat sami.[2]

Web 3.0<sup>30</sup> je oproti tomu založený na sdílení konkrétních informací, které mají standardizovanou strukturu a jsou popsány z hlediska svého významu. Někdy bývá Web 3.0 označován

<sup>28</sup>Označení Web 1.0 není používáno od začátků internetu, objevuje se až s označením Web 2.0.

<sup>29</sup>Termín Web 2.0 se začíná objevovat v době, kdy je statický obsah webových stránek postupně nahrazován sdíleným prostorem pro společnou tvorbu obsahu. Poprvé je termín použit v roce 1999, ujímá se ale až po roce 2002.

<sup>30</sup>Termín Web 3.0 se objevuje poprvé v roce 2006.

také jako *Sémantický Web*, přestože dosavadní definice obou pojmů se liší. V kontextu Sémantického Webu už internet není distribuovaným systémem propojených dokumentů, ale spíše distribuovaným systémem propojených informací.[55]

Tento koncept otevírá zcela nové možnosti při používání internetu. Zásadní důraz je kladen na to, aby byly sdílené informace strojově čitelné a aby byl i jejich význam strojově analyzovatelný tak, aby počítače mohly sdílené informace zpracovávat, propojovat a nabízet uživatelům odpovědi na dotazy, které by jinak museli hledat uživatelé sami.

Stejně jako má každý internetový dokument svůj unikátní identifikátor - URL<sup>31</sup>[56], musí mít i každá informace sdílená na internetu ten svůj. Za tímto účelem je využíván *Uniform Resource Identifier* (URI)<sup>32</sup>. [48]

Sdílené informace jsou z hlediska syntaxe reprezentovány jednotně a to pomocí značkovacího jazyka *eXtensible Markup Language* (XML).[57] Aby byly sdílené informace použitelné mezinárodně, je používáno jednotné kódování *Unicode* [47].

Pro účely reprezentace informací sdílených na internetu slouží RDF, neboli *Resource Description Format*[51]. RDF je primárním nástrojem pro reprezentaci sdílených informací, informace jsou reprezentovány grafy tvořenými trojicemi *subjekt-predikát-objekt*. Jednotlivé části této trojice potom mohou být URI odkazem, literálem, nebo prázdným uzlem (*blank node*). Tyto části se souhrnně nazývají zdroje (nebo také entity).

Zdroj RDF může reprezentovat v podstatě cokoli - fyzické věci, abstraktní koncepty, dokumenty, čísla nebo textové řetězce. Zdroje reprezentované literály (literálové hodnoty) mají datové typy definované standardy XML (například číslo, textový řetězec, datum), nebo mohou být bez definice datového typu. URI odkazují na tzv. slovníky, které definují význam těchto entit. Z principu unikátnosti URI jsou tyto entity globálně platné, pokud se tedy stejné URI objevuje v RDF grafu vícekrát, odkazuje vždy na stejnou entitu. Struktura URI odkazu by měla obsahovat odkaz na slovník, kterým je odkazovaná entita popsána. Samotný slovník potom může být také reprezentován RDF grafem. Na rozdíl od URI odkazů, které mohou být používány i mimo slovník, kterým jsou odkazované entity definovány, čisté uzly nemají charakter unikátnosti, a proto mohou být používány pouze jako součást grafu, kterým jsou definovány. Přesto jsou často užitečné, například při definování seznamů.[52]

Na obrázku 2.5 je XML kód ve formátu N-Tripplé[53], který je pomocí RDF reprezentován jako graf vizualizovaný obrázkem 2.6. Z tohoto RDF grafu můžeme vyčíst například následující informace:

- Bob, který je narozený 4. 7. 1990 se zajímá o obraz Mona Lisa,
- obraz Mona Lisa byl namalován Leonardem Da Vinci,
- Bob se zná s Alicí.

Počáteční uzly hran grafu (první entity řádků XML) jsou objekty, hrany grafu (druhé entity XML) jsou predikáty a koncové uzly hran (třetí entity XML) jsou objekty, přičemž predikáty jsou vlastností subjektů, jejichž hodnotou jsou objekty.

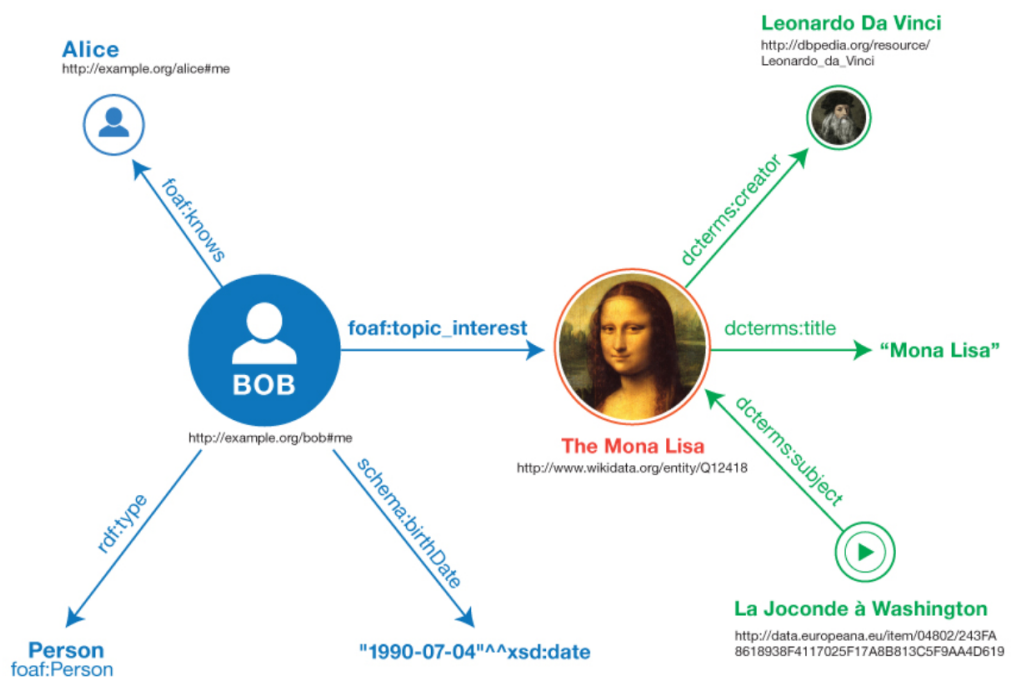
Na tomto jednoduchém příkladu RDF grafu (který využívá několik slovníků) je demonstrováno, jakým způsobem jsou data pomocí RDF reprezentována, jak mohou být sdílena a jak jsou čtena.

```

<http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .
<http://example.org/bob#me> <http://schema.org/birthdate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/topic_interest> <http://www.wikidata.org/entity/Q12418> .
<http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/title> "Mona Lisa" .
<http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/creator> <http://dbpedia.org/resource/Leonardo_da_Vinci>
<http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619> <http://purl.org/dc/terms/subject> <http://

```

Obrázek 2.5: Příklad XML kódu RDF grafu ve formátu N-Trippl [52]



Obrázek 2.6: RDF reprezentovaný graf [52]

Za účelem využívání standardizovaných entit a slovníků může být používáno *RDF Schema* [54] (RDFS). RDFS je jak syntaktickým, tak sémantickým rozšířením RDF, definuje řadu standardizovaných tříd pro specifikaci ontologií, ne vždy je ale dostačujícím nástrojem. Pro specifikaci detailnějších ontologií je používán *Web Ontology Language* (OWL). [50] OWL je jazykem založeným na deskriptivní logice a nabízí více konstruktů pro specifikaci ontologií než RDFS. Syntaxe OWL je stejně jako syntaxe RDFS založena na RDF, OWL tedy může být používán jako další standardizovaný slovník RDF. Sémantika jazyka OWL vychází ze sémantiky deskriptivní logiky.

RDF datové zdroje, stejně jako RDFS a OWL ontologie mohou být dotazovány jazykem *RDF Query Language* (SPARQL). [58] SPARQL je dotazovacím jazykem podobným SQL, zdrojem dat i výsledkem dotazu SPARQL jsou ale RDF reprezentovaná data. SPARQL slouží také jako protokol pro přístup k RDF reprezentovaným datům.

Vlastní logika aplikací Sémantického Webu je definována pomocí výše uvedených nástrojů. Tyto nástroje jsou široce využívány v aplikacích Sémantického webu. Stejně tak data ukládaná Semantic Big Data Historianem jsou reprezentována frameworkem RDF a popsána OWL ontologií.

---

<sup>31</sup>URL - Uniform Resource Locator

<sup>32</sup>URL používané pro identifikaci internetových dokumentů je samo podmnožinou URI



# Kapitola 3

## Experimentální část

Cílem experimentální části práce je navrhnout způsob transformace dat zpracovávaných Semantic Big Data Historianem pro efektivní analýzu těchto dat pomocí definovaného analytického nástroje KNIME a tuto transformaci následně implementovat pomocí analytické komponenty. Úkolem analytické komponenty je kromě samotné transformace sémanticky uložených dat také umožnění předzpracování transformovaných dat za pomoci Big Data nástrojů a tím snížení výpočetní náročnosti koncové analýzy prováděné nástrojem KNIME.

V následujících kapitolách jsou popsána analyzovaná data, architektura Semantic Big Data Historianu a samotná analytická komponenta.

### 3.1 Analyzovaná data

Data pro analýzu jsou uložena pomocí platformy Semantic Big Data Historian, která je určena pro integraci dat různorodých datových zdrojů z oblasti průmyslové automatizace za využití Big Data technologií. Typickými datovými zdroji jsou:

- senzory snímající informace o aktuálním stavu strojů,
- ERP<sup>1</sup> systémy,
- MES<sup>2</sup> systémy,
- další systémy z vyšší úrovně řízení,
- externí datové zdroje.

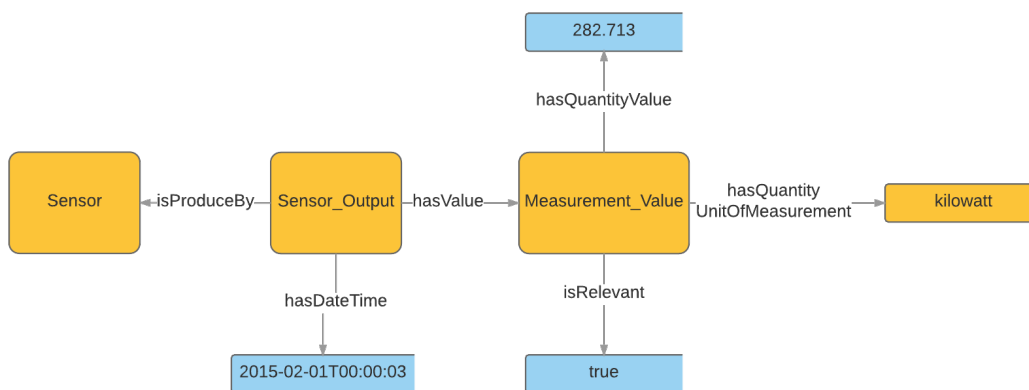
Semantic Big Data Historian data z výše uvedených zdrojů transformuje a ukládá ve formátu klíč-hodnota do indexovaných HDFS souborů. Tato sémanticky uložená data jsou popsána sdílenou SHS ontologií. Proces integrace heterogenních dat je popsán v [19]. Část SHS ontologie je zjednodušeně vizualizována diagramem 3.1. Diagram popisuje pouze reprezentaci uložených dat - nezobrazuje většinu konceptů a vazeb mezi nimi.

Pro účely této práce jsou používána data z vodní elektrárny.

---

<sup>1</sup>Enterprise Resource Planning

<sup>2</sup>Manufacturing Execution Systems



Obrázek 3.1: Zjednodušená vizualizace reprezentace sémanticky uložených dat

## 3.2 Popis architektury

Analyzovaná data jsou uložena na Hadoop serveru, konkrétně ve sdíleném souborovém systému HDFS ve formě trojic<sup>3</sup>. Na HDFS je připojen Hive, který pomocí definovaných metadat zpřístupňuje data ve formě tabulek. Data je tedy možné dotazovat pomocí Hive QL<sup>4</sup>[49] dotazů, které jsou následně převáděny na MapReduce joby. Přestože jsou data přístupná pomocí Hive QL dotazů nad definovanými tabulkami, jsou data stále uložena sémanticky - sloupce tabulek jsou {subjekt, predikát, objekt}. Aby mohl uživatel takto uložená data analyzovat pomocí analytického nástroje KNIME, musí data nejdříve transformovat a transformovaná data uložit. Průběh analýzy bez použití analytické komponenty je popsán diagramem 3.2.

Analytická komponenta mění tuto architekturu a zjednodušuje pro uživatele koncovou analýzu. Při spuštění jsou načtena metadata tabulek definovaných v Hive<sup>5</sup> tak, aby mohl uživatel jednoduše zvolit zdroj dat pro analýzu. Zároveň jsou načtena existující KNIME workflow - uživatel má tedy možnost zvolit, jakou analýzu bude nad zvolenými daty spouštět. Po zvolení datového zdroje a KNIME workflow je vygenerován transformační dotaz.

Sémanticky uložená data jsou transformována pomocí vygenerovaného Hive QL dotazu do běžného tabulkového formátu pro umožnění analýzy těchto dat pomocí analytického nástroje KNIME. Transformační dotaz je generovaný automaticky na základě datového zdroje (tabulky) zvoleného uživatelem<sup>6</sup>.

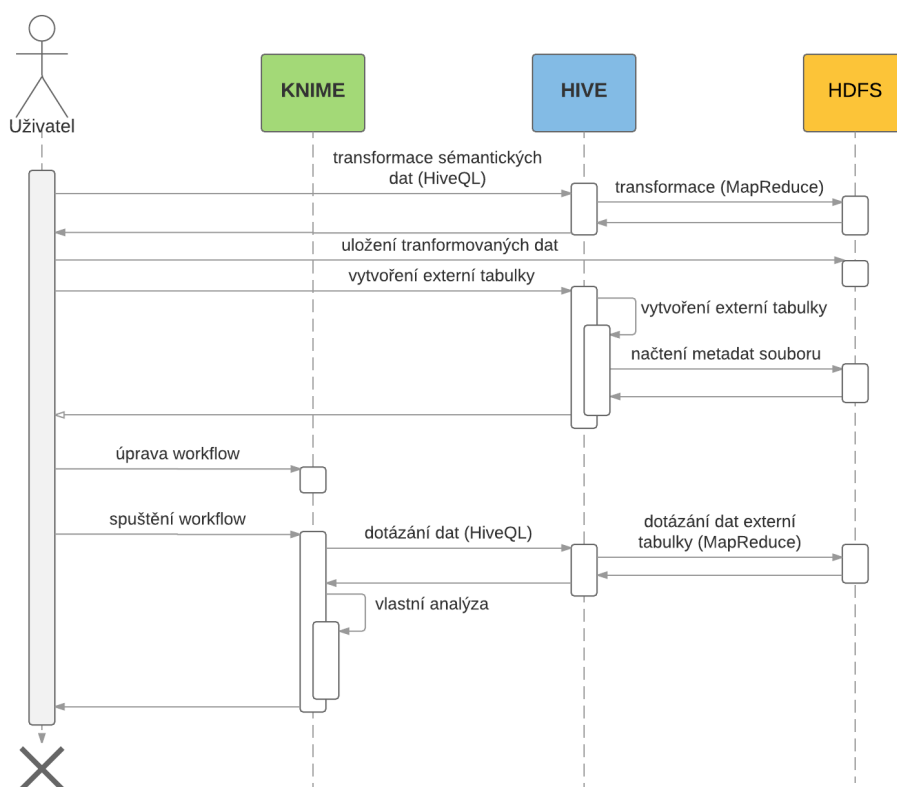
Uživatel má možnost tento vygenerovaný transformační dotaz upravit pro potřeby konkrétní analýzy. Tímto prvním krokem analýzy dat může být filtrování dat nebo například jejich agregace. Objem dat analyzovaných pomocí KNIME workflow se tak může výrazně zmenšit, a díky tomu je možné data analyzovat efektivněji - rychleji. Předpokladem je, že

<sup>3</sup>Data jsou uložena sémanticky ve formě trojic subjekt-predikát-objekt, trojice jsou ale reprezentovány páry klíč-hodnota.

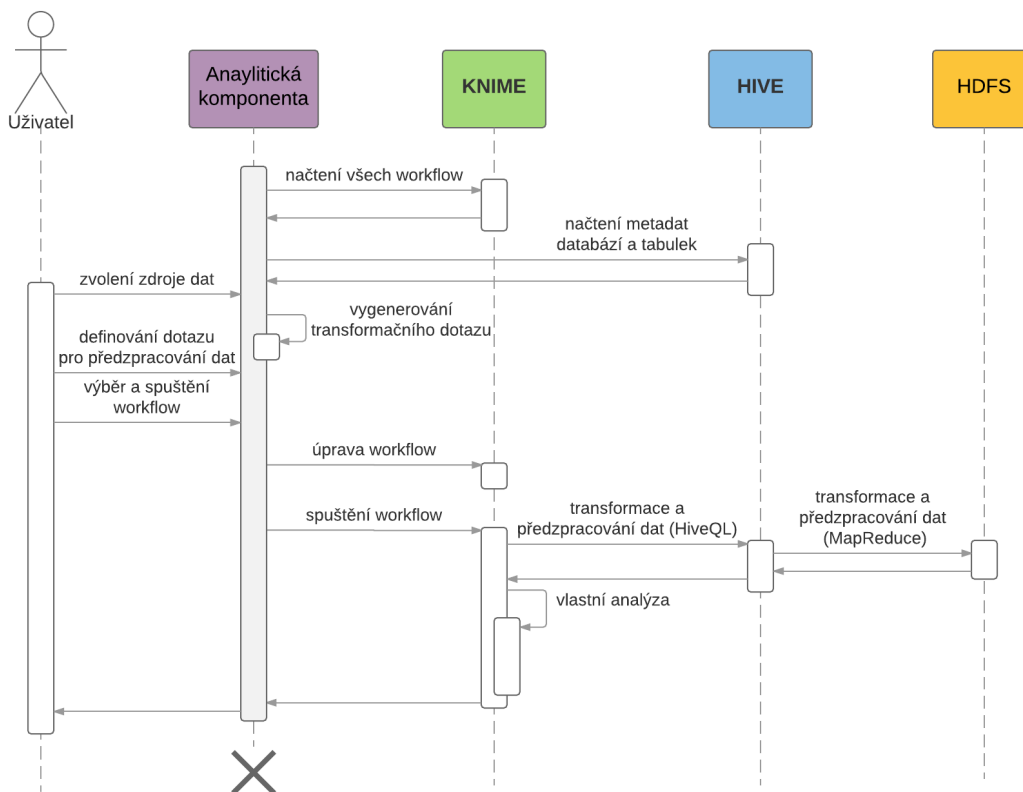
<sup>4</sup>Hive Query Language - vlastní implementace jazyka SQL

<sup>5</sup>Analytická komponenta je připojena na Hive pomocí JDBC konektoru.

<sup>6</sup>Ontologie popisující sémanticky uložená data je ale pevně implementována v analytické komponentě.



Obrázek 3.2: Průběh analýzy bez použití analytické komponenty



Obrázek 3.3: Průběh analýzy s použitím analytické komponenty

pokud je nejnáročnější část analýzy vykonána pomocí Big Data nástrojů, které data zpracovávají paralelně, bude celá analýza rychlejší, než pokud by byl celý soubor dat zpracováván pomocí KNIME workflow.

Při spuštění analýzy je vygenerovaný (a případně upravený) transformační dotaz uložen jako součást spouštěného KNIME workflow, díky čemuž může KNIME pracovat přímo s dotázanými daty bez nutnosti jejich materializace. Následně je spuštěno KNIME workflow, jehož první částí je transformace sémanticky uložených dat a jejich předzpracování dat a až poté samotná analýza.

Průběh analýzy s použitím analytické komponenty je zachycený na diagramu 3.3.

### 3.3 Popis funkcí analytické komponenty

Z výše popsané architektury analytické komponenty vyplývají základní funkční požadavky na analytickou komponentu:

- načtení metadat z Hivu,

- načtení KNIME workflow,
- vygenerování transformačního dotazu,
- úprava KNIME workflow,
- spuštění KNIME workflow,
- uložení výsledků analýzy.

### 3.3.1 Načtení metadat z Hivu

Vzhledem k úkolu analytické komponenty zjednodušit analýzu dat je žádoucí, aby uživatel mohl při změně nastavení workflow vycházet pouze z informací, které mu komponenta poskytuje. Nejinak je tomu při výběru zdroje dat. Analytická komponenta dotazuje Hive o metadata - názvy databází, názvy a strukturu tabulek, které Hive obsahuje tak, aby mohl uživatel jednoduše vybrat, nad kterou tabulkou bude analýza spuštěna.

Komunikace komponenty s Hive serverem je možná několika způsoby. Jako první byla testována komunikace pomocí příkazové řádky operačního systému<sup>7</sup>. Tento způsob řešení je sice funkční, nicméně časová náročnost komunikace skrz příkazovou řádku je neúměrně velká i pro velice rychle vykonatelné dotazy. Například dotaz na názvy všech databází v tomto případě trvá více než dvě vteřiny. Z toho důvodu bylo toto řešení nahrazeno připojením pomocí ODBC konektoru. V tomto případě je časová náročnost na komunikaci s Hive serverem zanedbatelná. Připojení pomocí ODBC konektoru nicméně vyžaduje běh služby *hiveserver2*.

### 3.3.2 Načtení KNIME workflow

KNIME workflow jsou definována pomocí XML souborů, přičemž vlastní XML nastavení má každá komponenta (uzel) ve workflow. I samotné workflow má vlastní XML nastavení, kde jsou definovány jeho základní atributy, namapovány jednotlivé uzly workflow a jejich propojení. Načtení informací o workflow a jednotlivých uzlech je důležité pro další funkce analytické komponenty.

Zjednodušená struktura XML nastavení workflow je následující:

- základních atributů workflow (verze KNIME, se kterou je workflow kompatibilní, název workflow, informace o jeho autorovi),
- definice uzlů, ze kterých je workflow složeno,
- definice propojení jednotlivých uzlů,
- atributy pro vykreslení workflow v KNIME.

---

<sup>7</sup>Předpokládá se, že Hive server běží na UNIXovém operačním systému, příkazovou řádkou je míněna některá z interpretací příkazové řádky UNIXu, například BASH. Při reálném použití by příkaz byl realizován pomocí SSH, protože Hive server je spuštěn na jiném zařízení než analytická komponenta. Jak je popsáno dále, tento přístup ale nebyl zvolen.

```
<config>
  <!--basic workflow attributes-->
  <config key="workflow_credentials"/>
  <config key="nodes">
    <config key="node_1">
      <!--basic node attributes-->
      <config key="ui_settings">
        <config key="extrainfo.node.bounds">
          <!--attributes of processed data-->
        </config>
      </config>
    </config>
  </config>
  <!--more nodes-->
</config>
<config key="connections">
  <config key="connection_0">
    <!--edge definition-->
  </config>
  <!--more edges-->
</config>
<config key="workflow_editor_settings">
  <!--knime editor attributes-->
</config>
</config>
```

Struktura XML nastavení jednotlivých uzlů se odvíjí od typu uzlu, vzhledem k velkému množství typů uzlů zde není uvedena. Pro účely načtení workflow není podstatná, lze vyjít z informací o uzlech uvedených v nastavení workflow samotného.

### 3.3.3 Vygenerování transformačního dotazu

Po zvolení datového zdroje je vygenerován Hive QL dotaz, který slouží pro transformaci sémanticky uložených dat do formátu analyzovatelného pomocí aplikace KNIME. Vygenerovaný Hive QL dotaz může vypadat například následovně:

```
SELECT
  sensors.subject as sensor ,
  regexp_replace(substr(s2.object, 2, 19), 'T', '␣') as createDate ,
  cast(s4.object as decimal(18,6)) as value ,
  s5.object as unit
FROM bp.sample sensors
JOIN bp.sample s1
  ON (s1.object = sensors.subject AND s1.predicate =
      '<http://purl.oclc.org/NET/ssnx/ssn#isProducedBy>')
JOIN bp.sample s2
  ON (s2.subject = s1.subject AND s2.predicate =
      '<http://www.rockwellautomation.com/...#hasDateTime>')
```

```

JOIN bp.sample s3
  ON (s3.subject = s2.subject AND s3.predicate =
      '<http://purl.oclc.org/NET/ssnx/ssn#hasValue>')
JOIN bp.sample s4
  ON (s4.subject = s3.object AND s4.predicate =
      '<http://www.rockwell...#hasQuantityValue>')
JOIN bp.sample s5
  ON (s5.subject = s3.object AND s5.predicate =
      '<http://www.rockwell...#hasQuantityUnitOfMeasurement>')

```

Dotaz je vázaný na konkrétní ontologii popisující sémanticky uložená data. Datovým zdrojem je jedna tabulka, pro získání hodnoty každého sloupce musí být tabulka napojená sama na sebe. Transformační dotaz může být chápán jako vyhledávací algoritmus grafu.

Hive dotaz převádí na sérii MapReduce jobů, kterými jsou zpracovávána data uložená v HDFS.

### 3.3.4 Úprava KNIME workflow

V případě, že chce uživatel měnit nastavení některého z uzlů workflow nebo workflow samotného, je z XML souboru vytvořen DOM Dokument, který je následně možné prohlédávat a upravovat pomocí XPath. Díky tomuto univerzálnímu přístupu může uživatel měnit hodnotu libovolného atributu uzlu nebo workflow.

Pokud se uživatel rozhodne provedené změny uložit, je nejprve vytvořena záloha původního nastavení<sup>8</sup>, uživatel tedy může provedené změny jednoduše vrátit. Zálohována je pouze poslední verze souboru, pokud existuje starší záloha, je přepsána. V opačném případě by objem adresáře se zálohami narůstal a bylo by nutné jej udržovat, ať už manuálně, nebo automaticky.

### 3.3.5 Spuštění KNIME workflow

Samotné spuštění KNIME workflow je implementováno pomocí příkazové řádky operačního systému, workflow je spouštěno v *batch módu*<sup>9</sup>. Přestože ani v tomto případě není využití příkazové řádky operačního systému příliš efektivní<sup>10</sup>, architektura aplikace KNIME neumožňuje jinou alternativu<sup>11</sup>.

### 3.3.6 Uložení výsledků analýzy

Jednou z nevýhod spouštění workflow pomocí batch módu je, že KNIME po úspěšném vykonání workflow nevrátí výsledek analýzy. Výsledky tedy musí být získány jiným způsobem.

<sup>8</sup>Zálohovaný soubor je uložen do složky `./backup`

<sup>9</sup>`org.knime.product.KNIME_BATCH_APPLICATION` - mód aplikace KNIME pro spouštění workflow pomocí příkazové řádky.

<sup>10</sup>Jedinou zpětnou vazbou o výsledku průběhu workflow jsou logy aplikace KNIME, ve kterých je nutné potřebné informace dohledávat.

<sup>11</sup>Tedy za předpokladu, že není žádoucí otevírat grafické rozhraní aplikace - což není, analytická komponenta by tímto přišla o výhody, které přináší z hlediska jednoduchosti spouštění analýz.

bem, než přímo pomocí analytické komponenty, protože takové řešení by bylo implementačně velice náročné. Tento problém je možné řešit na úrovni samotného workflow pomocí uzlů exportujících výsledek do souboru (například CSV). Je ovšem třeba brát ohled na to, že workflow může být rozvětvené, potom výsledná data z každé větve workflow musí být exportována samostatně. Z tohoto důvodu je zavedena následovná jmenná konvence uzlů pro export dat "export\_x.y", kde "x" je název výsledného exportovaného souboru a "y" je jeho přípona. Díky této konvenci je možné jednoduše měnit cílové umístění pro exportované soubory.

### 3.3.7 Rozpad časové náročnosti analýzy

Analýzy jsou prováděny nad velkým objemem dat, z toho důvodu může být rychlost a efektivita analýzy zásadním faktorem pro využitelnost celého konceptu. Optimalizace spouštěných analýz bude proto při používání analytické komponenty jednou z nutných činností. Samotné optimalizační nástroje budou spíše tématem možného budoucího rozšíření analytické komponenty, nicméně jako základ pro tyto nástroje je navržena a implementována funkce pro časové vyhodnocování jednotlivých částí analýzy. Z hlediska optimalizace je zásadní, aby bylo možné rozlišit čas části analýzy vykonávané Big Data nástroji (předzpracování dat) a čas konečné analýzy pomocí nástroje KNIME. Implementovaná funkce rozpadá čas analýzy na úrovni jednotlivých uzlů KNIME workflow. Tyto údaje je nicméně nutné čerpat z logů, které jsou vytvářeny při spuštění KNIME workflow, což do budoucna nemusí být výhodný přístup<sup>12</sup>.

## 3.4 Implementace

Analytická komponenta je implementovaná jako desktopová aplikace s grafickým rozhraním v programovacím jazyce Java. Aplikace je určena pro Linuxové operační systémy<sup>13</sup>. Pro správu knihoven je použit Apache Maven. [42] Architektura aplikace je vícevrstvá, skládá se z následujících vrstev:

- model<sup>14</sup>,
- DAO<sup>15</sup>,
- services<sup>16</sup>.

Nad těmito vrstvami je vystavěno grafické uživatelské prostředí. Hlavní stránka grafického uživatelského rozhraní je popsána obrázkem 3.4.

---

<sup>12</sup>Velikost vytvářených logů roste s objemem analyzovaných dat a zároveň je závislá na implementaci jednotlivých uzlů KNIME workflow. Rychlost zpracování těchto logů může být tedy na základě těchto parametrů velmi různá.

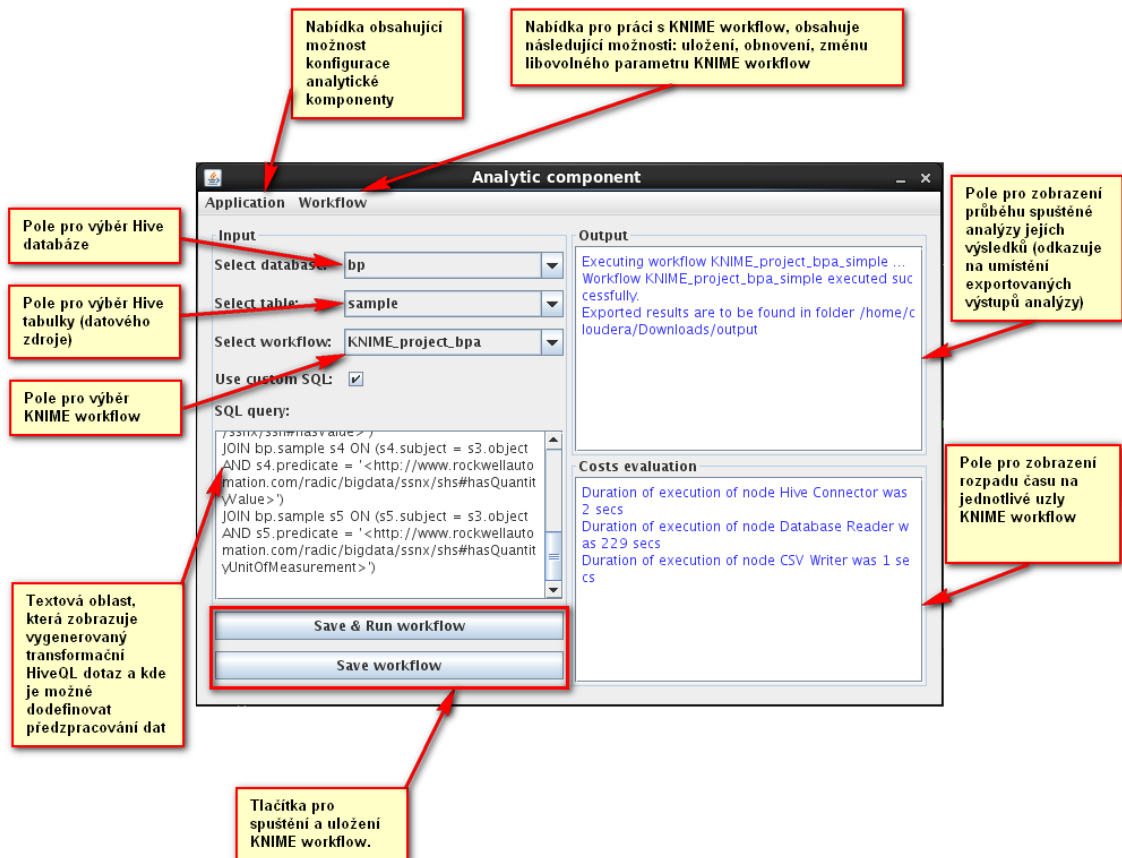
<sup>13</sup>Aplikace byla vyvíjena a testována na operačním systému CentOS verze 5.5.

<sup>14</sup>Vrstva model obsahuje třídy odpovídající entitám, se kterými analytická komponenta pracuje.

<sup>15</sup>Vrstva DAO (Data Access Object) obsahuje třídy pro základní práci s entitami definovanými třídami ve vrstvě model.

<sup>16</sup>Vrstva services obsahuje třídy poskytující služby, které jsou volány grafickým uživatelským rozhraním.





Obrázek 3.4: Popis hlavní stránky grafického uživatelského rozhraní

### 3.5 Návrh budoucího rozšíření

Vytvořená analytická komponenta poskytuje základní funkce pro analýzu sémanticky uložených dat pomocí analytického nástroje KNIME. Níže jsou uvedeny návrhy na rozšíření aplikace, které mohou značně zvýšit její přidanou hodnotu.

V kapitole [Popis architektury](#) je zmíněna nutnost optimalizace analýz z hlediska jejich výkonu. Tato nutnost nevznikla vytvořením analytické komponenty, nýbrž je dána objemem analyzovaných dat, která jsou ukládána Semantic Big Data Historianem. Analytická komponenta nabízí možnosti, jak tuto optimalizaci usnadnit. Současná implementace umožňuje uživateli definovat předzpracování analyzovaných dat pomocí Hive QL dotazu. Závisí ale na samotném uživateli zda se rozhodne této možnosti využít a pokud ano, tak jak velkou část analýzy se rozhodne tímto způsobem vykonat. Možným rozšířením tohoto konceptu je automatické vyhodnocování toho, co je vhodné analyzovat na lokálním zařízení pomocí KNIME a co je vhodné analyzovat paralelně pomocí Semantic Big Data Historianu. Ideální rozdělení analýzy na tyto dvě části může přinést její velké zrychlení. Pokud bude definováno rozdělení analýzy na obě části, bude také možné uvažovat automatické převádění části analýzy definované pomocí KNIME workflow na Hive QL kód a tuto část analýzy tedy provádět paralelně pomocí Big Data nástrojů.

Cílem celého konceptu Semantic Big Data Historianu s použitím analytické komponenty a analytického nástroje KNIME je umožnit efektivní analyzování uložených dat. Data jsou analyzována různými algoritmy, přičemž velká část logiky analýzy je definována pomocí KNIME workflow. Analytická komponenta umožňuje uživateli výběr z definovaných workflow - tedy zvolení analýzy, která bude nad transformovanými daty spuštěna. Ne vždy uživatel ale vystačí s předdefinovanými analýzami. Pokud je potřeba data v krátké době analyzovat způsobem, který neodpovídá žádnému předdefinovanému workflow, musí uživatel vytvořit workflow nové. Tento přístup ale není dostatečně pružný pro případy, kdy je požadována *ad-hoc analýza*. Pro tyto případy by byl užitečný nástroj, pomocí kterého by bylo možné sestavit nové workflow pružněji. Dalším z možných rozšíření analytické komponenty je generování nových KNIME workflow z předdefinovaných částí. Uživatel by měl možnost zvolit z jakých algoritmů by se měla nová analýza skládat a analytická komponenta by dle těchto požadavků vytvořila nové KNIME workflow. Samotné algoritmy by přitom byly již předdefinovány. Tímto způsobem by bylo možné velice pružně vytvářet nové analýzy a zkoumat data z nových úhlů pohledu. Z informací uvedených v kapitole [KNIME](#) vyplývá, že by pro tento účel mohlo být vhodné využití specifických uzlů KNIME workflow - metanodů.

Vytvořená analytická komponenta je určena pro práci se daty, která jsou sémanticky uložena podle konkrétní ontologie používané frameworkem Semantic Big Data Historian. Dalším z možných rozšíření analytické komponenty je zobecnění implementace komponenty v tomto smyslu. Transformační Hive QL dotaz by mohl být dynamicky generován na základě ontologie (specifikované některým z jazyků RDF nebo OWL) popisující uživatelem vybraný datový zdroj.

Používání analytické komponenty v praxi jistě přinese mnoho dalších návrhů na její rozšíření.

# Kapitola 4

## Závěr

Cíl této práce, tedy navržení způsobu transformace dotazů pro zpracování dat ukládaných Semantic Big Data Historianem za účelem efektivní analýzy těchto dat pomocí analytického nástroje KNIME, byl splněn pomocí implementace analytické komponenty splňující tyto požadavky.

Cílem teoretické části práce byl detailní popis technologií a konceptů, které jsou používány, ať už přímo, nebo zprostředkovaně, analytickou komponentou. Velký důraz byl přitom kladen na způsoby zpracování a reprezentace dat, především na koncepty, jejichž použití může vést k větší efektivitě zpracování dat.

Byly popsány charakteristiky a koncepty Big Data a nástroje frameworku Apache Hadoop, které za pomoci těchto konceptů umožňují efektivní zpracování velkého objemu různorodých dat. Z těchto nástrojů je pro samotnou práci nejpodstatnější distribuovaný souborový systém HDFS, který je využíván frameworkem Semantic Big Data Historian k ukládání sémanticky popsaných dat.

Dále byla pozornost věnována analytickým nástrojům, respektive nástrojům, které umožňují zpracovávat data uložená v HDFS. První z těchto nástrojů, framework MapReduce, umožňuje paralelní zpracování dat uložených v HDFS. Ostatní popsané nástroje využívají pro zpracování dat právě MapReduce, a proto byla věnována velká pozornost jeho architektuře. Dále byl popsán framework Apache Hive, který je v kontextu frameworku Semantic Big Data Historian využíván jako primární nástroj pro dotazování a zpracování dat externími aplikacemi, nebo přímo uživatelem. Hive je pro tento účel vhodný mimo jiné proto, že umožňuje dotazování dat uložených v HDFS pomocí vlastní implementace jazyka SQL - Hive QL. Posledním článkem řetězce analytických nástrojů využívaných frameworkem Semantic Big Data Historian je KNIME - nástroj pro analýzu dat pomocí vizuálně sestavovaných workflow. Právě KNIME je využíván pro koncovou analýzu sémanticky popsaných dat ukládaných Semantic Big Data Historianem.

V poslední kapitole teoretické části práce byly popsány principy sémantického popisu dat. Sémantickým popisem dat je chápána ontologie, která je formální, explicitní specifikací konceptualizace. Byly uvedeny způsoby formální reprezentace ontologie, především potom jazyky založené na deskriptivní logice RDF a OWL.

Na základě těchto informací byla navržena architektura analytické komponenty a způsob její spolupráce s existujícími nástroji, které dle zadání využívá - tedy analytickým nástrojem KNIME a Big Data nástroji používanými frameworkem Semantic Big Data Historian.

Implementovaná analytická komponenta umožňuje mimo jiné automatické generování transformačních dotazů a definování dotazů pro prvotní zpracování transformovaných dat.

Automatické generování transformačních dotazů a následné použití transformovaných dat pro analýzu definovanou pomocí workflow analytického nástroje KNIME přináší uživateli zjednodušení analýzy sémanticky popsaných dat. Uživatel není nucen před analýzou data manuálně transformovat a může se soustředit na samotnou analýzu dat a její výsledky. Uživatel také díky automatickému generování transformačního dotazů nemusí znát ontologii, která je popisuje.

Díky možnosti definovat prvotní předzpracování dat pomocí Big Data nástrojů může také uživatel rozložit výpočetní náklady analýzy. Náklady mohou být rozloženy mezi distribuovaný Semantic Big Data Historian, respektive fyzická zařízení používaná tímto frameworkem pro zpracování velkého objemu dat, a zařízení, které provádí analýzu dat za pomoci analytického nástroje KNIME. I v tomto případě přináší analytická komponenta výrazné zjednodušení práce uživatele. Uživatel má možnost definovat tuto prvotní část analýzy pomocí Hive QL dotazu. Nemusí přitom data předzpracovaná pro účel další analýzy pomocí KNIME workflow již nikam přesouvat nebo s nimi jinak manipulovat.

V práci jsou také navrženy možnosti dalšího rozšíření analytické komponenty. Tato rozšíření by přinesla další zefektivnění a zjednodušení analýzy velkého objemu sémanticky popsaných dat a to především z hlediska účelovosti a výpočetní efektivity analýzy.

# Literatura

- [1] ANSI Information technology — Common Logic (CL) – A framework for a family of logic-based languages. Technical report, ISO, 2013.
- [2] Alcatel Lucent. New communication behaviours in a Web 2.0 world — Changes challenges and opportunities in the era of the Information Revolution. Technical report, 2008.
- [3] BAADER, F. et al. The Description Logic Handbook: Theory, Implementation and Applications. Technical report, Cambridge University Press, 2003.
- [4] BERTHOLD, M. R. et al. KNIME: The Konstanz Information Miner. Technical report, ALTANA Chair for Bioinformatics and Information Mining Department of Computer and Information Science University of Konstanz, 2007. Dostupné z: <[http://www.uni-konstanz.de/bioml/bioml2/publications/Papers2007/BCDG+07\\_knime\\_gfkl.pdf](http://www.uni-konstanz.de/bioml/bioml2/publications/Papers2007/BCDG+07_knime_gfkl.pdf)>.
- [5] BORST, W. N. Construction of Engineering Ontologies. Technical report, Institute for Telematica and Information Technology, University of Twente, 1997.
- [6] CHAUDHRI, V. et al. *Open Knowledge Base Connectivity* [online]. 1998. [cit. 26.4.2016]. Dostupné z: <<http://www.ai.sri.com/~okbc/okbc-2-0-3.pdf>>.
- [7] CLOUDERA. Hadoop and HDFS: Storage for next generation data management. Technical report, Cloudera, 2016. Dostupné z: <<http://www.cloudera.com/content/dam/www/static/documents/whitepapers/hadoop-and-hdfs.pdf>>.
- [8] DAS, S. K. *Deductive Databases and Logic Programming*. : Addison-Wesley Pub, 1st edition, 1992. ISBN 0201568977.
- [9] DEAN, J. – GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. *Google, Inc.* 2008, 51, 1, s. 107–113.
- [10] FOREMAN, J. W. *Using Data Science to Transform Information into Insight*. Crosspoint Boulevard Indianapolis : WILEY, 1st edition, 2014. ISBN 978-1-118-66146-8.
- [11] FRANCONI, E. *Description Logics Tutorial Course Information* [online]. 2002. [cit. 26.4.2016]. Dostupné z: <<https://www.inf.unibz.it/~franconi/dl/course/>>.

- [12] GENESERETH, M. – FIKES, R. Knowledge interchange Format Version 3.0 Reference Manual. Technical report, Computer Science Department Stanford University Stanford, 1994.
- [13] GENESERETH, M. – NILSSON, N. Logical Foundations of Artificial Intelligence. Technical report, Stanford University, 1987.
- [14] GRUBER, T. A Translation Approach to Portable Ontology Specifications. Technical report, Knowledge Acquisition, 1993.
- [15] HELLERSTEIN, J. M. Programming a Parallel Future. Technical Report UCB/EECS-2008-144, EECS Department, University of California, Berkeley, Nov 2008. Dostupné z: <<http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-144.html>>.
- [16] IDC. *Data Growth, Business Opportunities, and the IT Imperatives* [online]. 2014. [cit. 1.4.2014]. Dostupné z: <<http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>>.
- [17] Intel Corporation. Extract Transform and Load Big Data with Apache Hadoop. Technical report, 2013. Dostupné z: <<https://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf>>.
- [18] Intel Corporation. Optimizing Hadoop Deployments. Technical report, 2010. Dostupné z: <<http://www.intel.com/content/dam/doc/white-paper/cloud-computing-optimizing-hadoop-deployments-paper.pdf>>.
- [19] JIRKOVSKÝ, V. – OBITKO, M. Semantic Heterogeneity Reduction for Big Data in Industrial Automation. Technical report, Czech Technical University in Prague and Rockwell Automation Research and Development Center, 2014.
- [20] KENNEDY, K. – MCKINLEY, K. Optimizing Hadoop Deployments. Technical report, Department of Compute Science Rice University, 2010. Dostupné z: <<http://www.cs.utexas.edu/users/mckinley/papers/par-mem-ics-92.pdf>>.
- [21] KNIME.com AG. *KNIME Big Data Extensions* [online]. 2016. [cit. 26.4.2016]. Dostupné z: <<https://www.knime.org/knime-big-data-connectors>>.
- [22] KONSTANTIN SHVACHKO, S. R. H. K. – CHANSLER, R. The Hadoop Distributed File System. Technical report, Yahoo, 2010. Dostupné z: <<http://storageconference.us/2010/Papers/MSST/Shvachko.pdf>>.
- [23] LANEY, D. 3D Data Management: Controlling Data Volume, Velocity and Variety. *META Group*. 2001, 1, 1, s. 1–4.
- [24] LEE, R. et al. YSmart: Yet Another SQL-to-MapReduce Translator. Technical report, Department of Computer Science and Engineering Ohio State University Center for Comprehensive Informatics Emory University Facebook Data Infrastructure Team, 2011. Dostupné z: <<http://web.cse.ohio-state.edu/hpcs/WWW/HTML/publications/papers/TR-11-7.pdf>>.

- [25] MILLER, G. *WordNet: A Lexical Database for English* [online]. 1995. [cit. 26. 4. 2016]. Dostupné z: <<http://nlp.cs.swarthmore.edu/~richardw/papers/miller1995-wordnet.pdf>>.
- [26] NewVantage Partners. *Big Data Executive Survey 2012 - Consolidated Summary Report* [online]. 2012. [cit. 31.12.2012]. Dostupné z: <<http://newvantage.com/wp-content/uploads/2012/12/NVP-Big-Data-Survey-2012-Consolidated-Report-Final1.pdf>>.
- [27] NIKOLAOS PAPAILIOU, D. T. I. K. – KOZIRIS, N. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Technical report, Cisco IBSG, 2011. Dostupné z: <[http://www.cisco.com/c/dam/en\\_us/about/ac79/docs/innov/IoT\\_IBSG\\_0411FINAL.pdf](http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf)>.
- [28] NOY, N. – MCGUINNESS, D. *Ontology Development 101: A Guide to Creating Your First Ontology* [online]. 2000. [cit. 26. 4. 2016]. Dostupné z: <[http://protege.stanford.edu/publications/ontology\\_development/ontology101-noy-mcguinness.html](http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html)>.
- [29] OBITKO, M. *Industry 4.0 and Big Data* [online]. 2015. [cit. 26. 4. 2016]. Dostupné z: <[http://www.stech.cz/Portals/0/Konference/2015/03%20Industry/PDF/03\\_obitko.pdf](http://www.stech.cz/Portals/0/Konference/2015/03%20Industry/PDF/03_obitko.pdf)>.
- [30] OBITKO, M. Translations between Ontologies in Multi-Agent Systems. Technical report, Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University in Prague, 2007.
- [31] PHILLIPS, P. White Paper: An insight into Business Intelligence. Technical report, Brio Technology, 1997. Dostupné z: <<http://www.computerweekly.com/feature/White-Paper-An-insight-into-Business-Intelligence>>.
- [32] RapidMiner. *RapidMiner* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://rapidminer.com/>>.
- [33] SANJAY GHEMAWAT, H. G. – LEUNG, S.-T. The Google File System. Technical report, Google, 2003. Dostupné z: <<http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf>>.
- [34] SIEB, C. – MEINL, T. – BERTHOLD, M. R. Parallel and distributed data pipelining with KNIME. Technical report, ALTANA Chair for Bioinformatics and Information Mining Department of Computer and Information Science University of Konstanz, 2007. Dostupné z: <[https://kops.uni-konstanz.de/bitstream/handle/123456789/5542/20Parallel\\_and\\_Distributed\\_Data\\_Pipelining\\_with\\_KNIME.pdf](https://kops.uni-konstanz.de/bitstream/handle/123456789/5542/20Parallel_and_Distributed_Data_Pipelining_with_KNIME.pdf)>.
- [35] SMITH, B. Beyond Concepts: Ontology as Reality Representation. Technical report, Department of Philosophy University at Buffalo and Institute for Formal Ontology and Medical Information Science Saarland University, 1998.
- [36] SOWA, J. F. *Knowledge Representation: Logical Philosophical and Computational Foundations*. : BrooksCole, 1st edition, 2000. ISBN 0534949657.

- [37] STAAB, S. – STUDER, R. *Handbook on ontologies*. : Springer, 2st edition, 2009. ISBN 978-3-540-92673-3.
- [38] The Apache Software Foundation. *Apache Derby* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://db.apache.org/derby/>>.
- [39] The Apache Software Foundation. *Apache HBase™ Reference Guide* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<http://hbase.apache.org/book.html#replication>>.
- [40] The Apache Software Foundation. *Mahout 0.12.0 Features by Engine* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<http://mahout.apache.org/users/basics/algorithms.html>>.
- [41] The Apache Software Foundation. *Mahout Recommender Overview* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://mahout.apache.org/users/recommender/recommender-documentation.html>>.
- [42] The Apache Software Foundation. *Apache Maven Project* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<http://maven.apache.org/>>.
- [43] The Apache Software Foundation. *Apache Spark™* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://spark.apache.org/>>.
- [44] The Apache Software Foundation. *Apache Thrift* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://thrift.apache.org/>>.
- [45] The Apache Software Foundation. *MapReduce NextGen aka YARN aka MRv2* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site>>.
- [46] The Common Lisp Foundation. *Common-Lisp.net* [online]. 2015. [cit. 26. 4. 2016]. Dostupné z: <<https://common-lisp.net/>>.
- [47] The Unicode Consortium. *Unicode* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<http://unicode.org/>>.
- [48] Tim Berners-Lee and Roy Fielding and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. Technical report, W3C and Day Software and Adobe Systems, 2005.
- [49] Treasure Data. *Hive (SQL-style) Query Language* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://docs.treasuredata.com/articles/hive>>.
- [50] W3C. *OWL Web Ontology Language* [online]. 2014. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/owl-ref/>>.
- [51] W3C. *RDF 1.1 Concepts and Abstract Syntax* [online]. 2014. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/rdf11-concepts/>>.
- [52] W3C. *RDF 1.1 Primer* [online]. 2014. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/>>.



- [53] W3C. *RDF 1.1 N-Triples* [online]. 2014. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/n-triples/>>.
- [54] W3C. *RDF Schema 1.1* [online]. 2014. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/rdf-schema/>>.
- [55] W3C. *Semantic Web* [online]. 2016. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/standards/semanticweb/>>.
- [56] W3C. *Uniform Resource Locator* [online]. 1994. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/Addressing/URL/url-spec.html>>.
- [57] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. 2008. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/REC-xml/>>.
- [58] W3C. *SPARQL Query Language for RDF* [online]. 2008. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/TR/rdf-sparql-query/>>.
- [59] WHITE, T. *Hadoop: The Definitive Guide*. Gravenstein Hwy N, Sebastopol : O'Reilly Media, Inc., 3rd edition, 2012. ISBN 978-1-449-31152-0.
- [60] World Wide Web Consortium. *Incubator Activity - W3C Semantic Sensor Network Incubator Group* [online]. 2005. [cit. 26. 4. 2016]. Dostupné z: <<https://www.w3.org/2005/Incubator/ssn/>>.



# Příloha A

## Obsah přiloženého CD

- *analyticka\_komponenta* (spustitelný soubor analytické komponenty)
- *bakalarska\_prace* (text bakalářské práce ve formátu PDF)
- *dokumentace* (dokumentace analytické komponenty)
  - *javadoc* (vygenerovaný javadoc)
  - *diagramy* (diagramy balíčků a tříd)
- *vzorova\_data*
  - *datovy\_zdroj* (neupravený zdroj testovacích dat)
  - *importovateln\_data* (upravený zdroj testovacích dat)
  - *vzorove\_KNIME\_workflow* (vzorové workflow)
- *zdrojove\_kody*
  - *analyticka\_komponenta* (zdrojový kód analytické komponenty)
  - *bakalarska\_prace* (zdrojový kód bakalářské práce ve formátu LaTeX)
- *README.TXT* (soubor definující požadavky ke spuštění a korektní funkčnosti analytické komponenty)