Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Optimalizace vyhodnocování XPath dotazů v systému Xomoda

Jan Mašek

Vedoucí práce: Ing. Adam Šenk

12. května 2015

Poděkování

Děkuji mému vedoucímu Ing. Adamu Šenkovi za vedení bakalářské práce a za projevenou důvěru, ochotu a zejména trpělivost při pronikání do tajů Big Data a systému Xomoda.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Mašek, Jan. *Optimalizace vyhodnocování XPath dotazů v systému Xomoda.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se zabývá možností využití Big Data technologií pro vyhodnocování Xpath dotazů. V první části jsou shrnuty hlavní poznatky z oblasti uchování a zpracování dat. Druhá část se zaměřuje na systém Xomoda, který pro vyhodnocování Xpath dotazů využívá technologie open-source frameworku Hadoop. Součástí práce je návrh a implementace vyhodnocování XPath os preceding a following v systému Xomoda.

Klíčová slova XPath, XML, Hadoop, HBase, MapReduce, Xomoda, Big Data

Abstract

This thesis is focused on application Big Data technologies for XPath evaluation. The main findings in field of data storage and processing are introduced in the first part of document. The second part of the thesis is dedicated to Xomoda system. This system implements XPath query evaluation using technologie from open-source framework Hadoop. The thesis includes design and implementation of XPath axes preceding and following in Xomoda system.

Keywords XPath, XML, Hadoop, HBase, MapReduce, Xomoda, Big Data

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Každý den, každou minutu i každou sekundu je v digitálním světě generováno obrovské množství dat. Firmy, které chtějí tyto data využít, případně je poskytovat dále (vyhledávače, sociální sítě), je musí zvládat efektivně ukládat a rychle analyzovat. To je se vzrůstajícím objemem problém. Mnoho subjektů je nuceno pracovat s objemy dat, které byly ještě před několika lety nepředstavitelné. Již není možné uložit všechny data na jeden disk, či dokonce na jeden počítač, zjišťujeme, že relační databáze nedokáží uspokojit naše požadavky. Proto jsme nuceni využívat přístupy, které byly vyvinuty pro velké objemy dat, tzv. Big Data technologie.

Tato práce má za cíl seznámení se s technologiemi použitými v systému Xomoda a vylepšení tohoto systému. Tento systém ukazuje, jak je možné využít Big Data technologií pro vyhodnocování XPath dotazů nad velkými XML soubory.

V první části je práce zaměřena na aktuální technologie a poznatky z oblasti strukturalizace, serializace a také persistence dat. Poté následuje bližší pohled na technologie použité v systému Xomoda, zejména pak na nástroje z open-source frameworku Hadoop.

Druhá část práce se věnuje architektuře systému Xomoda, včetně vyhodnocování a ukládání dat.

Ve třetí části se pak nachází samotná realizace vyhodnocování dvou zbývajících os, které v systému Xomoda dosud nebyly implementovány.

Kapitola 1

Cíl práce

Cílem teoritcké části je analýza současných technologií. Zejména pak pochopení platformy Hadoop, databáze HBase, výpočetního modelu MapReduce a jejich provázanost, seznámení s jazykem XML a dotazovacím jazykem XPath.

Praktická část se zaměří a funkční prototyp Xomoda. Na jeho práci s nástroji z frameworku Hadoop, zejména HBase a MapReduce, na architekturu vyhodnocování a způsob ukládání dat. Dalším cílem je implementace os preceding a following a optimalizace způsobu ukládání pro jejich efektivní vyhodnocení.

Kapitola 2

Teoretické ukotvení

Tato kapitola se se zaměřuje na popis známých problémů a řešení. Ukazuje možné přístupy a pohledy na problematiku persistence a práce s daty.

Při jakékoliv práci s daty se programátor dostává do problému, že se celý dataset nevejde do operační paměti. Během zpracování musí data někam odkládat a pak je zase číst. Obecně se dá říct, že má dvě možnosti. Ukládat mezivýsledky na disk do souboru, nebo k persistenci využít databázi.

2.1 Jazyk XML

Data na disk můžeme ukládat jako binární soubory, nebo posloupnost znaků. To je ale pro přenos, a zejména čitelnost, značně nepraktické, protože musíme znát přesnou strukturu. Proto vznikly univerzálnější nástroje, jazyky, které jsou určeny ke strukturalizaci a serializaci dat. V současné době jsou nejpopulárnější jazyky XML, JSON, či YAML. Hlavní využití těchto jazyků je především ve výměně dat mezi programy. Například mezi zdrojovým systémem a systémem, který data analyzuje.

Jazyk XML byl poprvé standartizován v roce 1998 organizací W3C [5]. Od té doby proběhlo několik revizí a v současné době se používá pátá edice původní verze XML 1.0.

XML k definování struktury dokumentu využívá XML značky, jinak také tagy. Díky značkám lze data strukturovat a lze tak přiřadit jednotlivým elementům jejich význam. XML nám tak dovoluje standartizovat reprezentaci dat a data přenášet mezi aplikacemi [6]. Mezi hlavní výhody XML patří variabilita, přehlednost a čitelnost.

Každá XML značka může mít kormě obsahu definovány další vlastnosti pomocí atributů (například atribut `id`). Příklad jednoduchého XML dokumentu ukazujue Obrázek 2.1.

Jednou z dalších vlastností XML je, že dokument lze reprezentovat jako strom. V 2.1 je kořenovým uzlem `Employees`. Ten má čtyři potomky `Employee`,

```
<?xml version="1.0" encoding="UTF-8"?>
<Employees>
  <Employee id="1">
    <age>55</age>
    <name>Pankaj</name>
    <gender>Male</gender>
    <role>Java Developer</role>
  </Employee>
  <Employee id="2">
    <age>35</age>
    <name>Lisa</name>
    <gender>Female</gender>
    <role>CEO</role>
  </Employee>
  <Employee id="3">
    <age>40</age>
    <name>Tom</name>
    <gender>Male</gender>
    <role>Manager</role>
  </Employee>
  <Employee id="4">
    <age>25</age>
  </Employee>
</Employees>
```

Obrázek 2.1: Příklad XML kódu.

každý s jiným atributem `id`. Každý uzel pak může mít atributy, obsahovat další potomky nebo text.

2.1.1 Jazyk XPath

Jazyk XPath je dotazovací jazyk pro XML dokumenty. Kromě použití čistě pro XML se s ním můžeme setkat například v oblasti testování softwaru, kde se používá pro adresaci HTML prvků.

Samotný dotaz je vlastně cestou a říká nám, jak se k požadovanému elementu dostaneme. Práci jazyka XPath si lze nejlépe představit na stromové reprezentaci XML dokumentu. Každá cesta vždy začíná před kořenovým uzlem a je posloupností jednotlivých kroků, které nás dovedou k požadovanému uzlu, nebo uzlům. Každý krok se skládá ze tří složek, které je možné libovolně kombinovat, případně vynechat[6].

- *Osa* určuje směr, kterým se budeme pohybovat v XML stromu.

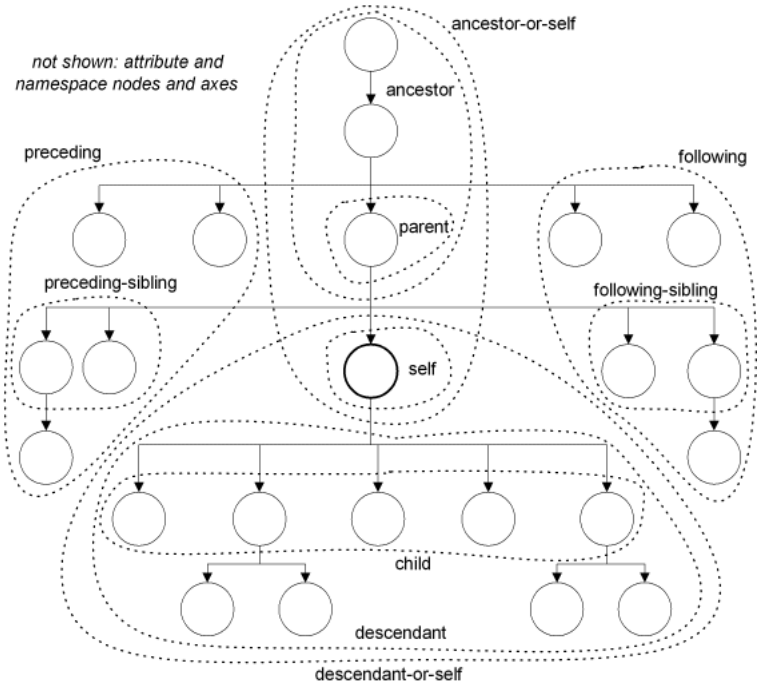
- *Test uzlu* určuje jméno nebo typ, které mohou být zpracovány v rámci kroku.
- *Predikát* přidává další kritéria pro výběr uzlu v daném kroku.

XPath osy Jazyk XPath definuje následující osy. Jejich význam lze nejlépe demonstrovat na XML stromu 2.2.

- *self* vyhodnocovaný uzel
- *child* přímí potomci uzlu
- *descendant* všichni potomci uzlu
- *descendant-or-self* uzel a všichni jeho potomci
- *parent* rodič uzlu
- *ancestor* předek uzlu
- *ancestor-or-self* uzel a všichni jeho předci
- *following-sibling* všichni následující sourozenci
- *preceding-sibling* všichni předcházející sourozenci
- *following* všechny uzly, které se v XML dokumentu nacházejí za vyhodnocovaným uzlem
- *preceding* všechny uzly, které se v XML dokumentu nacházejí před vyhodnocovaným uzlem
- *attribute* všechny atributy náležící vyhodnocovanému uzlu
- *namespace* deklarované jmenné prostory

Zkratky Pro zjednodušení a zkrácení dotazů se při zápisu některých os a příkazů používají zkratky. Nejpoužívanější jsou následující.

- *defaultní osa* Pokud není v daném kroku specifikována osa, je krok vyhodnocen v ose `child`.
- *@* Znak at je zkratka pro attribute.
- *//* Dvě lomítka jsou vyhodnoceny jako osa descendant-or-self (ekvivalent k `descendant-or-self::node()`)
- *.* Tečka je vyhodnocena jako osa self (ekvivalent k `self::node()`)
- *..* Dvě tečky jsou vyhodnoceny jako osa parent – podobně jako v operačních systémech tedy odkazuje na nadřazený uzel. (ekvivalent k `parent::node()`)
- *** Znak hvězdička je zástupný znak.



Obrázek 2.2: Znázornění XPath os na XML stromu [1].

```
/Employees//name[text()='Lisa']/parent::node()
```

Obrázek 2.3: Příklad XPath dotazu.

2.1.1.1 Příklad Xpath dotazu

Příkladem může být dotaz, který z ukázkového XML dokumentu (2.1) vybere zaměstnance, jehož jméno je `Lisa` (zobrazuje Obrázek 2.3).

Xpath v prvník kroku vybere kořenový uzel `Employees`. V dalším kroku z jeho potomků (přímých i nepřímých) vybere uzly, které se jmenují `name`. Z nich vybere ty, které obsahují text `Lisa`. U těchto uzlů zjistí jejich rodiče a ty vypíše. V našem případě bude výsledek pouze jeden a je ukázán na Obrázku 2.4. Bude jediný, protože je to jediný uzel, který odpovídá dotazu.

2.2 Databáze

Další z možností, jak řešit persistenci dat je databáze. Ty jsou pro práci s daty optimalizované a práci s daty zvládají velice efektivně. Pro práci s nimi byly vyvinuty dotazovací jazyky, které programátory dovolují specifikovat a filtrovat, jaká data program, případně programátor, požaduje.

```
<Employee id="2">
  <age>35</age>
  <name>Lisa</name>
  <gender>Female</gender>
  <role>CEO</role>
</Employee>
```

Obrázek 2.4: Výsledek XPath dotazu.

2.2.1 Relační databáze

Mezi nejpoužívanější databázové modely se v současné době řadí relační databáze. Data jsou pro účely uložení rozděleny do relací (tabulek). Každá tabulka obsahuje řádky, ve kterých jsou uloženy související data. Vztah mezi dvěma řádky různých tabulek je reprezentován pomocí odkazů, tzv. cizích klíčů.

Mezi hlavní výhody tohoto systému je logické uspořádání a důraz na integritu uložených dat. Hlavní požadavky na relační databáze a transakce s nimi prováděné lze definovat akronymem ACID[7].

- *Atomicity* Atomicita transakcí vyjadřuje požadavek, aby se během transakce provedly buď všechny její součásti (úpravy), nebo žádná.
- *Consistency* Jakákoliv transakce nesmí zanechat databázi v nekonzistentním stavu.
- *Isolation* Transakce nesmí zasahovat do jiné.
- *Durability* Databáze musí být odolná, tzn. že dokončená transakce musí přetrvat i v případě restartu.

2.2.1.1 SQL

Jazyk SQL je dotazovací jazyk, který byl vyvinut pro práci s relační databází. Umožňuje komplexní ovládání relační databáze, takže obsahuje příkazy pro následující typy operací[8].

- Definice dat (DDL)
- Manipulace s daty (DML)
- Řízení transakcí
- Řízení relace
- Nastavení systému
- Vnořené SQL příkazy

2.2.2 NoSQL databáze

Při použití ve velkých systémech relační databáze velmi zpomaluje požadavek konzistenci. K jeho zajištění je třeba během transakce části tabulek zamykat, takže k nim má přístup pouze jedna transakce. V případě, že přijde další transakce, která potřebuje pracovat s uzamčenou částí databáze, je nucena čekat, až bude předchozí transakce dokončena. To samozřejmě zvětšuje odezvu systému a může být velkým problémem.

Z tohoto důvodu byly formulovány požadavky, které více reflektují nároky velkých systémů. Jsou reprezentovány akronymem BASE[7].

- *Basic availability:* Je garantována odpověď pro každou transakci – úspěšné nebo neúspěšné provedení.
- *Soft state:* Stav databáze se může změnit i v době, kdy není vykonána žádná transakce (pro zajištění koncové konzistence).
- *Eventual consistency:* Databáze může být nekonzistetní, ale nakonec bude konzistentní.

Dalším důvodem pro přechod k NoSQL databázím může být neefektivní ukládání řídkých dat. V relační databázi se velice snadno může stát, že máme tabulky plné `null` hodnot. V NoSQL ukládáme pouze informace, které známe.

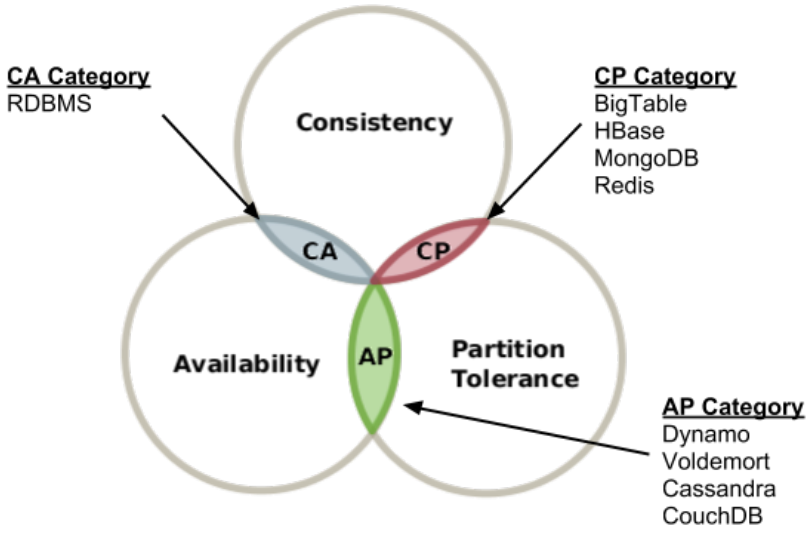
Způsobů jak realizovat NoSQL databáze je samozřejmě několik, takže se v dnešní době setkáváme s následujícími typy[9].

- *Column oriented:* Data jsou organizována do jednotek a tyto jednotky obsahují dvojice klíč/hodnota. Výhoda tohoto přístupu je, že každá jednotka může mít svou vlastní strukturu.
- *Key/value:* Tyto databáze pracují s dvojicemi klíč/hodnota. Lze si ji představit, jako velkou, distribuovanu hash tabulku.
- *Document:* Document je v tomto případě organizační jednotka, která obsahuje strukturovaná data, nejčastěji ve formátu JSON nebo XML.
- *Graph:* Hlavní jednotkou je uzel, který obsahuje informace o sobě samém a pak přímý odkaz na své sousedy–uzly se kterými je spojen nějakým vztahem/hranou.

2.2.3 CAP Theorem

S tématem NoSQL databází velice úzce souvisí CAP teorém, který představil v roce 2000 Eric Brewer. V této práci formuluje tři základní požadavky: [10]

- *Consistency:* Úložiště je v konzistentním stavu, všechny oblasti systému osahují aktuální data a změny jsou atomické.



Obrázek 2.5: CAP teorém a příklady technologií pro jednotlivé kombinace požadavků. Převzato z [2].

- *Availability:* Je garantována odpověď pro každý požadavek, ta může být úspěch, nebo neúspěch (chyba).

- *Partition-tolerance:* Systém zvládá pracovat a komunikovat, i přestože její součást selhala.

Teorém říká, že nelze vytvořit systém, který by splňoval všechny tři požadavky souběžně. Z toho vyplývá, že při výběru vhodného úložiště je vždy nutné zjistit, jaké vlastnosti jsou pro nás klíčové a podle toho vybrat vhodnou technologii. Obrázek 2.5 ukazuje, které technologie v současné době nabízí danou kombinaci požadavků.

2.3 Velká data

Pokud začneme uchovávat a zpracovávat opravdu velké množství dat, dostáváme se do problému, že data nelze efektivně uchovávat a ani zpracovávat na jednom fyzickém stroji. Proto je nutné přistoupit k rozdělení, uložení a zpracování dat na více strojích.

Mohlo by se zdá, že následující problémy jsou pouze teoretické, případně je nutné je řešit pouze v okrajových případech. Následující údaje jsou důkazem, že tyto problémy jsou skutečné a bude je nutné řešit stále častěji.

- V roce 2009 Google zpracovával více než 24 petabytes za den[11].

- V roce 2012, [12] uvádí, že Facebook používá úložiště s fyzickou diskovou kapacitou více než 100 petabytes v jediném HDFS (2.4.1.1) filesystému.

- Vytváření filmu Avatar vyžadovalo diskové úložiště více než 1 petabyte[13].

Při uložení dat na více strojů, mluvíme o distribuovém úložišti. Příkladem takového úložiště může být HDFS, který je součástí frameworku Hadoop. Tomuto úložišti je věnována část 2.4.1.1.

S tím i částečně souvisí obecně systém uložení dat. Relační databáze nastavují dost striktní pravidla, jejichž dodržování stojí mnoho času, proto se některé větší aplikace uchylují k NoSQL databázím.

Dalším problémem při práci s velkými objemy dat je způsob zpracování. Nelze již uvažovat o klasickém sekvenčním přístup, kdy vstup zpracováváme v jednom vlákně a data postupně načítáme do paměti a zpracováváme. Je nutné zpracování rozdělit na co nejmenší kroky, které lze vykonávat samostatně, nezávisle na ostatních. Abychom co nejvíce zkrátili dobu výpočtu, je potřeba minimalizovat přenosy dat po pomalých sítích a výpočty přesunout co nejblíže k datům. Tzn. provádět maximum výpočtů na strojích, na kterých jsou uložena data.

2.3.1 Bigtable

Bigtable je koncept, který byl vytvořen společností Google a popsán v roce 2006 v [14]. Hlavním cílem tohoto konceptu bylo navrhnout úložiště, které dovoluje spolehlivě uchovávat petabyty dat na tisících strojích. Google uvádí, že implementaci Bigtable využívá více než 60 produktů této společnosti, například Google Analytics, Google Finance, Orknut a Google Earth.

Data v Bigtable databázi jsou organizována do jednotek, které se skládají ze tří složek: řádky, sloupce a časové značky.

$$(row : string, column : string, time : int64) \rightarrow string$$

Tyto jednotky jsou pak spojovány do celků, které se jmenují tablets. Ty jsou pak distribuovány a ukládány na filesystém[14].

2.3.2 Výpočetní model MapReduce

Problém paralelizace je často netriviální a velice náročný problém, a proto byl vytvořen výpočetní model MapReduce, který díky svému návrhu umožňuje relativně jednoduchou distribuci výpočtu.

MapReduce vychází z principů funkcionálního programování. Do výpočtu vstupuje množina dvojic klíč/hodnota a výstupem je také množina dvojic klíč/hodnota. Celý výpočet, který se nazývá MapReduce job, se skládá ze dvou základních funkcí, `map` a `reduce`, které definuje programátor.

$$\textbf{map}\ (k1, v1) \rightarrow list(k2, v2)$$
$$\textbf{reduce}(k2, list(v2)) \rightarrow list(v3)$$

Do funkce `map` vstupuje vždy jedna dvojice klíč/hodnota ze vstupní množiny a její vyhodnocování probíhá nezávisle na zbývajících hodnotách na vstupu. Tato funkce produkuje dvojice klíč/hodnota, které mohou být zpracovány ve funkci `reduce`. `Reduce` funkce během jednoho volání zpracovává vždy jeden klíč a množinu hodnot, které byly vyprodukovány během funkce `map` a byl jim přiřazen tento klíč. Výstupem funkce map je množina hodnot[11].

Jednoduchost distribuce výpočtu vychází z návrhu. Každé volání funkce `map` je nezávislé na ostatních vstupech, takže všechny volání lze vykonávat paralelně. Podobně i každé volání funkce `reduce` je nezávislé na ostatních voláních, tudíž je možné i tuto funkci volat paralelně s ostaními.

Příklad použití MapReduce

Typickým příkladem pro použití modelu MapReduce je WordCount. Úloha je následující. Máme dokument a rádi bychom zjistili četnosti jednotlivých slov, které se v dokumentu nacházejí.

V prvním kroku se dokument rozdělí na řádky a každý řádek vstoupí do vlastní `map` funkce. Tato funkce spočítá jednotlivé počty slov. Poté, pro každé unikátní slovo bude emitována dvojice klíč/hodnota, tedy `(slovo,počet)`. Tyto dvojice jsou v dalším kroku podle klíče shromážděny a vstupují do funkce `reduce`. Pokud se tedy slovo `pes` vyskytovalo v dokumentu na prvním řádku jednou, na třetím dvakrát a na šestém řádku čtyřikrát, do funkce reduce vstupuje dvojice klíč/hodnoty, tedy `(pes,<1,2,4>)`. Funkce `reduce` tyto částečné součty sečte a výsledkem bude, že v dokumentu se slovo `pes` nacházelo celkem sedmkrát.

Takto každý reducer spočítá četnost jednotlivých slov v dokumetu a tím získáme četnosti všech slov, které se nacházejí v dokumentu.

2.4 Použité technologie

Pro zajištění kompatibility a stability všech součástí nutných pro chod a konfiguraci distribuovaného clusteru jsem využil produkt CDH společnosti Cloudera, který hlídá závislosti jednotlivých verzí nástrojů balíku Hadoop a zajišťuje tak jejich bezproblémový chod.

2.4.1 Hadoop

Apache Hadoop je framework, který obsahuje nástroje pro práci s velkými daty. Je vyvíjen v jazyce Java pod licencí Apache License, version 2.0, která je kompatibilní s GNU GPL v3 [15], takže se jedná o svobodný open-source software. Hlavním vývojářem je Apache Software Foundation.

Tento balík open-source nástrojů pokrývá vše potřebné pro práci a obsluhu distribuovaného systému. Pro názornost uvedu několik nejdůležitějších komponent.

- Distribuované úložiště (HDFS)
- Implementace výpočetního modelu MapReduce (Hadoop MapReduce)
- Nástroj pro plánování úloh a zdrojů (Hadoop YARN)
- Distribuovaná databáze (HBase)
- Službu pro plánování a koordinaci distribuovaného výpočtu (ZooKeeper)

2.4.1.1 HDFS

HDFS je distribuovaný filesystém. Podobně jako mnoho distribuovaných systému, je navrhnut tak, aby mohl být provozován na bežně dostupném hardwaru. Vývojáři Hadoop jdou ještě dále a uvádějí, že HDFS je připraven, aby mohl běžet na nízkonákladovém hardwaru. Podporou tohoto tvrzení je skutečnost, že HDFS je velmi odolný proti chybám, které mohou při použití levného hardwaru nastávat velice často. Detekce chyb a rychlá obnova dat jsou pro Hadoop prioritou.

Data jsou vždy uložena na několika fyzických zařízeních, velké soubory jsou ukládány jako sekvence bloků a ty jsou distribuovány a replikovány. Samotnou replikaci a distribuci řídí hlavní uzel, který se v terminologii HDFS nazývá NameNode. Uzly na kterých jsou uložena data se nazývají DataNode. Ty opakovaně posílají řídícímu uzlu informace o svém stavu (Heartbeat) a o blocích (Blockreport), které jsou na nich uloženy.

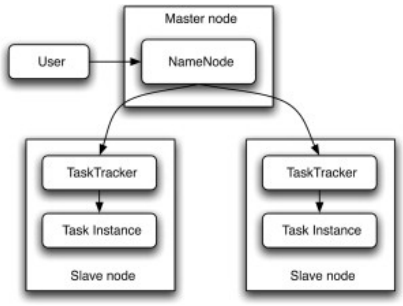
Systém je koncipován jako write-once-read-many, takže pokud je soubor jednou uložen, neměl by už být měněn. Není tedy koncipován jako klasický uživatelský filesystem, ale je optimalizován například pro MapReduce aplikace nebo pro aplikace stahující webové stránky (tzv. web crawler). [16]

2.4.1.2 HBase

HBase byla vytvořena v roce 2007 a je open-source implementací Bigtable. Je součástí projektoru Apache Hadoop. Primárně pracuje s HDFS, ale existuje podpora i pro jiné filesystémy. Rozdíly mezi HBase a BigTable jsou především v názvech (například tablet z BigTable je v HBase region), ale jsou zde i drobné rozdíly ve funkcionalitě. Ty jsou většinou způsobeny buď nejasnou definicí v [14], případně tím, že HBase je živým systémem a stále se vyvíjí[17].

2.4.1.3 Hadoop MapReduce

Hadoop MapReduce je framework, který je součástí Hadoop projektu a je implemntací výpočetního modelu MapReduce pro HBase.



Obrázek 2.6: Distribuce výpočtu v Hadoop MapReduce. Převzato z [3].

Celý MapReduce program se nazývá `Job`, jeho vykonání se skládá z jednotlivých úloh na každé části dat. Tyto úlohy se nazývají `Tasks`.

Celý výpočet řídí hlavní uzel (Master node), na kterém běží instance třídy `Jobtracker`, která příjímá požadavky (Job), které rozdělí na jednotlivé úlohy (Tasks) a ty předá výpočetním uzlům (Slave nodes). Na těchto uzlech běží instance `TaskTracker`, která má za úkol vykonat nad určenými daty požadovaný úkol. `TaskTracker` vykonává jednotlivé úlohy v oddělených java procesech, takže v jednom okamžiku může být vykonáváno několik úloh paralelně[3].

Kapitola 3

Analýza: Systém Xomoda

Xomoda je funkční prototyp, jež je popsán v [4]. Cílem tohoto projektu bylo ukázat, jak lze využít Big Data technologie pro ukládání dat ve formátu XML a následné dotazování v jazyce XPath. Tato kapitola shrnuje poznatky z uvedené práce.

3.1 Podobné práce

Mapování dokumentů do NoSQL databází bylo popsáno v několika dokumentech. V [18] byly porovnávány tři možnosti mapování, nicméně vyhodnocování dotazů nebylo zkoumáno vůbec. Dokument [19] představuje techniku distribuovaného dotazování nad velkou množinou XML dokumentů, která je použita v technologii Amazon Cloud. Možnosti dotazování nad jedním velkým XML dokumentem v této práci zvažovány nejsou.

Distribuované vyhodnocování XPath dotazů je zkoumáno v [20]. Práce bohužel pokrývá pouze vyhodnocování některých os. Další nevýhodou je, že pro každý dotaz je XML dokument znovu parsován a indexován[4].

Existuje několik dalších prací, které se zabývají tématem vyhodnocování dotazů nad XML dokumenty, bohužel většina buď neimplementuje vyhodnocování všech, ale pouze některých XPath os, nebo vytváří nový dotazovací jazyk.

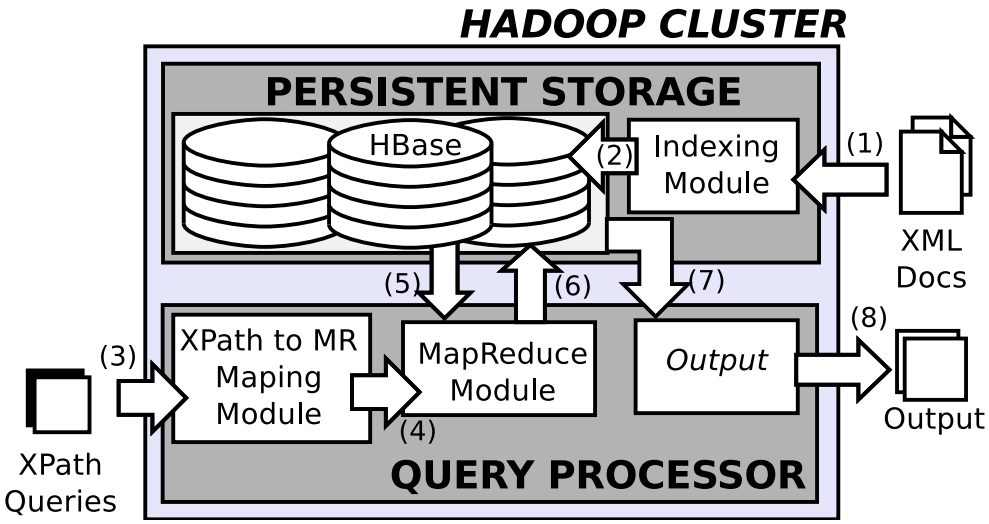
3.2 Architektura

K ukládání XML dokumentů a výsledků je použita databáze HBase.

O vyhodnocování XPath dotazů se stará Query processor (Obrázek 3.2). XPath dotazy jsou nejdříve převedeny na výpočet pomocí modelu MapReduce, vznikne `MapReduce Job`, který je předán modulu, který zajišťuje MapReduce výpočet (`MapReduce Module`). Ten tyto úkoly vykoná nad daty uloženými v HBase databázi. Toto předání ukazuje Obrázek 3.1. Důležité jsou

```
TableMapReduceUtil.initTableMapperJob(sourceTable, scan,
       Mapper.class, ImmutableBytesWritable.class,
       ImmutableBytesWritable.class, job);
TableMapReduceUtil.initTableReducerJob(targetTableName,
       Reducer.class, job);
```

Obrázek 3.1: Předání výpočtu MapReduce modulu.



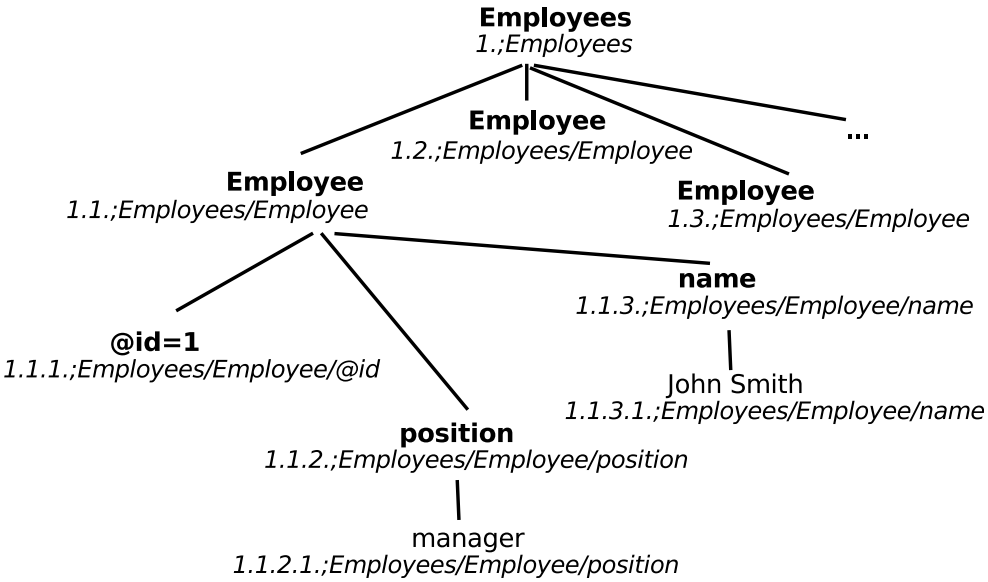
Obrázek 3.2: Architektura výpočtu systému Xomoda. Převzato z [4].

parametry `Mapper.class` a `Reducer.class`. Jsou to statické třídy, které napsal programátor a které obsahují hlavní logiku výpočtu a zajišťují výpočet XPath dotazu pomocí modelu MapReduce.

`MapReduce Module` zajistí distribuovaný výpočet nad daty, který jsou uloženy na počítačích v databázi HBase. Výsledek je uložen do nové tabulky v databázi. `Query processor` si tuto tabulku vyžádá a následně zrekonstruuje výslednou XML strukturu. Architekturu zachycuje.3.2.

3.3 Ukládání XML do úložiště

Každý XML dokument, se kterým chceme pracovat musí být zpracován v `Indexing module`. Toto zpracování dokument rozdělí na jednotlivé uzly, které jsou postupně uloženy do HBase databáze spolu s odpovídajícím indexem. Xomoda používá pro indexování dvě strategie. Hlavní cíle indexace jsou dva, snadná zpětná rekonstrukce XML stromu a efektivní vyhodnocení XPath os. Obě tyto metody zobrazuje obrázek 3.3.



Obrázek 3.3: Část XML stromu indexována pomocí strategií Dewey a Path index. Klíče jsou odděleny středníkem. Převzato z [4].

3.3.1 Dewey index

Tato strategie byla představena v [21]. Každému uzlu přidělí unikátní vektor (neboli také uspořádanou n-tici), který jednoznačně definuje jeho pozici v XML stromu. Kořenovému uzlu je přidělen jednorozměrný vektor s hodnotu `1`. Jeho přímým potomků bude přidělen dvourozměrný vektor, jehož první složka stejná, jako jejich rodič, takže `1`. Druhá složka bude unikátní a bude přidělena podle horizontálního pořadí. V další úrovni stromu proběhne přidělení stejně. První dvě složky budou stejné jako vektor rodiče a třetí bude postupně přidělena přímým potomkům daného rodiče. Další přidělení pracuje tak, že potomek je identifikován vektorem, který má dimenzi o jedna větší než jeho rodič a odpovídající složky vektoru má totožné se svým rodičem. Díky tomuto systému lze snadno určit hloubku, v jaké se uzel nachází a vztah mezi uzly (předchůdce, rodič, potomek, nebo sourozenec).

3.3.2 Path index

Tato adresace je klíčová pro vyhodnocování XPath os. K adresaci uzlů využívá, podobně jako dotazy formulované v jazyce XPath, cestu. Tato cesta vyjadřuje posloupnost jednotlivých uzlů, které vedou od kořene k danému uzlu. Indexy jednotlivých uzlů nejsou unikátní a jsou stejné pro sourozence (potomky jednoho rodiče). Příklad takové indexace ukazuje obrázek 3.3.

KEY	*nodeinfo*				
	PathIndex	**Name**	**Node Type**	**Text Val**	**LS**
1.	Employees	Employees	ELEM		1
1.1.	Employees/Employee	Employee	ELEM		60
1.1.1.	Employees/Employee/id	id	ATTR	1	3
1.1.2.	Employees/Employee/position	position	ELEM		3
1.1.2.1.	Employees/Employee/position		TEXT	CIO	3

Tabulka 3.1: Část XML dokumentu (Obrázek 3.3) uložená do HBase databáze. Převzato z [4].

3.3.3 Využití Wide Column Store

XML dokument je uložen v jedné tabulce a rozdělen na jednotlivé uzly. Podle rozdělení v 2.2.2 je způsob ukládání nejblíže typu Column-oriented. Odlišností databáze HBase je slučování vlastností do balíčků, tzv. column-families.

Klíčem každého záznamu v tabulce je Dewey index. Column family je využita pouze jedna a to `nodeinfo`. Součástí tohoto balíčku jsou následující vlastnosti.

- *PathIndex:* Tak jak je popsán v 3.3.2.
- *NodeType:* Údaj, který říká o jaký typ záznamu se jedná – může to být `element`, `attribute`, nebo `text`.
- *Name:* Jméno uzlu.
- *TextValue:* Obsah uzlu, který obsahuje text, nebo hodnota atributu uzlu.
- *LastSibling:* Poslední sourozenec. Využívá se při vyhodnocoví osy following-sibling.

Tabulka 3.1 ukazuje příklad takového uložení. Pokud pro daný záznam nemá smysl některý z vlastností uvádět, je vynechán, tzn. tato dvojice klíč/hodnota se vůbec neuvede.

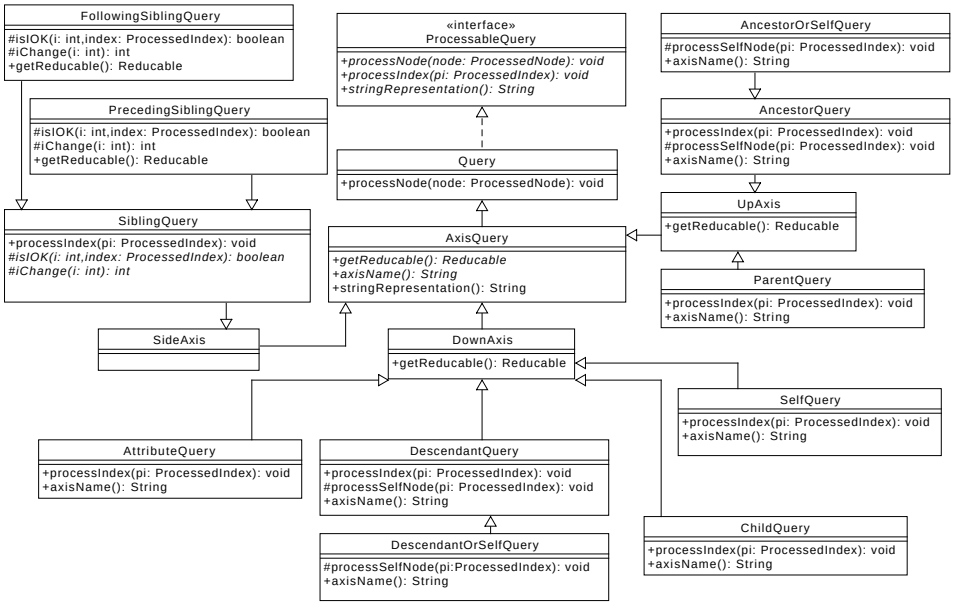
Výhodou tohoto uložení je, že je ve fázi Map možné vyhodnocovat každý uzel samostatně, nezávisle na ostatních.

3.4 Rozdělení XPath os

Xomoda implementuje vyhodnocování devíti XPath os. Tyto osy jsou rozděleny do dvou kategorií, které jsou důležité pro převod dotazu do modelu MapReduce. První kategorie se nazývá `down-axis`, patří do ní osa `self` ,`child`,

$$/Employees//node()/parent::name$$
$$XP_1 = (child :: Employees, descendant :: node(), parent :: name)$$

Obrázek 3.4: Příklad XPath dotazu a jeho rozkladu. Převzato z [4].



Obrázek 3.5: Diagram tříd pro vyhodnocení XPath dotazu.

`descendant` a `descendant-or-self`. Název této kategorie je odvozen od skutečnosti, že tyto osy označují pohyb směrem dolů v XML stromu.

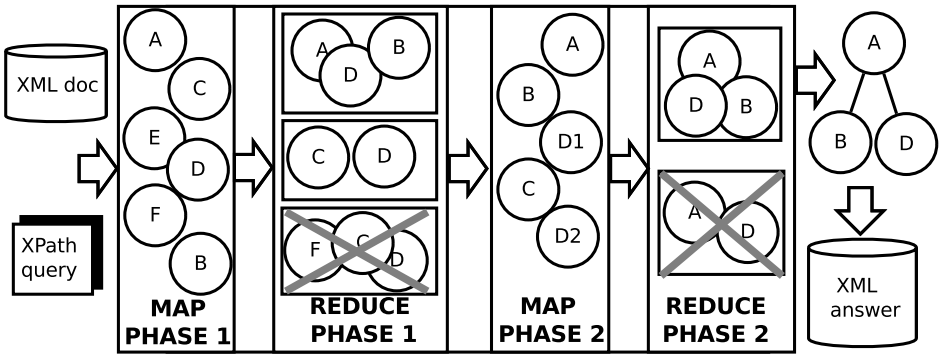
Druhá kategorie se nazývá `upside-axis`, protože vyhodnocování těchto os může vyžadovat pohyb vzhůru, nebo do stran v XML stromu. Patří sem osy `parent`, `ancestor`, `ancestor-or-self`, `following-sibling` a `preceding-sibling`.

Každý XPath dotaz se skládá z jednotlivých kroků (viz. 2.1.1). Pro účely vyhodnocování je proto dotaz rozdělen na posloupnost jednotlivých os. Příklad takového rozkladu ukazuje Obrázek 3.4

Způsob implementace s využitím dědičnosti ukazuje Obrázek 3.5. V diagramu jsou také zachyceny hlavní metody, které jsou volány během výpočtu.

3.5 Převod na MapReduce Job

Jak bylo řečeno výše, XPath dotaz je rozdělen na jednotlivé kroky. Jakkoliv dlouhá posloupnost kroků z kategorie `down-axis` může být zpracována v jedné fázi map. Pokud v je v dalším kroku posun z kategorie `upside-axis`, je nutné



Obrázek 3.6: Zpracování uzlů v modulem MapReduce.

krok vyhodnocovat ve fázi reduce. Pokud XPath dotaz obsahuje další krok, je nutné jej vykonat v novém MapReduce Job. Ten znovu může vyhodnocovat libovolné množství kroků z kategorie `down-axis` a maximálně jeden krok z kategorie `upside-axis`.

Map fáze V této fázi probíhají dvě základní operace. Jedná se o filtrování, kdy je rozhodnuto, zda daný uzel může být součástí potenciálního výsledku a dále probíhá určování emitovacího klíče a následná emitace uzlu. Daný uzel může být emitován s jedním, nebo několika klíči, případně být emitován nemusí. Uzly, který mají stejný emitovací klíč budou součástí stejného výsledku (budou na konci vyhodnocení použity pro kompozici jedné části výsledku, protože, jak bylo řečeno v 2.1, uzel může obsahovat další uzly nebo text). Pro každý uzel uložený v HBase může být vyhodnocena fáze map samostatně bez ohledu na ostatní uzly, takže vyhodnocování může probíhat paralelně.

Reduce fáze Uzly, které jsou emitovány se stejným emitovacím klíčem vstupují společně do jedné funkce reduce. V této fázi může docházet k další filtraci a kontrole. Pokud uzel nevyhovuje, je vyřazen, pokud splňuje požadavky, můžou výsledky z reduce fáze vstupovat do dalšího MapReduce Job nebo, pokud je funkce fáze reduce poslední, dojde k rekonstrukci výsledku a jeho uložení. Práci modulu MapReduce ilustruje Obrázek 3.6.

Kapitola 4

Realizace: Osy preceding a following

V systému Xomoda původně nebyly osy preceding a following vůbec implementovány. Vyhodnocování těchto os je obecně náročnější. Problematický je i převod vyhodnocování do modelu MapReduce.

4.1 Návrh

Příklad Mějme dotaz

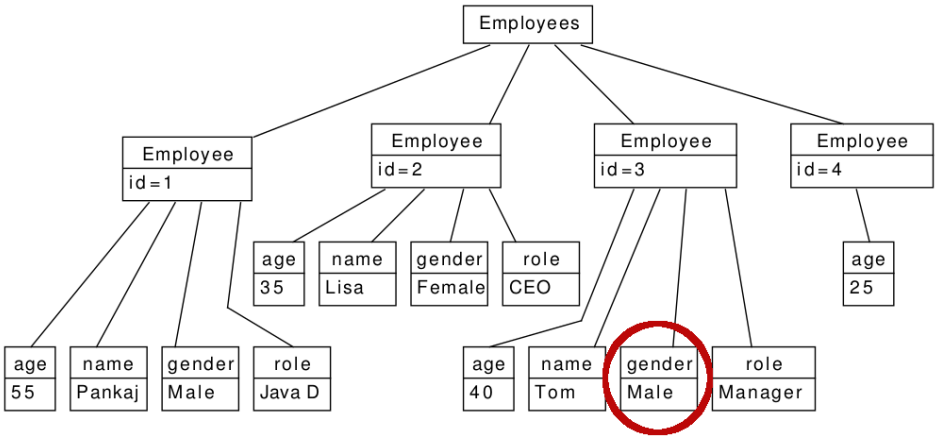
/Employees/Employee[@id="3"]/gender/following::node()

nad dokumentem 2.1. Během vyhodnocování je nejprve nutné najít referenční uzel, ten je v tomto případě pouze jeden. Ukazuje ho Obrázek 4.1 Poté dojde k vytvoření odpovědi. Odpovědí budou všechny uzly a jeich obsah, které se nacházejí vpravo v XML stromu od referenčního uzlu. Odpověď na dotaz je zobrazena na Obrázku 4.2. Povšimněme si, že pokud se ve výsledku osy following objeví uzel, tak výsledkem budou i všichni jeho potomci. Toto bude využito v další práci.

4.1.1 Porovnání vyhodnocování

Při vyhodnocování všech os z kategorie `down-axis`, lze již během fáze `map` u každého uzlu rozhodnout, zda bude součástí odpovědi nebo ne, takže daný uzel může být emitován s požadovaným indexem. Ve fázi `reduce` pak dochází pouze k sestavení odpovědi z emitovaných uzlů.

Běheme vyhodnocování os, které směřují vzhůru v XML stromu, tj. `parent`, `ancestor`, `ancestor-or-self` také dochází k výrazné redukci možných výsledků, takže jsou emitovány pouze uzly, které by mohly být výsledkem. Ve fázi



Obrázek 4.1: XML strom, podle dokumentu uvedeného v 2.1.

```
<role>Manager</role>

Manager

<Employee id="4">
 <age>25</age>
</Employee>

<age>25</age>

25
```

Obrázek 4.2: Výsledek XPath dotazu pro osu following.

reduce dochází pouze ke kontrole, zda opravdu existuje celá cesta složená z libovolného množství posunů z kategorie `down-axis`, na které by se nacházel vyhodnocovaný uzel.

Pro osy preceding-sibling a following-sibling je nutné ve fázi map stanovit referenční uzel. Pak je možné jej emitovat do všech jeho sourozenců. Výhodou vyhodnocování těchto os je, že již v této fázi lze množinu možných odpovědí značně omezit, protože odpovědí mohou být pouze přímí potomci uzlu, ke kterému se dostaneme kroky XPath dotazu bez posledního. Do fáze reduce se tedy vždy dostanou uzly, které jsou sourozenci. Pokud je některý z uzlů referenčním, objeví se mezi nimi a má nastavenou speciální značku (flag). V této fázi tedy dojde k nalezení uzlu se značkou a jako odpověď budou vyhodnoceny uzly, které se nacházejí před/za ním, takže mají lexikograficky menší/větší Dewey index. Pokud se mezi emitovanými uzly žádný referenční

uzel nenachází, odpovědí není žádný z emitovaných uzlů.

Problém os following a preceding je, že ve fázi map nelze nijak omezit množinu množných odpovědí. Můžeme pouze stanovit referenční uzel/uzly. Nelze určit, zda uzel v výsledku určitě bude nebo určitě nebude a to z toho důvodu, že každý uzel je vyhodnocován samostatně a daný uzel nemá žádnou informaci o tom, jak vypadá jeho okolí a celý XML strom.

4.1.2 Možnosti řešení

Pokud bychom použili řešení hrubou silou, museli bychom emitovat každý uzel s každým možným indexem, protože mohou být odpovědí. Problémem by byla i emitace referenčního uzlu, museli bychom zjistit celou strukturu XML dokumentu (zejména hloubku jednotlivých větví) a pak referenční uzel emitovat do všech možných odpovědí, abychom je tak označili jako správné. To by bylo samozřejmě výpočetně velmi náročné a neefektivní.

Po zvážení několika variant výpočtu jsem přistoupil ke dvěma optimalizacím, které spolu úzce souvisí. První je omezení emitace uzlů pouze na druhé patro XML stromu. Druhým je pak rozdělení výpočtu na dvě MapReduce úlohy.

4.1.3 Omezení emitace

První optimalizace řeší problém neznalosti emitovacího indexu během fáze map. Všechny uzly jsou emitovány s indexem odpovídajícího předchůdce z druhého patra XML stromu. Příloha A ukazuje, jak jsou uloženy jednotlivé uzly v distribuované databázi. Během fáze map tedy dochází k tomu, že každý uzel je emitován s Dewey indexem svého rodiče z druhého patra XML stromu, takže například uzel `age` s indexem `1:3:2:` je emitován s indexem `1:3:`. Takže pro účely emitace dojde pouze k ořiznutí indexu.

Během této fáze také dochází k testování, zda uzel není referenční. Pokud uzel odpovídá požadavkům, je emitován s nastavenou značkou (flag) se svým příslušným emitovacím indexem ořízntým podle pravidla uvedeného výše. Dále je pak emitován i s indexem všech následujících, případně předchozích uzlů z druhého patra.

Pokud například vyhodnocujeme osu following, zjistíme, že referenční uzel je například `Lisa` s indexem `1:2:3:1:`, pak bude tento uzel emitován jako referenční s indexy `1:2:`, `1:3:` a `1:4:`. Pokud bychom vyhodnocovali osu preceding, byl by emitován s indexy `1:2:` a `1:1:`.

Do fáze reduce pak vstupují všechny uzly z daného podstromu a navíc referenční uzly. Nejdříve je vyhledán nejlepší referenční uzel a následně dochází k eliminaci nevhodných uzlů.

Eliminace probíhá na základě Dewey indexu. Index vyhodnocovaného uzlu je lexikograficky porovnán s indexem referenčního uzlu a vyřazen, případně vynechán v odpovědi.

Výsledné uzly pak vstupují do druhé fáze vyhodnocování, jak je popsáno v následující části.

4.1.4 Rozdělení výpočtu

Důvodem tohoto rozdělení je skutečnost, že pokud je uzel odpovědí, pak jsou odpovědí i všichni jeho přímí i nepřímí potomci. Protože v první fázi výpočtu jsou nalezeny všechny kořenové uzly možných odpovědí. Ve druhé fázi tyto uzly vstoupí do vyhodnocování osy descending-or-self jako vstupní uzly. Tato osa je již v systému implementována a optimalizována. Tím je zajištěno, že budou v odpovědi zahruty všechny její součásti.

4.2 Implementace

Vyhodnocování os following a preceding je velmi podobné a dalo by se nazvat symetrickým. Implementace jednotlivých funkcí je rozdělena na dvě části. V první části jsou popsány funkce a třídy volány ve fázi map, ve druhé pak ty, které jsou volány z fáze reduce.

4.2.1 Fáze map

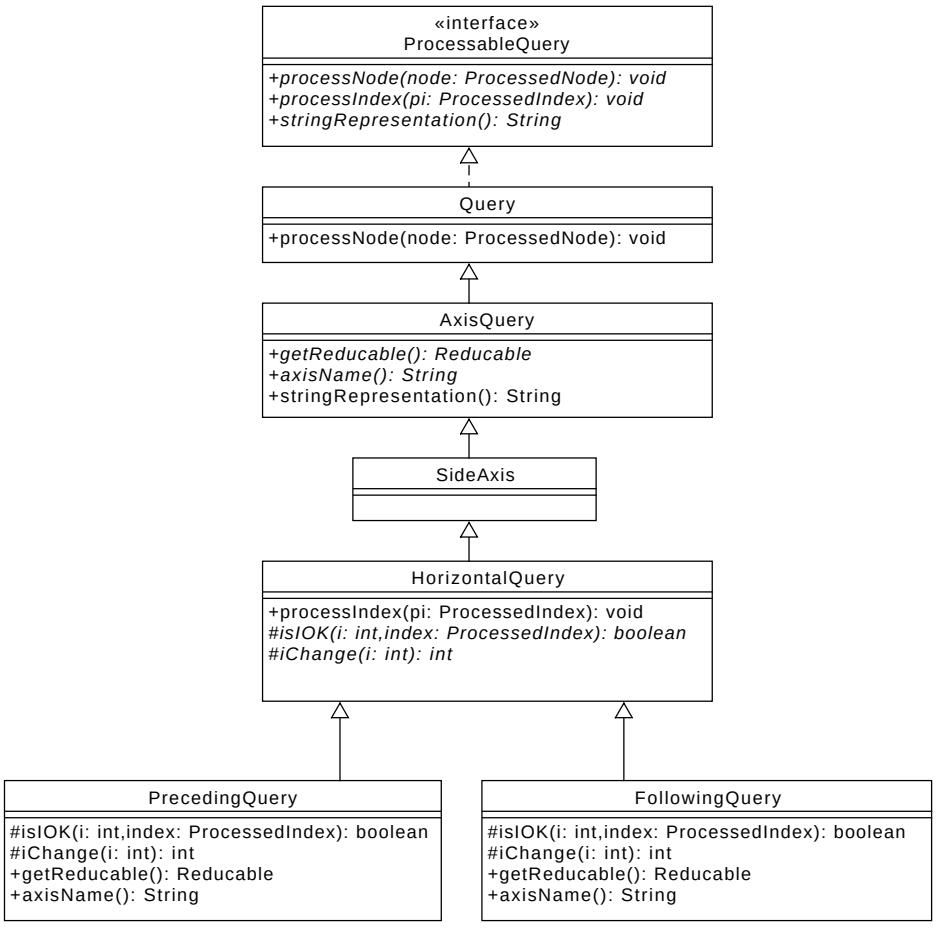
U obou os je třeba v prvním kroku najít referenční uzel a v druhém jej pak emitovat do skupin, které obsahují jeho následovníky nebo předchůdce, takže rozdíl je pouze v tom, zda index inkrementujeme nebo dekrementujeme. Z tohoto důvodu jsem využil dědičnosti a implementoval třídy `HorizontalQuery`, `PrecedingQuery` a `FollowingQuery`. Tyto třídy ukazuje diagram 4.3 v návaznosti na stávající třídy systému.

Třída `HorizontalQuery` obsahuje hlavní výpočetní logiku pro vyhodnocování obou os. Třídy `PrecedingQuery` a `FollowingQuery` pak implentují pouze odlišné funkce pro posuv a kontrolu při emitaci, které jsou volány během vyhodnocování, konkrétně funkce `isIOK(int i, ProcessedIndex index)` a `iChange(int i)`.

4.2.2 Problém osy following a úprava indexace

V systému Xomoda je každý uzel zpracován samostatně a zná pouze svou polohu v XML stromu, ale už nezná informaci o velikosti, případně šířce stromu. Z toho vyplývá problém pro vyhodnocování osy following. Pokud je nalezen referenční uzel, nelze jednoduše zjistit, do jaké šíře je nutné index emitovat. Při vyhodnocování osy preceding není nutné toto řešit, protože jsou uzlům přidělovány indexy od čísla jedna, takže jednoduše emitujeme do všech uzlů od těch s indexem jedna až po index referenčního uzlu.

Po zvážení možností byla zvolena varianta, která během indexace přidá každému uzlu informaci o šířce XML stromu na druhém patře. V příloze A



Obrázek 4.3: Diagram tříd HorizontalQuery, PrecedingQuery a FollowingQuery.

je, stejně jako v reálné databázi, tento údaj uveden jako `distributionmax`. V systému Xomoda tuto funkcionalitu zajišťuje třída `XMLReindexer2`, která využívá výsledků vytvořených třídou `XMLreindexer`, která byla implementována v původním verzi. A je spolu s ní volána během ukládání XML dokumentu do databáze.

4.2.3 Fáze reduce

V této fázi dochází nejdříve k vyhledání nejlepšího referenčního uzlu a poté k vyřazení uzlů, které se nacházejí na špatné pozici vzhledem k referenčnímu uzlu (před uzlem u osy following a za uzlem u osy preceding). Vyhodnocování je, podobně jako ve fázi map, symetrické, proto byla hlavní logika

implementována v abstraktní třídě `HorizontalReducer`. Tato třída během výpočtu využívá metody `boolean betterPosition(String deweyIndex)` a `boolean correctPosition(String deweyIndex)`, které jsou specifické pro jednotlivé osy, a proto jsou implementovány ve třídách `PrecedingAxisReducer` a `FollowingAxisReducer`. Tyto třídy jsou pak přiděleny pro vyhodnocení daných os.

4.2.4 Implementace rozdělení výpočtu

Rozdělení výpočtu os preceding a following a na výpočet kořenových uzlů odpovědí a vyhodnocení výsledku jako osy descendant-or-self nad kořenovými uzly (jak je popsáno v 4.1.4) je realizováno ve třídě `XomodaXPathQuery` ve funkci `processInput(List<Query> queries)`. Tato funkce je volána konstruktorem této třídy při vytváření nového XPath dotazu.

4.3 Testování

K testování jsem použil framework JUnit. Hlavní pozornost byla věnována testování nově implementovaných tříd. První část testování byla zaměřena na správnou funkci a korektní práci těchto tříd. Ve druhé fázi pak byla testována funkčnost systému na složitějších dotazech a větších souborech. Ke kontrole výsledků byly použity třídy, které systém Xomoda již obsahoval a byly použity pro testování v předešlých verzích, zejména `ResultComparator` a `SaxonXpathProcessor`. Tyto třídy pro vyhodnocování používají nástroje balíku SAX.

4.3.1 Testování fáze map

Pro účely testování tříd `HorizontalQuery`, `PrecedingQuery` a `FollowingQuery` byly vytvořeny testovací třídy `PrecedingQueryTest` a `FollowingQueryTest`. Testují především správné oříznutí indexu a dále správnou emitaci referenčního uzlu.

4.3.2 Testování fáze reduce

Tato část testování se zaměřila na testování na práci tříd `HorizontalReducer`, `PrecedingAxisReducer` a `FollowingAxisReducer`. Důležité je správný výběr referenčního uzlu a vyřazení uzlů s nevhodnou polohou.

4.3.3 Ověření správného rozdělení výpočtu

Dalším testováním byla ověřena správnost rozdělení výpočtu osy preceding/following podle 4.1.4 ve třídě `XomodaXPathQuery`, ve funkci `processInput(List<Query> queries)`. O tento test se stará třída `TestDivide`.

4.3.4 Komplexní testy

K prověření celkové funkčnosti vyhodnocování byla vytvořeny třídy `PrecedingCompareTest` a `FollowingCompareTest`, které obsahují složitější dotazy nad většími daty.

Závěr

Abych se mohl věnovat vylepšování systému Xomoda, bylo nejdříve nutné seznámit se s použitým technologiemi. V tomto ohledu práce v první kapitole nabízí ucelený přehled současných možností a přístupů ke zpracování a analýze dat a zaměřuje se dále na oblasti, na které je orientován systém Xomoda. Takže obecně na technologie pro Big Data a dále pak na konkrétní komplexní řešení v podobě frameworku Hadoop.

Samotnému systému Xomoda a jeho architektuře je věnována celá kapitola. V ní jsou shrnuty všechny podstatné myšlenky a techniky, na kterých je systém Xomoda vystavěn, včetně způsobu uložení do databáze a techniky vyhodnocení.

Dalším cílem byla implementace dvou zbývajících os. Vyhodnocení těchto os nebyla dosud v systému implementována z důvodu větší výpočetní náročnosti a obtížné transformace výpočtu do modelu MapReduce. Po zvážení všech možností jsem přistoupil ke kompromisu v podobě dvou hlavních úprav oproti ostatním osám – omezení emitace a rozdělení výpočtu do dvou MapReduce úkolů. Tím se mi podařilo vyřešit problém osy preceding, nicméně, aby tyto úpravy mohly být použity pro osu following, bylo nutné optimalizovat datové úložiště, což byl další z cílů této práce. Tato optimalizace spočívá v přidání údaje o šířce XML stromu ve druhé úrovni do databáze.

Dále jsem měl navrženou optimalizaci implementovat. Vzhledem k tomu, že implementace vyhodnocování osy following byla podmíněna funkční optimalizací, je popsána v kapitole společně s návrhem a implemenací os preceding a following. Dopad navržené optimalizace je především v tom, že díky ní bylo možné realizovat vyhodnocování osy following s rozumnou náročností.

Další možnosti ve vývoji systému Xomoda jsou obrovské. Je možné se dále orientovat na další optimalizaci a další urychlení výpočtu. Je také možné zabývat se škálovatelností tohoto systému a s tím související optimální velikost clusteru. Výhodou jazyků XML a XPath je, že jsou velmi populární a všeobecně používané.

Literatura

[1] XPath Axes and their Shortcuts [online]. [cit. 2015-04-22]. Dostupné z: https://our.umbraco.org/media/upload/0562fd58-c6db-4fa8-a432-68b28f11c3f2/7x1B0.gif

[2] NoSQL and its use in Ceilometer [online]. [cit. 2015-04-22]. Dostupné z: http://robertgreiner.com/uploads/images/2014/CAP-full.png

[3] Wang, L.; Tao, J.; Ranjan, R.; aj.: G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, ročník 29, č. 3, 2013: s. 739–750.

[4] Šenk, A.; Valenta, M.; Benn, W.: Distributed Evaluation of XPath Axes Queries over Large XML Documents Stored in MapReduce Clusters. 2014: s. 253–257, ISSN 1529-4188, doi:10.1109/DEXA.2014.59.

[5] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-xml-19980210.*, 1998. Dostupné z: http://www.w3.org/TR/1998/REC-xml-19980210

[6] Jim Melton, S. B.: *Morgan Kaufmann Series in Data Management Systems : Querying XML : XQuery, XPath, and SQL/XML in Context*. Morgan Kaufmann, 2006, ISBN 9780080540160.

[7] Vaish, G.: *Hadoop For Getting Started with NoSQL*. Packt Publishing, 2013, ISBN 9781849694988.

[8] Oracle Database SQL Reference 10g Release 1 (10.1) [online]. [cit. 2011-04-24]. Dostupné z: http://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_1001.htm

[9] Tiwari, S.: *Professional NoSQL*. Wrox, 2011, ISBN 9780470942246.

[10] Gilbert, S.; Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002, doi:10.1145/564585.564601.

[11] Dean, J.; Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Communications of the ACM*, ročník 51, č. 1, 2008: s. 107–113, ISSN 0001-0782.

[12] Under the Hood: Hadoop Distributed Filesystem reliability with Namenode and Avatarnode [online]. [cit. 2011-04-24]. Dostupné z: `https://www.facebook.com/notes/facebook-engineering/under-the-hood-hadoop-distributed-filesystem-reliability-with-namenode-and-avata/10150888759153920`

[13] Believe it or not: Avatar takes 1 petabyte of storage space, equivalent to a 32 YEAR long MP3. [online]. [cit. 2011-04-24]. Dostupné z: `http://thenextweb.com/2010/01/01/avatar-takes-1-petabyte-storage-space-equivalent-32-year-long-mp3/`

[14] Chang, F.; Dean, J.; Ghemawat, S.; aj.: Bigtable: A Distributed Storage System for Structured Data. Google. *Inc., Novembro*, 2006.

[15] Free Software Foundation: *Various Licenses and Comments about Them - GNU Project [online]*. [cit. 2011-04-24]. Dostupné z: `https://www.gnu.org/licenses/license-list.en.html`

[16] Borthakur, D.: HDFS architecture guide. *Hadoop Apache Project*, 2008.

[17] Taylor, R. C.: An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics*, ročník 11, č. Suppl 12, 2010: str. S1.

[18] Strnad, P.; Macek, O.; Jira, P.: Mapping XML to Key-Value Database. In *DBKDA 2013, The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications*, 2013.

[19] Camacho-Rodríguez, J.; Colazzo, D.; Manolescu, I.: Building Large XML Stores in the Amazon Cloud. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, IEEE, 2012, s. 151–158.

[20] Bidoit, N.; Colazzo, D.; Malla, N.; aj.: Partitioning XML documents for iterative queries. In *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, ACM, 2012, s. 51–60.

[21] Tatarinov, I.; Viglas, S. D.; Beyer, K.; aj.: Storing and querying ordered XML using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, ACM, 2002, s. 204–215.

Příloha **A**

Způsob uložení XML dokumentu z příkladu 2.1 v systému Xomoda

Tabulka A.1: Uložení dokument XML dokumentu (Obrázek 2.1) do HBase databáze. Část první.

Dewey Index	**Node info**
1:	type:ELEMENT; distributionmax:4; biggestsibling:1; pathindex:Employees/; name:Employees;
1:1:	type:ELEMENT; distributionmax:4; biggestsibling:4; pathindex:Employee/Employees/; name:Employee;
1:1:1:	type:ATTRIBUTE; distributionmax:4; biggestsibling:5; pathindex:id/Employee/Employees/; name:id; textval:1;
1:1:2:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:age/Employee/Employees/; name:age;
1:1:2:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/age/Employee/Employees/; textval:55;
1:1:3:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:name/Employee/Employees/; name:name;
1:1:3:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/name/Employee/Employees/; textval:Pankaj;
1:1:4:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:gender/Employee/Employees/; name:gender;
1:1:4:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/gender/Employee/Employees/; textval:Male;
1:1:5:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:role/Employee/Employees/; name:role;
1:1:5:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/role/Employee/Employees/; textval:Java Developer;
1:2:	type:ELEMENT; distributionmax:4; biggestsibling:4; pathindex:Employee/Employees/; name:Employee;
1:2:1:	type:ATTRIBUTE; distributionmax:4; biggestsibling:5; pathindex:id/Employee/Employees/; name:id; textval:2;
1:2:2:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:age/Employee/Employees/; name:age;
1:2:2:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/age/Employee/Employees/; textval:35;
1:2:3:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:name/Employee/Employees/; name:name;
1:2:3:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/name/Employee/Employees/; textval:Lisa;
1:2:4:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:gender/Employee/Employees/; name:gender;
1:2:4:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/gender/Employee/Employees/; textval:Female;

Tabulka A.2: Uložení dokument XML dokumentu (Obrázek 2.1) do HBase databáze. Část druhá.

Dewey Index	**Node info**
1:2:5:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:role/Employee/Employees/; name:role;
1:2:5:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/role/Employee/Employees/; textval:CEO;
1:3:	type:ELEMENT; distributionmax:4; biggestsibling:4; pathindex:Employee/Employees/; name:Employee;
1:3:1:	type:ATTRIBUTE; distributionmax:4; biggestsibling:5; pathindex:id/Employee/Employees/; name:id; textval:3;
1:3:2:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:age/Employee/Employees/; name:age;
1:3:2:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/age/Employee/Employees/; textval:40;
1:3:3:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:name/Employee/Employees/; name:name;
1:3:3:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/name/Employee/Employees/; textval:Tom;
1:3:4:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:gender/Employee/Employees/; name:gender;
1:3:4:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/gender/Employee/Employees/; textval:Male;
1:3:5:	type:ELEMENT; distributionmax:4; biggestsibling:5; pathindex:role/Employee/Employees/; name:role;
1:3:5:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/role/Employee/Employees/; textval:Manager;
1:4:	type:ELEMENT; distributionmax:4; biggestsibling:4; pathindex:Employee/Employees/; name:Employee;
1:4:1:	type:ATTRIBUTE; distributionmax:4; biggestsibling:2; pathindex:id/Employee/Employees/; name:id; textval:4;
1:4:2:	type:ELEMENT; distributionmax:4; biggestsibling:2; pathindex:age/Employee/Employees/; name:age;
1:4:2:1:	type:TEXT; distributionmax:4; biggestsibling:1; pathindex:/age/Employee/Employees/; textval:25;

Příloha B

Seznam použitých zkratek

DDL Data Definiton Language

DML Data Manipulation Language

HDFS Hadoop Filesystem

HTML HyperText Markup Language

JSON JavaScript Object Notation

NoSQL Not only SQL

SAX Simple API for XML

SQL Structured Query Language

XML Extensible Markup Language

XPath XML Path Language

YAML Ain't Markup Language

W3C World Wide Web Consortium

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS