Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Studijní obor: Podniková informatika

Integrace Big Data a datového skladu

Autor diplomové práce : Bc. Vladislav Kiška

Vedoucí diplomové práce : doc. Ing. Ota Novotný, Ph. D.

Oponent diplomové práce : Ing. Valeria Kerol

Rok odevzdání: 2017

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 10. dubna 2017 …….

Bc. Vladislav Kiška

Poděkování

Na tomto místě bych rád poděkoval vedoucímu diplomové práce doc. Ing. Otovi Novotnému Ph.D. za jeho čas a cenné rady při vedení této práce.

Abstrakt

Diplomová práce se zabývá problémem datové integrace Big Data platformy a podnikového datového skladu. Hlavním cílem je vytvoření přenosového systému, který pomocí vhodně zvoleného nástroje bude přenášet data z datového skladu na tuto platformu a bude udržovat informace o všech realizovaných přenosech.

V teoretické části se práce soustředí na představení pojmu Big Data, stručný vývoj těchto technologií a faktory, které vedly k potřebě těchto technologií. Dále jsou představeny hlavní principy a vlastnosti těchto technologií a přínos jejich implementace do podniku. Práce popisuje také nástroje a přístupy označované jako Business Intelligence, jejich typické použití v podniku a jejich vztah k technologiím Big Data. Dílčí podkapitola se také věnuje systému Hadoop a nejpopulárnějším technologiím, které s ním souvisí.

Praktická část se věnuje konkrétní vzorové implementaci tohoto aparátu, který bude realizovat přenosy z klasické relační databáze, představující datový sklad, do clusteru několika počítačů provozujících systém Hadoop. Součástí praktické části je také přehled několika možných nástrojů, které se aktuálně používají pro nahrávání dat do Hadoopu a návrh databázového schématu metadat, které bude sloužit k řízení celého systému a udržování informací o proběhlých přenosech.

Klíčová slova

Big Data, Business Intelligence, datové sklady, Hadoop, Sqoop, datová integrace

Abstract

Master thesis deals with a problem of data integration between Big Data platform and enterprise data warehouse. Main goal of this thesis is to create a complex transfer system to move data from a data warehouse to this platform using a suitable tool for this task. This system should also store and manage all metadata information about previous transfers.

Theoretical part focuses on describing concepts of Big Data, brief introduction into their history and presents factors which led to need for this new approach. Next chapters describe main principles and attributes of these technologies and discuss benefits of their implementation within an enterprise. Thesis also describes technologies known as Business Intelligence, their typical use cases and their relation to Big Data. Minor chapter presents main components of Hadoop system and most popular related applications.

Practical part of this work consists of implementation of a system to execute and manage transfers from traditional relation database, in this case representing a data warehouse, to cluster of a few computers running a Hadoop system. This part also includes a summary of most used applications to move data into Hadoop and a design of database metadata schema, which is used to manage these transfers and to store transfer metadata.

Key words

Big Data, Business Intelligence, data warehouse, Hadoop, Sqoop, data integration

Obsah

1 Úvod

V současném světě přibývají informace a data nebývale vysokým tempem. Mezi běžnými uživateli stoupá obliba různých sociálních sítí, pomocí kterých si vyměňují zprávy, příspěvky a také jiný multimediální obsah. Díky rozšíření internetu a pokroku dalších technologií je možné k internetu připojovat stále nová zařízení a automatizovat většinu běžných úkonů. Všechna tato zařízení, sociální sítě a další aplikace tak generují obrovské množství dat, a to každý den, navíc závratnou rychlostí. Dostáváme se tak do situace, kdy na jedné straně je technologicky možné, díky zrychlování přenosových rychlostí internetu a klesající ceně místa na disku, všechna tyto data uchovávat, ale na druhé straně už není možné je efektivně zpracovávat. Klasické databázové prostředky a přístupy, které se používají už mnoho let, totiž přestávají při řešení takto objemově náročných úloh stačit.

A právě na tomto místě přicházejí ke slovu technologie souhrnně označované jako Big Data, které nabízejí nové metody a přístupy k řešení této problematiky. Big Data technologie jsou tak nástrojem, jak tyto závratné objemy dat v přijatelném čase zpracovávat, analyzovat a případně v nich hledat další užitečné informace. Data zpracovávána těmito metodami se vyznačují svým velkým objemem, rychlostí jakou vznikají a také jejich samotnou podobou. Jedná se totiž většinou o data, která bývají zpravidla málo nebo vůbec strukturovaná a jako typické příklady můžeme uvést automatická logovací data ze serverů a jiných přístrojů, textové příspěvky a zprávy ze sociálních sítí, geolokační data, popřípadě také multimediální obsah jako jsou obrázky a videa.

Jedna z hlavních aplikací těchto nových technologií se nachází v analýze dat. Společnosti nyní mohou uchovávat a zpracovávat data, které dříve kvůli jejich složité struktuře a velkému objemu přehlížela, protože klasický přístup analýzy nebyl pro tato data vhodný a efektivní. Nyní se však situace díky těmto technologiím změnila a společnosti nyní mohou v těchto datech hledat nové a cenné znalosti. Proto bude důležité tyto technologie propojit s dříve vyvinutými a používanými analytickými postupy, které jsou označovány jako Business Intelligence.

Big Data problematice je v poslední době věnována nemalá pozornost a věřím, že většina lidí, pohybujících se v profesích spojených s informačními technologiemi, alespoň okrajově slovní spojení „Big Data“ někdy zaslechla. S tím také souvisí úskalí tohoto pojmu, nejedná se totiž o pojem jasně definovaný s přesně standardizovanou terminologií. Jedná se spíše o soubor různých aplikací a přístupů, které většinou řeší rámcově stejný problém, ale často mají naprosto odlišnou dílčí aplikaci v celém systému.

Přesto, že se jedná o oblast informačních technologií, které je věnována velká pozornost a ve které se nové informace a články objevují závratným tempem, může být na první pohled těžké se v ní zorientovat.

1.1 Cíl práce

Hlavním cílem této práce je navržení a realizace přenosového systému, který pomocí vhodně zvoleného nástroje, bude přenášet data z datového skladu na Big Data platformu a bude udržovat potřebné informace o realizovaných přenosech.

Práce postupně seznamuje čtenáře s pojmem Big Data, prezentuje přehled aplikací a postupů, které se v této oblasti používají a které jsou s tímto pojmem spjaty. Následně diskutuje vztah těchto technologií s přístupy označovanými jako Business Intelligence, které jsou zaměřeny na získávání znalostí a zpracovávání z dat, které má podnik k dispozici. Práce tak prezentuje, jak je možné tyto nové přístupy pro analýzu dat propojit s těmi aktuálně používanými. V práci tedy bude uveden jak obecný teoretický přehled daných technologií, tak návrh vzorového řešení, které ilustruje jednu dílčí část integrace ekosystému aplikací Big Data s řešením datového skladu, která povede k vytvoření komplexní analytické platformy.

Práce si tak klade za úkol naplnit tyto dílčí cíle:

- V teoretické části představit, co to Big Data jsou a co tyto data typicky představují. Jaké jsou hlavní vlastnosti a principy těchto technologií a ilustrovat stěžejní postup zpracování dat pomocí těchto technologií. Součástí této části bude i představení systému Hadoop a aplikací, které s ním nejvíce souvisí.
- Popsat aplikace a přístupy označované jako Business Intelligence, jejich využití v podniku a jaký přínos pro podnik představují. Součástí kapitoly bude také vztah těchto technologií k Big Data.
- Dílčí část praktické části se bude věnovat nástrojům, které slouží k přenášení dat z externích zdrojů do Big Data platformy a výběr té nejvhodnější pro přenos dat z relačního datového skladu.
- V praktické části bude dílčím cílem navrhnuté řešení pro datovou integraci mezi Big Data platformou a klasickým datovým skladem implementovat pomocí virtualizačních nástrojů.

1.2 Struktura práce

Tato práce bude rozdělena do několika částí, které budou logicky odpovídat dílčím cílům této práce.

1.2.1 Teoretická část

- Co to vlastně Big Data jsou a jaký je jejich vztah ke klasickému zpracování dat. Přehled hlavních principů, které se aplikuji při zpracování Big Data.
- Popis aplikací a přístupů, které se nazývají Business Intelligence, jaká je jejich funkce a na jakých principech fungují. Jaké je místo těchto aplikací v podniku a jaký přínos pro podnik mají.

- Přehled hlavních dílčích technologií a implementací, které spadají do ekosystémů platformy Hadoop a k čemu se jednotlivé části používají.
- Možnosti, jakou podobu může v podniku mít centrální datová platforma vytvořená integrací Business Intelligence a Big Data nástrojů.

1.2.2 Praktická část

- Přehled nástrojů pro přenos dat mezi Big Data platformou a klasickým datovým skladem, popis jejich návrhů a funkčnosti a výběr toho nejvhodnějšího pro danou implementaci.
- Návrh architektury přenosového systému pro přenos dat z datového skladu do Big Data platformy, včetně zajištění systému metadat a řízení tohoto procesu.
- Realizace výše uvedeného návrhu tohoto aparátu, pomocí virtualizačních nástrojů, který bude demonstrovat použitelnost této systému.
- Doplňující diskuze o omezeních a vlastnostech přenosového systému a o možných způsobech dalšího rozvoje.

1.3 Očekávané výstupy a přínosy

Hlavním přínosem práce je nastínění toho, jak mohou aplikace Big Data pomoci v řešení problémů Business Intelligence a jak tyto aplikace zasadit do stávající architektury podniku společně s už provozovaným datovým skladem. Práce udává přehled základních druhů těchto technologií a přístupů a prezentuje, ke kterým úlohám je vhodné, kterou technologii využít.

V praktické části práce bude prezentována realizace navrženého systému, který bude integrovat data z datového skladu do Big Data platformy. Hlavní přínos práce tak bude v představení, jak tyto technologie vhodně využít, jaké je místo těchto technologií v souvislosti s datovým skladem a jaké jsou možnosti pro realizaci přenosů na tuto platformu.

1.4 Předpoklady a omezení práce

Protože záběr daných technologií je velice široký, práce se bude ve své praktické části soustředit pouze na jednu dílčí oblast, kterou je způsob realizace vhodného přenosu dat z datového skladu na Big Data platformu.

Praktická část předpokládá základní architekturu používaných technologií na: relační datový sklad na jedné straně a cluster počítačů provozujících systém Hadoop (doplněného o systém Hive) na straně druhé. Práce nebude tyto prvky měnit, pouze je vhodně doplňovat a propojovat.

2 Rešerše odborných zdrojů

Mnoho informací o Big Data je možné čerpat z odborných článků, kterých na toto téma existuje poměrně velké množství. Například články *Big Data: A Survey* (1) a *Big Data Analytics* (2) uvádí přehledný obecný úvod do problematiky Big Data a jsou vhodné i pro čtenáře, kteří se s touto problematikou potkávají poprvé. Prezentují základní problémy, které tyto technologie řeší, oblasti jejich aplikace, přínosy těchto technologií nebo také aktuální situaci, jakým způsobem v současném světě vzniká velké množství dat a jakou mají tato data povahu. Tyto články se snaží na danou oblast nahlížet v širších souvislostech a zasadit ji do celkového kontextu, většinou bez příliš velkého technického detailu.

Obdobnému obecnému uvedení do problematiky tohoto tématu se věnují i rozsáhlejší publikace, jako například *Big Data: A Revolution that Will Transform how We Live, Work, and Think* (3), která na jedné straně diskutuje, jak tyto technologie pozitivně ovlivní naši společnost, a na straně druhé čtenáře seznamuje s riziky nesprávného použití těchto technologií pro celou společnost.

Z publikací v českém jazyce je vhodné zmínit knihu *Big Data a NoSQL databáze* (4), která se důsledně věnuje základním principům zpracování pomocí těchto aplikací a je více náročná na technologické znalosti čtenáře. Velice dobře rozebírá princip distribuovaného zpracování úloh a problémy, které tato decentralizace s sebou nese. Velká část publikace se věnuje různým typům NoSQL databází, jejich návrhu, příkladům použití a způsobu, jak s daty pracují.

Velice užitečné jsou odborné články popisující konkrétní dílčí nástroje, které byly publikovány přímo autory těchto technologií. Tyto články se detailně věnují konkrétnímu návrhu dané aplikace a faktorům, které stály za jejím vznikem. Většinou předpokládají alespoň základní znalost dané problematiky a na těchto znalostech se snaží stavět. Studováním těchto zdrojů je možné získat náhled na to, jak různí autoři řešili obdobné úlohy nebo jak tyto technologie na sebe navazují.

Například *Hive - A Warehousing Solution Over a Map-Reduce Framework* (5) je řešení, které vzniklo ve společnosti Facebook a článek prezentuje systém pro snadnější analytické dotazování nad uloženými daty. Článek *Pig Latin: A Not-So-Foreign Language for Data Processing* (6) nabízí odlišný přístup ke stejné problematice a prezentuje nový dotazovací jazyk tentokrát ale vyvinutý ve společnosti Yahoo.

Velice důležitým článkem je *MapReduce: Simplified Data Processing on Large Clusters* (7) od autorů ze společnosti Google, který popisuje princip zpracování MapReduce. Novější technologie Spark popsaná v *Spark: Cluster Computing with Working Sets* (8) pocházející z Berkeley, zase popisuje efektivnější výpočetní přístup, který nahrazuje model MapReduce, a podobně.

Tématu Business Intelligence a datových skladů se věnuje publikace *Business Intelligence: Jak využít bohatství ve vašich datech* (9), která čtenáře seznamuje s těmito přístupy a popisuje i celý cyklus

zavedení těchto technologií do podniku, včetně jejich provozování. Jako další zdroj, který popisuje využití těchto technologií v kontextu celého podniku, výborně posloužil titul *Podniková informatika* (10).

Kvalifikační práce ostatních autorů se zabývají často aplikacemi těchto technologií v různých odvětvích a v možných konkrétních příkladech použití. Několik prací si jako téma vybralo jednu určitou dílčí technologii a následně popisují její detailní implementaci a diskutují příklady použití v praxi.

Z kvalifikačních prací, které se věnují těmto technologiím, určitě stojí za zmínku *Big Data v technológiách IBM* (11), která čtenáře nejprve seznamuje v obecné rovině s pojmem Big Data, vysvětluje základní principy, následně prezentuje vzorovou implementaci těchto technologií. K této implementaci navíc používá distribuci nástrojů poskytovaných společností IBM, čímž se práce dost vymyká obdobně zaměřeným pracím.

Práce *Hadoop NoSQL databáze* (12) je více technicky zaměřena a soustředí se hlavně na problematiku NoSQL databází. Z těchto typu technologií si autor k detailnímu popisu vybral databázi Apache HBase a čtenáři detailněji představuje práci s touto databází.

Využití Big Data v bankovním prostředí (13) a *Analýza Big Data v oblasti zdravotnictví* (14) jsou práce, které se zaměřují na konkrétní využití Big Data technologií v určitém oboru nebo dílčí oblasti aplikace. První práce prezentuje několik vzorových scénářů, které by se mohly uplatnit v praxi a na základě dotazníkového šetření porovnává jejich vhodnost pro realizaci. Druhá vysvětluje pojem Big Data z pohledu jeho aplikace ve zdravotnictví a v praktické části dokonce realizuje analýzu se zaměřením na choroby, jejíž výsledky byly konzultovány s odborníky ze zdravotnictví.

Oproti tomu práce *Nové trendy v Business Intelligence - Zaměření na Big Data a Hadoop* (15) se zase zabývá obecnějším propojením těchto aplikací s Business Intelligence. Práce sleduje vývoj posledních let v těchto technologiích a diskutuje možné použití a propojení těchto technologií v datové analytice.

3 Big Data

Samotný pojem Big Data je možné vysvětlit několika odlišnými způsoby, které na jeho podstatu nahlížejí z různého úhlu pohledu. Obecně lze tento pojem specifikovat jako nový přístup ke zpracování obrovského množství dat, které vznikají vysokým tempem, jejich podoba není pevně strukturovaná a klasické přístupy pro zpracování dat pro ně přestávají být dostačující. Tato definice určitě dobře vystihuje některé aspekty tohoto pojmu, ale pro dobrý celkový přehled, je potřeba vysvětlit problematiku Big Data v širších souvislostech. Navíc se jedná o problematiku velice aktuální, a její vývoj je v poslední době jedním z nejrychlejších na poli informačních technologií, možná i proto pro pojem Big Data najdeme v literatuře mnoho různých vysvětlení. V současné době neexistuje společná, jednotná definice a většina autorů tak při definici této problematiky akcentuje její různé oblasti (1, 2).

Proto se první část této práce bude věnovat hlavním oblastem a prvkům, které pojem Big Data zastřešuje a které jsou důležité pro pochopení co to Big Data jsou, jak se s nimi pracuje a jak těchto technologií efektivně využít. Následující kapitola popisuje, jak probíhal vývoj zpracování a uchovávání dat v historii, a jak vznikla potřeba pro Big Data technologie.

3.1 Historie zpracování dat

Ukládání a zpracovávání informací v databázových systémech provází počítače a informační technologie od jejich vzniku a je jejich nedílnou součástí. To, jak data ukládat a jak s nimi následně pracovat, bylo předmětem dlouholetého vývoje a na tomto poli se objevilo mnoho různých přístupů a technologií, jejichž modely bychom mohli stručně seřadit do následujících etap:

1) Lineární model – jedná se o úplně prvotní model pro uchovávání, který vlastně věrně kopíruje svoji reálnou předlohu – papírovou kartotéku. Je realizován jako množina jednotlivých záznamů, které jsou za sebou lineárně uchovávány a nemají mezi sebou jiné propojení než určení předchůdce a následovníka.
2) Hierarchický model – je dalším modelem, který rozšiřuje koncepci původního lineárního modelu a byl i hojně využíván v praxi. Přináší do modelu novou relaci mezi záznamy a tou je jejich propojení v roli předek-následovník. Zavádí tak hierarchickou strukturu, ve které jsou záznamy uspořádaný do grafu stromu. Přes jasné vylepšení oproti lineárnímu modelu, však tento model není dostatečně výkonný a efektivní a byl nahrazen modelem relačním.
3) Relační model – se objevil v 70. letech 20. století a je založen na matematickém aparátu množin a relací mezi nimi. Umožňuje pomocí relačních klíčů jednolitých záznamů zachytit rozličné vztahy mezi nimi a jeho popis se tak nejvíce přibližuje reálnému světu. Záznamy je tak možné mezi sebou pomocí těchto relací propojovat a propojené entity dávat do souvislostí.

4) Objektový model – poslední model je vlastně rozšíření původního relačního modelu o pojem objekt, jak je znám z objektově orientovaného programování. Rozšiřuje běžnou logiku zpracování o prvek zapouzdření do entity, aby se nejvíce podobala své předloze v reálném světě, a spolu s jejími vlastnostmi definuje i funkce, které může tento objekt vykonávat. (16)

3.2 Relační model a jeho vztah k Big Data

Relační model je tak nyní nejpoužívanějším principem při práci s daty a nabízí mnoho výhod oproti jiným modelům. Jedná se o robustní model, který je založen na pojmu matematické relace, která staví na teorii množin. Data jsou tak uspořádána do souhrnu relací, které mají podobu tabulek. Každá tabulka je zadána schématem relace, které popisuje jednotlivé její sloupce, což jsou atributy dané entity. Každá entita okolního světa je tedy záznam realizovaný jako jeden řádek v tabulce a hodnoty sloupců udávají hodnoty jejich atributů. V jednotlivých n-ticích se v jednotlivých atributech mohou také objevovat reference na jiné relace (je zde uveden identifikátor, který jednoznačně označuje záznam v odkazované tabulce.), tyto hodnoty pak spojují jednotlivé relace v různých schématech podle jejich vzájemného vztahu a vytvářejí tak systém, jak namodelovat a reprezentovat vztahy mezi jednotlivými entitami tak, jak je tomu v okolním světě (17).

Mezi další vlastnosti relačního modelu patří:

- každý řádek odpovídá jedné n-tici relace a žádné dva řádky nejsou stejné,
- pořadí řádků a sloupců je bezvýznamné,
- žádné dva sloupce (atributy) nemají stejný název,
- hodnoty ve sloupcích jsou atomické. (17)

Relační model také přináší sadu integritních a normalizačních omezení, která mají za cíl optimalizovat systém uchovaných dat a co nejlépe modelovat popisovanou realitu. Integritní omezení zajišťují například: doménovou integritu – atributy nabývají hodnot pouze platných v dané množině hodnot pro daný atribut; entitní integritu – pro každou relaci je dán primární klíč (složený z jednoho nebo více sloupců), který danou relaci jednoznačně identifikuje; referenční integritu – cizí klíč odkazující na jinou relaci je platný (17).

Normalizace si klade za cíl optimalizovat návrh databázových struktur tak, aby výsledná struktura byla co nejvíce optimalizovaná a efektivní. Postupná dekompozice těchto relací a vazeb mezi nimi navrhuje novou strukturu tak, aby byly zachovány závislosti a přitom byla odstraněna redundance dat. Proces normalizace probíhá tak, že struktura databáze je postupně optimalizována pouze pro danou úroveň normalizace a tu následně označujeme jako normální formu. Formy mají hierarchickou návaznost a každá další úroveň musí obsahovat všechny úrovně nižší. Postupnou aplikací těchto forem dostáváme návrh struktur relačního schématu do žádoucího normalizovaného stavu. (16)

Výše uvedené principy spolu s více než 40 lety vývoje těchto relačních databází přinesly ve výsledku silný a robustní aparát, na kterém lze v dnešní době stavět spolehlivé databázové aplikace. Jedním z nejdůležitějších prvků těchto relačních aplikací je silný důraz na udržení konzistence dat a podpory transakčních operací nad touto databází. Konzistence databáze říká, že různí uživatelé, kteří se nad touto databázi dotazují na určitá data a zároveň průběžně probíhá i aktualizace těchto dat, tak vždy v daný okamžik dostanou stejná, konzistentní data. Toho je dosaženo pomocí podpory transakčního zpracování, které je pro udržení konzistence v databázi kritické. Transakční zpracování staví na několika principech, pro které se v databázovém prostředí vžila zkratka ACID. Podle ní musí transakce v systému splňovat tyto vlastnosti:

- ***A**tomicity* – neboli nedělitelnost, pokud je součástí transakce několik operací, budou provedeny všechny korektně nebo žádná.
- ***C**onsistency* – zajišťuje, že při aplikaci dané transakce, databáze přejde z jednoho konzistentního stavu opět do nového konzistentního stavu.
- ***I**solation* – izolovanost transakce říká, že při jejím výkonu nebude ovlivněna jinými operacemi, které v databázi probíhají.
- ***D**urability* – trvanlivost transakce znamená, že po její aplikaci se změny opravdu do databáze promítnou a korektně uloží (3, 16).

Moderní relační systémy musí splňovat požadavky zajišťující konzistentnost dat v databázi, která je pro některé druhy aplikací naprosto kriticky důležitá. Například v systému, který spravuje finanční zůstatky na účtech, je nepřijatelná chyba, jejíž vlivem by se při přenosu peněz mezi dvěma účty peníze z jednoho účtu odečetly, ale už by nebyly korektně připsány na účet protistrany (16).

Z architektonického hlediska typický relační databázový systém funguje jako jeden server, na kterém jsou uložena veškerá data a k němu se připojují různí klienti, ať už uživatelé nebo jiné aplikace, které s daty na serveru potřebují pracovat. Databázový systém na tomto serveru tak funguje jako prostředník mezi klienty a samotnými daty a řídí veškeré operace, které s daty mají operovat. Dokáže tak řešit problémy konkurenčního přístupu více uživatelů ke stejným datům a to tak, aby byla zachována jejich správnost a konzistence. Tyto vlastnosti zajištuje právě pomocí aplikace výše uvedených přístupů. Moderní relační databáze jsou výsledkem dlouholetého vývoje světových poskytovatelů IT produktů a jsou k takovému způsobu použití uzpůsobeny a optimalizovány (17).

Právě na tomto místě začíná rozpor mezi klasickým relačním zpracováním a potřebami zpracování označovaným jako Big Data. V současné době se totiž rapidně zvyšuje objem dat, které lidstvo produkuje a které je možné zpracovávat. Udává se, že devadesát procent současných dat na internetu bylo vyprodukováno v posledních dvou letech a přírůstek dat se bude i v dalších letech nadále zvyšovat. Technický pokrok totiž přinesl nejen nové technologie a způsoby, jak data generovat, ale také jak je uchovávat a přenášet mezi jednotlivými výpočetními uzly. Kapacita počítačových disků se

stále zvyšuje a jejich cena přitom klesá. Je tedy možné uchovávat data v mnohem větších objemech a přitom stále za přijatelnou cenu. Stejně tak se pořád zvyšuje i přenosová rychlost internetu a je možné efektivně sdílet data mezi body, které geograficky dělí velká vzdálenost. Navíc dalším technologickým fenoménem poslední doby je termín „Internet of Things“, tedy vzájemné propojení a spolupráce nejrůznějších přístrojů pomocí internetové sítě. Všechny tyto prvky tak mají za následek stále větší množství produkovaných dat a k tomu, aby zpracování všech těchto dat bylo vůbec možné, slouží právě technologie Big Data (1, 18).

Existují tedy reálné možnosti, jak velké množství dat uchovávat i jak je efektivně distribuovat a zvyšuje se i počet přístrojů, které je mohou generovat. Data označované jako Big Data mají většinou tyto vlastnosti:

- objem těchto dat je obrovský, stejně tak jako rychlost, se kterou vznikají,
- jejich zdroje jsou velice rozličné a sama data mají různorodou povahu,
- data jsou málo nebo vůbec strukturovaná.

Všechny výše uvedené faktory tak vytvářejí nové úkoly a výzvy pro zpracování těchto množství dat a právě na tomto místě přestává běžný relační přístup stačit. Objemy dat jsou už tak veliké a jejich podoba tak špatně strukturalizovaná, že jejich zpracování v centrální relační databázi není proveditelné v reálně uspokojivém čase. Big Data jsou tak novými možnostmi, jak k tomuto problému přistupovat (1, 2).

Podstata tohoto zpracování je hlavně postavena na decentralizovaném propojení velkého množství uzlů, které zpravují určitou část dat a jejich kombinovaný výpočetní výkon dokáže celý soubor dat zpracovat mnohem rychleji a efektivněji než centralizovaný databázový systém. Nevýhodou decentralizace je zajištění vzájemného propojení těchto uzlů, jejich orchestrace a zajištění dostupnosti dat v tomto systému, i když už jsou data rozmístěna na více uzlech. V decentralizovaném prostředí je také mnohonásobně těžší zajistit některé principy relačního modelu jako je například zajištění transakcí, které je přitom pro relační model jednou ze základních a hlavních vlastností. Protože jsou však operace nad těmito daty většinou analytického charakteru, popřípadě zajištění konzistence není prioritou tohoto systému, je právě výhodné použití decentralizovaného přístupu. Ztrácíme tak některé vlastnosti práce s daty, které však nejsou pro naše účely zásadní, výměnou za efektivní a rychlé zpracování (3).

Big Data jsou z pohledu datových modelů a databázových systémů dalším přístupem, jak s daty pracovat, ale přesto je jejich spojitost se současně stále hojně používaným relačním modelem trochu odlišná. Oproti výše uvedeným příkladům, které postupně nahrazovaly model předcházející, u zpracování Big Data však nemůžeme jednoznačně říci, že se jedná o nové paradigma, které nahradí současné relační zpracování. To je pořád velice kvalitním a vyspělým aparátem, které má své místo v určitých úlohách a které se bude vyskytovat i nadále. Principy Big Data však hlavně nabízejí nové

přístupy, jak efektivně řešit nové specifické úlohy a jsou tak novým doplňujícím aparátem, který bude doplňovat zpracování klasickým relačním způsobem (3, 18).

3.3 Zdroje a vznik Big Data

Použití Big Data technologií je do velké míry provázáno s daty, se kterými tyto technologie pracují. Většina těchto dat má specifickou, nestrukturovanou podobu, dosahují obrovských objemů a jsou velice složité na zpracování. Tyto faktory tak byly vlastně impulzem pro vývoj Big Data technologií.

Původ těchto dat úzce souvisí s rozvojem informačních technologií a samotného internetu. Díky jejich rapidnímu rozvoji v posledních letech, je možné připojovat k internetu další zařízení a provozovat datově velice náročné aplikace. Tyto aplikace tak většinou produkují velké množství datových souborů, jejichž obsah je generován automaticky. Typickými představiteli takových souborů, se kterými Big Data technologie často pracují, jsou: logy webových serverů a aplikací, data generována senzory a různými stroji, online transakce, výsledky dotazů do vyhledávacích systémů apod. Stejně tak do této oblasti spadají další geografické, dopravní nebo zdravotnické informace (1).

Dobrým zdrojem pro Big Data jsou i sociální sítě a každodenní činnost jejich uživatelů. Tyto oblasti lidského chování generují obrovské množství obsahu v podobě různých zpráv, příspěvků, emailů a dalšího obsahu těchto sítí. Součástí zpracování tak mohou být i multimediální soubory jako jsou fotografie, videa a zvukové stopy (1).

Většina těchto informací však bude uložena v podobě nestrukturalizovaných nebo částečně strukturalizovaných textových dat. Nejčastějšími formáty pro toto zpracování jsou formáty CSV – zřetězený seznam hodnot rozlišený pouze speciálním oddělovačem, JSON – formát původně využívány na webu, který data řadí do dvojic klíč:hodnota a jednotlivé atributy hierarchicky zanořuje nebo velmi rozšířený formát pro výměnu textových informací XML – značkovací jazyk vycházející ze stejných principů jako HTML, data jsou ohraničena jednotlivými speciálními značkami, která data popisují a částečně strukturují (3).

Všechny obdobné datové soubory byly dříve společnostmi většinou přehlíženy, protože jejich informační hodnota vzhledem k jejich objemu byla nízká a jejich nestrukturovanost a heterogennost byla na zpracování příliš náročná. Právě tento fakt mění použití Big Data technologií, které nyní umožňují efektivně získávat znalosti i z těchto zdrojů. Protože většina těchto dat je dost odlišného charakteru je důležitou součástí efektivního zpracováni těchto dat i výběr té správné technologie (1, 7).

3.4 Uchovávání dat

Big Data technologie mají často odlišný přístup k uchovávání dat. Zatímco relační, transakční databáze ukládají upravená a transformovaná data v okamžiku vytvoření a pouze po potřebnou dobu, u Big Data aplikací je většinou výhodnější data uložit v původní podobě a zpravidla na mnohem delší dobu.

Transakční databáze mají také odlišný architektonický návrh a snaží se data ukládat v co nejvíce normalizované podobě a data jsou minimálně duplikována. Oproti tomu data uchovávaná v Big Data technologiích bývají často duplikována, nemají pevně danou strukturu a dokáží tak efektivněji reagovat na změny této struktury (3).

Hlavní rozdílnost ale spočívá v tom, že do relačních systémů vkládáme data už ve zpracované podobě a ve struktuře, která už se nebude měnit. V Big Data technologiích je výhodnějším přístupem data ukládat v původní nezměněné podobě a díky škálovatelnosti celého systému vlastně i na neomezenou dobu. Následné zpracování a dotazování nad daty probíhá až poté, co jsou data uložena v clusteru a tyto úlohy se tak mohou flexibilně v budoucnosti měnit. Transakční systém, tak data ukládá vyčištěná a připravená k předem známému použití, Big Data technologie ukládají data v surové podobě a až poté nad nimi staví potřebné dotazy. K uložení dat tak dochází zpravidla pouze jednou, následně už se data neaktualizují a neupravují, pouze se mění algoritmy, které je zpracovávají (3).

3.5 SQL a NoSQL databáze

Před popisem decentralizovaného uložení dat a vysvětlení decentralizovaného algoritmu jejich zpracování, je vhodné nejdříve nastínit problematiku technologií, které toto zpracování implementují. Jedná se totiž o principy tak odlišné od klasického relačního přístupu, že si zaslouží speciální pozornost.

Jak již bylo uvedeno v předcházející části práce, relační model byl po dlouhou dobu hlavním a vlastně i jediným způsobem, jak uchovávat data. Spolu s jeho návrhem vznikl také speciální jazyk, pomocí kterého se celá logika relační práce s daty implementovala. Jedná se o jazyk SQL (Structured Query Language), který se objevil ve stejné době jako relační schéma. Obsahuje příkazy jednak pro definici struktur, ve kterých budou data uložena (DDL – Data Definition Language), tak příkazy, které slouží k zapisování, mazání, aktualizování a v neposlední řadě také dotazování dat (DML - Data Manipulation Language). Tento soubor příkazů nabízí silný aparát, jak databáze vytvářet a jak s nimi i následně pracovat. Dotazování probíhá na principu propojování tabulek do složitějších relací za uživatelem zvolených podmínek a je možné jimi realizovat i velice složité dotazy. Jazyk jako takový je definován vlastní ASCI specifikací a téměř všechny komerční i open-source relační databáze používají pro komunikaci uživatele s databází jazyk SQL. Většina základních funkcí je totožná, ale existují speciální příkazy a varianty zápisu tohoto jazyku, které nejsou součástí oficiální specifikace a jsou specifikem daného výrobce databáze. Po dlouhou dobu byl jazyk SQL vlastně jediným standardem, jak pracovat s databází (17).

Přesto, že relační schéma bylo hlavní větví vývoje databází, společně s tímto přístupem vznikaly také jiné přístupy, jak uchovávat data, které ale nebyly tak populární. Většinou se jednalo o pokusné databázové systémy, které byly vyvíjeny na akademické půdě, popřípadě v malých začínajících společnostech a byly často určeny pouze k jednomu specializovanému účelu. Protože většinou

nepodporovaly relační schéma v klasické podobě a k práci s databázi nepoužívaly jazyk SQL, vzniklo pro ně souhrnné označení NoSQL databáze. (Což můžeme přeložit jako Not-only-SQL nebo Not-SQL.) (3)

Databázové systémy NoSQL pracují na úplně jiných principech, které v klasickém transakčním prostředí nebyly výhodnější než relační přístup, a tak se jejich použití hojně nerozšířilo. Ale jak začaly nabývat na důležitosti požadavky na zpracování typické pro Big Data, začalo se v této oblasti těchto databází více využívat, protože principy na kterých pracují, se výborně hodily do decentralizovaného, denormalizovaného prostředí (3, 19).

Základní typologie NoSQL databází lze rozdělit do níže uvedených, hlavních kategorií:

- databáze typu „klíč-hodnota“ – mají velice jednoduchý datový model, objekty jsou uchovávány jako hodnota, která je provázána s klíčem, který na ni odkazuje,
- sloupcové databáze – vychází z principu, že každý záznam nemusí mít vyplněny všechny sloupce, jak je tomu u relačního modelu; tato databáze tak má pružné schéma, které vyhovuje nestrukturovaným datům, je efektivní pro zpracování a je vhodné pro decentralizaci,
- dokumentové databáze – slouží primárně k ukládání dokumentů, které jsou ve formátu JSON popřípadě XML; tyto formáty samy o sobě popisuji data, která uchovávají; tyto databáze pak umožňují přistupovat k těmto dokumentům a vyhledávat v jejich obsahu (3, 19).

3.6 Decentralizace a paralelní zpracování

Jedním z hlavních principů Big Data technologií je decentralizace celého systému do většího počtu uzlů a rozdělení zpracovávané úlohy mezi tyto uzly. Zpracování této úlohy následně probíhá paralelně a mnohem rychleji, než kdyby ji zpracovával jeden server. Přes jasnou výhodu a sílu takto kombinovaného výpočetního výkonu, přináší tento způsob zpracování dat poměrně významné problémy a omezení.

Hlavní problémy tohoto přístupu spočívají v zajištění konzistence systému, dodržení dostupnosti dat a zajištění, že systém bude odolný i vůči rozpadu (pokud se síť rozpadne na několik částí, budou nadále fungovat). Výše uvedené vlastnosti jsou poměrně problematické k zajištění a decentralizovaný systém musí umět nabízet metody a funkcionalitu, jak tyto jednotlivé problémy alespoň na nějaké úrovni uspokojivě řešit (7).

Problematice, jak zajistit tyto vlastnosti systému, se věnuje takzvaný CAP teorém (zkratka ze slov: consistency, availability a partition tolerance), který říká, že v decentralizovaném systému jsme schopni zabezpečit pouze dvě ze tří výše uvedených vlastností a to na úkor vlastnosti třetí. Postupem času tento teorém neobstál v diskuzi odborné veřejnosti, a proto ho bylo nutné doplnit o další doplňující podmínky. Přesto přehledně ukazuje vlastnosti decentralizovaných systémů a ukazuje, že při použití těchto systémů musíme hledat optimální vyvážení v podpoře těchto vlastností. Bývá totiž

obvyklou praxí, že decentralizovaný systém při volbě úrovně zajištění těchto vlastností, volí nejlepší, nejvíce prospěšný a efektivní kompromis (3).

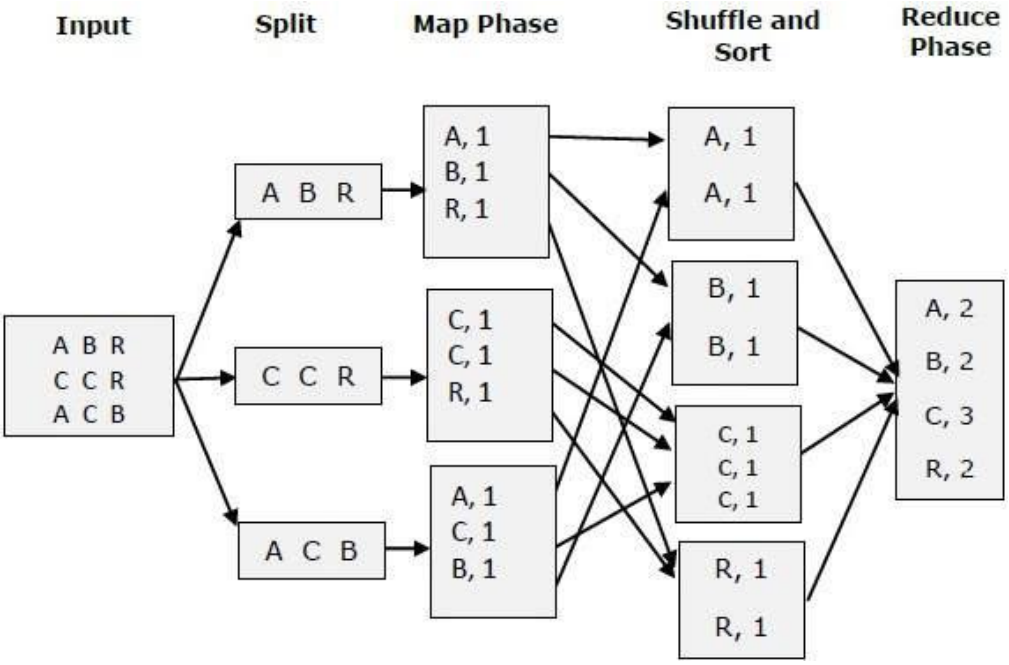
3.7 Zpracování pomocí frameworku MapReduce

MapReduce je programovací model, který je jedním z nejčastěji využívaných principů v oblasti distribuovaného zpracování pomocí technologií Big Data. Snaží se řešit problém zpracování příliš velkého množství informací pomocí rozdělení této úlohy do několika samostatných výpočetních uzlů, kde proběhne dílčí část zpracování a následné sloučení dílčích výsledků.

Pouhé zpracování těchto dat je ve své podstatě vlastně jednoduchý úkol, problém nastává až v situaci, kdy objem těchto dat naroste do obrovských rozměrů a data nejsou dobře strukturována. Zpracování pak v tomto objemu není možné dokončit v přijatelném časovém horizontu na jednom serveru a řešením je tak problém rozdistribuovat na celou síť počítačů, které pak úlohu řeší společně. MapReduce je model, který určuje jednak zpracování dílčích úkolů na jednotlivých uzlech, tak následně i spojení výsledků ze všech uzlů dohromady. Protože tento model musí pracovat v distribuovaném prostředí, je nutné, aby se dokázal efektivně vyrovnat s problémy, jako jsou výpadky sítě nebo nefunkčnost jednotlivých uzlů (7).

3.7.1 Princip zpracování

Tento programovací model byl původně hojně využíván na obrovských clusterech počítačů ve společnosti Google, která se také velkou měrou podílela na jeho vývoji. Ostatní implementace často vycházejí z tohoto původního návrhu, a tak v další podkapitole je tento model popsán tak, jak byl navržen a používán ve společnosti Google.



Obr. č. 1 Příklad zpracování MapReduce úlohy (zdroj: www.tutorialspoint.com, Map Reduce - Introduction, 2016)

Prvním krokem pro to, aby bylo možné data zpracovat, je nutné je rovnoměrně rozdělit po celé síti uzlů. Data jsou tak nejdříve rozdělena do menších bloků, které mají typicky velikost mezi 16 až 128 MB, a ty jsou pak rozdistribuovány po všech uzlech. Navíc, aby se předešlo možnému výpadku systému, když některý z uzlů nebude dostupný, jsou tyto menší bloky navíc ještě replikovány. Většinou je každý blok nakopírován na tři různé počítače v clusteru, protože výpadek jednoho počítače bývá poměrně častý jev, ale výpadek všech tří uzlů, které skladují stejný kousek dat, už je velice nepravděpodobný. Navíc pokud systém zjistí, že některý uzel selhal a není dostupný, zkopíruje veškeré bloky souborů, který daný uzel obsahoval, do jiného počítače v clusteru. Tyto bloky systém získá z ostatních uzlů, na kterých se bloky vyskytují. Je tak zajištěno, že každý blok dat bude vždy uložen právě na třech počítačích v clusteru (7).

Hlavní princip zpracování spočívá v rozdělení celého procesu do dvou samostatných operací, které se nazývají „map“ a „reduce“. Funkce map je konfigurována uživatelem systému a ze vstupních dat vygeneruje mezivýsledek v podobě: klíč – hodnota. Tato funkce je rozdělena mezi uzly systému a probíhá pak na každém uzlu zvlášť, pouze s daty, které má tento uzel na starosti. Tyto mezivýsledky jsou výsledkem algoritmu, který zadal uživatel. Jako typickou úlohu pro zpracování můžeme například uvažovat analýzu webových logů serveru a počet kolikrát bylo přistupováno k jednotlivým URL adresám serveru. Potom výsledkem operace map bude dvojice (URL adresa, počet výskytů: 1) (7).

Výsledkem je tedy velké množství těchto dvojic, které bývají často duplicitní a slouží jako vstup pro další funkci tohoto modelu, kterou je funkce reduce. Tato funkce je také specifikována uživatelem a má za úkol sloučit dohromady výsledky funkce map do menší množiny dat. Při tom hodnoty agreguje, filtruje či seskupuje, podle toho, jak byl její algoritmus nadefinován. Aplikace funkce reduce na výše uvedený příklad by tak měla za následek součet jednotlivých výskytů dané URL adresy v procházených webech, v podobě dvojice: (URL adresa, suma výskytů). Poté, co jsou veškeré dílčí výsledky ze všech uzlů sloučeny dohromady, je tento výsledek odeslán zpět uživateli. V našem případě by tak uživatel obdržel seznam URL adres a ke každé adrese počet jejich výskytů, tedy kolikrát se v analyzovaných souborech objevila (7).

Tento model funguje na „master-slave“ principu, kdy některé vybrané uzly jsou nadřazeny ostatním uzlům a rozdělují mezi ostatní uzly dílčí úkoly a řídí tak zpracování celé úlohy. Podřízené uzly pouze dostávají zadání od hlavního uzlu a vracejí mu své dílčí výsledky. V tomto distribuovaném prostředí je tak master zodpovědný za přiřazování jednotlivých map a reduce úkolů jednotlivým podřízeným uzlům a to nejlépe tak, aby žádné uzly nebyly nevyužity a aby jejich zatížení bylo rovnoměrné. Hlavní uzel tak se tak snaží maximalizovat výkon celého clusteru (7).

Na velkém clusteru počítačů dochází pravidelně k výpadkům jednotlivých počítačů a tak cluster musí umět efektivně tento problém řešit. Hlavní uzel má informace o všech uzlech, které v clusteru pracují a to včetně toho, jaké bloky jsou na tomto uzlu uloženy, jakou úlohu právě uzel zpracovává a v jakém je

stavu. Hlavní uzel se na stav podřízeného uzlu periodicky dotazuje, a pokud není možné získat zpětnou odpověď, je prohlášen za chybný uzel. V tom případě je úloha, kterou tento uzel zpracovával označena jako nedokončená a řídicí uzel se snaží najít jiný funkční uzel, který není momentálně vytížen a který by mohl tuto úlohu dokončit. Celý cluster je tak neustále kontrolován a úlohy jsou flexibilně přiřazovány uzlům tak, aby celé zpracování bylo co nejefektivnější. Nevýhodou této architektury je fakt, že hlavní uzel je kritické místo celého systému, na kterém je běh tohoto systému závislý. Tento problém je možné řešit několika způsoby. Jedním z nich je periodické ukládání aktuálního stavu hlavního uzlu a v případě poruchy je možné označit za mastera jiný výpočetní stroj, ten pak přebere databázi posledního uloženého stavu a pokračuje od tohoto bodu. Dalším možným způsobem, který byl využíván právě ve společnosti Google, je předpoklad, že pravděpodobnost pádu řídicího uzlu je tak nízká, že tento stav nebude speciálně ošetřován a pokud nastane, výpočet celé úlohy se zastaví a po obnovení master uzlu se spustí znovu (7).

Výpočetní uzly, které zpracovávají data, nejsou pro velké společnosti finančně náročně zařízení, ale jedná se většinou o běžný komoditní hardware, který je poměrně lehce dostupný a snadno vyměnitelný. Doporučuje se pouze zvýšená konfigurace několika hlavních master uzlů, protože jejich úloha v clusteru je nejdůležitější. Protože jednou z důležitých částí tohoto výpočtu je komunikace po interní sítí, která nemá neomezenou propustnost, je dobré tímto zdrojem zbytečně neplýtvat. Master uzel tak dokáže vzít v potaz fyzické zařazení jednotlivých počítačů v síti a podle toho upravit rozdělení úloh v clusteru. Pokud to je možné, snaží se úlohy dělit tak, aby probíhaly na stroji, který obsahuje odpovídající repliky daných dat, popřípadě se je snaží přiřadit uzlu, který s tímto uzlem sdílí stejnou lokální síť. Výsledkem je pak minimální zatížení připojení po internetu, protože velké množství úloh je realizováno v rámci jedné lokální sítě nebo dokonce přímo na jednom uzlu (7).

Celá tato implementace klade velký důraz na to, aby veškerá koordinace, paralelizace úloh, zotavení z chyb a celé řízení výpočtu na clusteru, bylo co nejvíce automatizované a aby koncový uživatel byl od něj co nejvíce odstíněn. Finálním výsledkem je tak jednoduché rozhraní, které slouží uživateli pro výpočet i velice složitých úloh a které je přitom dostatečně silné a flexibilní. Uživatel se soustředí pouze na samotnou analytickou úlohu a systém samotný se stará o to, aby rovnoměrně rozděloval úlohy, agregoval dílčí výsledky a řešil problémy s dostupností jednotlivých uzlů systému (7).

3.7.2 Nevýhody modelu MapReduce

Využití frameworku MapReduce, přes všechny svoje výhody a přínosy, není vždy přijímáno kladně. Mezi hlavní nevýhody patří například fakt, že ústřední myšlenka tohoto algoritmu stojí hlavně na využití hrubé výpočetní síly celého clusteru a nevyužívá ostatní optimalizační techniky, které se běžně používají v jiných systémech. Některé složitější dotazy a spojování dílčích výběrů z různých souborů, nemusí být prováděno vždy efektivně a často dochází k velkému kopírování mezivýsledků. Tato

myšlenka se navíc neobjevila někdy v posledních letech, ale jedná se o algoritmus známý už více jak dvacet let, pouze technický pokrok dalších komponent umožnil její využití v tak velkém měřítku.

Úlohy, které mají zpracovávat data v clusteru, jsou většinou napsány ve vyšších programovacích jazycích, jako je Java nebo Python. Napsat opravdu složité úlohy není triviální a oproti jiným způsobům dotazování nad databází, je tento způsob méně efektivní a více časově náročný. Navíc znalost těchto jazyků také klade větší nároky na koncové uživatele tohoto systému (3).

Mezi další nevýhody tohoto frameworku patří to, že úlohy, které efektivně řeší, jsou poměrně specifické a musí být pro použití frameworku vhodné. MapReduce není univerzálním řešením na každý problém zpracování dat a měl by být použit pouze tam, kde to bude mít smysl. V mnoha jiných problémech je dobré se držet staršího, ale v daném okamžiku efektivnějšího relačního zpracovávání dat (3).

Poslední nevýhodou tohoto systému je také vysoká složitost a odborná náročnost správné konfigurace celého systému. Většina parametrů a nastavení není vždy na první pohled zřejmá. Následný provoz tohoto systému a provádění úkolů jako je optimalizace, nasazování nového softwaru a opravování chyb, v prostředí, které je distribuované na velké množství jednotlivých počítačů, je tak náročný úkol (3).

3.8 Přínosy a aplikace Big Data

Hlavní přínos Big Data technologií je odlišný přístup k celé problematice zpracování dat a možnost řešit takové typy úloh, které předtím nebyly standardními prostředky řešitelné. Poskytují tak celý nový zdroj informací, který může razantně vylepšit konkurenceschopnost organizací. Ty pak mohou disponovat mnohem podrobnějšími a detailnějšími analýzami. Oblasti aplikací těchto technologií je možné nalézt ve zdravotnictví, výrobních procesech nebo administrativě veřejného sektoru. Pro společnosti mohou být Big Data možností jak doplnit a vylepšit jejich stávající Business Intelligence platformu (1, 2).

Big Data umožňují:

- zlepšení plánování a odhadů,
- detailnější analýzu aktuálního stavu procesů v samotném podniku,
- přesnější a efektivnější detekce příležitostí na trhu,
- mnohem lepší zacílení marketingu podle chování uživatelů,
- přesnější segmentaci zákazníků,
- vyhodnocování podnikových procesů v reálném čase (odhalení podvodu, schvalovací procesy, apod.) (2).

Oblasti konkrétních aplikací těchto technologií jsou velmi široké a podle (1) je možné je rozdělit do těchto hlavních kategorií:

- Analýza strukturovaných dat – Mnoho současných aplikací používaných ve společnostech produkuje velké objemy dat, které přesto, že jsou často zpracovávány pomocí Business Intelligence nástrojů, jsou vhodné i pro analýzu pomocí Big Data. Nový přístup umožňuje na těchto datech uplatnit nové a mnohem náročnější algoritmy.
- Analýza textu – Zpracování velkého množství textových informací, které společnost uchovává v dokumentech, emailech, webech a dalších zdrojích. Toto zpracování se soustředí na nalezení dalších užitečných znalostí a často se v této oblasti používají algoritmy pro zpracování přirozeného jazyka.
- Analýza webového obsahu – Tuto kategorii je možné dále rozdělit do tří dalších podkategorií: analýza obsahu – co daná stránka obsahuje a jak je možné z toho získat užitečné informace; analýza struktury – jak jsou stránky a odkazy v nich provázány a co tyto vazby představují; analýza použití – jak uživatelé k těmto stránkám v průběhu času přistupují a jak stránek využívají.
- Multimediální analýza – Soustředí se na analýzu zdrojů, které mají podobu obrazu, videa nebo zvukových stop a obsahuje mnoho rozličných úloh jako například automatický popis a kategorizace těchto zdrojů nebo rozpoznávání určitých vzorů v těchto datech.
- Analýza sociálních sítí – umožnuje sledování chování uživatelů těchto sítí a vztahy mezi nimi. Je možné analyzovat jejich chování, vyměňovaný obsah s jinými uživateli a mnoho dalších informací.
- Analýza mobilních dat – rozšíření mobilních telefonů a jejich možnosti neustále stoupají a také generují velké množství dat. Tato data jsou navíc oproti jiným typům dat poměrně odlišná, protože navíc mohou obsahovat informace jako je například poloha nebo data ze senzorů telefonu. Tyto faktory otevírají prostor pro naprosto nové typy úloh.

4 Datové sklady a Business Intelligence

Big Data jako technologie umožňuje zpracovat velké množství dat a hledat v nich další užitečné informace a souvislosti, které může společnost využít k svému prospěchu. Obdobný úkol si také berou za cíl technologie a přístupy označované jako Business Intelligence, jejichž historie je starší než používání technologií označovaných jako Big Data. Přesto, že v některých ohledech jsou oba tyto světy odlišné, je možné je oba propojit a využít tak jejich kombinovaného potencionálu. Tento úkol je však poměrně složitý a při jeho řešení je třeba odpovědět na mnoho náročných otázek. Následující kapitola se bude věnovat základním principům a nástrojům, na kterých je Business Intelligence řešení postaveno.

4.1 Business Intelligence

Tento termín se poprvé objevuje až v roce 1989 přesto, že nástroje a principy, které dnes zastřešuje, se začaly vyvíjet už koncem 70. let 20. století. Jedná se o specializovanou oblast informatiky a podnikových systémů, která si klade za cíl řešit manažerské a plánovací úlohy, podporuje manažerské rozhodování převážně využitím interních informací a dat. Termín jako takový obsahuje rozličné množství někdy i velice rozdílných technologií, přístupů a dílčích aplikací. Výsledná správná implementace této platformy v daném podniku je tak kombinací různých nástrojů a technik tak, aby co nejlépe vyhovovala potřebám daného podniku a aby co nejefektivněji podporovala jeho rozhodovací procesy. Nástroje Business Intelligence nahlíží na podnik jako na celek a snaží se výsledná data prezentovat vždy z úhlu, který je aktuálně potřeba. Řešení dokáže poskytovat potřebná data uživatelům na všech úrovních podniku, od různých reportů pro operativní manažery až po komplexní analytické služby, poskytované vrcholovým manažerům podniku.

Tak, jak v posledních letech stále narůstá konkurence v tržním prostředí, a jak se zvyšují nároky na organizace a jejich manažery, je stále důležitějším faktorem pro úspěch podniku, disponovat co nejvíce aktuálními, komplexními a správnými informacemi. Tyto informace dokáží efektivně podporovat manažery při výběru toho nejlepšího rozhodnutí a mají tak ve svém důsledku zásadní podíl na úspěchu dané společnosti. Business Intelligence je tedy nástroj, jak mít tyto informace při rozhodování k dispozici, a proto se také jedná o oblast informačních technologií, která se v současné době těší velké pozornosti a která je vnímaná společnostmi jako strategická oblast pro jejich rozvoj (9, 10, 20).

4.2 Oblasti aplikace

Oblasti, ve kterých je možné řešení Business Intelligence aplikovat, se nacházejí vlastně ve všech částech podniku.

4.2.1 Finance

Jednou z nejvýznamnějších je oblast financí a hospodaření s prostředky podniku. BI nástroje umožňují prezentovat finanční výkonost podniku a to včetně rozdělení na jeho jednotlivá oddělení, výrobní jednotky nebo nákladová střediska. Je tak možné identifikovat, které oddělení čelí finančním problémům a včas přijmout patřičná opatření.

Nástroje nacházejí uplatnění v sestavování finančních plánů a prognózování dalšího finančního vývoje podniku nebo v konsolidaci finančních výkazů a jejich prezentaci odpovědným zaměstnancům. Je také možné efektivně tvořit reporty, které vyžadují různé externí organizace, které regulují podnikatelský sektor a jejichž tvorba by byla v transakčních systémech mnohem těžším úkolem. Neposledními úkoly může být důkladná analýza ziskovosti podniku a jeho nákladů nebo řízení rizik podniku (10).

4.2.2 Marketing

Marketing je v oblasti Business Intelligence stále důležitější oblastí, kde se tyto technologie aplikují. Hlavní myšlenkou této aplikace je lepší porozumění zákazníkům dané organizace a důkladnému vyhodnocování marketingových kampaní. Je následně možné nové kampaně lépe zaměřit na cílové zákazníky a dosahovat tak lepších výsledků.

Business Intelligence tak pomáhá lépe definovat segmenty zákazníků a jednotlivé zákazníky správně zařadit, vyhodnocovat marketingové kampaně a vyhodnocovat úspěšnost jednotlivých produktů v rámci daných klientských segmentů (10).

4.2.3 Výroba

Ve výrobním procesu je možné sledovat mnoho klíčových ukazatelů jako je doba dodávky výrobků, stavy zásob a výrobků, výkonnost jednotlivých fází výrobního procesu. Je tak možné zkoumat trendy jednotlivých částí procesu na základě historických dat a automaticky upravovat výrobní plán podle aktuálních potřeb (10).

4.2.4 Logistika

Aplikace v této oblasti sledují náklady jednotlivých dopravců, dodržování jejich termínů a míru, jak často se u nich objevuje poškození zboží nebo reklamace. Na základě toho je možné optimalizovat složení využívaných dopravců a snížit náklady na logistiku (10).

4.2.5 Lidské zdroje

BI technologie pomáhají sledovat informace o zaměstnancích v podniku, prezentovat jejich kompetence a schopnosti, sledovat plány jejich profesního rozvoje. Další oblastí může být sledování výkonnosti jednotlivých zaměstnanců, jejich nákladů a které lidské zdroje a profesní kompetence jsou v podniku aktuálně potřeba (10).

4.2.6 Webová analýza

Jednou z aplikací, která je v poslední době hodně populární, je analýza chování uživatelů na webové stránce společnosti. Jedná se o sledování toho, jací uživatelé používají tento informační kanál, které dílčí stránky na tomto webu navštěvují a jak vypadá chování jednotlivých klientských segmentů v tomto kanálu. Tato analýza pomáhá optimalizovat podobu webu tak, aby uživatelé mohli lehce najít hledaný obsah a práce s tímto webem pro ně byla co nejpříjemnější (10, 11).

4.3 Transakční a analytické zpracování

Analytické úlohy jsou většinou naprosto odlišné povahy než úlohy, které se objevují během běžného provozu podniku a také jejich řešení vyžaduje jiný přístup a technologie. Dochází tak k rozdělení do dvou hlavních oblastí, které se označují jako transakční a analytické zpracování. Obě úlohy sice operují nad stejnými daty, ale liší se povaha jejich výstupů a také odběratelů, kterým jsou tyto výstupy určeny.

Podnik používá ke svému chodu mnoho různých transakčních systémů. Tyto systémy slouží hlavně k zajištění běžného operačního provozu podniku a jejich hlavním úkolem je správně a rychle realizovat a ukládat transakce, které během provozu podniku vznikají. Tyto databázové systémy jsou tak postaveny zpravidla na klasickém relačním databázovém modelu, který je vhodně normalizován. Výsledná normalizovaná databáze tak dokáže velice rychle provádět změny a udržovat stav datových objektů tak, aby co nejvíce odpovídaly stavu reálných entit, které má popisovat, a tento stav se také může za den několikrát změnit (9).

Podstata těchto systémů tak spočívá v jejich pružné reakci na měnící se okolí a v co nejrychlejší poskytování detailních dat uživateli. Pro toto zpracování se používá termín „OnLine Transaction Processing“ neboli OLTP.

V průběhu času se začaly zvyšovat počty systémů, se kterými daný podnik operuje a které potřebuje ke svému provozu a stejně tak se začaly i zvětšovat objemy dat, které jsou v těchto systémech uloženy. Navíc většina těchto systémů je zaměřena pouze na úzce specializovanou oblast a má k tomu odpovídajícím způsobem navržený i datový model. Pokud tedy budeme blíže zkoumat jaké informace a atributy jsou uloženy pro daný objekt reálného světa v jednotlivých systémech, můžeme přijít na to, že každý systém akcentuje rozdílné vlastnosti podle toho, jaká je jeho hlavní oblast zaměření. Dalším problémem může být také odlišné pojmenování jednotlivých domén, rozdílné použití datových typů a celkově odlišného architektonického návrhu dané databáze (9).

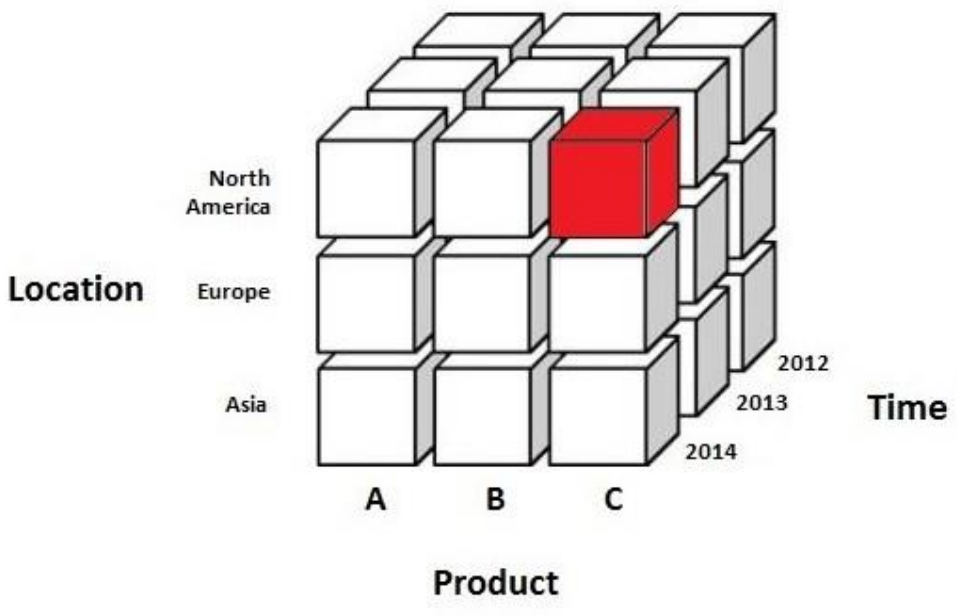
V takovém komplexnějším prostředí je stále těžší zodpovědět analytické otázky, které jsou potřebné pro správnou funkci rozhodovacích procesů. Transakční systémy slouží hlavně k rychlé reakci a poskytování dat na detailní úrovni granularity. Oproti tomu analytické úlohy mají za úkol prezentovat data v agregované podobě přes několik různých oddělení, z několika různých transakčních systémů.

Častým požadavkem je také zobrazení daných dat z různých úhlů pohledu, jako je například výsledek pro danou geografickou oblast nebo výrobní jednotku a tyto pohledy mezi sebou pružně měnit. Důležitým požadavkem je také zobrazení dat nejen v daném okamžiku, ale zobrazení vývoje dané metriky v čase a zobrazení historického trendu (9, 21).

Tyto úlohy jsou sice řešitelné i v prostředí transakčních systémů, ale protože tyto technologie nejsou primárně určeny k těmto úlohám, bylo by toto řešení málo flexibilní a těžkopádné. Transakční systémy jsou totiž zatěžovány kontinuálně menšími úlohami, které musí řešit co nejrychleji a analytické dotazy, které by data agregovaly přes několik různých oblastí, by byly na tyto systémy příliš časově náročné. BI technologie jsou proti tomu zaměřeny právě na zpracování těchto složitých dotazů, které jsou většinou předpočítány dopředu a to včetně dílčích mezivýsledků pro různé dimenze. Koncový uživatel tak může velice pružně měnit zadání analytické úlohy podle požadované dimenze a výsledná data jsou mu poskytnuta mnohem rychleji, než kdyby se při každé drobné úpravě dotazu musela data spočítat znovu, jak by tomu bylo v prostředí transakčního systému. Pro tento druh analytického zpracování dat se používá označení „OnLine Analytical Processing“, zkráceně OLAP. Toto řešení je postaveno na takzvaném multidimenzionálním pohledu na data, který bude vysvětlen v další dílčí podkapitole (9, 21).

4.4 Multidimenzionální databáze

Hlavní přínos použití multidimenzionálních databází spočívá v pružnosti, s jakou toto řešení dokáže měnit úhel pohledu na zkoumaná data a umožňuje tak velice efektivní analýzu dat. Stěžejním principem je uložení dat v hlavní multidimenzionální tabulce, která je propojená na několik dimenzionálních tabulek. Tyto dimenze představují různé úrovně, ze kterých je možné na data nahlížet. Můžeme tak mít časovou dimenzi, která zobrazuje data pro jednotlivé měsíce, kvartály nebo roky, popřípadě používáme geografickou dimenzi, která umožňuje sledovat data v kontextu daného města, kraje nebo celého státu. Uživatel při práci s touto databází podle své potřeby upravuje nastavení těchto dimenzí a dostává odpovídající agregované data pro úroveň detailu podle jeho požadavků. Pro ilustraci práce s tímto typem databáze se pro jednoduchost reprezentuje v podobě kostky, jejíž jednotlivé strany představují její datové dimenze. Výsledná hodnota hledaná uživatelem leží v průsečíku všech hodnot dimenzí (9).



Obr. č. 2 Grafická reprezentace multidimenzionálního pohledu na data (zdroj: openit.com, Faster Analysis with OLAP, 2014)

Výše uvedený obrázek tak představuje objem prodaných produktů typu C v severní Americe za rok 2014. Multidimenzionální databáze tak obsahuje agregované hodnoty pro jednotlivé úrovně dimenzí a pro jejich kombinace. Tyto hodnoty jsou vypočteny dopředu a je tak možné pružně měnit zadání úlohy.

Tento datový model je totiž odlišný od klasického relačního normalizovaného návrhu. Data v něm totiž nejsou uložena pouze v normalizované podobě, ale pro různé úrovně agregace obsahují duplicitní informace. Přepočítávat jednotlivé agregace vždy znovu, během práce s těmito daty, by trvalo zbytečně dlouho a nebylo by dostatečně efektivní (9, 10).

4.5 Základní vrstvy BI řešení

Zpracování OLAP úloh probíhá v několika krocích, které na sebe navazují. Tyto kroky je možné, podle jejich zaměření, rozdělit do několika hlavních vrstev. Každá z těchto vrstev má dosti specifický účel a používá specializované technologie. Sloučením jejich funkce je tak možné získat data ze zdrojových systémů, vhodně je zpracovat a následně přehledně prezentovat koncovým uživatelům v podobě přehledných tabulek a reportů. Finální uživatelé těchto reportů tak jsou odstíněni od interní implementace tohoto řešení a nemusí ani ovládat dovednosti nutné pro práci s daty v databázovém systému. Zpracování dat je možné rozdělit do níže uvedených vrstev:

- vrstva zdrojových systémů – jedná se o transakční systémy podniku, které obsahují data, nad kterými pak probíhá datová analýza. Jsou zdrojem informací pro celé Business Intelligence řešení a jsou často označovány jako primární zdroje, sloužící k pořizování dat.

- ETL vrstva – tato vrstva si klade za cíl přenos dat ze zdrojových systémů, provedení patřičných úprav a následné uložení do struktur vhodných pro analytické zpracování. Zkratka vznikla ze složení počátečních písmen slov: Extraction, Transformation a Load, tedy jednotlivé dílčí části tohoto procesu. Druhou variantou tohoto procesu zpracování označované jako ELT, tedy jednotlivé fáze procesu jsou v odlišném pořadí a poté, co jsou data získána ze zdrojového systému, jsou hned uložena v cílové oblasti. Fáze transformace a úpravy tak probíhá v databázovém prostředí, kde budou uloženy i výsledné struktury. Volba, kterou z těchto dvou variant vybrat, je závislá na konkrétní implementaci dané BI platformy.
- vrstva pro uchovávání dat – jedná se o technologie, které uchovávají data ve vhodné podobě pro další analytické zpracování. Mezi tyto technologie patří databázové systémy označované jako datové sklady, datová tržiště, operační datové sklady a další. Datovým skladům bude věnována další podkapitola.
- vrstva pro analýzu dat – v této vrstvě dochází k přípravě dat pro finální prezentaci výsledných dat uživatelům. Jsou zde vyčítány agregované datové objekty a propojení těchto dat k odpovídajícím hodnotám dimenzí. Připravují se zde reporty a ostatní objekty, které pak budou sloužit uživatelům pro přehled informací a rychlé dotazování nad daty. Další úlohou této vrstvy je také příprava dat pro data mining, což je specializovaná úloha, která si klade za cíl hledání dosud neznámých souvislostí a informací ve velkých objemech dat.
- vrstva pro prezentaci dat – poslední vrstvou je prezentace napočtených dat koncovým uživatelům. Data mohou být poskytována přes webové rozhraní nebo přes specializované aplikace. Hlavním cílem těchto aplikací je vytvořit příjemné uživatelské prostředí, se kterým budou uživatelé pracovat a které bude efektivně jejich práci podporovat. Aplikace dokážou vytvářet také přehledné grafické sestavy nebo sestavovat grafy podle uživatelských požadavků (9).

4.6 Přínosy Business Intelligence pro podnik

Jak již bylo řečeno výše, jednou z hlavních Business Intelligence je rychlé a flexibilní řešení analytických úloh, které mají významnou roli v rozhodovacích procesech podniku. Tyto technologie umožňují přehledný a komplexní pohled na podnik i jeho dílčí části. Je tak možné rychle identifikovat vznikající problém a včas podniknout potřebná opatření pro to, aby se zamezilo negativním dopadům na podnik. Manažeři společnosti, která využívá BI technologie, mají při svých rozhodnutích dostatek podkladových informací a mohou tak zvolit správné řešení.

Důvodem, proč mohou tyto technologie kvalitně podporovat rozhodování v podniku je to, že integrují data z různých informačních systémů a snaží se hledat odpovědi na otázky v kontextu celého podniku. Dokážou také detailně analyzovat výkonnost jednotlivých procesů nebo úspěšnost jednotlivých produktů. Je tak možné optimalizovat tyto procesy, případě upravovat produktové portfolio.

V neposlední řadě je velkým přínosem těchto nástrojů zvýšení datové kvality. Informace o jednom subjektu mohou být uloženy v několika různých systémech, a protože je náročné udržet všechny tyto informace synchronizované a aktuální, stává se poměrně často, že tato data se napříč různými systémy liší. Nástroje Business Intelligence mohou sloužit jako centrální místo, které bude všem okolním systémům poskytovat aktuální data. Vyšší datová kvalita umožňuje lepší funkci celé společnosti a menší chybovost jejich procesů (9, 10, 20).

4.7 Datové sklady

Datové sklady jsou speciální formou databázového systému, jehož primárním cílem je uchovávání velkého množství dat z různých datových zdrojů a provádění velkých analytických dotazů nad těmito daty. Zdrojem těchto dat většinou bývají ostatní transakční systémy podniku. Protože celý datový sklad slouží jiným uživatelům a je určen pro jiné typy úloh, bývá odlišná i jeho struktura. Většinou jsou data rozdělena do dílčích databází, které odpovídají dílčím oblastem, které popisují. Tyto databáze obsahují faktové tabulky, které udržují sledované hodnoty a tabulky dimenzí, které uchovávají strukturu jednotlivé úrovně dimenzí. Přesto, že data v datovém skladu mohou být v normalizované podobě, není to pravidlem a pro některé úlohy je výhodnější data ve skladu denormalizovat. Celý tento databázový systém je většinou centralizovaný a využívá relačního zpracování dat (20, 22).

Zpracování v datovém skladu probíhá většinou v jednodenních intervalech většinou v době, kdy není sklad uživateli aktivně využíván. Data z externích systémů jsou přenesena, transformována a uložena do struktur datového skladu. Data jsou poté předpočítávána do agregovaných podob, které budou nejvíce vyhovovat OLAP úlohám. Celý tento proces není triviální úlohou a většinou trvá i několik hodin. Následně jsou transformovaná data připravena pro uživatele skladu. Je kladen velký důraz na jednotný časový údaj, pro který jsou data vypočtena a pro který jsou platná (22).

4.7.1 Principy uložení dat v datovém skladu

Protože celý datový sklad funguje na jiných principech, je odlišný i způsob jakým ukládá data. Tato data bývají:

- integrovaná – Data jsou ukládána v kontextu celého podniku a tak pro propojení dat z různých informačních zdrojů je potřeba je transformovat do jednotné podoby.
- subjektově orientovaná – Důležitým faktorem je podstata samotných dat a ne z jakého systému data pochází a jaká aplikace je vytvořila. V datovém skladu tak je cílová tabulka pro jednu entitu reálného světa, přestože pochází z několika dílčích transakčních systémů.
- časově orientovaná – Protože analytické úlohy jsou často závislé na časovém vývoji a sledování vývoje trendů v čase, tak datový sklad klade velký důraz na udržování celého

historického vývoje těchto dat. Oproti tomu transakční systémy většinou udržují pouze aktuální stav záznamů.

- neměnná – Z principu povahy úloh, které nad datovým skladem probíhají, není běžné data v datovém skladu měnit, ale pouze nad nimi zpracovávat dotazy. Data se tak do skladu uloží zpravidla pouze jednou a dále nejsou upravována (10).

4.7.2 Vývoj a architektura datového skladu

Jednou z typických architektonických vzorů, jak může vypadat datový sklad je jeho rozdělení podle několika fází zpracování. První úrovní bývá přípravná oblast pro uložení dat, které se přenesly ze zdrojových systémů. Dochází zde pouze k základní úpravě a filtrování dat a většina dat je přenesena v podobě, v jaké jsou uloženy ve zdrojovém systému. Další fází je uložení do takzvané integrační vrstvy datového skladu. Zde už jsou data uložena v požadované transformované a integrované podobě. Poslední fází je výpočet dílčích datových tržišť v prezenční vrstvě, což jsou specializovaná databázová schémata odpovídající dílčí analyzované oblasti. Nad těmito kontejnery poté operují další nástroje Business Intelligence, které data zobrazují koncovým uživatelům.

Vývoj datového skladu může probíhat několika možnými způsoby, jedním z nich je přístup pomocí tzv. „velkého třesku“. Tento přístup si klade za cíl analyzovat potřeby celého podniku najednou a následně navrhnout kompletní architekturu celého skladu. Výsledkem je nasazení komplexního řešení, nevýhodou vysoká složitost tohoto přístupu a velká pravděpodobnost neúspěchu. Další přístupy stojí na postupném přírůstkovém rozšiřování datového skladu. Jeden označujeme jako přístup „shora dolů“ a probíhá v analýze celého podniku a rozložení na dílčí podproblémy, které jsou následně implementovány. Tento postup je poměrně spolehlivý, ale může se časově prodloužit. Druhý způsob se označuje „zdola nahoru“ a je založen na implementaci dílčích problémů, které jsou následně integrovány dohromady. Výhodou je detailnější a důkladnější návrh dílčích řešení, nevýhodou je někdy náročnější integrace těchto řešení (9, 22).

4.7.3 Přínos datových skladů

Datové sklady hrají významnou roli v celém řešení Business Intelligence. Mají významný integrační přínos pro celý podnik a je to jeden ze základních stavebních kamenů, nad kterými stojí další BI technologie. Přesto, že jsou optimalizovány pro výpočet velice složitých dotazů, nemusí z povahy své architektury stačit řešit veškeré analytické potřeby podniku, proto je stále důležitější otázkou, jak efektivně využít kombinovanou sílu datových skladů a Big Data technologií (22).

5 Projekt Hadoop a doprovodné projekty

Další část tohoto textu se bude věnovat konkrétní implementaci nástrojů, které dokážou zpracovávat velké množství informací a slouží hlavně k analýze dat. Hadoop je jedním z nejvýznamnějších představitelů těchto technologií. Tento projekt má už několikaletou historii a spolu s ním se vyvíjí i mnoho příbuzných projektů, které s tímto projektem fungují a jeho funkčnost rozšiřují či doplňují. Dohromady tak umožňují vytvořit komplexní Big Data platformu, která je připravena řešit mnoho analytických problémů.

5.1 Apache Hadoop

Hadoop je stěžejním projektem těchto technologií a jedním z nejaktivněji vyvíjených projektů organizace Apache. Jak již název Apache napovídá, Hadoop je vyvíjen jako open source a je tak možné ho provozovat zdarma. Historie tohoto projektu začíná už v roce 2003, kdy byly publikovány články zaměstnanců společnosti Google, které se věnovaly zpracování velkého množství dat v distribuovaném prostředí velkého počítačového clusteru a konceptu Google File System, což byl speciální souborový systém určený pro toto distribuované prostředí. Tyto články podnítily další vývoj a pod záštitou společnosti Apache se začal vyvíjet projekt Nutch, ze kterého se po určité době vyvinul projekt Hadoop. Tento projekt vydal svou první verzi v roce 2006 a od té doby si získal mnoho příznivců a přispěvatelů (23).

Jedná se o výpočetní framework složený z několika dílčích knihoven a modulů, který slouží ke zpracovávaní velkého množství dat v distribuovaném prostředí. Je koncipován tak, aby tento výpočetní cluster byl složen z běžných komoditních počítačů a aby byl automatizovaně schopen řešit problémy výpadku jednotlivých uzlů v tomto clusteru. Projekt se dělí do několika hlavních částí, kterým se budou věnovat následující podkapitoly.

Pro zajímavost: tento projekt byl pojmenován po plyšové hračce v podobě slona, který patřil synovi jednoho z autorů projektu. Proto se také i slon stal oficiálním maskotem tohoto projektu.

5.1.1 Hadoop Common

Základní knihovna, která obsahuje důležité abstrakce potřebné pro chod ostatních částí celého projektu. Pro nasazení celého výpočetního clusteru je tento modul nezbytný.

5.1.2 MapReduce

Implementace MapReduce výpočetního modelu v prostředí aplikace Hadoop. Používá klasický výpočetní model MapReduce, který byl již v této práci prezentován. Poskytuje potřebné knihovny a funkce, které používají uživatelé při definování jejich MapReduce aplikací (23).

5.1.3 HDFS

Tato zkratka znamená „Hadoop Distributed File System“, což je modul, který vytváří distribuované prostředí, nad kterým pak probíhá další zpracování. Právě tato komponenta zajišťuje odolnost vůči chybě jednotlivých uzlů a to tím, že jednotlivé soubory rozděluje do dílčích bloků a ty pak distribuuje po celém clusteru a vhodně je replikuje. Práce s tímto systémem je intuitivní a ve velké míře odpovídá práci s jinými souborovými systémy a používá i podobné příkazy jako linuxové systémy (24).

5.1.4 YARN

Jedná se o komponentu, která v prostředí Hadoop clusteru řídí a spravuje zdroje, kterým je možné přidělovat úlohy k výpočtu, sleduje jejich zatížení, koordinuje jednotlivé uzly a řídí celý životní cyklus jednotlivých výpočetních úloh. Důležité je oddělení těchto organizačních úkonů do samostatného výpočtu. (25)

Tyto čtyři komponenty dohromady tvoří jádro výpočetního systému Hadoop a po jejich konfiguraci je možné Hadoop používat ke zpracování dat. Provozování takového clusteru by ale bylo velice náročné, jednak pro samotné administrátory tohoto systému, ale i pro koncové uživatele. Práce s daty a tvorba vždy nových úloh pro získání dat z tohoto clusteru by byla časově zdlouhavá a také náročná na schopnosti těchto uživatelů. Stejně tak i běžný provoz tohoto clusteru přináší nové úkoly, jako řízení zpracování jednotlivých úloh, kontrola stavu celého clusteru a řízení přenosu dat mezi jinými systémy. Tyto problémy měly za výsledek vznik několika dalších projektů, které se snaží dílčí problémy řešit, zvýšit tak použitelnost celého řešení a vylepšit komfort práce s tímto systémem. Níže jsou uvedeny některé nejdůležitější tyto projekty (většina je také vyvíjena v rámci organizace Apache).

5.2 Hive

Jedná se o modul, který je postaven nad Hadoopem a zpřístupňuje infrastrukturu datového skladu nad tímto uložištěm dat. Je tak možné se poměrně jednoduše na data v HDFS dotazovat, tvořit sumarizace a analyzovat data. Uživatelé se mohou využitím této komponenty dotazovat pomocí speciálního jazyka, který se nazývá HiveQL a je velmi podobný známému jazyku pro dotazování nad relačními databázemi SQL. Právě toto je hlavní myšlenka, která stála za vznikem tohoto projektu. Uživatelé běžně pracující s databázovými systémy většinou ovládají jazyk SQL a může pro ně být těžší začít používat jiný jazyk a psát v něm MapReduce úlohy. Hive řeší tento problém a pro uživatele je výsledná práce s Hadoopem dosti podobná práci s běžnou relační databází (5).

Architektonicky používá Hive svoji vlastní relační databázi, do které si ukládá metadatové informace o datech v HDFS. Tuto metadatovou databázi, která popisuje data uložená v clusteru, mohou využívat i ostatní projekty operující nad Hadoopem. Tato metadata popisují data v textových souborech uložených v HDFS, jak jsou jednotlivé záznamy rozděleny do sloupců a jaký je datový typ tohoto sloupce. Výsledkem je abstrakce tohoto textového souboru do podoby běžné relační tabulky (5).

Hive jako takový funguje jako samostatný proces, který přijímá požadavky od uživatelů v podobě HiveQL dotazů, překládá je na MapReduce úlohy (může ale využívat i jiné implementace), které jsou poté odeslány do clusteru.

Výhodou je tedy příjemné uživatelské rozhraní pro dotazování nad Hadoopem a také vývoj těchto dotazů je rychlejší než kdyby uživatel musel pokaždé psát novou MapReduce úlohu. Nevýhodou je, že jazyk HiveQL, který se k tomu používá, zatím neobsahuje veškeré možnosti SQL a pro některé typy úloh musí uživatel použít jiného rozhraní než je Hive (5, 26).

5.3 Pig

Pig řeší dosti podobný problém, jako výše uvedený projekt Hive. Také se jedná o nástroj, který si klade za cíl usnadnění dotazování dat soubory v HDFS a abstrahuje toto dotazování do vyššího jazyka. Tento jazyk se jmenuje „Pig Latin“ a oproti Hive se liší tím, že se nesnaží co nejvíce napodobovat SQL, ale jedná se spíše o procedurální jazyk.

Výhodou této implementace je také to, že vývoj dotazů v aplikaci Pig je mnohem více pod kontrolou uživatele. Zatímco v SQL se pouze deklaruje jaká operace má být provedena, v Pigu je možné detailně deklarovat, jaká konkrétní implementace operace se má použít. Tento přístup tak umožňuje uživatelům mít mnohem více pod kontrolou průběh celé úlohy.

Pig byl původně vyvinut ve společnosti Yahoo, kde si získal velkou oblibu a kolem roku 2007 přešel jeho vývoj do samostatného Apache projektu (6, 27).

5.4 Spark

MapReduce framework, který je součástí základního Hadoop projektu má přes své výhody i některá omezení. Je to hlavně velká lineárnost a přímočarost, jak probíhá zpracování MapReduce úlohy a také to, že dílčí mezivýsledky tohoto algoritmu jsou často ukládány na disk, což zpomaluje dobu výpočtu celé úlohy. Tyto problémy se snaží vyřešit projekt Spark.

Jedná se tedy také o výpočetní framework pro úlohy na distribuovaném clusteru počítačů, který je odolný vůči výpadku jednotlivých uzlů. Kromě toho dokáže zpracovávat úlohy v cyklech nebo paralelně a dokáže více využívat výpočtů v mezipaměti clusteru, proto je výpočet celé úlohy mnohem rychlejší. Spark navíc definuje vlastní abstrakce datových objektů a má také sadu vlastních speciálních příkazů pro práci s těmito daty.

Jednou z nástaveb hlavní části projektu Spark je také speciální modul Spark Streaming, který slouží k analýze dat, které nevznikají dávkově v určitých intervalech, ale kontinuálně. Jedná se například o data z různých senzorů nebo automatizované zápisy do logovacích souborů. Díky tomu můžeme tvořit analýzy, které operují nad daty téměř s nulovým zpožděním (8, 28).

5.5 Oozie

Jedná se o webovou Java aplikaci, která slouží k provozování a organizování dílčích MapReduce úloh a jejich propojování do složitějších celků. Tato data uchovává ve specializované databázi v podobě XML souborů, které celé zpracování popisují. Výsledkem této definice jsou tak orientované acyklické grafy, které popisují jednotlivé kroky a cesty nutné k realizaci dané úlohy. Oozie umožňuje řídit spouštění těchto úloh a inicializuje toto spouštění na základě času nebo na existenci určitých dat nutných ke zpracování. Mnoho pravidelných výpočetních úloh je tak možné v clusteru automatizovat. Velkou výhodou tohoto nástroje je integrace s dalšími moduly, které fungují v Hadoop prostředí. Uživatel tak není omezen pouze na plánování pouze MapReduce úloh, ale může spouštět také úlohy napsané v prostředí Hive, Pig, Spark nebo dalších. Tyto dílčí technologie a úkony je také možné kombinovat v rámci jedné velké výpočetní úlohy (29, 30).

5.6 Sqoop

Náplní tohoto projektu je zajištění přenosu dat mezi Hadoopem a klasickými relačními databázemi a dokáže tyto přenosy realizovat v obou směrech. Pro zpracování dat v Hadoopu je totiž nutné mít tyto data přeneseny z primárních transakčních systémů do tohoto prostředí. Přenos takto velkých objemů dat se musí vypořádat s několika dílčími problémy jako je konzistence přenesených dat nebo zatížení primárního systému. Právě tyto problémy se snaží Sqoop řešit a nabízí rozhraní pro přenos dat mezi těmito systémy, ve kterém se dají nové přenosy rychle a efektivně nadefinovat. Přenos může probíhat jednak jako nahrávání celých tabulek nebo dokáže realizovat nadefinované SQL dotazy. Navíc veškeré operace exportu i importu je možné naplánovat jako dílčí úlohy v prostředí aplikace Oozie. Sqoop tak je důležitým nástrojem pro integraci okolních systémů s Hadoopem a někdy se o něm mluví jako o hlavním představiteli ETL nástrojů v prostředí Hadoopu. Několik společností už implementovalo ovladač pro připojení do jejich databázového systému pomocí aplikace Sqoop a tak se stále rozšiřují možnosti pro přenos dat mezi Hadoopem a dalšími systémy pro uchovávání dat (31, 32).

5.7 Zookeeper

Zookeeper je jednou z klíčových aplikací, které mají na starosti provoz celého clusteru. Náplní této aplikace je řízení uzlů v distribuovaném prostředí. Protože většina aplikací, které se musí provozovat tímto decentralizovaným způsobem, bude v určité fázi řešit tyto koordinační a synchronizační problémy, snaží se Zookeeper tyto problémy vyřešit samostatně a poskytnout těmto aplikacím základ, nad kterým mohou dále stavět. Zookeeper tedy v clusteru počítačů řeší problémy synchronizace, koordinace, pojmenovaní, možných konfliktů mezi uzly a dalších problémů, které jsou v distribuovaném prostředí běžné. Přes poměrně velkou složitost této problematiky je kladen velký důraz na to, aby implementace Zookeeperu byla co nejjednodušší a v produkčním provozu rychlá. Princip Zookeeperu spočívá v tom, že samotná tato aplikace je distribuována po síti uzlů. Tyto uzly ke každé provedené operaci přidávají časovou značku a ta následně garantuje, že na ostatních uzlech

budou tyto operace provedeny přesně v pořadí, v jakém byly z původního uzlu odeslány. Operace, které se tak postupně provádějí na celém clusteru, bývají označeny za úspěšné až v okamžiku, kdy jsou správně provedeny aspoň na polovině všech uzlů v clusteru. Pro každou změnu je určen také hlavní uzel, a pokud nastane konflikt, je vyřešen podle onoho hlavního uzlu (33, 34).

Přesto, že použití Zookeeperu není vázáno na projekt Hadoop, stal velice často využívanou komponentou v tomto prostředí a jeho jednotlivé projekty ho využívají ke svému chodu.

5.8 Hue

Posledním zmíněným projektem je Hue. Ten na rozdíl od výše uvedených není vyvíjen organizací Apache, ale jeho autorem je společnost Cloudera. Přesto je i tento projekt licencován pomocí Apache licence a je tak možné ho používat zdarma.

Hue, zkratka pro „Hadoop User Experience“, vytváří přehledný webový interface pro práci s Hadoop clusterem a jeho dalšími moduly pro běžné uživatele. Je možné zde psát a spouštět Hive nebo Pig dotazy, tyto dotazy organizovat a plánovat pomocí Oozie, popřípadě se dotazovat i do běžných databázových systémů. Stejně tak je možné i procházet HDFS a v něm uložené soubory, vkládat a spouštět soubory pro MapReduce napsané v jiných jazycích nebo procházet a definovat popisy datových souborů v metadatové databázi (35).

6 Big Data a datový sklad v architektuře podniku

Big Data aplikace jsou tak stále poměrně novou skupinou technologií, které ještě nejsou ve většině společností plně využívány a to přesto, že mohou přinášet nové cenné informace ze zdrojů, které předtím nebylo možné efektivně zpracovávat. Správná implementace těchto technologií do architektury podniku tak bude jedním z nejdůležitějších problémů společností, které se pro využití Big Data aplikací rozhodnou. S touto výzvou také přichází otázka, co se stane s technologiemi Business Intelligence, které už aktuálně podnik používá a které donedávna byly hlavním nástrojem pro analýzu dat. Big Data technologie totiž mají s nástroji Business Intelligence mnoho společného, často je jejich primární oblastí zájmu stejný problém a některé oblasti jejich použití se mohou i vzájemně překrývat. Nabízí se tak myšlenka, jestli není možné klasické BI postupy úplně opustit a veškerou datovou analýzu v podniku nahradit pouze Big Data platformou. V současné době není pro většinu společností tento přístup možné řešení.

Hlavní důvod spočívá především v principech, na kterých jsou jednotlivé technologie vystavěny. Big Data aplikace pracují distribuovaně s obrovským množstvím dat, které jsou většinou nestrukturalizované a bez pevného datového modelu. V tomto prostředí ještě nejsou plně podporovány vlastnosti relačního zpracování, jako jsou transakce nebo tzv. „ACID“ vlastnosti, které jsou v postupech Business Intelligence samozřejmostí. Tato situace je také částečně zapříčiněna tím, že se jedná stále o nové technologie, které se postupně vyvíjejí, a tyto vlastnosti se eventuálně doplní v budoucnosti. Zatím však použití Big Data aplikací pro úlohy, které vyžadují tyto vlastnosti, může být neefektivní a zbytečně náročné (36, 37).

Business Intelligence nástroje oproti tomu pracují s daty, které jsou přehledně strukturovány v tabulkách, s pevným datovým modelem, garantují konzistenci dat a při práci s nimi je možné využít všech možností a optimalizací, které jsou v relačním zpracování k dispozici. Tyto technologie a přístupy už existují mnoho let a byly tak už mnohokrát prověřeny a optimalizovány v praxi.

Dalším aspektem při porovnávání těchto dvou oblastí může také být každodenní provozování, okolní prostředí a uživatelé těchto systémů. Okolní systémy, které používají zaběhnuté způsoby pro integraci s BI nástroji, nemusí být zatím stejně propojitelné i s novými Big Data technologiemi. Někteří koncoví uživatelé nové platformy zase nemusí mít potřebné technické dovednosti pro psaní úloh, které se zde používají, a tak i jednoduché dotazy a analýzy jsou pro tyto uživatele nerealizovatelné (38).

Nejvhodnějším aktuálním řešením je koexistence obou těchto platforem v podniku najednou. Je tak možné využít předností každé z těchto technologií, využít ji při zpracování úloh, pro které se bude daná technologie nejvíce hodit a maximálně tak zefektivnit práci s podnikovými daty. Obě platformy se budou vzájemně obohacovat a doplňovat a je proto důležité, aby bylo možné je i vhodně mezi sebou propojit (36, 37, 38).

7 Integrace Big Data platformy s datovým skladem

Big Data technologie poskytují další možnosti, jak analyzovat data, která má podnik k dispozici, a která předtím nebylo možné efektivně zpracovávat. To však neznamená, že současné způsoby, kterými jsou analýzy dat v podniku prováděny, jako jsou datové sklady a technologie označované jako Business Intelligence, budou najednou odstraněny a nahrazeny Big Data platformou. Vhodnější variantou je postupné využívání těchto technologií souběžně s používanými technologiemi a jejich postupná, vzájemná integrace. Tento přístup tak dokáže lépe využít jednotlivých předností obou technologií a každou z nich použít pro řešení takových druhů problémů, ve kterých budou nejefektivnější.

Toto budování Big Data platformy vedle už existující implementace datového skladu je však velice složitý úkol, ve kterém je potřeba se vypořádat s mnoha náročnými úkoly a architektonickými výzvami. Protože tento úkol je svým rozsahem daleko větší, než je rozsah této práce, bude se praktická část této práce soustředit pouze na jednu dílčí oblast a to, jak vhodně navrhnout automatický přenos dat z datového skladu do Hadoop clusteru.

Následující část práce tak prezentuje návrh uceleného aparátu technologií, který demonstruje možný způsob, jak Hadoop datově integrovat s podnikovým datovým skladem. Výchozí požadavky na tuto implementaci byly formulovány na základě požadavků nejmenované instituce, která v době vzniku práce zvažuje zavedení těchto technologií do chodu podniku a přenos dat na tuto platformu z datového skladu je jedním z mnoha problémů, které bude nutné vyřešit.

Navržená konfigurace a architektura tohoto systému pro přenos dat je v této práci realizována pomocí virtualizačních nástrojů, na kterých byl celý proces přenosu implementován a otestován a který demonstruje jeho funkčnost.

7.1 Zadání implementace

Zadání praktické části práce bylo formulováno na základě požadavků reálné společnosti, která si však nepřeje být jmenována. Práce vychází ze základních, obecných předpokladů a technických specifikací a bližší detaily o této společnosti nemají na další zpracování této technické problematiky hlubší vliv. Zadání práce a následná implementace byla konzultována s architektem datového skladu této instituce.

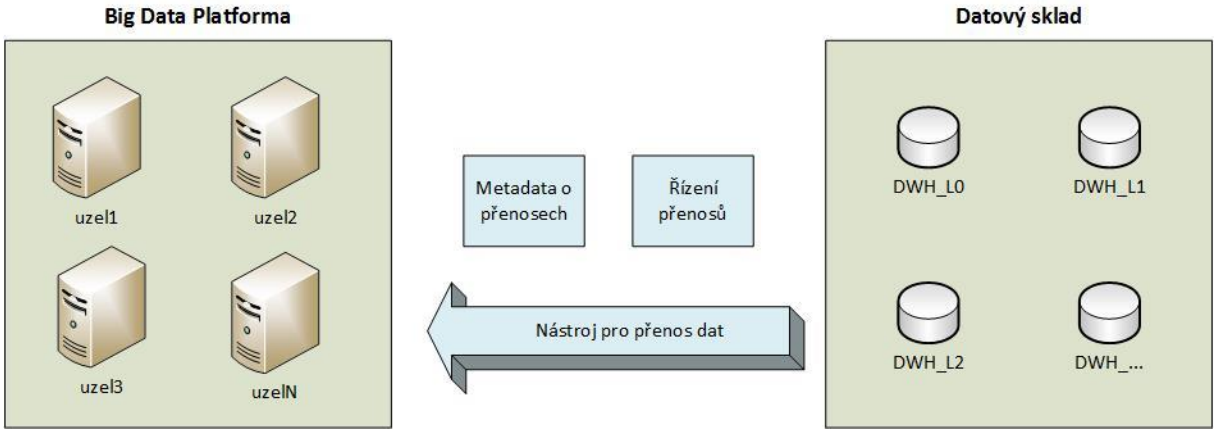
Základním cílem implementace, bylo navrhnout vhodný systém pro přenos dat z datového skladu do Hadoop clusteru. Společnost využívá datový sklad a denně poskytuje koncovým uživatelům mnoho analýz a reportů. Tato platforma je velice vyspělý systém, který je nadále aktivně rozvíjen. Vedle této platformy plánuje společnost implementovat Big Data řešení, které se bude s touto platformou vhodně doplňovat. Velké množství dat, které se aktuálně zpracovává v datovém skladu, bude důležitý vstupní podklad i pro analýzy a výpočty realizované na Big Data platformě. Z tohoto důvodu společnost hledá

systém, jak data automatizovaně a efektivně přenášet z datového skladu do Big Data platformy za použití vhodně zvolených nástrojů.

Právě této problematice se věnuje následující část této práce. Součástí je přehled aktuálně nejznámějších a nejpoužívanějších nástrojů, které umožňují přenos dat do Hadoop clusteru; výběr vhodného nástroje a konfigurace jeho použití; architektonický návrh propojení všech potřebných komponent a návrh systému metadat o proběhlých přenosech. Výsledný systém je implementován v testovacím prostředí, pomocí virtualizačních nástrojů.

Přesto, že práce jako taková není nutně vystavěna na technologických a architektonických detailech a metodikách, které společnost v prostředí svého datového skladu používá, jsou některé prvky systému navrhnuty s ohledem na tyto faktory (39).

7.2 Výchozí požadavky



Obr. č. 3 Výchozí architektura systému (zdroj: autor)

Na jedné straně systému je Big Data platforma, složená z několika samostatných výpočetních uzlů, a na straně druhé datový sklad, složený z dílčích databázových schémat. Cílem vzorové implementace je doplnit systém o tyto komponenty (v obrázku světle modře):

- nástroj, který bude data z datového skladu přenášet na Big Data platformu,
- systém metadat, který bude udržovat nastavení přenosů i informace o realizovaných přenosech,
- komponentu, která bude veškeré přenosy a metadata řídit.

Dále pak dopad do architektury a provozu datového skladu by měl být co nejmenší, protože sklad je denně aktivně využíván a neměl by být ovlivněn jeho běžný provoz.

Implementace přenosového aparátu představeného v této práci tak vychází ještě z následujících omezení a předpokladů, které byly nadefinovány zástupcem společnosti:

- Zdrojem dat je datový sklad implementován v relační databázi, rozdělený do několika úrovní podle zpracování: vrstva L0 – vstupní oblast pro nahrání dat ze zdrojových systémů; vrstva L1 – ve které se uchovávají transformovaná data a prezenční vrstva L2 – určená konkrétním skupinám uživatelů.
- Cílem pro data je cluster počítačů provozujících Hadoop, kde budou data uchovávána a zpracovávána.
- Data je po přenosu na Hadoop potřeba nahrát do aplikace Hive, která poskytuje funkcionalitu datového skladu v prostředí Hadoopu a umožňuje se nad daty dotazovat jazykem podobným SQL.
- Součástí práce není řízení uživatelského přístupu k datům ani detailní návrh bezpečnostních opatření tohoto systému (39).

7.3 Nástroje pro přenos dat do Hadoop

Prvním úkolem pro realizaci přenosu mezi Hadoopem a datovým skladem je výběr vhodného nástroje, pomocí kterého se budou přenosy realizovat. Důležitý faktor tohoto výběru je fakt, že zdrojem je relační databáze, která je analyticky orientovaná a ve které zpracování dat probíhá dávkovým způsobem v několika fázích.

Přenosy dat na platformu Hadoop mohou být realizovány několika možnými nástroji, které se od sebe mohou svým přístupem k dané problematice i velmi lišit. Prvním krokem praktické části této práce bylo tedy zanalyzovat možné způsoby, jak je možné data do této platformy nahrávat a na základě nároků na přenos dat vybrat tu nejvhodnější. Následující přehled jistě není úplný, ale představuje aktuálně jedny z nejčastěji používaných nástrojů.

7.3.1 Apache Sqoop

Tento nástroj je jeden z předních projektů, které se používají v prostředí Hadoopu a za dobu své existence se stal v oblasti své funkcionality téměř standardem. Přístup k datům v externích databázových systémech, přímo ze spouštěných aplikací na Hadoop clusteru, může totiž být neefektivní a zátěž kladená na tyto externí systémy by mohla omezit jejich schopnost kvalitně poskytovat jejich hlavní služby. Proto se v tomto případě používá nástroj Sqoop, jehož primární funkcionalita spočívá v exportování dat z relačních databázových systémů a v jejich nahrání do souborového systému Hadoopu. Stejně tak dokáže tento nástroj realizovat i opačný směr přenosu a data z Hadoopu do relační databáze nahrávat. Oproti základnímu rozhraní pro nahrávání dat, které Hadoop poskytuje, je tento přenos mnohem efektivnější a rychlejší a poskytuje také mnohem bohatší funkcionalitu.

S nástrojem se v základní podobě pracuje přes terminálové rozhraní a jednotlivé přenosy jsou tak realizovány pouze jako spuštění příkazu z příkazové řádky, doplněného o vhodné parametry. Základní parametry takového přenosu jsou například: adresa cílového serveru včetně cílové složky na HDFS, kde se mají data nahrát; adresa databázového serveru, který slouží jako zdroj, včetně uživatele pro připojení k této databázi nebo například celý SQL dotaz, který se použije k vybrání dané množiny dat pro export (31).

```
sqoop import --connect jdbc:mysql://db-address/db-schema
--query "select * from db-table" --username db-user --password db-
psswd --target-dir /hdfs/folder
```

Další dílčí parametry jednoho příkazu mohou určovat například: typ souboru, do kterého se mají data na Hadoopu uložit; zda se má cílová destinace před uložením vymazat; případně můžeme realizovat pouze inkrementální nahrání dat, která ve zdrojové tabulce přibyla od posledního Sqoop přenosu. Mezi velice užitečnou funkcionalitu patří také možnost data po přenesení na HDFS rovnou nahrát do systému Hive nebo HBase a to včetně vytvoření cílové tabulky v tomto systému.

Každý přenos pomocí Sqoopu se skládá ze dvou hlavních kroků: v první fázi si Sqoop připraví veškerá metadata o datech, která bude v dalším kroku přenášet, čehož dosáhne zanalyzováním metadat tabulky v relačním systému. Následně v druhém kroku připraví několik samostatných mapovacích procesů, které se spustí na jednotlivých uzlech Hadoop clusteru a do těchto uzlů se přenesou zmapovaná data z relační databáze. Tyto procesy jsou připraveny na základě metadat získaných v prvním kroku a právě tyto procesy realizují samotný přenos dat. Těchto přenosů může probíhat několik najednou pro různé množiny dat a data ze zdrojové tabulky jsou tak nahrána do HDFS paralelně a efektivněji.

Sqoop je aktuálně ve většině distribucí používán ve své první verzi, která je popsána výše, ale zároveň už existuje novější, kompletně přepracovaná verze tohoto systému, která bude fungovat na bázi centralizovaného Sqoop serveru, ke kterému budou jeho klienti přistupovat přes webové rozhraní (31, 40).

7.3.2 Apache Flume

Flume je jednou z aplikací, která se snaží vyřešit přenos velkých objemů dat, která jsou rozdistribuována po větším množství koncových počítačů, do souborového systému Hadoopu a jeho primární zaměření je přenos souborů ze souborového systému. Vychází z časté potřeby, že v běžně používaném prostředí Big Data platformy je jednou ze základních potřeb pravidelně přenášet soubory logovacích dat, která se generují v externích aplikačních systémech. Aplikační a webové servery generují během svého chodu velké množství nestrukturovaných dat v podobě aplikačních logů, které se svou povahou výborně hodí právě pro další analýzu na systému Hadoop. Tyto externí aplikace často bývají v odlišných data centrech, jiných počítačových sítích a doménách. Flume je komponentou, která umožňuje z těchto systémů data plnit rovnou do prostředí Hadoopu a celý tento proces automatizovat (41).

Základní použití této komponenty je architektonicky poměrně jednoduché. Na každý aplikační server, ze kterého chceme data přenášet, je nainstalován Flume agent, který je pak také individuálně nakonfigurován. Mezi základní komponenty každé takovéto Flume instance patří zdroj (Source), cíl (Sink) a přenosový kanál (Channel). Po spuštění tak bude každý agent očekávat nová data na zdroji, což může být například určitá složka na souborovém sytému, poté nová data nahraje do přenosového kanálu, tím může být místo v paměti nebo na disku a odtud se tyto data snaží nahrát do nadefinovaného cíle. Tento cíl může být buď konečná instance Hadoopu, kde budou data finálně uložena, nebo další Flume agent. Tato vlastnost umožnuje vytvářet celý přenosový proces složený z několika Flume uzlů a je tak možností, jak řídit rozhraní mezi několika systémy.

Celý přenos je založen na principu transakcí, které zajišťují, že data jsou z přenosového kanálu jednoho agenta smazána až tehdy, kdy jsou správné uložena do jeho cílové destinace. V jednotlivých přenosových kanálech tak jsou data uchovávána i po celou dobu, dokud nejsou doručena do cíle a celý přenos tak zajištuje, že je schopen zotavit se z pádu, když některé jeho uzly vypadnou z provozu.

Oproti starším verzím už Flume nepotřebuje ke svému chodu používat instance Zookeeperu (viz výše) a také už se mezi jeho instancemi nerozlišuje mezi nadřazenými a podřízenými uzly (master-slave architektura).

Základní předpoklad tohoto systému je takový, že každý agent má nadefinovanou časovou periodu, jak často kontroluje vznik nových dat a s touto periodou jsou nová data aktivně přenášena agentem do dalšího uzlu (41, 42).

7.3.3 Apache Kafka

Tato aplikace je také systém, jehož primární cíl je přenos dat z několika externích systémů do cílových destinací v co nejkratším čase, ale oproti aplikaci Flume zaujímá v celé architektuře podniku jiné místo a má také odlišný princip práce s daty.

Původně se Kafka začala vyvíjet v prostředí společnosti LinkedIn, která potřebovala nástroj pro zajištění efektivního přenosu dat mezi jednotlivými systémy, které společnost provozovala. Zamýšlela původně používat centrální komponentu, která bude akumulovat data odesílaná ze zdrojových systémů, a v případě dostupnosti odběratelů je bude na tyto cílové příjemce dat rozesílat. Pokud není možné zprávu doručit příjemci, bude tato informace držena v zásobníku tohoto systému, dokud nebude odběratel zase aktivní. Tento systém pro rozesílaní zpráv je označován jako tzv. „messaging system“ a existuje několik jeho různých implementací (např. RabbitMQ nebo ActiveMQ). Tato varianta se ukázala v tomto případě nepoužitelná a to zejména z toho důvodu, že systém nedokázal zpracovávat tak velké množství dat, které společnost potřebovala mezi systémy zpracovávat. Při zahlcení systému zprávami se celá jeho výkonnost radikálně snížila. Výsledkem řešení tohoto problému tak byl projekt Kafka (43).

Jedná se o distribuovanou síť počítačů, které dohromady vlastně tvoří výše uvedený systém pro zpracování dat. Díky jeho distribuovanému návrhu dokáže systém zpracovávat velké množství příchozích i odchozích dat a v případě potřeby větší datové propustnosti, lze zvyšovat počet uzlů v systému a tuto propustnost tak dále rozšiřovat. Centrálním prvkem návrhu systému je princip logovacích souborů, které jednak udržují pořadí posloupnosti zpráv, v jakém do celého systému přišly a zároveň jsou rozdistribuovány po jednotlivých uzlech, což zajišťuje stabilitu v případě individuálních výpadků. Systém tak svým návrhem garantuje, že data z něj budou odcházet ve stejném pořadí, v jakém do systému přišly a veškeré zprávy budou v systému uloženy do té doby, než budou korektně odebrány uživateli (44, 45).

Oproti aplikaci Flume největší rozdíl Kafky spočívá v tom, že data do koncových systémů aktivně neposílá, ale čeká, až si je systémy samy odeberou a korektnost přenosu potvrdí. Systém tak dokáže sloužit pro další systémy jako datový zásobník a každý odběratel dat je může odebírat různou rychlostí.

Aplikaci Kafka je tak možno rozšířit daleko za funkcionalitu pouhého nahrávání dat z relační databáze do Hadoopu, protože dokáže vytvořit centrální systémovou komponentu, přes kterou se budou realizovat veškeré přenosy v IT architektuře dané společnosti (44, 45).

7.3.4 Hadoop CLI a WebHDFS

Pro úplnost je uvedena i doplňující možnost, jak nahrávat data do Hadoopu, tím jsou příkazy, které obsahuje Hadoop v základním balíčku. Příkaz *PUT,* zadávaný přes příkazovou řádku (CLI), slouží k nahrání z lokace souborového systému počítače do HDFS. Tento příkaz je dostačující při manuální práci se systémem, popřípadě pro různé ad-hoc přenosy, ale stejně neřeší problém, jak dostat data z cílové destinace na souborový systém počítače v clusteru. WebHDFS je zase obdobné rozhraní pro tyto příkazy, které používá http protokol (23).

7.3.5 Výběr vhodného nástroje

Při výběrů nástroje bylo potřeba zvážit nejdůležitější vlastnosti systému, ve kterém se bude používat. V tomto případě to byl jednak typ zdroje, kde budou data pro přenosy vznikat a kterým je v tomto případě relační databáze a také to, že data budou zpravidla přenášena dávkově, ve větších objemech a jejich doba platnosti bude většinou celý jeden den.

Aplikace nástroje Flume je většinou v přenosech menších dat, které vznikají na zdrojovém systému a jsou co nejrychleji přenášeny do cílové destinace. Přesto, že dokáže data přenášet i z databází a jiných zdrojů, je primárně určen především k přenosu logů ze souborových systémů serverů.

Apache Kafka je také aplikací, která umožňuje přenášet data v co nejkratším možném čase. Kromě toho poskytuje také možnosti integrace několika zdrojových a cílových systémů najednou a díky svému distribuovanému návrhu jde i dobře škálovat. Tento aparát je však pro přenos mezi dvěma systémy zbytečně robustní a v případě datového skladu není plně využita ani rychlost Kafky.

Oba výše uvedené nástroje Flume a Kafka v tomto případě dostatečně nevyužijí jejich hlavní přednost, kterou je velice rychlý přenos dat do cílové destinace od doby jejich vzniku, a hodí se spíše pro přenos dat z transakčních systému.

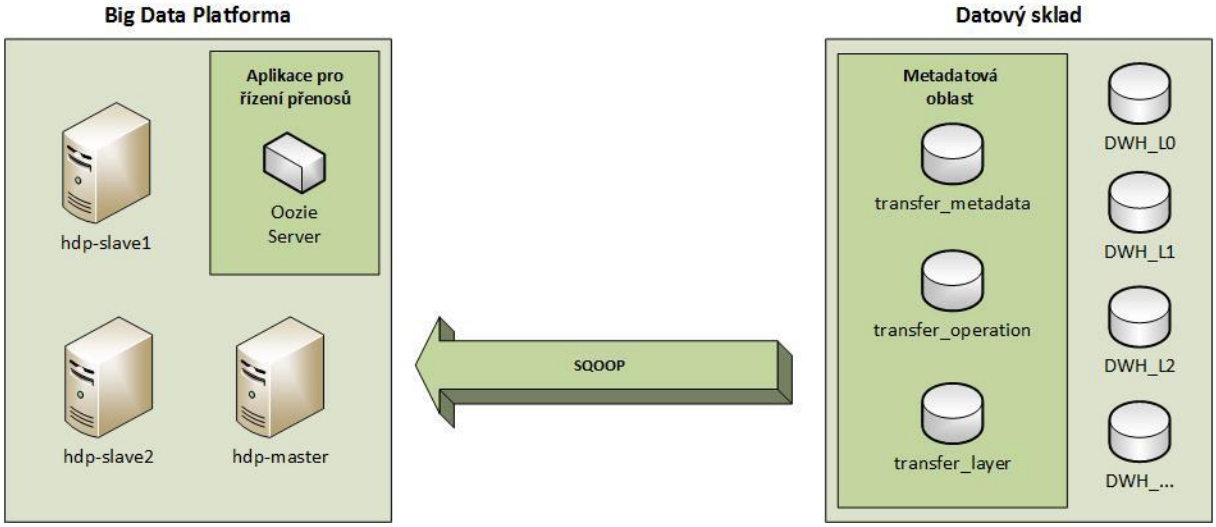
Nástroje, které obsahuje základní balíček Hadoopu by zase nemusely být schopné zpracovávat tak velké objemy dat a před jejich použitím by se data z databáze stejně musela nějakým způsobem předpřipravit.

Na základě těchto faktorů byl jako nástroj pro přenos vybrán Apache Sqoop, který se specializuje na přenos dát právě z relačních databází a dokáže velké objemy dat přenášet paralelně a efektivně. Hodí se také zejména pro dávkový přenos velkého objemu dat najednou, čímž přesně odpovídá typickému zpracování v datovém skladu.

Dalšími argumenty, proč z daných nástrojů zvolit právě Sqoop je také to, že tento projekt je volně k dispozici jako open-source a v případě, že se společnost rozhodne pro nasazení komerční distribuce od poskytovatelů jako je například Cloudera, Hortonworks nebo MapR, tak tyto distribuce už Sqoop implicitně obsahují.

7.4 Architektura přenosového systému

Celkový pohled na navrženou architekturu přenosového systému je ilustrován na níže uvedeném obrázku a jednotlivé jeho komponenty jsou blíže rozepsány v dílčích podkapitolách. Popis jednotlivých komponent se soustředí na hlavní funkci v celém systému a v popisu určitých omezení, které tyto části měly na jeho návrh. Na diagramu jsou hlavní doplněné komponenty zvýrazněny světle zelenou barvou.



Obr. č. 4 Architektonický návrh přenosového systému (zdroj: autor)

7.4.1 Datový sklad

Zdrojem dat pro realizované přenosy a výchozí komponentou přenosového aparátu bude relační datový sklad. Jak již bylo v práci uvedeno, jedná se o relační databázový systém, který je primárně určen k analytickým dotazům a reportům nad velkými objemy dat. Tato data bývají často uchovávána po delší časové období, jsou předmětově orientovaná a zpravidla v jednou uložených datech už nedochází k dalším změnám.

Data, která do datového systému proudí na denní bázi z ostatních externích systémů v podniku, bývají před uložením do datového skladu upravována, transformována a konsolidována do jednotné podoby, která umožňuje lépe nad těmito daty provádět analytické dotazy. V dalších krocích jsou nad některými těmito konsolidovanými daty prováděny další výpočty a tyto derivované údaje jsou také ukládány pro potřeby dalších analýz.

Jednotlivé fáze zpracování v průběhu dne zpracovávají rozličné množiny dat a plní různá dílčí databázová schémata v datovém skladu. Toto zpracování dat probíhá dávkově podle toho, jak jsou množiny dat ke zpracování poskytovány externími zdrojovými systémy. V tomto případě se předpokládá, že většina těchto datových množin se zpracovává jednou denně. Jednotlivé datové oblasti mají určené datum platnosti, ke kterému byly zpracovány, a každodenní zpracování těchto množin tak poskytuje možnost v datovém skladu analyzovat jejich historický vývoj v čase.

Jakmile je jedna část dat konsolidována k danému dni platnosti a uložena do datového skladu, je nutné celou tuto oblast přenést na prostředí Hadoop, aby bylo možné s těmito daty pracovat i na této druhé platformě. Protože jedním z hlavních přínosů datového skladu je jednotný a konsolidovaný pohled na data, je důležité přenášet celé datové oblasti najednou, aby pro korespondující data byla zachována jejich časová integrita.

7.4.2 Metadatová oblast

Jedním z hlavních požadavků na přenosový systém bylo také udržování veškerých metadat, které poskytnou uživatelům systému přehled o tom, jaké datové přenosy do Hadoopu už proběhly a jaké je aktuální datum platnosti přenesených dat. Stejně tak je nutné udržovat přehled o tom, pro které datové objekty v datovém skladu mají přenosy probíhat a jaké budou mít tyto přenosy parametry. V metadatech se tak udržuje přehled tabulek určených pro přenos, jejich rozdělení do skupin tabulek, které by mělo probíhat najednou a zařazení do procesu, které by mělo přenos realizovat. Oblast tak bude zajišťovat primárně dva hlavní úkoly:

- popisný – udává, jak mají být datové přenosy nakonfigurované a pomocí změn v těchto záznamech bude možné ovlivnit chování systému,
- provozní – zaznamenává veškerý každodenní provoz této oblasti.

Díky každodennímu sledování a analýze realizovaných přenosů, bude možné nastavení optimalizovat a výkonnost celého aparátu vylepšovat.

Pro potřeby uživatelů datového skladu, ať už se bude jednat o fyzické osoby nebo technické účty, které budou používat externí aplikace, je nad touto oblasti vystavěna jednoduchá vrstva pohledů, která jim bude užitečně informace zprostředkovávat.

Návrh základních principů vztahů jednotlivých entit v této oblasti a systém jejího řízení vychází ze systému řízení metadat, který se v dané společnosti používá pro koordinaci ETL procesů. Jedná se totiž o principiálně podobný proces, pro který se hodilo zachovat stejnou metodologii pojmenování a rozdělení entit.

7.4.3 Big Data platforma

Big Data platforma je v tomto systému cílem pro veškeré datové přenosy z datového skladu. Základní prvek této platformy je systém Hadoop, který zajištuje distribuované uložiště dat a základní zpracování těchto dat, pomocí frameworku MapReduce.

Další prvek systému, který je z pohledu vzorové implementace důležitý, je aplikace Apache Hive, která bude data v clusteru poskytovat koncovým uživatelům. (Data je po nahrání do souborového systému HDFS ještě potřeba dodatečně zpracovat pro potřeby této aplikace.) Jakmile jsou data správně nahrána do aplikace Hive je přenos pro daný objekt úspěšně ukončen.

V clusteru je vyhrazena speciální oblast na straně HDFS, která slouží jako rozhraní pro nahrání dat do systému a kam je bude aplikace Sqoop ukládat.

7.4.4 Nástroj Sqoop

Na základě výše uvedených informací, je pro samotnou realizaci přenosů vybrán nástroj Sqoop. Použití tohoto nástroje pro realizaci jednoho přenosu spočívá ve spuštění příkazu se správnými parametry. Je tak nutné kolem tohoto nástroje vystavět samostatný aparát, který bude pro jednotlivé tabulky konstruovat správnou podobu Sqoop příkazu. Detailní popis vlastností tohoto nástroje a jeho parametrů je popsán v samostatné podkapitole.

7.4.5 Aplikace pro řízení přenosů

Do systému zbývá doplnit poslední integrující komponentu, která bude mít na starosti spouštění jednotlivých částí přenosů, řízení návaznosti jejich kroků. Pro tento úkol tak je na straně Big Data platformy nainstalována aplikace Apache Oozie.

Oozie je serverová aplikace, napsaná také v jazyce Java, vyvíjená jako open-source pod záštitou Apache Foundation. Slouží k realizaci předdefinované série úloh a úkolů běžně prováděných v Hadoop clusteru. Tyto série úloh jsou v daném kontextu označovány jako „workflow“ a ty se skládají z dílčích úloh označované jako „joby“. (Pro přehlednost bude v práci nadále použita tato terminologie.) Jednotlivé workflow v Oozie tak jsou sledem několika různých jobů a směrovacích, rozhodovacích uzlů a workflow tedy mají podobu orientovaných acyklických grafů. Typy uzlů úloh odpovídají několika základním úlohám běžně prováděných v Hadoop clusteru jako je například spuštění MapReduce úlohy, provedení HDFS příkazu, spuštění Hive příkazu nebo dokonce i spuštění shell skriptu na jednom uzlu (46).

Pro tento úkol existují i jiné projekty (např. Apache Azkaban), které řeší podobnou úlohu v celém ekosystému kolem Hadoopu. (Eventuálně je možné řídit zpracování jobů v platformě pomocí plánování úloh přímo na úrovni operačního systému.) Důvod, proč je z těchto možností vybrána do vzorové implementace právě Oozie, je zejména to, že je tento nástroj implicitní součástí hlavních distribucí komerčních platforem a je v oblasti řízení úloh na Hadoopu velmi populárním řešením. Přesto, že konfigurace workflow v Oozie je nemalou částí této práce, stěžejní jsou dílčí úlohy, které jsou ve workflow postupně spouštěny a je tak možné toto řešení převést i do jiného systému pro řízení zpracování (46, 47).

7.5 Technická specifikace vzorové implementace

Pro vzorovou realizaci implementace distribuovaného systému, který se skládá z několika samostatných komponent, je použit virtualizační nástroj Oracle VM VirtualBox. Pomocí tohoto nástroje je možné vytvořit jednotlivé počítače v celém systému a simulovat interakce mezi nimi. Přesto, že jsou tyto počítače pouze virtuální a reálně jsou provozovány v rámci jednoho fyzického počítače, jejich chování a funkce je z programového pohledu stejná, jako by se jednalo o fyzické stroje. Nevýhodou této realizace je nemožnost simulovat výkonnostní testy a chování systému v různých úrovních zátěže, ale takový typ testování není předmětem této práce. Tento virtualizační program je ve své základní verzi poskytován jako open-source, pomocí GNU licence.

Pomocí tohoto nástroje bylo vytvořeno několik virtuálních počítačů, které budou simulovat na jedné straně Big Data platformu, jejíž centrem bude cluster počítačů provozujících Hadoop, a na druhé straně relační databázi představující datový sklad.

7.5.1 Instalace relační databáze

Zdrojem dat pro Big Data platformu je v tomto vzorovém případu relační databáze nainstalována na samostatný uzel a v celém systému tak zastává roli relačního datového skladu. Jako operační systém tohoto počítače je zvolen systém CentOS verze 7 a funkčnost datového skladu bude zajišťovat databáze MySQL verze 5.7.15.

Tato databáze je ve své základní verzi k dispozici také jako open source a dlouhou dobu byla i tímto způsobem komunitně vyvíjena. Díky těmto vlastnostem se za svou dobu své existence stala velice populární a patří k nejvíce využívaným typům databází na světě. Od roku 2010 přešel její vývoj pod správu společnosti Oracle, která na její další vývoj dohlíží. Přesto, že velká část její popularity pramení zejména z použití v menších typech projektů, je možné tuto databázi najít v architektuře systémů velkých společností jako je YouTube, Facebook nebo Twitter (48).

MySQL jako typ databáze není primárně navržena k realizaci velkých datových skladů a používá se zejména k provozování transakčních databázových systémů. Přesto existují i implementace datového skladu v MySQL, i když jsou, co se týče objemu dat, spíše menšího rozměru. K budování rozměrných podnikových datových skladů, lze využít speciální verze proprietárních databází, které jsou k tomuto účelu určeny (Oracle Database, MS SQL Server BI Edition) nebo využít typ databáze, jehož architektura je celá k takovému použití navržena (Teradata). V tomto konkrétním případě je MySQL databáze použita zejména proto, že se jedná o klasického představitele relačního databázového systému, používá standardní jazyk SQL (jehož velká část je implementována stejně všemi výrobci databází a liší se pouze v malém procentu příkazů) a práce s tímto systémem je dost podobná výše uvedeným databázím (48).

Architektura této databáze opisuje vzorový model architektury datového skladu a hlavní architektonické části tohoto rozložení se označují jako:

- vrstva L0 – vstupní vrstva, kam se nahrávají data z externích systému (zpravidla bez dalších úprav, ty se realizují až při uložení do další vrstvy),
- vrstva L1 – integrační vrstva, do této části se transformují data ze vstupní vrstvy v konsolidované podobě pro datový sklad, dochází k čištění dat a jejich validaci a také se zde udržuje jejich historie,
- vrstva L2 – prezentační vrstva, do těchto kontejnerů se plní specializované množiny dat podle jejich konkrétní reprezentace (49).

Vzorový datový sklad předpokládá stejnou architekturu a oproti ní speciálně doplňuje oblast metadat, která bude sloužit k řízení přenosů do Hadoop clusteru. Předpokládá se, že zpracování dat v takovém skladu bude probíhat stejně jako ve vzorové společnosti, to znamená dávkově, na denní bázi a postupně po jednotlivých vrstvách. Průběh zpracování dat v celém skladu by měl zůstat beze změny a nemělo by doplněním přenosového aparátu dojít k jeho ovlivnění. Oblast řízení metadat je tak komponentou, která doplňuje celkovou architekturu a bude detailně popsána v samostatné kapitole.

Pro externí aplikace, které budou do relační databáze přistupovat, bylo založeno několik technických uživatelů. Přesto, že součástí práce není detailní návrh řízení přístupu k datům v databázi, byli tito uživatelé vytvořeni s ohledem na tyto předpoklady a jejich oprávnění pro přístup do relační databáze se tak omezují pouze na oblasti a úkony nezbytně nutné k jejich fungování.

7.5.2 Instalace Hadoop clusteru a popis komponent

V modelovém příkladu byla vytvořena část Hadoop clusteru ze tří samostatných počítačů. Tento počet samozřejmě bude v produkčním prostředí mnohem vyšší, ale pro testování funkčnosti všech nástrojů je dostačující.

Na všechny tři virtuální stroje je nainstalován operační systém Ubuntu verze 14.04 LTS (Jedná se také o typ linuxové distribuce, ale tentokrát vycházející z jiné množiny systémů, než systém CentOS.) a protože veškerou administraci těchto systémů je možné zvládnout pouze pomocí protokolu SSH a příkazové řádky, je postačující i serverová verze tohoto systému, která implicitně neobsahuje uživatelské grafické rozhraní. Nejdůležitější komponentou je poté instalace samotného Hadoopu a jeho následná konfigurace. Ten je na všech třech strojích nainstalován ve verzi 2.7.2 a instrukce pro tuto instalaci je možno dohledat na oficiálních stránkách projektu, popřípadě je na internetu k dispozici několik návodů (51, 52). Přesný postup této instalace není předmětem této práce, proto je zde uvedeno pouze několik nejzákladnějších bodů, které bylo třeba realizovat na každém jednotlivém uzlu:

1. Nainstalování Javy, kterou Hadoop potřebuje ke svému běhu (openjdk-7-jdk).

2. Vytvoření dedikovaného uživatele pro spouštění procesů Hadoopu.
3. Nakonfigurování několika XML souborů – obsahují adresy jednotlivých počítačů; porty přes které služby budou komunikovat; určení lokace na souborovém systému pro zápis dat v Hadoopu, apod.
4. Vygenerování SSH klíčů pro propojení jednotlivých uzlů pomocí SSH protokolu – počítače využívají tento protokol ke vzájemné šifrované komunikaci.

Po těchto několika krocích a správném nastavení sítě (Jednotlivým počítačům byla přiřazena pevná IP adresa v rámci lokální sítě a zakázáno používání protokolu IPv6.), ve které musí být jednotlivé uzly vzájemně dosažitelné, zbývá spustit na jednotlivých strojích služby Hadoopu a systém je možné začít používat.

Konfigurace počítačů není pro všechny tři uzly totožná a počítače tak nemají v celém clusteru stejnou důležitost. Jeden uzel je povýšen nad ostatní jako hlavní a oproti ostatním na tomto uzlu běží i Hadoop procesy označované jako „NameNode“ a „Resource Manager“. První z těchto procesů je centrální komponentou HDFS systému, která udržuje přehled o místě uložení částí souborů po celém clusteru a aplikace s touto komponentou komunikují v každém případě, kdy chtějí provádět operaci se souborovým systémem clusteru nebo potřebují lokalizovat, kde se části souboru v clusteru nachází. Druhý proces je primárně určen k řízení MapReduce úlohy v clusteru. Udržuje přehled o tom, jak jsou jednotlivé uzly vytížené zpracováváním části úloh a také zodpovídá za přiřazování částí úlohy volným uzlům. Na ostatních uzlech běží pouze základní služby Hadoopu zvané „DataNode“ a „Node Manager“, které jsou podřízené dvěma výše uvedeným službám. DataNode má na starosti správu a ukládání dat na daném počítači a poskytuje data ostatním komponentám v clusteru, avšak až po tom, co je na tento DataNode odkáže nadřazený NameNode. Těmito komponentami mohou být jednak aplikace, které na daný uzel posílají úlohu k řešení nebo i jiné DataNody, které mezi sebou replikují soubory a zajišťují tak dostupnost dat v případě, že některý z uzlu přestane komunikovat. Druhá služba Node Manager udržuje obdobný přehled o daném uzlu, ale pouze v kontextu k nadřazenému Resource Manageru a hlavně má na starosti zpracování úloh v clusteru. Řídí výpočty na daném počítači, monitoruje spotřebovávané prostředky uzlu a tyto stavy reportuje zpět nadřazenému Resource Manageru, který tak řídí zpracování úlohy napříč celým clusterem (52).

Systém zpravidla vypadá tak, že obsahuje velké množství počítačů, na kterých běží pouze služby DataNode a Node Manager, ty fyzicky udržují data a zpracovávají části úloh, které jim byly přiřazeny a na samostatných strojích běží instance NameNode a Resource Manager, které celý cluster řídí. Tyto uzly se tak v rámci clusteru vyskytují jednou a jsou nejdůležitějšími komponentami, bez kterých není celý systém funkční. Proto se doporučuje serverům, které budou zastávat tuto roli, neinstalovat další služby Hadoopu a oproti ostatním uzlům je nastavit jako hardwarově silnější. Protože se jedná o úzká místa celého systému, bez kterých nebude systém fungovat, je možné nakonfigurovat ještě další záložní uzly, které dokážou v případě výpadku primárních uzlů, jejich funkčnost nahradit a zabránit

úplnému výpadku systému. Přesto ale není toto řešení úplně bezproblémové a chod systému bude v případě výpadku hlavních komponent ovlivněn. Celý systém Hadoopu byl totiž hlavně designován s důrazem na konzistenci dat, ale ne na jejich stoprocentní dostupnost (52).

Rozložení služeb a nastavení jednotlivých virtuálních počítačů v modelovém clusteru je tedy následující:

Název počítače	IP adresa	Paměť	Hadoop služby
hdp-master	192.168.56.31	4GB	DataNode, Node Manager, NameNode, Resource Manager
hdp-slave1	192.168.56.32	2GB	DataNode, Node Manager
hdp-slave2	192.168.56.33	2GB	DataNode, Node Manager

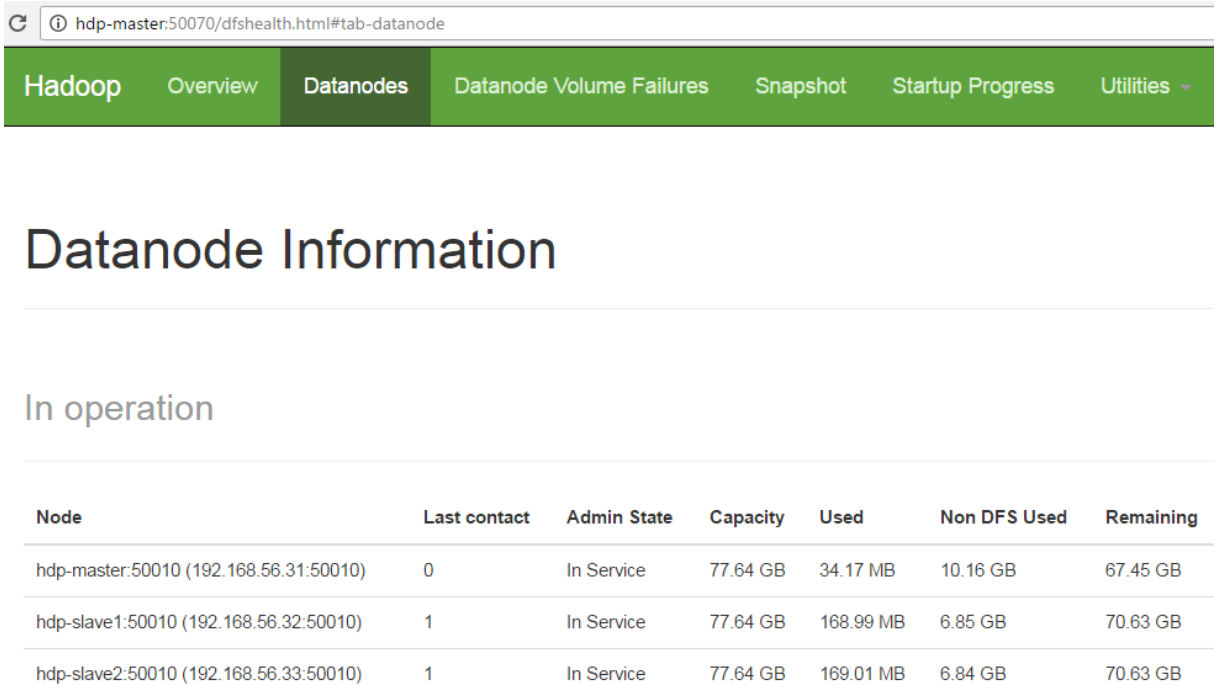
Tabulka č. 1 Přehled počítačů a nainstalovaných služeb (zdroj: autor)

Přesto, že oficiální doporučení je provozovat služby NameNode a Resource Manager na samostatných uzlech, v tomto případě jsou společně na hlavním uzlu, který provozuje i služby DataNodu a Node Manageru, protože tento systém demonstruje pouze funkčnost a integraci jeho komponent, a ne udržení celého systému v chodu při výpadku jeho jednotlivých uzlů. Ze stejného důvodu není v systému použito i speciálních serverů pro sekundární NameNode a Resoure Manager.

V clusteru je nastavena replikace na hodnotu dva, tedy každá část souboru musí existovat vždy na dvou různých uzlech. Pokud jeden z podřízených uzlů přestane fungovat (neuvažujeme výpadek hlavního uzlu, z výše popsaných důvodů), nenastane zhroucení systému, protože soubory budou vždy k dispozici z repliky na druhém uzlu. Pokud by došlo k většímu výpadku, systém tento fakt identifikuje a celý systém se přepne do módu „Safe Mode“, který znamená, že je možno z clusteru číst data, která jsou aktuálně dostupná, ale není možno je přepisovat či nějak měnit, protože by to mohlo ohrozit integritu vůči datům, která aktuálně jsou pouze na nedostupných strojích. Jakmile systém dokáže opět identifikovat pozici veškerých datových souborů (některé uzly začaly opět reagovat), je bezpečnostní mód zase vypnut. V produkčních prostředích se doporučená hodnota replikace udává alespoň na hodnotu tři, která už zajištuje velice vysokou odolnost vůči výpadku systému (52).

Tímto je zprovozněna funkčnost distribuovaného ukládání dat na Hadoopu a také spouštění MapReduce úloh na tomto clusteru. Z dalších komponent, které se běžně v produkčním prostředí celé Big Data platformy používají, jsou na clusteru nainstalovány pouze ty komponenty, které jsou relevantní z pohledu této integrační úlohy představené v práci. Nainstalovány byly na hlavní uzel a popis jejich instalace a konfigurace je vysvětlen v dílčích podkapitolách.

Součástí instalace Hadoopu je také jednoduchá implementace webového rozhraní, které je dostupné přes webový prohlížeč na hlavním uzlu clusteru, který zajišťuje funkci NameNode. Přes toto rozhraní je možné sledovat aktuální stav clusteru, jeho nastavení a přehled jednotlivých uzlů. (Je dokonce možné procházet soubory uložené v souborovém systému HDFS a také je případně stáhnout.)



Obr. č. 5 Grafické rozhraní systému Hadoop (zdroj: autor)

7.5.3 Instalace aplikace Hive

Pro snadnější dotazování nad daty uloženými v Hadoopu je v clusteru nainstalována rozšiřující komponenta Apache Hive, která poskytuje snadné rozhraní uživatelům, a ti mohou psát své dotazy v jazyku velmi podobném SQL. Interně je pak tento dotaz přeložen do MapReduce úlohy a spuštěn na Hadoop clusteru.

Aplikace musí být nainstalovaná na jednom uzlu počítače a nakonfigurována tak, aby mohla úlohy směrovat na hlavní uzel Hadoop clusteru. Součástí konfigurace musí také být vytvoření speciální složky na HDFS, kam bude aplikace Hive mít potřebná práva a kde budou veškerá její data uchovávána. Tato data, nad kterými se může aplikace dotazovat, jsou reálně standardně uchovávána na HDFS jako textové soubory a hodnoty těchto dat jsou pouze rozděleny oddělovačem. (Typů souborů, do kterých je možné data ukládat, je v aplikaci Hive několik, každý vhodný pro určitý typ dat.) Aplikace tak ke svému chodu potřebuje vlastní schéma v databázi, do kterého si ukládá informace o uložených souborech v HDFS a které například udávají, jak se jednotlivé datové sloupce v souborech jmenují a co mají za datový typ. Hive je tady nainstalován na centrální uzel hdp-master a ten je

doplněn o instanci databáze MySQL. (Ta na tomto uzlu nebude mít jinou úlohu, než ukládat metadata aplikace Hive a aplikace Oozie, viz níže.) Hive stačí dedikovat jedno samostatné schéma v databázi a veškeré potřebné tabulky jsou pak vytvořeny automaticky, pomocí přiloženého skriptu. MySQL je jednou z oficiálních podkladových databází, které Hive v základu podporuje, a jako další alternativu je možné použít databázi PostgreSQL nebo odlehčený typ databáze Apache Derby (53).

Součástí základní instalace je také aplikace HiveCLI, která spustí textové rozhraní příkazové řádky, přes které je možné posílat do systému Hive dotazy. Tato aplikace musí být na všech strojích, ze kterých se budou posílat příkazy ke zpracování. V nových verzích Apache Hive je k dispozici serverová verze aplikace HiveServer2, která vylepšuje bezpečnost a umožňuje řízení konkurenčního přístupu více uživatelů. K této verzi serveru se také poskytuje nové, bezpečnější rozhraní Beeline, které zde nahrazuje rozhraní HiveCLI. V této práci je pro jednoduchost použita starší verze systému Hive, včetně původního rozhraní HiveCLI (53).

7.5.4 Instalace nástroje Sqoop

Pro zahájení datového přenosu, je potřeba na stroji, který bude tento příkaz inicializovat, mít nainstalovaný nástroj Sqoop. Instalace nástroje je jednoduchá, spočívá pouze v rozbalení všech potřebných souborů do samostatné složky a cestu k této složce přidat do systémové proměnné operačního systému pro cestu ke spustitelným souborům, aby bylo možné příkaz Sqoop jednoduše volat z jakékoliv lokace pomocí rozhraní příkazové řádky.

Posledním krokem nutným k provozu této aplikace je doplnění používaných knihoven o soubor „connectoru“, což je knihovna nutná pro připojení Sqoopu k danému typu databáze. V tomto případě byl tak doplněn soubor JDBC MySQL „connector“ verze 5.1.40, který byl stažen z oficiálních stránek produktu MySQL.

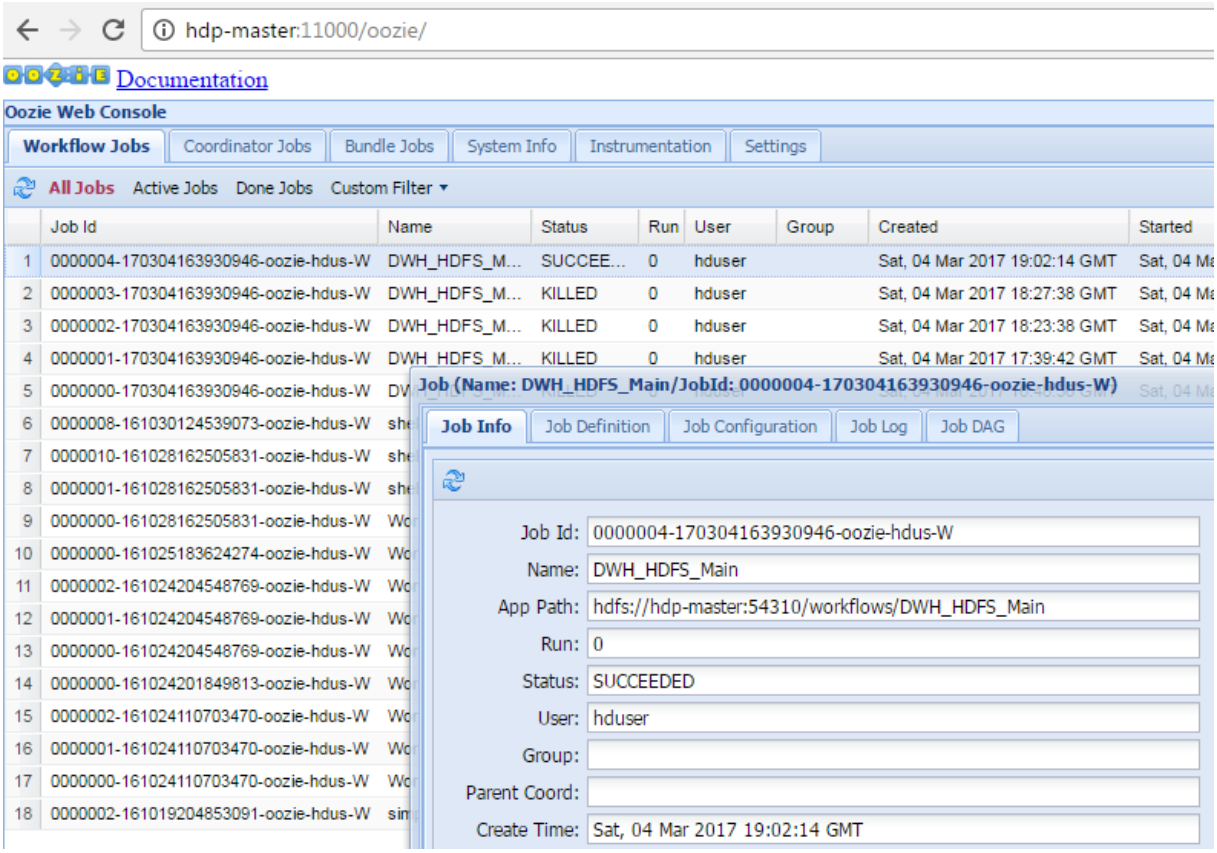
7.5.5 Instalace Oozie

Pro instalaci serverové aplikace Oozie je potřeba použít nástroj Maven, který automatizuje kompilaci programů a zajišťuje správnou návaznost všech souborů v dané aplikaci. Zdrojové kódy aplikace obsahují také skripty, které usnadňují správnou konfiguraci Mavenu a ke spuštění sestavení tak stačí spustit přiložený skript s několika parametry.

Oozie je nainstalována ve verzi 4.1.0, která je kompatibilní s Hadoopem verze 2.7.2. Následně je nutné pro potřeby aplikace založit samostatné schéma v relační databázi, kde si bude Oozie ukládat potřebná metadata. Je využita stejná instance MySQL databáze na uzlu hdp-master, která už poskytuje uložiště pro metastore aplikace Hive. K založení všech tabulek ve schématu slouží opět jeden z přiložených skriptů.

Posledním krokem ke spuštění Oozie je spuštění tzv. historyserver deamonu v Hadoop clusteru. Jedná se o proces, který poskytuje informace o dříve proběhlých úlohách na clusteru a Oozie s tímto procesem musí komunikovat, aby mohla správně informovat o stavu proběhlých workflow.

Oozie také implicitně obsahuje jednoduché webové rozhraní, které poskytuje přehled nad všemi proběhlými workflow.



Obr. č. 6 Grafické rozhraní systému Oozie (zdroj: autor)

7.6 Návrh řízení metadat

Klíčovým prvkem celého aparátu je oblast metadat, která bude udržovat jak informace o nastavení jednotlivých entit v systému, tak provozní informace o již realizovaných přenosech.

7.6.1 Výchozí řízení metadat v datovém skladu

Návrh rozdělení entit v této oblasti vychází z terminologie, která se používá v datovém skladu společnosti a základním konceptem této terminologie je pojem „datové oblasti“. Datová oblast je množina tabulek v části určitého databázového schématu plněného specifickým procesem. Tímto procesem tak může být plnění pomocí ETL nástroje nebo následné transformační zpracování. Jedno databázové schéma může být složeno z několika datových oblastí, a tak i jedno schéma bývá plněno několika procesy.

Pro každou datovou oblast se určuje aktuální datum platnosti, které udává časovou platnost záznamů v této oblasti a na konci každého plnění se v metadatech toto datum aktualizuje.

Cílem celého tohoto systému je řízení závislostí při zpracování dat v datovém skladu v jeho jednotlivých částech. Datové oblasti a jejich data platnosti tak poskytují nástroj, jak data koordinovat a fáze zpracování realizovat na správných množinách dat a ve správný čas. Příklad řízení může vypadat například takto: Pro výpočet dat v určité datové oblasti je potřeba zpracování dvou oblastí v předcházejících úrovních skladu. Pokud se transformační proces jedné této oblasti opozdí (například kvůli chybě v transformaci, nedostupnosti zdrojového systému apod.) nemůže začít výpočet dané datové oblasti, protože k danému datu platnosti nejsou všechny zdrojové datové oblasti připravené – opožděná datová oblast má stále datum platnosti starší než druhá oblast. Jakmile je problém vyřešen a zpracování proběhne, je datum platnosti správné pro všechny potřebné zdroje a navazující proces může transformovat data do své cílové databáze a po úspěšném ukončení tohoto procesu se aktualizuje datum platnosti pro navazující oblast. Díky tomuto systému je tak zajištěno zpracování pouze se správnou množinou dat a celková integrita těchto dat (39).

7.6.2 Rozdělení entit v přenosovém systému

Na základě výše uvedeného pojmu datová oblast v datovém skladu, je tento termín použit i v metadatové oblasti pro přenos dat z datového skladu na Hadoop. Z pohledu přenosového aparátu je každá datová oblast tvořena několika tabulkami a každou datovou oblast můžeme přenášet určitým workflow. Mezi datovým skladem tak můžeme nadefinovat několik různých workflow pro přenos a každé nakonfigurovat samostatně. Příkladem může být třeba to, s jakou frekvencí se bude workflow spouštět. Můžeme tak mít důležitější workflow, které bude přenášet ty datové oblasti, které je třeba přenést na Big Data platformu co nejdříve, a toto workflow bude spouštěno každých pět minut. Druhé workflow, pro přenos méně důležitých dat může být spouštěno pouze jednou za hodinu apod.

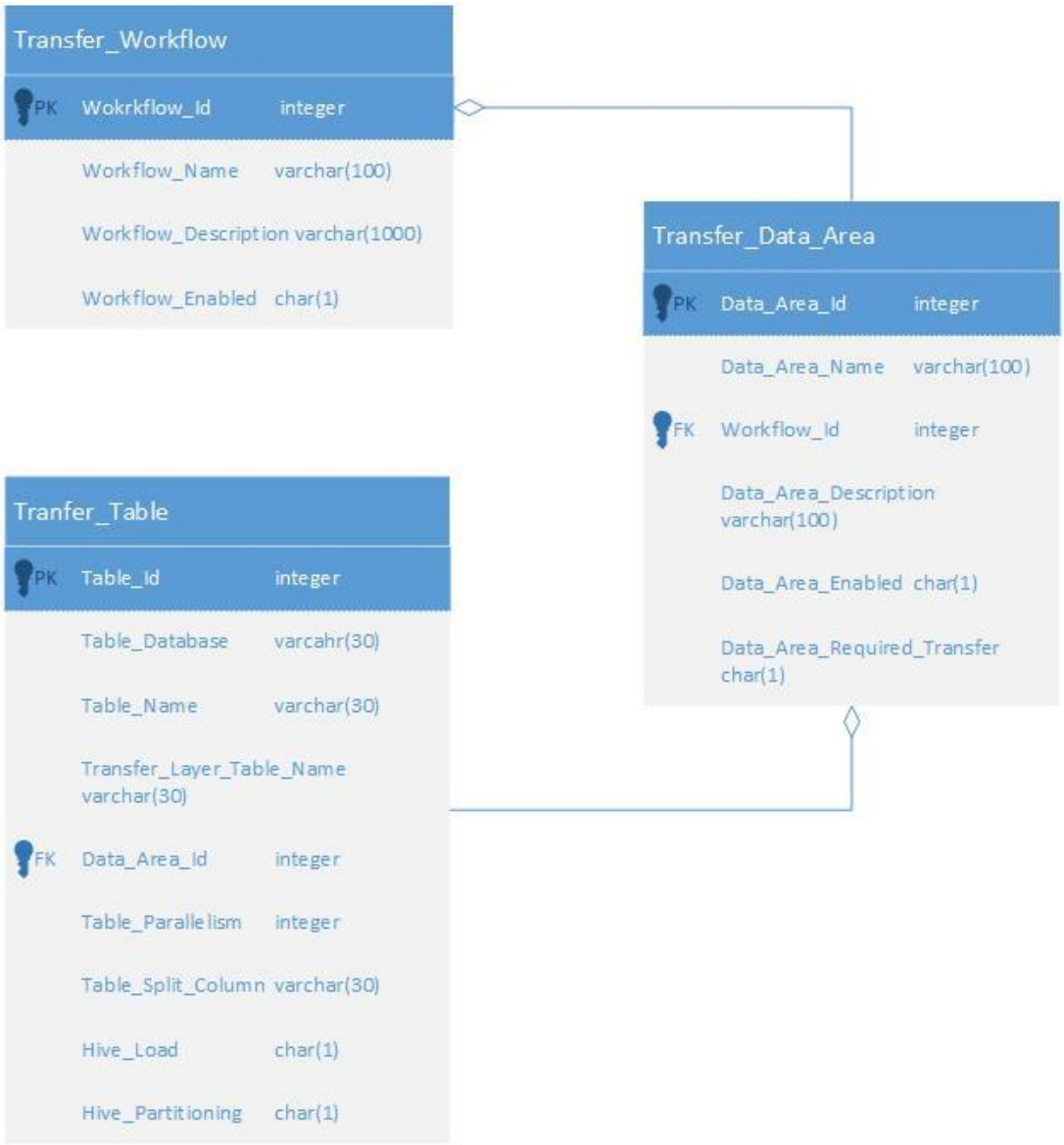
Výsledné rozdělení entit pro přenos dat a jejich vzájemný vztah je tedy následující:

- workflow – hlavní entita, která odpovídá jednomu procesu nadefinovanému v aplikaci Oozie a může přenášet množinu několika datových oblastí,
- datová oblast – konsolidovaná část databázového schématu, skládající se z několika tabulek, které mají stejné datum platnosti a měly by být přenášeny společně,
- tabulka v datové oblasti – nejmenší jednotka v systému; objekt určený k přenosu.

7.6.3 Datový model metadatové oblasti

Metadatová oblast je prakticky realizována jako tři samostatná databázová schémata: transfer_metadata, transfer_operation a transfer_layer.

transfer_metadata



Obr. č. 7 ER Diagram – oblasti transfer_metadata (zdroj: autor)

V schématu budou udržovány veškeré konfigurace nutné k realizaci přenosu a nastavení vztahu mezi jednotlivými entitami. Při každém běhu se do této oblasti realizuje dotaz, pomocí kterého se budou nastavovat parametry pro spuštění příkazu Sqoop pro jednotlivé tabulky.

Přehled objektů a jejich nejdůležitějších vlastností (vynechány identifikátory a popisné sloupce):

1. Transfer_Workflow – nejvyšší entita; zde slouží hlavně jako přehled všech přenosových workflow, které jsou nakonfigurované v Oozie serveru a k přiřazení datových oblastí k jednomu přenosovému workflow.
    1.1. Workflow_Enabled – příznak, zda toto workflow, může realizovat přenosy (hodnoty: Y, N)
2. Transfer_Data_Area – jedna datová oblast skládající se z několika tabulek
    2.1. Enabled_Flag – dostupnost datové oblasti se má kontrolovat pro přenos (hodnoty: Y, N)
    2.2. Data_Area_Required_Transfer – informace, která se použije pro ostatní procesy v datovém skladu; pokud je nastaveno na Y, je vyžadováno, aby před provedením změn v dané datové oblasti, proběhl přenos do HDFS
3. Trasnfer_Table – tabulka určená pro přenos
    3.1. Enabled_Flag – je povolen přenos konkrétní tabulky v dané datové oblasti (hodnoty: Y, N)
    3.2. Table_Database – název schématu, kde se přenášená tabulka nachází
    3.3. Table_Name – celý název přenášené tabulky
    3.4. Transfer_Layer_Table_Name – název pohledu v oblasti transfer_layer, ze kterého bude aplikace Sqoop číst data
    3.5. Table_Parallelism – na kolik částí se má nahrání dat Sqoopem rozdělit
    3.6. Table_Split_Column – podle kterého sloupce se mají data rozdělovat
    3.7. Hive_Load – tabulka bude nahrána do aplikace Hive (hodnoty: Y, N)
    3.8. Hive_Partitioning – pro přenos do aplikace Hive se použije funkce partition (hodnoty: Y, N)

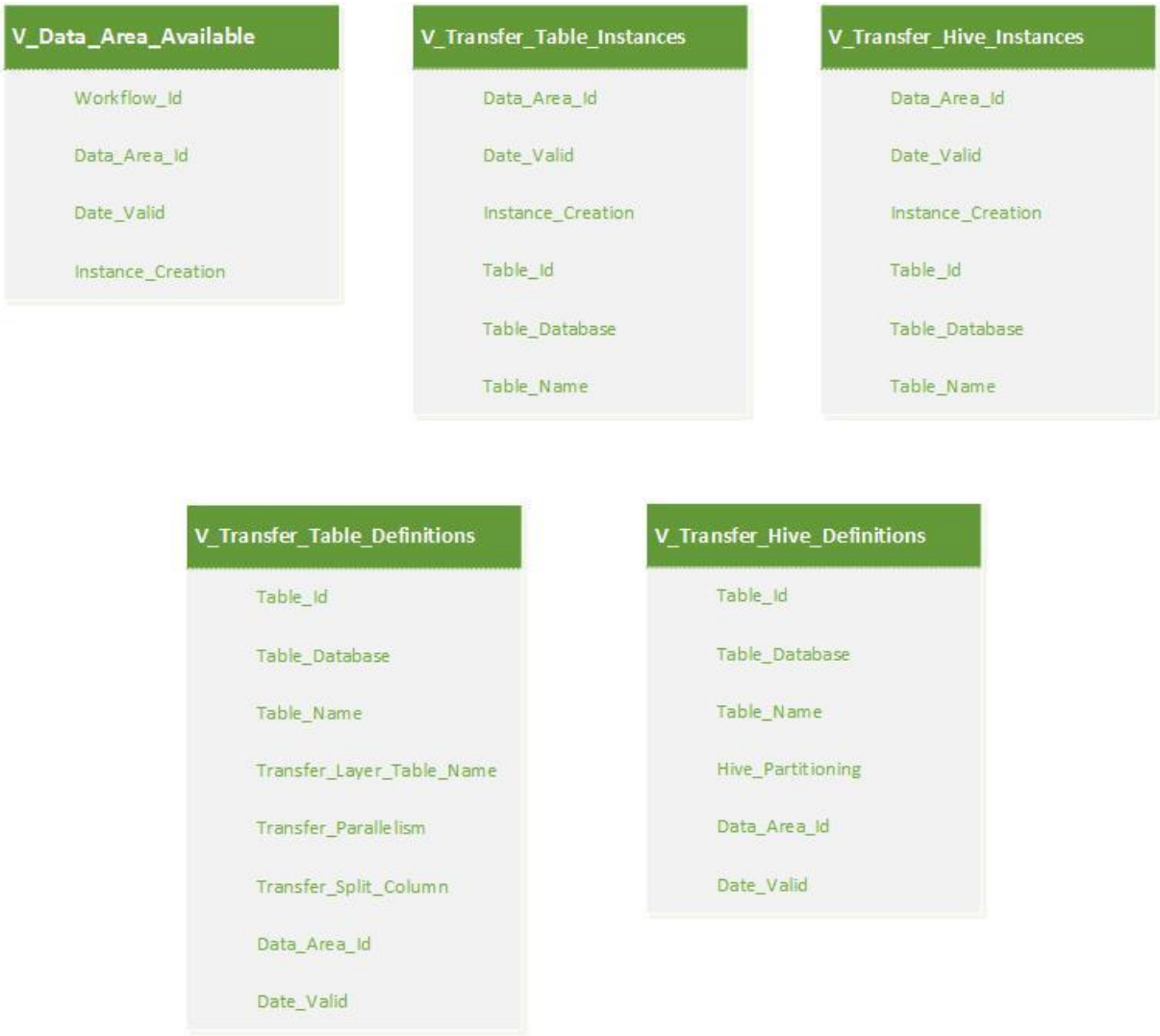
transfer_operation

Obr. č. 8 ER Diagram – transfer_operation (zdroj: autor)

Oblast, která bude uchovávat veškeré informace o realizaci přenosů. To znamená časové značky začátku a konce realizovaných přenosů, ať už na úrovní celé datové oblasti nebo jejich podřízených tabulek a pro jednotlivé entity bude uchovávat aktuální stav přenosu (probíhá, připraven k přenosu, ukončen apod.).

1. Transfer_(Data_Area|Table|Hive)_Instance – tabulky udržující informace o přenosech a pro jednotlivé entity se dost podobají
   1.1. Transfer_Status – aktuální stav dané entity (hodnoty: READY – připraven k přenosu, RUNNING – přenos běží, SUCCEEDED – přenos úspěšně dokončen, FAILED – přenos zhavaroval)
   1.2. Transfer_Start – začátek přenosu
   1.3. Transfer_End – konec přenosu
2. V_Data_Area_Available – výstupní pohled, který udává, které datové oblasti jsou připraveny k přenosu
3. V_Transfer_(Table|Hive)_Instances – pohledy, které vracejí seznam tabulek, pro které se má v daném běhu založit záznam v Transfer_(Table|Hive)_Instance (vysvětleno níže)
4. V_Transfer_(Table|Hive)_Definitions – pohledy vracející parametry pro nastavení dalších částí přenosu

Součástí oblasti je také několik výstupních pohledů, pomocí kterých se budou dílčí fáze procesu dotazovat na potřebné provozní informace.



Obr. č. 9 ER Diagram – transfer_operation, pohledy (zdroj: autor)

transfer_layer

Doplňující výstupní vrstva, která bude pouze poskytovat data externímu procesu a ke které bude přistupovat technický uživatel, který bude nastaven pro použití aplikací Sqoop.

V tomto schématu budou pouze databázové pohledy na objekty v datovém skladu, které jsou určeny k přenosu a bude to jediná oblast, kde bude mít technický uživatel právo číst data. Tato vrstva tak umožní snadnější řízení přístupu technického uživatele k objektům v datovém skladu. Místo toho, aby uživatel musel žádat o přístupy do různých schémat, má právo pouze do této výstupní vrstvy. Tato vrstva sice bude muset mít právo data číst z ostatních kontejnerů ve skladu, ale zprostředkovaně se budou tyto objekty poskytovat pouze pomocí pohledů. To znamená, že externí proces bude mít k dispozici pouze ty tabulky, ke kterým je vytvořen pohled a je tak možné přístup omezit pouze na

vybrané objekty a ne na celý kontejner. Další výhodou je také to, že je možné data pro export v pohledech ještě upravit, např. nepřenášet všechny, ale pouze vybrané sloupce tabulky nebo omezit výstupní množinu dat, kterou pohled vrací.

7.6.4 Nastavení uživatelů

V databázovém systému byli pro přenosový aparát vytvořeni dva uživatelé:

- **u_transfer_data** – uživatel sloužící k přihlášení k databázi pomocí aplikace Sqoop a k přenosu dat z oblasti transfer_layer (právo na čtení)
- **u_transfer_meta** – uživatel používaný v shell skriptech k aktualizaci dat o přenosu v oblasti transfer_operation (právo na čtení i zápis) a nastavení parametrů přenosu na základě dat v transfer_metadata (právo na čtení)

7.6.5 Příprava nových objektů k přenosu

Pokud je potřeba připravit novou oblast k přenosu nebo do stávající oblasti doplnit nové tabulky, je postačující doplnit nové objekty do konfiguračních tabulek v transfer_metadata a nastavit tak parametry pro jejich přenos. Pro každou tabulku je také ještě potřeba vytvořit odpovídající pohled v transfer_layer na tabulku ve zdrojovém schématu.

Při vytvoření úplně nové datové oblasti je potřeba zajistit, aby po ukončení zpracování všech jejich objektů, byla tato skutečnost zaznamenána do metadat ve schématu transfer_operation. Pomocí tohoto záznamu je pak inicializováno přenosové workflow. To, jakým způsobem bude tato režie realizována, je už mimo rozsah této práce a při implementaci a testování přenosů se předpokládá existence odpovídajícího záznamu.

7.7 Přenosové workflow

Jedno přenosové workflow je realizované jako jedna instance workflow v Oozie serveru a jedná se tedy o acyklický orientovaný graf, složený z dílčích úkolů a směrovacích uzlů. Aplikace podporuje několik základních operací na Hadoop clusteru (např. zavolání příkazu Sqoop se zadanými parametry), ale protože v systému, tak jak je navržen, se počet opakování jednotlivých příkazů různí podle aktuálního běhu, nejsou tyto základní kroky použity. Místo těchto úkonů je ve workflow použit typ „Shell action“, který spustí shell skript a ve kterém je implementována veškerá logika daného kroku.

V Oozie se tyto grafy implicitně deklarují pomocí speciálních značek v XML souboru a pomocí interního příkazu lze kontrolovat správnost této definice. Zadání kroků v podobě značek do XML souboru není uživatelsky přívětivé a komerční platformy poskytují pro jeho zadání grafické rozhraní, kde je konstrukce workflow pohodlnější. Definice výsledného workflow je součástí přílohy práce.

Workflow jako takové funguje na principu, že s určitou periodou kontroluje metadatovou oblast v datovém skladu a na základě dotazu do výstupního pohledu určí, zda jsou aktuálně některé datové oblasti pro toto workflow připraveny k přenosu. Pokud jsou pro dané workflow takové datové oblasti připraveny k přenosu, workflow tuto oblast v metadatech označí jako tu aktuálně přenášenou a pokračuje dalším krokem, který tabulky přenese do přípravné oblasti v HDFS a následně i do aplikace Hive.

Pokud žádné datové oblasti připravené nejsou, workflow tuto skutečnost zaznamená, je patřičně ukončeno a po zadaném časovém intervalu je spuštěno znovu. Pro řízení tohoto spouštění se použije tzv. coordinator job, což je v Oozie typ řídicího workflow, které má za úkol pouze spouštět ostatní workflow na základě splnění časových podmínek, dostupnosti dat nebo jiné externí událostí. Oozie dovoluje běh několika instancí jednoho workflow najednou a v této implementaci také k takovému souběžnému běhu může dojít, ale každý běh bude zpracovávat jinou datovou oblast.

Hlavní výhodou přenosového workflow je jeho obecný návrh, který podporuje přenos různých datových oblastí pomocí jednoho procesu. Počet přenášených tabulek a způsob jejich uložení je proměnlivý a je vždy konfigurován až za běhu podle aktuálního nastavení. Přínosem je tak snadné rozšíření přenosu o nové oblasti, popřípadě snadná úprava konfigurace přenosu, aby probíhal co nejefektivněji.

7.7.1 Kroky workflow

Níže je vysvětlena funkce dílčích kroků přenosového workflow a jejich vzájemná posloupnost a návaznost je reprezentována pomocí EPC diagramu.

CheckSource.sh

První shell skript spustí SQL dotaz do pohledu V_Data_Area_Available schématu transfer_operation v datovém skladu a na základě vrácených výsledku určí, které datové oblasti se mají přenést nebo jestli vůbec je nějaká oblast k dispozici a pro tento konkrétní běh se má workflow ukončit.

Pokud nalezne datové oblasti určené k přenosu, vybere nejstarší z nich a v tabulce Transfer_Data_Area_Instance změní status daného záznamu na aktuálně přenášený. S takto označeným záznamem bude pracovat další krok workflow.

SqoopImport.sh

Tento krok sestavuje Sqoop příkazy pro jednotlivé tabulky a jeden po druhém je spouští. Příkazy jsou opět sestaveny na základě SQL dotazu, tentokrát do schématu transfer_metadata, které pro každou tabulku obsahuje aktuálně nastavené konfigurační parametry. Veškeré tabulky jsou nahrány do přípravné oblasti v HDFS, pojmenované jako „\stage“.

Na začátku skriptu jsou pro všechny přenášené tabulky vytvořeny záznamy v transfer_operation.Transfer_Table_Instance s aktuální časovou značkou, datumem platnosti a příznakem, že je tabulka připravena k přenosu. Na konci každého úspěšného Sqoop příkazu jsou tyto údaje o proběhlém přenosu zaznamenány zpět do Transfer_Table_Instance.

HiveImport.sh

Poté co jsou veškeré tabulky přeneseny do HDFS na Hadoop clusteru, jsou některé z nich, podle konfigurace v transfer_metadata, nahrány do aplikace Hive.

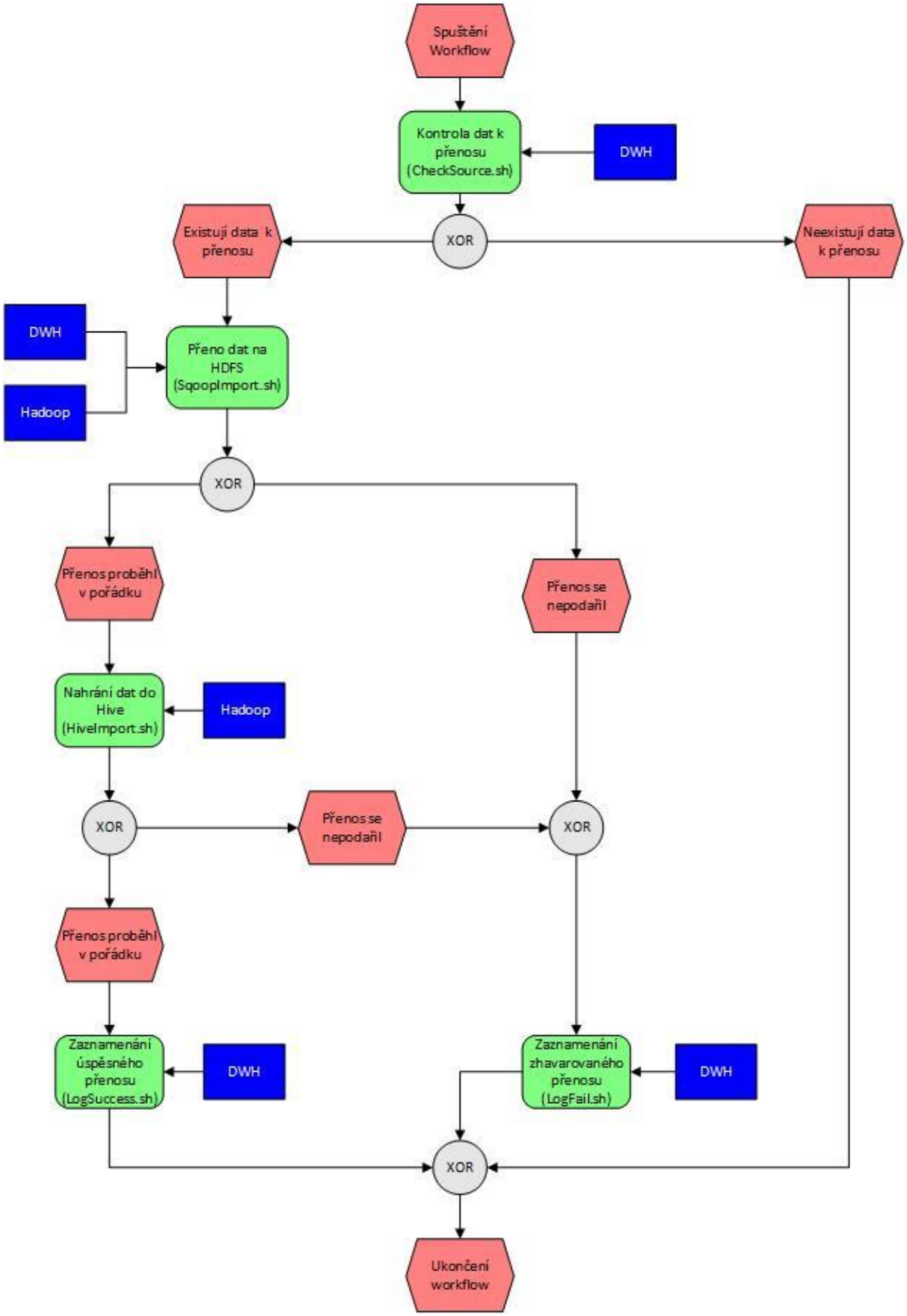
LogSuccessMetadata.sh

Finální fáze jedné instance přenosu. Spustí opět SQL příkaz, který do metadatové tabulky v transfer_operation zaznamená úspěšné dokončení a čas přenosu celé datové oblasti.

LogFailMetadata.sh

Pokud přenos v nějakém kroku z jakéhokoliv důvodu havaruje, je tato skutečnost zaznamenána do transfer_operation a odpovídající záznam je nastaven na „FAILED“. Pokud je potřeba přenos po opravě příčiny pádu opakovat, stačí vytvořit nový záznam pro danou oblast se stejným datem platnosti.

Návaznost jednotlivých kroků je zobrazena na EPC diagramu níže.



Obr. č. 10 EPC diagram – průběh přenosového workflow (zdroj: autor)

7.7.2 Nastavení přenosů Sqoopu

Aplikace Sqoop je komponenta systému, která přímo realizuje datové přenosy. Každý běh Sqoopu je nastaven vždy až při jeho spuštění a to podle aktuálního nastavení pro danou tabuli v transfer_metadata.

Nejdůležitějším parametrem, který pomáhá ovlivnit dobu přenosu, je nastavení počtu paralelních připojení do zdrojové databáze. Každé toto připojení je samostatná mapovací úloha v clusteru, která přenáší podmnožinu dat v tabulce. Základní nastavení je čtyři připojení a celá tabulka je tak rozdělena na čtvrtiny. Rozdělení dat se děje podle primárního klíče tabulky, případně podle sloupce, který je možno specifikovat jako jeden z parametrů. Důležitým faktorem je, aby primární klíč tabulky nebo parametrem specifikovaný sloupec měly rovnoměrně distribuované hodnoty a rozdělené množiny tak byly přibližně stejně velké. Díky tomu bude průběh úloh clusteru a nahrání dat probíhat rovnoměrně. V případě, že tabulka nemá primární klíč nebo nemůžeme vybrat vhodný sloupec, je lepší nastavit počet připojení na jedna, tabulka pak bude celá nahrána pouze pomocí jednoho procesu. Protože v této konkrétní implementaci je vždy jako zdrojový objekt použit databázový pohled, nad kterým není definován primární klíč, je specifikace sloupce pro rozdělení dat v transfer_metadata povinné (31).

Vhodný počet paralelních připojení pomocí Sqoopu se může lišit v konkrétním prostředí a bude potřeba patřičně otestovat, při jakém nastavení bude systém podávat nejlepší výsledky. Vždy je však nutné mít na paměti vytížení databáze a její provozní omezení, protože při nastavení příliš vysokého počtu připojení může být výsledek kontraproduktivní.

Mezi další užitečné vlastnosti Sqoopu patří:

- **ukládání jobů** – Sqoop umožňuje uložení parametrů často používaného příkazu jako tzv. jobu, který je pak možno opakovaně spouštět bez nutnosti pokaždé definovat všechny parametry. Jednotlivé joby jsou uchovávány buď v rámci daného počítače, případně je možné vytvořit tzv. Sqoop metastore, což je nástroj, který uchovává jednotlivé joby a umožnuje je sdílet přes TCP/IP protokol.
- **inkrementální přenos –** Umožňuje přenášet ze zdrojové tabulky pouze ta data, která v ní přibyla od posledního přenosu. K dispozici jsou dvě alternativy, jak s novými záznamy v tabulce pracovat: append – data v tabulce konstantně přibývají (např. události s konstantě narůstajícím identifikátorem), v tom případě jsou přeneseny pouze nově vytvořené záznamy nebo last-modified – v tomto případě se záznamy na zdroji mohou i měnit a jsou tak přenášeny záznamy nové a změněné. Další parametry ovlivňují, pomocí kterého sloupce se posuzuje, zda bude záznam přenesen a jaká je poslední přenesená hodnota v tomto sloupci. Inkrementální přenos se vhodně doplňuje s předešlou vlastností ukládaných jobů a pokud je v těchto jobech použito inkrementální nahrávání, Sqoop udržuje poslední přenášenou hodnotu automaticky (31).

Přesto, že mohou být tyto vlastnosti v určitých případech velice užitečné, nejsou v této implementaci použity. Ukládání jobů není implementováno, protože by znamenalo další operační zátěž při změnách systému a úpravě nastavení existujících příkazů. Navíc nastavení těchto jobů by bylo „ukryto“ na dalším místě v systému. Takto je veškerá konfigurace přehledně uložena na jednom místě, spolu s nastavením celého aparátu v transfer_metadata. Každý Sqoop příkaz je vystavěn znovu pomocí aktuálních parametrů a není potřeba měnit nastavení daného jobu. Spolu s nepoužíváním uložených jobů je také opuštěno použití inkrementálního přenosu. Pokud bude potřeba tuto funkčnost pro určité objekty realizovat, vhodným řešením bude omezit množinu dat, kterou poskytuje výstupní pohled v databázi transfer_layer. Pohled může k omezení množiny přenášených dat používat například metadata v transfer_operation nebo jakoukoliv jinou tabulku v datovém skladu a logicky tak vytvořit inkrementální množinu dat. Pro uživatele tak bude přehlednější při práci se systémem prověřit aktuální data vrácená pomocí tohoto pohledu než hledat v nastavení Sqoopu poslední přenesenou hodnotu odpovídajícího jobu.

7.7.3 Použití nástroje Sqoop s databázi Teradata

Následující podkapitola se bude v krátkosti věnovat použití nástroje Sqoop k přenášení dat z analytického databázového systému Teradata, který se používá jako platforma pro provozování datového skladu. Protože se jedná o rozsáhlé a náročné řešení, je konfigurace a použití tohoto nástroje diskutováno pouze teoreticky a není součástí vzorové implementace.

Architektura systému Teradata

Jedná o relační databázový systém, který je od základu navrhnut tak, aby co nejvíce vyhovoval složitým analytickým dotazům, a je proto primárně určen k budování datových skladů. Hlavním principem tohoto řešení je masivní paralelní zpracování každého dotazu. Jeden fyzický server je složen z několika dílčích výpočetních jednotek (*Access Module Processors* neboli *AMP*) a každá z nich pracuje pouze s dílčí části celé databáze. Většinou má každá datová tabulka některé sloupce určené jako primární index (je však možné mít tabulky i bez primárního indexu), a pomocí hashe, který se spočítá na základě indexu, je vybrán AMP na který bude záznam uložen. V ideálním případě je tak tabulka rozdělena po všech AMPech v systému a zpracování tabulky je maximálně paralelizované. Každý AMP tak vlastně zpracovává pouze svou část dat, kterou nesdílí s jinými AMPy. Jedná se tak o komplexní softwarové řešení, které stojí také na speciálně nakonfigurovaném hardwarovém návrhu (54).

Princip samotného zpracování je dost podobný principu zpracování pomocí Big Data technologií, oproti nim však zde je velký rozdíl v architektuře celého systému. Jednotlivé výpočetní uzly jsou totiž provozovány v rámci jednoho serveru, jsou více infrastrukturně seskupeny dohromady a není potřeba tolik řešit odolnost jednotlivých uzlů proti výpadku. Přesto i takové situace mohou v tomto systému nastat a databáze má postupy, jak se s nimi vypořádat (54).

Scoop connector

Při použití Sqoopu s Teradatou je potřeba doplnit jeho instalaci o patřičnou knihovnu konektoru, stejně tak jak se doplnil konektor pro databázi MySQL ve vzorové implementaci. Tato knihovna poté poskytuje další funkcionalitu nástroje, kterou je možné konfigurovat pomocí dalších parametrů při zadávání příkazu Sqoop. Tento konektor v obdobných variantách naimplementovali různí poskytovatelé komerčních distribucí Big Data platforem. Za zmínku určitě stojí i oficiální implementace obdobného nástroje vydaná společností Teradata, nesoucí název *Teradata Connector for Hadoop*, která se používá obdobně jako nástroj Sqoop (55, 56).

Velice důležitá je vlastnost konektoru, pomocí které je rozdělena vstupní množina dat při přenosu mezi těmito dvěma prostředími. (Popis vychází z implementace konektoru poskytovaného společností Cloudera, ale tato funkčnost bývá u jiných poskytovatelů implementována obdobně.) Vstupní množina dat je totiž rozdistribuována po jednotlivých AMPech na základě hashe daného záznamu. To, jak bude Sqoop data z těchto AMPů získávat, tak bude mít velký vliv na efektivitu systému (55).

Možné parametry pro konfiguraci rozdělení přenosu jsou:

- split.by.amp – Vstupní množina je rozdělena podle uložení na jednotlivých AMPech a následně je pro každý AMP vytvořeno jedno připojení, které data přenáší právě z daného AMPu. Tento postup je doporučovaný, ale je možné ho použít pouze u novějších verzí Teradaty.
- split.by.value – Rozdělení je tvořeno obdobně jako u běžného použití příkazu Sqoop a data tabulky jsou rozdělena podle daného sloupce a jeho konkrétních hodnot.
- split.by.hash – Rozdělení je opět založeno na rozmezí hodnot, tentokrát ale podle hodnoty hashe, který se používá k rozdělení dat po AMPech.
- spli.by.partition – Doporučená možnost při přenosu velkých tabulek, jehož výkonnost je ale hodně ovlivněna tím, zda a jak je zdrojová tabulka rozdělená do tzv. „partitions“. Což je způsob interního rozdělení dat při jejich uložení v Teradatě (55).

Výše uvedené parametry poskytují uživatelům několik možností, jak data přenášet z tohoto systému co nejefektivněji, v kontextu dané databázové technologie. Příkaz Sqoop je tedy možné více optimalizovat pro použití s konkrétním databázovým systémem a při zvažování možností jeho použití je důležité mít tento fakt na paměti.

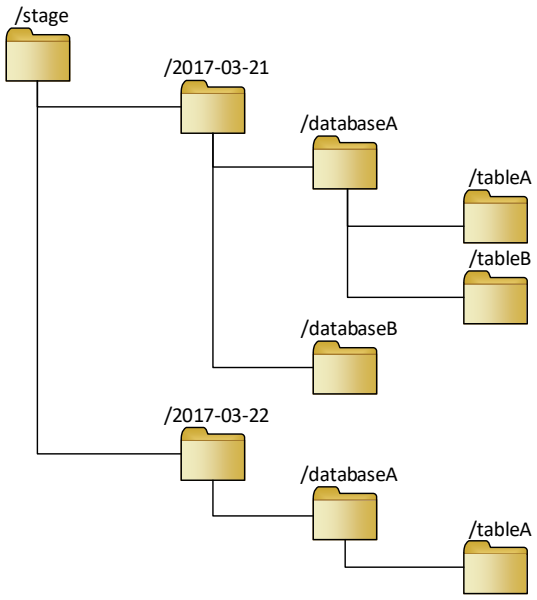
7.7.4 Přenos dat do HDFS a do aplikace Hive

Nahrání dat do HDFS clusteru je realizováno ve dvou fázích:

Přenos na HDFS

Pro aplikaci Sqoop je cílovou destinací oblast HDFS v clusteru a další zpracování je prováděné samostatným navazujícím procesem. Pro všechny Sqoopem přenášená data je vyhrazena složka \stage\, kam jsou dočasně data přenesena a odkud se budou nahrávat do aplikace Hive nebo čekat na manuální nebo jiné zpracování.

Protože HDFS funguje obdobně jako klasický souborový systém běžných operačních systému, je rozdělení dat řízeno pomocí několika úrovní adresářů. První úroveň je datum platnosti přenášených dat, druhou je název databáze odkud byla tabulka přenesena a třetí je název tabulky. V poslední podsložce už budou uložené datové soubory tak, jak je tam nahrála aplikace Sqoop. (Ta je přenáší v podobě např. „part-m-00000“, přitom souborů může být ve složce několik, podle toho na kolik připojení bylo nahrání rozděleno a jsou identifikovány číselnou příponou.) Struktura složek tedy pro několik tabulek může vypadat následovně:



Obr. č. 11 Uspořádání nahrávaných dat do souborového systému HDFS (zdroj: autor)

Data tak na nejvyšší úrovni souborového systému budou dělena podle jejich data platnosti a bude tak možné snímky tabulky i několik dnů uchovávat přesně tak, jak přišly ze zdrojového systému. Odmazávaní už nepotřebných dat v oblasti stage může probíhat dávkově po určité periodě, například v době, kdy není cluster příliš vytěžován. Realizace tohoto procesu ale není součástí této práce.

Nahrání do Hive

V druhé fázi jsou některé tabulky, podle konfigurace v metadatech, nahrány do předem připravených tabulek v aplikaci Hive. V tomto kroku se předpokládá, že data ze zdrojových tabulek mají stejnou strukturu jako tabulka ve zdrojové databázi. Pokud by bylo nutné v datovém skladu definici dat upravit (např. přidat nebo odebrat sloupec), je nutné tuto úpravu provést také v cílové Hive tabulce.

Pro nahrání dat se použije příkaz LOAD DATA, který se pokusí ze zadané lokace v souborovém systému HDFS nahrát všechny soubory, které odpovídají zadané masce. V masce je možno použít i zástupné znaky a je tak možné cíleně zpracovat i několik souborů najednou. Díky tomu je tak možné nahrát několik souborů, které vytvoří Sqoop díky svému paralelnímu nahrání dat. (Všechny soubory vytvořené Sqoopem odpovídají masce part-*.) Pokud nahrání proběhne v pořádku, jsou soubory ze zdrojové lokace v HDFS odmazány (57).

Struktura schémat v aplikaci Hive je obdobná rozdělení schémat v klasickém relačním systému. Aby se předešlo možné kolizi stejně pojmenovaných databázi v Hivu a v datovém skladu, jsou názvy databází v Hivu doplněny o předponu DWH_.

Další funkcí, které mohou některé tabulky využívat, je tzv. partitioning. Jedná se o rozdělení Hive tabulky podle dalšího sloupce, které umožňuje efektivněji se nad tabulkou dotazovat. (Reálně se jedná o další úroveň v souborovém systému HDFS.) Tabulky budou v tomto případě rozdělované podle data platnosti, tak jak byly přeneseny ze zdrojového systému. Dosáhne se tak přehledného uložení denních snímků v dané tabulce, tak jak se data v ní měnila v průběhu času. Pokud data z nějakého důvodu z datového skladu v daný den nepřijdou, bude partition pro dané datum platnosti vynechána. Tabulky, u kterých je žádoucí jejich snímky uchovávat, musí mít nastaveny odpovídající parametr v transfer_metadata, ostatní tabulky v Hive nebudou partitioning využívat a vždy budou přemazány novějšími daty. Pro nahrání dat s využitím této funkčnosti je potřeba pouze rozšířit příkaz LOAD DATA o daný parametr a hodnotu ukládané partitiony (57).

Jeden vzorový příkaz pro nahrání jedné množiny souborů do Hive, tak může vypadat například takto:

```
LOAD DATA INPATH "/stage/2017-03-02/SchemaName/TableName/part-*"
OVERWRITE INTO TABLE DWH_SchemaName.TableName partition (Load_Data =
"2017-03-02");
```

Hive také poskytuje ještě další způsob, jak umožnit práci s daty uloženými v souborech a tím je použití tzv. external table. Při založení tabulky je možné tabulku označit jako externí a specifikovat cestu k souborům v HDFS, ze kterých se má tabulka skládat. Hive místo toho, aby data ukládal v adresářové struktuře určené pro aplikaci Hive, použije při práci s tabulkou data v zadané lokalitě. Tato volba je vhodná například pro velké množství souborů, které budou vytvářeny jiným procesem a budou v dané lokalitě kontinuálně přibývat.

V této implementaci se neuvažuje použití této funkcionality. Data jsou místo toho konsolidována do klasické tabulky a rozdělována podle partitioningu, který umožňuje sledování vývoje dané entity podle jednotlivých dat platnosti. Vhodným kandidátem pro externí tabulku tak mohou být data, která obsahují pouze události, které se vyskytly v určitém čase a pro danou událost není možné sledovat vývoj její změny v průběhu času.

7.7.5 Příklad vzorového přenosu

Vzorový příklad představuje jeden běh přenosového aparátu a detailně popisuje jeho kroky.

1. Výchozí situace pro začátek přenosu: V datovém skladu bylo dokončeno zpracování datové oblasti a všechny tabulky této oblasti jsou tak připraveny k přenosu. Na konci tohoto zpracování byl uložen záznam do transfer_operation.Data_Area_Instance se statusem, že daná datová oblast je připravena pro přenos (Transfer_Status = „READY“).
2. V systému Oozie je spuštěno workflow DWH_HDFS_Main, které přenáší danou datovou oblast. Prvním krokem je spuštění skriptu CheckSource.sh, který jako první krok spustí dotaz do pohledu tranfer_operation.V_Data_Area_Available. Dotaz vrátí identifikátor oblasti a datum platnosti dat a tyto údaje jsou uloženy do proměnných v Oozie workflow, aby s nimi mohly pracovat navazující kroky. Přenos tedy bude zahájen a další příkaz změní status oblasti na „přenášená“ (Transfer_Status = „RUNNING“).
3. Dále je spuštěn krok SqoopImport, který pro všechny tabulky dané oblasti vytvoří záznam v tabulce Transfer_Table_Instance se zadaným datem platnosti. Zároveň také všechny předešlé nedokončené instance označí na „FAILED“. (Kvůli opakovatelnosti zohlední pouze tabulky, které nemají k danému datu platnosti dokončený přenos. viz níže) A následně pro každou tabulku spustí Sqoop příkaz. Po každém dokončeném přenosu je záznam v Transfer_Table_Instance označen jako dokončený (Transfer_Status = „SUCCEEDED“).
4. Navazující krok HiveImport získá seznam všech tabulek, které se mají nahrát do Hive, podle příznaku v Transfer_Table_Instance. Pro jednotlivé tabulky spustí příkaz load data a pro tabulky, které používají partitioning použije jejich datum platnosti. Po konci každého příkazu je doplněn čas dokončení do metadat v transfer_operation.
5. Po dokončení všech kroků je spuštěn skript LogSuccessMetatada.sh, který pro danou datovou oblast nastaví status na úspěšně dokončený (Transfer_Status = „SUCCEEDED“) a doplní také časovou značku této události. Pokud v jakýkoliv předchozí krok havaruje, pokračuje workflow krokem LogFailMetadata a status přenosu dané datové oblasti je nastaven na nedokončený (Transfer_Status = „FAILED“).

7.7.6 Opakovatelnost přenosů

Pokud přenos zhavaruje je označen jeho stav na „FAILED“ a poté, co je opravena příčina tohoto pádu, je možné celé worflow spustit znovu tím, že do tabulky Transfer_Data_Area_Instance bude vložen nový záznam. Takto opakované workflow však už nebude přenášet tabulky, pro které k danému datu platnosti už přenos skončil úspěšně. Předpokládá se, že už nahrané tabulky jsou v pořádku a zbývá pouze přenést ty nedokončené. Po opravě tak workflow bude přenášet pouze ještě nezpracované objekty a nebude se tak zbytečně prodlužovat celková doba běhu workflow. Pokud by však bylo potřeba už přenesená data nahrát znovu, například kvůli chybě v datech, bude třeba všechny odpovídající záznamy označit v Transfer_Table_Instance jako „FAILED“.

7.7.7 Omezení navrhnutého systému

Při vývoji systému, a při následném testování jeho funkčnosti, vyplynulo několik požadavků na celý přenosový systém, které je pro tuto konkrétní implementaci nutné respektovat.

Kroky celého procesu jsou vykonávány pomocí workflow v Oozie a to spouští úkony v clusteru pomocí frameworku YARN. Tato komponenta řídí zpracování každé úlohy v clusteru a to tak, že úlohu rozdělí do menších částí a tyto části pak přiřadí jednotlivým uzlům v clusteru ke zpracování. Přiřazení úloh jednotlivým uzlům se děje na základě aktuálního vytížení a také dat, které jsou na uzlu uložena. Uzel se pak snaží svoji část úlohy zpracovat, a pokud se mu to nepodaří, například z důvodu nějaké chyby nebo výpadku uzlu, řídící komponenta YARN určí ke zpracování této části úlohy uzel jiný. Výběr uzlů ke zpracování je tak do velké míry náhodný (25).

YARN přesto poskytuje možnosti, jak řídit zpracování na uzlech clusteru. Pomocí konceptu tzv. „labels“ můžeme jednotlivé uzly oštítkovat a celý cluster rozdělit do několika disjunktních množin (každý uzel může mít nejvýše jeden štítek). Druhý konceptem jsou tzv. „queues“ neboli fronty, které umožňují řídit výpočetní kapacity pro jednotlivé aplikace v clusteru. Jedná se o stromově rozdělenou strukturu, která určuje kolik dané kapacity clusteru si může která fronta alokovat pro své výpočty a při zadání nové úlohy v clusteru je možné specifikovat, do které fronty se úloha zařadí. Fronty tak můžou rozdělit kapacity uzlu například podle oddělení ve společnosti nebo podle druhu zpracovávaných úloh a je tak například možné pro kritické úlohy v clusteru mít vytvořenu specifickou frontu, která zajistí, že její úlohy budou mít vždy dostatek výkonu. (Fronty je navíc možné nastavit i tak, aby nevyužité kapacity byly sdíleny s ostatními frontami.) Koncepty „queues“ a „labels“ je možné společně kombinovat a tak například určit, že některou množinu oštítkovaných uzlů, může k výpočtům používat pouze určitá fronta apod. (23, 52)

Oozie workflow je spuštěno v podobě MapReduce úlohy na clusteru, každý jeho jeden krok je přiřazen náhodnému uzlu a tento uzel se snaží přiřazený krok zrealizovat. Protože logika přenosu je realizována pomocí shell skriptů, pokusí se tento vybraný uzel provést tento skript. Z tohoto faktu vznikají dvě hlavní omezení systému:

- Na všech uzlech clusteru, je nutné mít nainstalovanou a nakonfigurovanou aplikaci HiveCLI (popřípadě novějšího klienta Beeline), aplikaci Sqoop a klienta MySQL, aby bylo možné spouštět v shell skriptech tyto příkazy.
- Protože skript může být spuštěn na jakémkoliv uzlu v systému, je nutné, aby každý uzel byl schopný se připojit k externímu serveru, na kterém se provozuje datový sklad. Nastavení počítačové sítě, ve které jsou uzly clusteru provozovány proto musí tento fakt respektovat.

Výhodou této realizace je to, že systém není svázán s funkčností jednoho daného uzlu, ale může probíhat na uzlu libovolném. Rizikovým místem systému se tak stávají prvky Hive a Oozie, které ale v produkčním prostředí mohou být nainstalovány na více uzlů a při kombinaci s duplikovanou podkladovou databází, kterou tyto aplikace používají, bude zajištěna i jejich vysoká dostupnost. Celý přenosový systém, na straně Big Data platformy, se tak stane odolným vůči výpadku náhodného uzlu.

Pokud by tato zvolená implementace nebyla v produkčním prostředí vhodná, je možné „shell akce“ ve workflow zaměnit za tzv. „ssh akce“, což je spuštění daných příkazů na specifickém uzlu, pomocí protokolu ssh. Tím bude zajištěno, že skripty bude provádět vždy ten stejný stroj. (Například pokud pro komunikaci s externími systémy mimo síť je určen pouze určitý uzel v clusteru.)

8 Možnosti pro další vývoj systému

Následující kapitola se věnuje dalším možnostem jak efektivnost a funkčnost přenosového systému dále rozšířit. Vzhledem k primárnímu zaměření práce, je už implementace těchto prvků mimo její rozsah a jsou zde diskutovány pouze teoreticky.

8.1 Zvýšení paralelismu přenosu

Jak bylo uvedeno výše, aplikace Sqoop umožňuje zvýšit efektivitu přenosu dat tím, že tabulku rozdělí na několik disjunktních množin a každou z nich pak pomocí samostatného připojení do zdrojové databáze nahraje do cílové lokace na HDFS. Přenos jedné tabulky tak probíhá paralelně.

Při přenosu datové oblasti, která se bude většinou skládat z několika tabulek, však přenos jednotlivých objektů probíhá pořád sériově: další tabulka se začíná nahrávat až tehdy, kdy je nahrána tabulka předchozí. Proto dalším možným přístupem, jak ještě zvýšit efektivitu celého aparátu, by tak mohla být paralelizace spouštěných Sqoop příkazů a nahrávání několika tabulek najednou (58).

Jedna z implementací pak může vypadat tak, že v shell skriptu bude nad seznamem tabulek k přenosu probíhat cyklus, ve kterém se budou jednotlivé příkazy Sqoop spouštět paralelně, ale pouze do uživatelem zadaného limitu paralelních běhů. Pokud je přenos dokončen, sníží se proměnná udávající počet aktuálně spuštěných Sqoop procesů, v seznamu tabulek se ukazatel přesune na další tabulku a v shell cyklu se spustí další Sqoop příkaz. Cyklus tak projde celým seznamem tabulek a zajistí, že se vždy bude provádět několik příkazů Sqoopu najednou. (Detailní implementace viz: https://github.com/sourygnahtw/hadoopUtils/blob/master/scripts/sqoop/sqoopTables.sh)

V této implementaci se vychází z toho, že většina tabulek pro přenos není objemově velká, a tak je počet paralelních připojení Sqoopu nastaven na jedna. Cílem tohoto přístupu je tak minimalizace času, který je potřebný na získání metadat o tabulce a rozdělení úlohy do několika procesů. Urychlení při paralelním přenosu malého objemu dat totiž není takové, jako s tím spojená režie nutná k získání metadat a nastavení celého běhu. Jinými slovy, objemově malé tabulky je efektivnější nahrát pouze pomocí jednoho připojení. (58)

Dalším rozšířením tohoto řešení může být zakomponování nastavení různého počtu připojení pro různé tabulky v přenosu. Aparát by se potom neřídil limitem současně běžících Sqoop příkazů, ale celkovým součtem všech realizovaných připojení do zdrojové databáze, nezávisle na tom, kolik Sqoop příkazu je provádí. (Například pokud se limit připojení do databáze nastaví na 16 a aktuálně se využívá 12 připojení, je možné paralelně spustit Sqoop příkaz, který má počet připojení nastaven na 4 apod.)

Správné nastavení celého aparátu by se pak optimalizovalo až v konkrétním prostředí a hlavními faktory by byl celkový počet připojení do zdrojové databáze a optimální počet připojení pro jednotlivé

tabulky. Tato vlastnost pro zvýšení přenosové rychlosti však v modelové implementaci postrádala smysl, protože v takto virtualizovaném prostředí není možné relevantně porovnat výkonnostní změny v chování systému.

8.2 Automatické aktualizace definice v Hive

Jedním z problémů při provozování přenosů mezi dvěma platformami je, že při změnách ve struktuře přenášených objektů je většinou třeba tyto struktury upravit hned na několika místech. V případě, že dojde například k přidání nebo odebrání sloupců ve zdrojové tabulce v datovém skladu, bude třeba tuto definici patřičně opravit i v systému Hive. Tento fakt přináší jednak větší pracnost pro provozovatele těchto řešení a také je zde větší prostor pro chybně provedenou konfiguraci.

Řešením by mohlo být přidání kroku do přenosového workflow, který by před nahráním dat do tabulky v Hivu definici zdrojových dat porovnal s definicí dat v cílové aplikaci. Pokud by došlo k rozdílu, byla by cílová tabulka založena znovu tak, aby vyhovovala datům ze zdroje.

Získání definice zdrojových tabulek by se dalo realizovat pomocí dotazu do metadatového schématu, ve kterém si databáze udržuje informace o všech objektech, které obsahuje (např. v databázi MySQL to je „information_schema“). Takové schéma obsahuje většina databází a reálně se jedná o sadu běžných databázových tabulek, ze kterých je možno data získat běžným SQL dotazem.

Oproti tomu definice tabulek z cílové aplikace Hive je možno získat buď obdobným způsobem, protože k udržování informací o objektech používá Hive schéma v relační databázi (viz výše uvedený Hive metastore) nebo je možné k tomuto účelu použít službu HCatalog. Je to součást standardní instalace Hive a poskytuje vrstvu, přes kterou je možné pracovat s definicemi objektů uložených v metastoru systému Hive. Definici objektů tak je možné získat pomocí jednoduchých příkazů přes toto rozhraní. Navíc je možné k této službě přistupovat i přes webové rozhraní WebHCat (26).

Aby definice ze zdrojového a cílového systému byly porovnatelné, je třeba specifické datové typy jedné platformy převést na odpovídající typ v cílovém systému. (Například Hive v prvních verzích nepodporoval datový varchar a pro ukládání datových řetězců se používal datový typ string, který zase není definován v MySQL. Proto by bylo nutné definovat pravidlo, aby typ varchar v MySQL byl v Hivu definován jako string, apod.) Seznam pravidel pro překlad definice, aby vyhovovaly systému Hive, by tak bylo nutné navrhnout podle konkrétní zdrojové databáze a používaných datových typů.

Jedním ze způsobů, jak na identifikaci změny definice reagovat, by mohlo být přejmenování aktuální tabulky a vytvoření tabulky znovu, podle aktuální definice dat, které v daném běhu přišly ze zdroje. Přejmenování by probíhalo do podoby: [názevtabulky]_[datumpřenosu] (např. SampleTable_20170104) a byla by tak zachována stará definice tabulky s informací datumu platnosti, kdy ke změně došlo.

Výhodou tohoto řešení je poměrně jednoduché zajištění opravy změněné definice tabulky, udržení všech dříve nahraných dat a odvrácení havárie daného přenosu. Nevýhodou je, že uživatelé, kteří nebudou o této změně včas informováni, mohou pracovat s novou verzí tabulky a postrádat přitom data, která zůstala uložena ve staré tabulce. Pokud navíc dojde k větší změně tabulky a některé sloupce budou odstraněny, mohou některé úlohy a dotazy přestat fungovat úplně. Pro tyto případy by tak bylo vhodné vždy tuto změnu struktury zaznamenat do řídicích metadat a následně provést ručně opravu navazujících úloh, popřípadě správně přeplnit data z historických tabulek do tabulky nové.

9 Závěr

Práce seznamuje čtenáře s pojmem Big Data, co tyto nové technologie představují a jaké jsou jejich vlastnosti a charakteristiky. Popisuje princip zpracování pomocí Big Data technologií a popisuje, s jakými typy dat tyto nástroje nejčastěji pracují a také jak tato data vznikají. Dále se práce zabývá přehledem technologií Business Intelligence a prezentuje jejich možné aplikace a přínos pro podnik. Následně popisuje, jak tyto technologie vhodně propojit s Big Data přístupy, které mají v analýze dat velké uplatnění. Součástí práce je také popis systému Hadoop, který je jeden z předních představitelů Big Data technologií, a také s ním nejčastěji používaných doplňujících komponent.

V praktické části, kapitola 7, je naplněn hlavní cíl práce, a to vytvoření návrhu přenosového systému pro datovou integraci mezi klasickým datovým skladem a Big Data platformou provozující systém Hadoop. V kapitole 7.3 se diskutují možné způsoby, jak je možné data do systému Hadoop nahrávat a jaký nástroj bude v konkrétní implementaci použit. Celý aparát je doplněn o návrh databázového schématu metadat, pomocí kterého se bude systém konfigurovat a který také bude udržovat informace o jeho provozu. Této části včetně rozdělení schémat a objektů, které budou obsahovat, se věnuje podkapitola 7.6. Dílčím cílem byla realizace tohoto návrhu, čehož bylo v práci dosaženo pomocí virtualizačních nástrojů a detailní popis technické instalace je obsažen v kapitole 7.5. Ostatních dílčích cílů bylo dosaženo zejména v teoretické části, kde kapitola 3 obecně vysvětluje pojem Big Data a možnosti aplikací těchto technologií a kapitola 4 se věnuje přístupům Business Intelligence a jejich přínosu pro podnik.

Hlavního cíle této práce tedy bylo úspěšně dosaženo, stejně tak jako všech cílů dílčích. Přehled všech cílů, spolu s kapitolami, které se naplnění těchto cílům věnují, je uveden níže v tabulce.

Cíl práce	**Číslo kapitoly**
Návrh přenosového systému	**kapitola 7**
Popis technologií Big Data	kapitola 3
Přehled technologií Business Intelligence	kapitola 4
Popis nástrojů pro přenos dat	kapitola 7.3
Realizace vzorové implementace	kapitola 7 (technický popis: 7.5)

Tabulka č. 2 Přehled splněných cílů a odpovídajících kapitol (zdroj: autor)

Jako další možnosti rozvoje systému se jeví například možnost přidání funkcionality, která bude automaticky přenášet definice objektů mezi oběma platformami a bude reagovat na změny struktur vhodným způsobem.

10 Zdroje

(1) CHEN, M., SHIWEN, M. a Y. LIU. *Big Data: A Survey*. [online] Springer. 2014. [cit. 10. 9. 2016] Dostupné z: http://www.ece.ubc.ca/~minchen/min_paper/BigDataSurvey2014.pdf

(2) RUSSOM, P. *Big Data Analytics*. [online] TDWI best practices report, 2011.[cit. 10. 9. 2016] Dostupné z: https://vivomente.com/wp-content/uploads/2016/04/big-data-analytics-white-paper.pdf

(3) MAYER-SCHÖNBERGER, V. a K. CUKIER. *Big Data: A Revolution that Will Transform how We Live, Work, and Think* . [online] New York: Houghton Mifflin Harcourt, 2013. ISBN 978-0-544-00269-2 [cit. 2016-10-26] Dostupné z: https://books.google.cz/books?id=uy4lh-WEhhIC

(4) HOLUBOVÁ, I., KOSEK, J., MINAŘÍK, K. a D. NOVÁK. *Big Data a NoSQL databáze*. Praha: Grada Publishing, a.s., 2015. ISBN 978-80-247-5466-6

(5) THUSSO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P. a R. MURTHY. *Hive – A Warehousing Solution Over a Map-Reduce Framework*. [online] VLDB Endowment, 2009 [cit. 2016-11-13] Dostupné z: http://www.vldb.org/pvldb/2/vldb09-938.pdf

(6) OLSTON, CH., REED, B., SRIVASTAVA, U., KUMAR, R. a A. TOMKINS. *Pig Latin: A Not-So-Foreign Language for Data Processing*. [online] ACM, 2008. [cit. 2016-11-12] Dostupné z: http://www.dcs.bbk.ac.uk/~dell/teaching/cc/paper/sigmod08/p1099-olston.pdf

(7) DEAN, J. a S. GHEMAWAT. *MapReduce: Simplified Data Processing on Large Clusters*. [online] ACM, 2008. [cit. 2016-11-13] Dostupné z: https://www.usenix.org/legacy/publications/library/proceedings/osdi04/tech/full_papers/dean/dean_html/

(8) ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J. a S. SHENKER. *Spark: Cluster Computing with Working Sets*. [online] HotCloud, 2010. [cit. 2016-11-09] Dostupné z: https://amplab.cs.berkeley.edu/wp-content/uploads/2011/06/Spark-Cluster-Computing-with-Working-Sets.pdf

(9) NOVOTNÝ, O., POUR, J. a D. SLÁNSKÝ. *Business Intelligence: Jak využít bohatství ve vašich datech*. Praha: Grada Publishing, a.s., 2005. ISBN 80-247-1094-3

(10) GÁLA, L., POUR, J. a Z. ŠEDIVÁ. *Podniková informatika*. Praha: Grada Publishing, a.s., 2015. ISBN 978-80-247-5457-4

(11) ŠOLTYS, M. *Big Data v technológiách IBM*. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky.

(12) ŠVAGR, L. *Hadoop NoSQL databáze* Praha, 2016. Bakalářská práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky.

(13) DVORSKÝ, B. *Využití Big Data v bankovním prostředí*. Praha, 2016. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky.

(14) NOVÁKOVÁ, M. *Analýza Big Data v oblasti zdravotnictví*. Praha, 2016. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky.

(15) KORKISCH, J. *Nové trendy v Business Intelligenec. Zaměření na Big Data a Hadoop*. Praha, 2014. Bakalářská práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky.

(16) KOCH, M. a B. NEUWIRTH. *Datové a funkční modelování*. Brno: Cerm. 2010. ISBN 978-80-214-4125-5

(17) ZENDULKA, J. a I. RUDOLFOVÁ. *Databázové systémy*. Studijní opora. FIT VUT v Brně 2006.

(18) SAGIROGLU, S. a D. SINANC. *Big data: A review*. [online] IEEE, 2013. [cit. 2017-01-14] Dostupné z: https://xa.yimg.com/kq/groups/72986399/1585974627/name/06567202.pdf

(19) HAN, J., et al. *Survey on NoSQL Database*. [online] IEEE, 2011 [cit. 2017-03-14] Dostupné z: http://s3.amazonaws.com/academia.edu.documents/38476517/06106531.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1492545399&Signature=NtP63nbCyyAwwvtOExLntoKh5hk%3D&response-content-disposition=inline%3B%20filename%3DSurvey_on_NoSQL_Database_PCN_and_CAD_Cen.pdf

(20) CHAUDHURI, S., DAYAL, U. a V. NARASAYYA. *An overview of business intelligence technology*. [online] ACM, 2011. [cit. 2017-03-21] Dostupné z: http://web.ist.utl.pt/~ist13085/fcsh/sio/casos/BI-Tech2.pdf

(21) TYRYCHTR, J. *Provozní a analytické databáze – Teoretické základy*. [online] Praha: ČSVIZ, 2015. [cit. 2017-02-12] Dostupné z: https://books.google.cz/books?isbn=8087968026

(22) LABERGE, R. *Datové sklady – Agilní metody a business intelligence*. Brno: Computer Press, 2012. ISBN 978-80-251-3729-1

(23) APACHE SOFTWARE FOUNDATION. Apache Hadoop 2.7.2 *hadoop.apache.org* [online] © 2014 [cit. 2016-11-18] Dostupné z: http://hadoop.apache.org/docs/r2.7.2/

(24) APACHE SOFTWARE FOUNDATION. HDFS Users Guide. *hadoop.apache.org* [online] © 2016 [cit. 2016-11-18] Dostupné z: http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

(25) APACHE SOFTWARE FOUNDATION. Apache Hadoop YARN. *hadoop.apache.org* [online] © 2016 [cit. 2016-12-10] Dostupné z: http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html

(26) APACHE SOFTWARE FOUNDATION. Apache Hive. *cwiki.apache.org* [online] © 2016 [cit. 2016-12-10] Dostupné z: https://cwiki.apache.org/confluence/display/Hive/Home

(27) APACHE SOFTWARE FOUNDATION. Welcome to Apache Pig. *pig.apache.org* [online] © 2012 [cit. 2016-12-10] Dostupné z: https://pig.apache.org/

(28) APACHE SOFTWARE FOUNDATION. Apache Spark – Lightning-fast cluster computing. spark.apache.org [online] © 2016 [cit. 2016-12-11] Dostupné z: http://spark.apache.org/docs/latest/

(29) APACHE SOFTWARE FOUNDATION. Oozie, Workflow Engine for Apache Hadoop. *oozie.apache.org* [online] © 2014 [cit. 2016-12-11] Dostupné z: http://oozie.apache.org/

(30) LUBLINKSY, B. a M. SEGEL. Introduction to Oozie. *infog.com* [online] © 2011 [cit. 2016-01-14] Dostupné z: https://www.infoq.com/articles/introductionOozie

(31) APACHE SOFTWARE FOUNDATION. Sqoop User Guide (v1.4.2). *sqoop.apache.org* [online] © 2016 [cit. 2016-12-15] Dostupné z: https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html

(32) APACHE SOFTWARE FOUNDATION. Apache Sqoop – Overview. *blogs.apache.org* [online] © 2011 [cit. 2016-12-15] Dostupné z: https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

(33) APACHE SOFTWARE FOUNDATION. ZooKeeper: A Distributed Coordination Service for Distributed Applications. *zookeeper.apache.org* [online] © 2013 [cit. 2016-12-15] Dostupné z: https://zookeeper.apache.org/doc/trunk/zookeeperOver.html

(34) MACKRORY, S. How-to: Use Apache ZooKeeper to Build Distributed Apps (and Why). *blog.cloudera.com* [online] © 2013 [cit.2016-15-16] Dostupné z: http://blog.cloudera.com/blog/2013/02/how-to-use-apache-zookeeper-to-build-distributed-apps-and-why/

(35) CLOUDERA. Hue - Hadoop User Experience. *gethue.com* [online] © 2016 [cit. 2016-12-16] Dostupné z: http://gethue.com

(36) AWADALLAH, A. a D. GRAHAM. Hadoop and the Data Warehouse: When to Use Which. *assets.teradata.com* [online] © 2012 [cit. 2017-03-26] Dostupné z: http://assets.teradata.com/resourceCenter/downloads/WhitePapers/EB-6448.pdf?processed=1

(37) SWOYER, S. Inside Facebook's Relational Platform. *tdwi.org* [online] © 2013 [cit. 2017-03-26] Dostupné z: https://tdwi.org/Articles/2013/05/06/Facebooks-Relational-Platform.aspx

(38) KUMAR, V. Will Hadoop replace or augment your Enterprise Data Warehouse?. *ibmbigdatahub.com* [online] © 2013 [cit. 2017-03-26] Dostupné z: http://www.ibmbigdatahub.com/blog/will-hadoop-replace-or-augment-your-enterprise-data-warehouse

(39) Konzultace s architektem datového skladu. Praha. 2016

(40) HORTONWORKS. Apache Sqoop – Overview. *hortonworks.com* [online] © 2017 [cit. 2017-02-12] Dostupné z: https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

(41) HOFFMAN, S. Apache Flume – Distributed Log Collection for Hadoop. *javaarm.com* [online] © 2013 [cit. 2017-01-08] Dostupné z: http://javaarm.com/file/apache/flume/book/Apache.Flume-Distributed.Log.Collection.for.Hadoop_1st.Edition_2013.pdf

(42) APACHE SOFTWARE FOUNDATION. Flume 1.7.0 User Guide. *flume.apache.org* [online] © 2012 [cit. 2017-01-26] Dostupné z: https://flume.apache.org/FlumeUserGuide.html

(43) KREPS, J. Putting Apache Kafka to Use for Event Streams. *youtube.com* [online] © 2015 [cit. 2017-02-01] Dostupné z: https://www.youtube.com/watch?v=el-SqcZLZlI

(44) CLOUDERA. Kafka Reference Architecture. *cloudera.com* [online] © 2015 [cit. 2017-02-03] Dostupné z: https://www.cloudera.com/content/dam/www/marketing/resources/datasheets/kafka-reference-architecture.pdf.landing.html

(45) APACHE SOFTWARE FOUNDATION. Kafka 0.10.2 Documentation. *kafka.apache.org* [online] © 2016 [cit. 2017-02-09] Dostupné z: https://kafka.apache.org/documentation/

(46) KEMPF, R. Open Source Data Pipeline – Luigi vs Azkaban vs Oozie vs Airflow. *galado.com* [online] © 2017 [cit. 2017-03-04] Dostupné z: https://www.galado.com/2017/01/workflow-managers/

(47) RATHBONE, M. 6 Reasons You Shouldn't Use Cron to Schedule Your MapReduce and Spark Jobs. *beekeeperdata.com* [online] © 2016 [cit. 2017-03-04] Dostupné z: http://beekeeperdata.com/posts/hadoop/2016/07/19/Cron-Alternatives-For-Hadoop.html

(48) ORACLE. MySQL – The world's most popular open source database. *mysql.com* [online] © 2017 [cit. 2017-01-17] Dostupné z: https://www.mysql.com

(49) RANGARAJAN, S. Data Warehouse Design – Inmon versus Kimball. *tdan.com* [online] © 2016 [cit. 2017-02-11] Dostupné z: http://tdan.com/data-warehouse-design-inmon-versus-kimball/20300

(50) APACHE SOFTWARE FOUNDATION. Hadoop: Setting up a Single Node Cluster. *hadoop.apache.org* [online] © 2016 [cit. 2017-02-11] Dostupné z: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

(51) NOLL, M. G. Running Hadoop on Ubuntu Linux (Multi-Node Cluster). *michael-noll.com* [online] © 2017 [cit. 2017-02-11] Dostupné z: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

(52) APACHE SOFTWARE FOUNDATION. Apache Hadoop. *wiki.apache.org* [online] © 2016 [cit. 2017-03-01] Dostupné z: https://wiki.apache.org/hadoop/

(53) APACHE SOFTWARE FOUNDATION. AdminManual Installation. *cwiki.apache.org* [online] © 2016 [cit. 2017-03-02] Dostupné z: https://cwiki.apache.org/confluence/display/Hive/AdminManual+Installation

(54) TERADATA. Introduction to Teradata. *info.teradata.com* [online] © 2016 [cit. 2017-03-18] Dostupné z: http://www.info.teradata.com/download.cfm?ItemID=1007177

(55) CLOUDERA. Using the Cloudera Connector Powered by Teradata. *cloudera.com* [online] © 2016 [cit. 2017-03-18] Dostupné z: https://www.cloudera.com/documentation/other/connectors/teradata/1x/topics/cctd_use_tpcc.html

(56) TERADATA. Teradata Connector for Hadoop Tutorial. *developer.teradata.com* [online] © 2013 [cit. 2017-03-18] Dostupné z: https://developer.teradata.com/sites/all/files/Teradata%20Connector%20for%20Hadoop%20Tutorial%20v1%200%20final.pdf

(57) APACHE SOFTWARE FOUNDATION. LanguageManual DML. *cwiki.apache.org com* [online] © 2017 [cit. 2017-03-02] Dostupné z: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML

(58) LUANGSAY, S. Using Sqoop to fetch many tables in parallel. *hortonworks.com* [online] © 2016 [cit. 2017-01-22] Dostupné z: https://community.hortonworks.com/articles/23602/sqoop-fetching-lot-of-tables-in-parallel.html

11 Seznam obrázků

12 Seznam tabulek

13 Příloha

13.1 Oozie workflow

```
<workflow-app xmlns='uri:oozie:workflow:0.4' name='DWH_HDFS_Main'>
        <start to='CheckSource'/>

                <action name='CheckSource'>
                        <shell xmlns="uri:oozie:shell-action:0.1">
                                <job-tracker>${jobTracker}</job-tracker>
                                <name-node>${nameNode}</name-node>

                                <exec>${CheckSourceScript}</exec>
                                        <argument>${SourceIPAddress}</argument>
                                        <argument>${SourceMetaUser}</argument>
                                        <argument>${SourceMetaPassword}</argument>
                                        <argument>${workflowId}</argument>
                                <file>${CheckSourceScript}#${CheckSourceScript}</file>
                                <capture-output/>
                        </shell>

                        <ok to="CheckSourceDecision" />
                        <error to="CommonFail" />
                </action>

                        <decision name='CheckSourceDecision'>
                        <switch>
                                <case to="SqoopImport">
                                        ${wf:actionData('CheckSource')['DataAreaAvailable']}
                                </case>
                                <default to='NoDataAvailable'/>
                        </switch>
                </decision>

                <action name='SqoopImport'>
                         <shell xmlns="uri:oozie:shell-action:0.1">
                                 <job-tracker>${jobTracker}</job-tracker>
                                 <name-node>${nameNode}</name-node>

                                 <exec>${SqoopImportScript}</exec>
                                      <argument>${SourceIPAddress}</argument>
                                      <argument>${SourceMetaUser}</argument>
                                      <argument>${SourceMetaPassword}</argument>
                                      <argument>${SourceDataUser}</argument>
                                      <argument>${SourceDataPassword}</argument>
                                      <argument>${workflowId}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaId']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaDateValid']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaCreation']}</argument>
                                 <file>${SqoopImportScript}#${SqoopImportScript}</file>
                         </shell>

                         <ok to="HiveImport" />
                         <error to="LogFailMetadata" />
                 </action>

                 <action name='HiveImport'>
                        <shell xmlns='uri:oozie:shell-action:0.1'>
                                <job-tracker>${jobTracker}</job-tracker>
                                <name-node>${nameNode}</name-node>

                                <exec>${HiveImportScript}</exec>
                                        <argument>${SourceIPAddress}</argument>
                                        <argument>${SourceMetaUser}</argument>
                                        <argument>${SourceMetaPassword}</argument>
                                        <argument>${workflowId}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaId']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaDateValid']}</argument>

        <argument>${wf:actionData('CheckSource')['DataAreaCreation']}</argument>


                                <file>${HiveImportScript}#${HiveImportScript}</file>
                        </shell>
```

```
                        <ok to='LogSuccessMetadata' />
                        <error to='LogFailMetadata' />
                 </action>

        <!-- Metadata Logging -->
        <action name='LogSuccessMetadata'>
                <shell xmlns='uri:oozie:shell-action:0.1'>
                        <job-tracker>${jobTracker}</job-tracker>
                        <name-node>${nameNode}</name-node>

                        <exec>${LogSuccessMetadataScript}</exec>
                                <argument>${SourceIPAddress}</argument>
                                <argument>${SourceMetaUser}</argument>
                                <argument>${SourceMetaPassword}</argument>
                                <argument>${workflowId}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaId']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaDateValid']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaCreation']}</argument>
                        <file>${LogSuccessMetadataScript}#${LogSuccessMetadataScript}</file>
                </shell>

                <ok to='END' />
                <error to='CommonFail' />
        </action>

         <action name='LogFailMetadata'>
                <shell xmlns='uri:oozie:shell-action:0.1'>
                        <job-tracker>${jobTracker}</job-tracker>
                        <name-node>${nameNode}</name-node>

                        <exec>${LogFailMetadataScript}</exec>
                                <argument>${SourceIPAddress}</argument>
                                <argument>${SourceMetaUser}</argument>
                                <argument>${SourceMetaPassword}</argument>
                                <argument>${workflowId}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaId']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaDateValid']}</argument>
<argument>${wf:actionData('CheckSource')['DataAreaCreation']}</argument>
                        <file>${LogFailMetadataScript}#${LogFailMetadataScript}</file>
                </shell>

                <ok to='END' />
                <error to='CommonFail' />
        </action>
         <!-- Metadata Logging -->

        <kill name="CommonFail">
                <message>Workflow failed, metadata saved, error message:
[${wf:errorMessage(wf:lastErrorNode())}]</message>
        </kill>

        <kill name="NoDataAvailable">
                <message>Workflow killed, no data to transfer.</message>
        </kill>

        <end name='END' />
</workflow-app>
```

13.2 CheckSource.sh

```
#!/bin/bash

# Check Source database for Data Area to transfer

# Arguments
#1 IP address of source database
#2 Metadata database user
#3 Metadata database user password
#4 Workflow Id

OIFS="$IFS" ; IFS=$'\n' ; oset="$-" ; set -f

SourceIPAddress=$1
SourceMetaUser=$2
```

```
SourceMetaPassword=$3
WorkflowId=$4

# Query the source database for Data Area to transfer

i=0
while IFS=$OIFS read -a line
do
  DataArea[$i]=${line[0]}";"${line[1]}";"${line[2]}

  i=$i+1
done < <(mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'select
Data_Area_Id, Date_Valid, date_format(Instance_Creation, "%Y-%m-%d_%H:%i:%s") from
transfer_operation.V_Data_Area_Available where Workflow_Id = "'$WorkflowId'";')



# Check if there exists Data Area to Transfer and then process the first one

if [ ${#DataArea[@]} -eq 0 ]; then
 echo "DataAreaAvailable=false"
else
 echo "DataAreaAvailable=true"

 IFS=";" read DataAreaId DataAreaDateValid DataAreaCreation <<< ${DataArea[0]}
 echo "DataAreaId=$DataAreaId"
 echo "DataAreaDateValid=$DataAreaDateValid"
 echo "DataAreaCreation=$DataAreaCreation"



# Set only the one Data Area to transfer

mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Data_Area_Instance set Transfer_Status = "RUNNING", Transfer_Start
= current_timestamp where Data_Area_Id = '$DataAreaId' and Date_Valid = "'$DataAreaDateValid'"
and date_format(Instance_Creation, "%Y-%m-%d_%H:%i:%s") = "'$DataAreaCreation'";'

fi
```

13.3 SqoopImport.sh

```
#!/bin/bash

# Execute Sqoop commands for each table in Data Area

# Script arguments

SourceIPAddress=$1
SourceMetaUser=$2
SourceMetaPassword=$3
SourceDataUser=$4
SourceDataPassword=$5

WorkflowId=$6

DataAreaId=$7
DataAreaDateValid=$8
DataAreaCreation=$9


# Add Sqoop command to PATH

export SQOOP_HOME=/usr/local/sqoop/sqoop
export PATH=$PATH:$SQOOP_HOME/bin

OIFS="$IFS" ; IFS=$'\n' ; oset="$-" ; set -f
#Logging Information

echo "### Starting Sqoop Import ###"
echo "############################"
echo "User: `whoami`"
echo "Node: `hostname`"
echo "Location: `pwd`"

#
```

```
mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Table_Instance set Transfer_Status = "FAILED" where
Transfer_Status in ("RUNNING", "READY") and Data_Area_Id = '$DataAreaId' and Date_Valid =
"'$DataAreaDateValid'";insert into transfer_operation.Transfer_Table_Instance select Table_Id,
Date_Valid, current_Timestamp as Instance_Creation, Table_Database, Table_Name, Data_Area_Id,
"READY" as Transfer_Status, null as Transfer_Start, null as Transfer_End from
transfer_operation.V_Transfer_Table_Instances where Data_Area_Id = '$DataAreaId' and
Date_Valid = "'$DataAreaDateValid'" and date_format(Data_Area_Instance_Creation, "%Y-%m-
%d_%H:%i:%s") = "'$DataAreaCreation'";'


i=0
while IFS=$OIFS read -a line
do
 TransferTable[$i]=${line[0]}";"${line[1]}";"${line[2]}";"${line[3]}";"${line[4]}";"${line[5]}

 i=$i+1
done < <(mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'select
Table_Id, Table_Database, Table_Name, Transfer_Layer_Table_Name, Table_Parallelism,
Table_Split_Column from transfer_operation.V_Transfer_Table_Definitions where Data_Area_Id =
'$DataAreaId' and Date_Valid = "'$DataAreaDateValid'";')


i=0
for i in "${TransferTable[@]}"
do
  IFS=";" read TableId TableDatabase TableName TransferLayerTableName TableParallelism
TableSplitColumn <<< $i



  # Log start of Sqoop load
  mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Table_Instance set Transfer_Status = "RUNNING", Transfer_Start =
current_timestamp where Table_Id = '$TableId' and Date_Valid = "'$DataAreaDateValid'" and
Transfer_Status = "READY";'

  sqoop import --bindir ./ --connect jdbc:mysql://$SourceIPAddress/transfer_layer --username
$SourceDataUser --password $SourceDataPassword --table $TransferLayerTableName --target-dir
/stage/$DataAreaDateValid/$TableDatabase/$TableName -m $TableParallelism --split-by
$TableSplitColumn --delete-target-dir || { exit 1; }

  # Log end of Sqoop load
mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Table_Instance set Transfer_Status = "SUCCEEDED", Transfer_End =
current_timestamp where Table_Id = '$TableId' and Date_Valid = "'$DataAreaDateValid'" and
Transfer_Status = "RUNNING";'

done
```

13.4 HiveImport.sh

```
#!/bin/bash

# Load data into Hive for selected tables transfered in previous step

# Script arguments

SourceIPAddress=$1
SourceMetaUser=$2
SourceMetaPassword=$3

WorkflowId=$4

DataAreaId=$5
DataAreaDateValid=$6
DataAreaCreation=$7


# Add Hive command to PATH

export HIVE_HOME=/usr/local/hive/hive
export PATH=$PATH:$HIVE_HOME/bin
```

```
OIFS="$IFS" ; IFS=$'\n' ; oset="$-" ; set -f

#Logging Information

echo "### Starting Hive Import ###"
echo "##############################"
echo "User: `whoami`"
echo "Node: `hostname`"
echo "Location: `pwd`"

mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Hive_Instance set Transfer_Status = "FAILED" where Transfer_Status
in ("RUNNING", "READY") and Data_Area_Id = '$DataAreaId' and Date_Valid =
"'$DataAreaDateValid'";insert into transfer_operation.Transfer_Hive_Instance select Table_Id,
Date_Valid, current_Timestamp as Instance_Creation, Table_Database, Table_Name, Data_Area_Id,
"READY" as Transfer_Status, null as Transfer_Start, null as Transfer_End from
transfer_operation.V_Transfer_Hive_Instances where Data_Area_Id = '$DataAreaId' and Date_Valid
= "'$DataAreaDateValid'" and date_format(Data_Area_Instance_Creation, "%Y-%m-%d_%H:%i:%s") =
"'$DataAreaCreation'";'


i=0

while IFS=$OIFS read -a line
do
 HiveTable[$i]=${line[0]}";"${line[1]}";"${line[2]}";"${line[3]}";"${line[4]}

 i=$i+1
done < <(mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'select
Table_Id, Table_Database, Table_Name, Hive_Partitioning, Date_Valid from
transfer_operation.V_Transfer_Hive_Definitions where Data_Area_Id = '$DataAreaId' and
Date_Valid = "'$DataAreaDateValid'";')


i=0
for i in "${HiveTable[@]}"
do
  IFS=";" read TableId TableDatabase TableName HivePartitioning DateValid <<< $i

  # Log start of Hive load
  echo 'Loading Hive Table: '$TableDatabase'.'$TableName

  mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Hive_Instance set Transfer_Status = "RUNNING", Transfer_Start =
current_timestamp where Table_Id = '$TableId' and Date_Valid = "'$DateValid'" and
Transfer_Status = "READY";'

  if [ "$HivePartitioning" == "N" ]; then
    hive -e 'load data inpath "/stage/'$DateValid'/'$TableDatabase'/'$TableName'/part-*"
overwrite into table DWH_'$TableDatabase'.'$TableName';' || { exit 1; }
  else
    hive -e 'load data inpath "/stage/'$DateValid'/'$TableDatabase'/'$TableName'/part-*"
overwrite into table DWH_'$TableDatabase'.'$TableName' partition (Load_Date =
"'$DateValid'");' || { exit 1;}
  fi


  # Log end of End load
  echo 'Loadind Hive Table '$TableDatabase'.'$TableName' finnised.'

  mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Hive_Instance set Transfer_Status = "SUCCEEDED", Transfer_End =
current_timestamp where Table_Id = '$TableId' and Date_Valid = "'$DataAreaDateValid'" and
Transfer_Status = "RUNNING";'

done
```

13.5 LogSuccessMetadata.sh

```
#!/bin/bash

# Log Successful run of current processed Data Area

# Arguments
```

```
#1 IP address of source database
#2 Metadata database user
#3 Metadata database user Password
#4 Workflow Id

SourceIPAddress=$1
SourceMetaUser=$2
SourceMetaPassword=$3
WorkflowId=$4

DataAreaId=$5
DataAreaDateValid=$6
DataAreaCreation=$7

# Log Success Metadata

mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Data_Area_Instance set Transfer_Status = "SUCCEEDED", Transfer_End
= current_Timestamp where Data_Area_Id = '$DataAreaId' and Date_Valid = "'$DataAreaDateValid'"
and date_format(Instance_Creation, "%Y-%m-%d_%H:%i:%s") = "'$DataAreaCreation'";'
```

13.6 LogFailMetadata

```
#!/bin/bash

# Log Successful run of current processed Data Area

# Arguments
#1 IP address of source database
#2 Metadata database user
#3 Metadata database user Password
#4 Workflow Id

SourceIPAddress=$1
SourceMetaUser=$2
SourceMetaPassword=$3
WorkflowId=$4

DataAreaId=$5
DataAreaDateValid=$6
DataAreaCreation=$7

# Log Success Metadata

mysql -u $SourceMetaUser -p$SourceMetaPassword -h $SourceIPAddress -s -N -e 'update
transfer_operation.Transfer_Data_Area_Instance set Transfer_Status = "FAILED", Transfer_End =
current_Timestamp where Data_Area_Id = '$DataAreaId' and Date_Valid = "'$DataAreaDateValid'"
and date_format(Instance_Creation, "%Y-%m-%d_%H:%i:%s") = "'$DataAreaCreation'";'
```