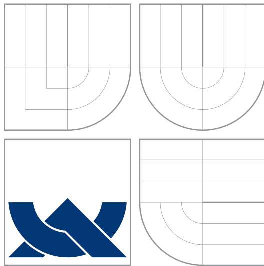
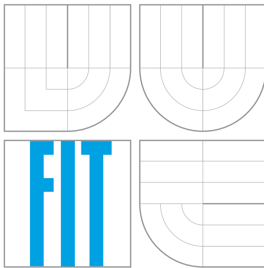


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ V SÉMANTICKY OBOHACENÝCH TEXTECH

SEARCH IN SEMANTICALLY-ENRICHED TEXTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ DOSTÁL

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2016

Abstrakt

Tato práce se zabývá dotazováním nad rozsáhlými textovými daty, obohacenými o sémantické informace, zejména o vyznačení pojmenovaných entit zmíněných v textu. Text nejprve rozebírá způsoby indexace částečně stukturovaných textových dat a přístupy k jejich dotazování. Jádrem práce je potom návrh a implementace dotazovacího systému, který analyzuje dotazy uživatele a požadavky na rozsah a strukturu vracených informací, převádí je do formátu vyhledávacího systému MG4J, na výsledku aplikuje zadaná omezení a formátuje jej do požadovaného tvaru. Výsledky práce jsou zhodnoceny na základě statistik časových odezev sady dotazů nad indexem anglické Wikipedie, které jsou také porovnány s údaji získanými pomocí dosavadního systému dotazování.

Abstract

This thesis deals with searches within large-scale text data, which are enriched by semantic information, especially the highlighting of named entities. The thesis begins by analysing methods used for the indexing of partly structured text data and the methods used for searching. The main body of the thesis consists of the description and implementation of a query engine, which analyses queries and requirements on range and structure. This query is later converted into an MG4J search engine format and applies restrictions on the results. These results have been tested on time response, as well as on a number of results, which are then compared to older results obtained from English Wikipedia.

Klíčová slova

index, dotazovací systém, MG4J, pojmenované entity, obohacená data, Java

Keywords

indexing, search engine, MG4J, named entity, enriched data, Java

Citace

Jiří Dostál: Vyhledávání v sémanticky obohacených textech, bakalářská práce, Brno, FIT VUT v Brně, 2016

Vyhledávání v sémanticky obohacených textech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Jiří Dostál
18. května 2016

Poděkování

Velice rád bych poděkoval panu Doc. RNDr. Pavlu Smržovi, Ph.D. za odborné vedení práce, trpělivost při konzultacích a za veškerou pomoc, kterou mi při tvorbě práce poskytl.

© Jiří Dostál, 2016.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Indexace dat **3**
- 1.1 Úvod . . . 3
- 1.2 Organizace dat . . . 4
  - 1.2.1 Centralizovaný index . . . 4
  - 1.2.2 Distribuovaný index . . . 4
- 1.3 Proces indexace . . . 5
  - 1.3.1 Příprava dokumentů . . . 5
- 1.4 Existující systémy . . . 7
  - 1.4.1 Lucene . . . 7
  - 1.4.2 Elastic Search . . . 8
  - 1.4.3 MG4J . . . 8

2 Dotazování dat **11**
- 2.1 Google Advanced Search . . . 11
  - 2.1.1 Symboly . . . 11
  - 2.1.2 Operátory . . . 11
  - 2.1.3 Modifikátory URL . . . 12
- 2.2 SPARQL . . . 12
  - 2.2.1 Jazyk . . . 12
- 2.3 Dotazování MG4J . . . 12
  - 2.3.1 Operátory . . . 13

3 Návrh aplikace **14**
- 3.1 Úvod . . . 14
- 3.2 Vývojové prostředí . . . 14
  - 3.2.1 Servery . . . 14
  - 3.2.2 Psaní zdrojových textů . . . 15
- 3.3 Skriptovací a programovací jazyky . . . 15
  - 3.3.1 Bash . . . 15
  - 3.3.2 Java . . . 15
- 3.4 Použité technologie . . . 16
  - 3.4.1 REST . . . 16
  - 3.4.2 Serializační formáty . . . 18

4 Implementace **22**
- 4.1 Vstupní data . . . 22
  - 4.1.1 Přepínače . . . 22
  - 4.1.2 Konfigurační soubor . . . 22

4.1.3 Vnitřní reprezentace vstupu 22
4.2 Logování a chybové hlášky 23
4.3 Nastavení a přístup k serverům 23
4.4 Dotazování serverů 24
4.5 Modifikace dotazu 24
4.6 Personalizace odpovědi 24
4.6.1 Proces zpracování zprávy 25
4.7 Postfiltering 25

5 Statistiky a experimenty **27**
5.1 Časová odezva 27
5.2 Počet výsledků 28
5.3 Porovnání dosavadních výsledků 29

6 Komplikace, limity a rozšíření **31**
6.1 Komplikace při implementaci 31
6.1.1 Překlad zdrojových textů 31
6.1.2 Spuštění vlastního serveru 31
6.1.3 Paměťová omezení 31
6.1.4 Slabá komunita systému *MG4J* 32
6.1.5 Nedostupnost serverů 32
6.2 Limity práce 32
6.2.1 Nedoručené výsledky od serveru 32
6.3 Možná rozšíření 32

7 Závěr **33**

Literatura **34**

Přílohy **36**
Seznam příloh 37

A Obsah CD **38**

B Překlad a spuštění **39**
B.1 Překlad aplikace 39
B.2 Generování dokumentace 39
B.3 Příklady spuštění 39

C Konfigurační soubor **40**

Kapitola 1

Indexace dat

Obsahem této kapitoly jsou informace o indexování dat, co tento pojem obnáší, jaký je význam indexace dat a rozbor několika nejpoužívanějších přístupů k tomuto procesu. Indexace dat je stěžejním prvkem procesu zpracování velkého množství dat a je velmi důležité, aby indexace byla provedena správným a optimálním způsobem pro řešení konkrétního problému.

1.1 Úvod

V době, kdy lidstvo začalo shromažďovat různé informace z několika okruhů, začala vznikat myšlenka logické strukturalizace dat. Právě rozdělení a označení dat správným způsobem usnadňuje přístup k informacím v nich obsažených. Dříve, například v případě kartoték či jiných fyzických datových skladů, se využívalo seskupování podle specifických znaků či podle abecedy. V případě kartotéky obsahující zdravotní karty pacientů pak nebyl problém rychle a snadno nalézt hledanou osobu. Dalším atributem, podle kterého by mohlo být provedeno seskupení je například rok narození. Vyhledání osoby v takovém fyzickém systému by pak bylo ještě jednodušší. Těchto atributů bychom jistě nalezli mnoho a využití těchto klíčů by znamenalo značné zlepšení v rychlosti nalezení hledané položky.

Je zřejmé, že organizace dat hraje velkou roli při práci s rozsáhlou databází. V dnešním světě se nejspíše setkáme s databází elektronickou. Tento trend má mnoho výhod, avšak nese s sebou i některé nevýhody. Výhodou je například možnost skrytí pozadí databáze tak, aby koncový uživatel nebyl zatížen procesy, které k výkonu své práce nepotřebuje. Například zařazení nové položky by v případě fyzického datového skladu znamenalo nalézt místo, kam položku vložit, vyhradit pro ni místo, označit ji atd. V elektronické podobě by to pro něj znamenalo pouze vyvolání příkazu "vlož položku" a nastavení hodnot. O samotný proces vložení se následně postará samotný systém. Další výhodou je možnost odkazovat se na několik atributů současně, množství atributů nemusí být ničím omezeno. Rychlost nalezení shody na hledanou položku je také nesrovnatelná, i přes to, že výpočetní výkon strojů zpracovávajících dotazy je omezený.

Mezi prvky, které jsou v elektronických datových skladech užitečné, ale současně mohou být na obtíž je například jejich velikost. Zpracováváme-li sadu dat, která nepřesahuje určité množství, zvolíme jinou organizaci položek, než v případě rozsáhlých kolekcí, které nazýváme Big Data [7].

Nevýhodou takových systémů je jejich závislost na technické výbavě. Stroj, který pracuje s takovou databází, musí mít dostatečný výkon pro uložení a procházení těchto položek.

Zřídka se setkáme s případy, kdy je tato operace prováděna běžnými stolními počítači nebo laptopy. Běžným přístupem je využití serverů nebo výpočetních clusterů. Návrh a vytvoření takového systému vyžaduje odborné znalosti z oblasti informačních technologií, tudíž je často potřeba do procesu organizace dat přizvat profesionály z oboru nebo si nechat systém navrhnout na míru.

1.2 Organizace dat

Indexace obecně značí proces, během kterého jsou získaná data upravena tak, aby přístup k nim byl rychlý a efektivní. Indexem tedy rozumíme množinu dat doplněnou o struktury, pomocí kterých k datům přistupujeme. Tento proces usnadňuje vyhledávání nad těmito daty, jelikož není potřeba procházet všechna data k nalezení shody. Vytvoření a udržování indexů je pro velká data často proces velmi náročný, proto se s indexací nejčastěji setkáváme na výpočetních clusterech, které jsou schopny operaci rozdělit na více strojů a urychlit tak samotné zpracování.

Při vytváření indexu je nutné předem promyslet, jakým způsobem budou položky organizovány, jak k nim bude přistupováno a jak a kde budou uloženy. Mezi základní aspekty patří především to, zda systém bude centralizovaný či distribuovaný.

1.2.1 Centralizovaný index

Všechny komponenty indexu jsou uloženy centrálně na jednom stroji, který současně provádí veškeré operace. Výhodou centralizovaného návrhu je jeho jednoduchost při implementaci a poměrně levná pořizovací cena výpočetní techniky. Dále není potřeba navrhovat komunikační protokol pro komunikaci mezi jednotlivými uzly, jelikož se vše odehrává v rámci jednoho serveru.

Centralizovaný systém však není příliš rozšířený kvůli nízké efektivitě při práci s rozsáhlými daty, jelikož úroveň paralelismu je v tomto případě poměrně nízká. Další vlastností je využití sdílené paměti napříč celým indexem. [13]

1.2.2 Distribuovaný index

Distribuovaný index se od centralizovaného liší počtem procesů či fyzických strojů, na kterých je index spuštěn. Jednotlivé stroje zpracovávají různé komponenty indexu. Systém je tedy tvořen množinou prvků, které nazýváme uzly. Tyto uzly jsou mezi sebou propojeny sítí a navzájem spolu komunikují pomocí zpráv.

Oproti centralizovanému indexu je distribuovaný index mnohem častěji používaný. Důvodem je rychlost, kterou index pracuje s velkými kolekcemi dat. Vyšší rychlost je umožněna především díky vysoké úrovni paralelismu. Všechny uzly běží asynchronně a nezávisle na sobě, což je důsledkem absence sdílené paměti. Každý uzel má svou vlastní paměť, je tedy možné operace provádět úplně paralelně.

Nevýhodou tohoto návrhu indexu je vyšší náročnost na výpočetní techniku a složitější implementace. [13]

Svá úskalí má distribuovaný index i ve svém dotazování. Ne všechny operace lze totiž provádět plně distribuovaně. Snadno lze provést výběr či zápis do konkrétního indexu, problém nastává v případě operace `JOIN`. Operace `JOIN` je v distribuovaných indexech proces velmi náročný a pro jeho zpracování je potřeba počítat s větší časovou odezvou. Problém je v tom, že data z uzlů, které mají být spojeny musí být přenesena na centrální uzel, kde je

operace spojení provedena. Přístupů k operaci `JOIN` v distribuovaných systémech je několik a stále se objevují nové výzkumné techniky [3].

1.3 Proces indexace

V této části práce jsou popsány způsoby, jakými se zpracovávají vstupní data a princip vytvoření indexu. Ačkoliv jsou dotazovací systémy často navrženy s ohledem na konkrétní řešený problém, následujících několik kroků je vždy alespoň přibližně dodrženo.

1.3.1 Příprava dokumentů

Prvním krokem v procesu indexace dat je nashromáždění, příprava a zpracování relevantních dokumentů, ve kterých bude později možné vyhledávat. Tato část obnáší především:

- Převedení dokumentů do předem definovaného normalizovaného stavu
- Rozčlenění dokumentu na menší celky, které bude možné vyhledávat
- Izolace metadat z příslušných částí celku
- Identifikace potenciálně indexovatelných elementů v dokumentech
- Odstranění *stop slov* (slova, která sama o sobě nenesou žádný význam)
- Získání kořenů zbylých slov
- Výpočet váh slov v dokumentech (pouze pokud je to nutné pro daný případ)
- Sestavení indexu

Přípravná fáze

Jelikož jsou často dokumenty nashromážděny z různých zdrojů, je běžné, že se budou lišit svým formátem. Prvním krokem pro správné zpracování dokumentů je jejich normalizace do společného formátu, který bude dále jednotně zpracováván. Špatná volba datové struktury a velikost jednotlivých dokumentů může mít velký vliv na pozdější zpracování.

Nalezení indexovatelných výrazů

V tomto kroku se zabýváme problémem, které části textu chceme mít uložené v indexu a jak toho dosáhnout. Zde se setkáváme s lingvistickými problémy, které do velké míry řeší disciplína *Natural Language Processing*, která se zabývá zpracováním přirozeného jazyka. Jedním příkladem, kdy může nastat problém se správným rozpoznáním výrazů, jsou například sousloví. Sousloví jsou ustálená slovní spojení, která nesou význam jednoho slova. Vhodným příkladem z angličtiny je například “hot dog”. Samotná slova “hot” a “dog” mají naprosto jiný význam, než ve společném znění. Sousloví je tedy vhodné indexovat jako celek, nikoliv jako více slov. Podobných problémů bychom nalezli mnoho, avšak toto není předmětem práce, nýbrž NLP.

V této fázi vytváření indexu je nutné dodržovat předem stanovená pravidla, podle kterých bude indexace probíhat. Je nevyhnutelné, aby při tomto procesu byla prováděna manuální kontrola, následovaná vyhodnocením, zda jsou dosavadní výsledky smysluplné či nikoliv, případně množinu pravidel upravit tak, aby se výsledek co nejvíce přibližoval předpokladu.

Odstranění stop slov

Stop slova jsou slova, která sama o sobě nenesou žádný význam. Do této kategorie bychom z angličtiny mohli zařadit slova jako “the”, “a”, “and”, “but” .... Odstraněním těchto slov dosáhneme nižších požadavků na systémové zdroje a urychlíme samotné zpracování dotazu. Toto si můžeme dovolit v případě, že pro daný projekt stop slova nebudou znamenat změnu významu dokumentu, nebo že výrazně neovlivní výsledky vrácené na odeslaný dotaz uživatelem. Tento proces nemusí být vždy výhodný, protože například “to be or not to be” se skládá pouze ze stop slov. Efektivnějších výsledků lze dosáhnout manuálním zpracováváním nebo algoritmy k tomu určenými.

Lemmatizace

K nalezení kořenu slova často stačí odstranit předpony a přípony slova. Tuto akci je vhodné provádět rekurzivně, dokud se nedostaneme na úplný základ slova. Jedním z důvodů, proč je vhodné tento krok provést, je snížení počtu unikátních slov ve slovníku. To má pozitivní vliv na velikost celého indexu, a tudíž i na jeho prohledávání a jednoduchost. Dalším důvodem, proč jsou kořeny slov klíčové, je fakt, že pokud uživatel hledá například slovo “stroj”, pravděpodobně bude mít i zájem o výsledky zahrnující slova “strojař”, “strojní” nebo “strojírenský”. Bez znalosti kořene slova bychom také nemohli správně vyhodnotit dotaz na slovo, které není napsáno v úplném znění.

Algoritmy pro získání kořene slova bychom také mohli nalézt v oboru NLP. Důležité je však promyslet, zda chceme slovo úplně “očistit”, nebo jen převést do základního tvaru.

Rozšíření slov o metadata

Tento krok není nutný pro každou situaci, ale je vhodný pro situace, kdy navrhujeme systém, který bude dále používán jako nástroj pro dolování informací.

Jádrem tohoto procesu je obohatit množinu jednotlivých výskytů slov o metadata, která popisují, o jaký typ informace se jedná. V případě analýzy webových stránek se může jednat například o rozpoznání odkazů, nadpisů, zvýrazněných slov a podobně. Takto rozpoznaná data jsou potom přidána do datové struktury našeho indexu.

Jelikož se tato práce zabývá především zpracováním textových dat, je v tomto kroku důležitý pojem *NER (Named Entity Recognition).* Tento pojem lze přeložit jako rozpoznání pojmenovaných entit, což značí, o jaký typ výrazu se jedná. Rozpoznat tedy lze jména, názvy měst, letopočty, data úmrtí/narození, umělecká díla a podobně. Díky tomuto procesu lze následně do jisté míry odpovídat na dotazy typu “Jaké obrazy namaloval Leonardo DaVinci” nebo “Kdy zemřel Albert Einstein”.

Pojmenované entity v textu přidávají velkou hodnotu jednotlivým slovům, tudíž celková hodnota nashromážděných dat razantně stoupne. Přidání pojmenovaných entit je výhodné především u systémů, které pracují s velkým počtem dat a může obsahovat stejná slova, avšak různého významu. Pojmenovanými entitami lze potom rozlišit, který z významů je relevantní pro daný dotaz. Pro systémy umožnující vyhledávání na sémantické úrovni je potom tento proces nezbytný. Je však nutné počítat s vyššími paměťovými nároky celého systému.

Přiřazení váhy výrazům

V nejjednodušším případě lze váhy jednotlivých slov vyjádřit pomocí binární reprezentace a to tím způsobem, že je výrazu přiřazena váha *0* v případě, že se konkrétní výraz v

dokumentu nenachází a *1* v případě, že je výraz v dokumentu obsažen. Často je však nutné, aby vyhledávací nástroj byl komplexní, a tudíž se pro označení významu výrazu v daném dokumentu používá stupnice s větším počtem hodnot než pouze *1* a *0*. Zvolená stupnice může být v každém systému různá, avšak musí být zachováno stejné hodnocení napříč celým systémem. Taková stupnice může například nabývat hodnot od *0* až po *10*. Je-li výraz pro daný dokument klíčový, bude mu logicky přiřazeno vyšší číslo než v případě, že je výraz málo vypovídající o obsahu celého dokumentu. Správné přiřazení váhy výrazu je důležité pro výsledné řazení výsledků podle relevance. Algoritmů pro přiřazení správné váhy výrazu je mnoho a jedná se o poměrně složitý postup. Většina algoritmů je stavěna na procesu výpočtu frekvence výrazu napříč dokumentem. Tato frekvence je potom porovnávána s frekvencí napříč celým indexem a následně je vypočítána váha daného výrazu. Tento postup je však často nepřesný, protože ne všechny výrazy jsou vhodným "diskriminátorem". Příkladem je slovo "the", které je v textech často použito mnohokrát, avšak jeho frekvence výskytů nevypovídá o tom, že se jedná o článek popisující problematiku členů v anglickém jazyce. Opačným příkladem je výraz *Leonardo*, kde frekvence výskytů značně napoví, že se může jednat o článek o životě Leonarda DaVinci. Je tedy nutné brát v potaz i ostatní faktory než samotnou frekvenci výskytů, což řeší hodnotící funkce jako *BM25* [8] nebo *TF.IDF* [2].

Tyto váhy jsou nejčastěji uloženy přímo do indexu, aby nebylo nutné provádět proces přiřazení váhy výrazu při zpracovávání dotazu od uživatele, což by výrazně ovlivnilo rychlost odezvy vyhledávače.

Sestavení indexu

Sestavením indexu se rozumí vytvoření finální datové struktury, která v sobě nese všechny dosud nastřádané informace o výrazu v daném dokumentu. Podoba indexu se často liší, avšak nejčastěji se vytváří tak zvaný *invertovaný index*, což je struktura podobná tabulce, která v řádcích obsahuje všechny výrazy ze všech dokumentů seřazené podle abecedy doplněné o informace jako počet výskytů v celém indexu, počet výskytů v jednotlivých dokumentech, jejich konkrétní umístění v dokumentu, jeho váhu či pojmenovanou entitu v kontextu dokumentu.

1.4 Existující systémy

V dnešní době je snadné použít již existující systém pro řešení našeho problému. Těchto systému je na internetu k nalezení několik, a to jak komerční tak volně šiřitelné či s dostupnými zdrojovými kódy. Dva důležité systémy pro tuto práci jsou shrnuty níže.

1.4.1 Lucene

Prvním zmíněným není komplexní systém, nýbrž opensource knihovna pro fulltextové vyhledávání nad naindexovanými daty. Zmíněna je zde především kvůli své široké komunitě a mnoha komerčním využitím. Přímé použití knihovny není doporučeno nezkušeným vývojářům, jelikož se jedná o rozsáhlý a poměrně komplikovaný projekt. Vhodnějším řešením je nalezení systému, který je na této knihovně postaven. Mezi tyto systémy patří především Elastic Search[1], Apache Solr[2], Apache Nutch[3] či Compass[4].

[1] https://www.elastic.co/products/elasticsearch
[2] http://lucene.apache.org/solr/
[3] http://nutch.apache.org/
[4] http://www.compass-project.org/

1.4.2 Elastic Search

Systém Elastic Seach je vyvíjen společností Elastic[5]. Jedná se o společnost zaměřenou na shromažďování, ukládání a vyhledávání nad rozsáhlými daty.

Popis systému

Elastic Search je fulltextový vyhledávač, ke kterému jsou dostupné zdrojové kódy, tudíž se jedná o opensource, konkrétně je vydáván pod licencí Apache, která požaduje zachování původního autorství, avšak je možné jej upravovat, distribuovat a dále používat podle vlastního uvážení [5]. Jedná o systém disponující vysokou dostupností, rychlostí a škálovatelností. Systém je vytvořen v programovacím jazyce Java s využitím knihovny Lucene[6] 1.4.1. Elastic Search je jeden z nejvíce rozšířených systémů svého typu.

Vlastnosti systému

Elastic Search je rozšířený díky své variabilitě a široké možnosti použití. To je umožněno díky propracovanému API, pomocí kterého je možné přistupovat k drtivé většině funkcí knihovny Lucene. API je postaveno na architektuře REST (Representational State Transfer) a umožňuje komunikaci se systémem pomocí zasílání JSON (JavaScript Object Notation) zpráv HTTP komunikací. Mimo samotné API lze nalézt spoustu knihoven obsluhující právě Elastic Search.

Dalším důležitým aspektem je jeho rychlost zpracovávat data v reálném čase. Lze tedy aplikovat různé filtry a omezení, které modifikují výslednou množinu odpovědí a tyto výsledky jsou téměř okamžitě poskytnuty uživateli.

Elastic Search funguje jako propracovaný distribuovaný systém s částečně automatickým monitorováním serverů, na kterých je spuštěn index. Toto monitorování umožňuje například snadné přidání události po dosažení určitého zatížení serveru. To lze využít pro dosažení rovnoměrného zatížení všech uzlů. Podobně lze využít monitoring v případě, že některý z uzlů nepracuje správně nebo je v chybovém stavu.

Tento software oplývá mimo jiné i primitivní umělou inteligencí, což umožňuje efektivní nalezení výsledku podle kontextu dokumentu, geografické polohy či data. Lze jej také využít pro napovídání uživateli s dotazem, automatické vyplňování webových formulářů nebo opravovat vstupní dotazy od uživatele. Samozřejmostí je pak vícejazyčné vyhledávání [1].

V *Elastic Search* lze indexovat i sémanticky obohacená data, problém je však s jejich dotazováním, nicméně existuje několik rozšíření, které tento problém řeší, případně lze systém doplnit vlastním kódem.

1.4.3 MG4J

Druhým zde zmíněným systémem je MG4J (Managing Gigabytes For Java). Tento projekt není tolik rozšířený jako Elastic Search 1.4.2, avšak je pro tuto práci klíčový. Tento software je vyvíjen pod licencí *GNU Lesser General Public License*[7].

[5] http://www.elastic.co
[6] http://lucene.apache.org/
[7] http://www.gnu.org/copyleft/lesser.html

Popis systému

Systém je vyvíjen na univerzitě Università degli Studi di Milano[8] profesorem Sebastiano Vigna. Celý systém je implementován v jazyce Java, jak název napovídá. Jedná se o open-source aplikaci, která je vydávána pod licencí GNU Lesser General Public License[9]. Jedná se o velmi rozsáhlý projekt, avšak s podstatně menší uživatelskou základnou, než výše zmíněný Elastic Search. Systém je stále vyvíjen (poslední update 12. dubna 2016). Největší komunitu lze nalézt na Google Groups[10]. MG4J je dostupný v Maven repozitáři[11] v několika variacích (mg4j / mg4j-big) a různých verzích. Součástí balíků je i poměrně rozsáhlý manuál a javadocs nápověda. Za nevýhodu tohoto konceptu považuji absenci Git repozitáře.

Vlastnosti systému

- **Indexace** – Stavebním kamenem MG4J je výkonné indexování dat, konkrétně analýza a zpracování rozsáhlých textových kolekcí. Index generuje krátké pasáže textu, které jsou snadno čitelné a lze je využít pro zběžnou kontrolu, zda jsou data správně naindexována.

- **Efektivita** – Systém je vytvořen tak, aby čas potřebný k nalezení relevantního výsledku byl ve všech případech co nejnižší, tudíž je vhodné systém použít jak pro malé kolekce dat, tak i pro stovky milionů dokumentů.

- **Množina výsledků** – MG4J pro každý dotaz vyhodnotí systém odpovídající dokumenty a do hlavního vlákna programu vrátí speciální množinu obsahující tyto výsledky. Díky takto získaným výsledkům je možná implementace operátorů, pomocí nichž lze výsledky ještě více zpřesnit.

- **Výrazové operátory** – Výrazové operátory jsou unární či binární operátory, které lze použít v těle dotazu. Takovými operátory jsme schopni specifikovat různá omezení a dodatky v dotazu. Příkladem je specifikace omezení vzdálenosti mezi jednotlivými hledanými výrazy či označení výrazů identifikačními čísly, na které následně aplikujeme nějaké podmínky. Více o tomto tématu je zmíněno v kapitole X.

- **Flexibilita** – MG4J poskytuje určitou flexibilitu při vytváření indexů. Uživatel si může zvolit, zda je pro něj výhodnější index, který je velmi efektivní, avšak zabírá mnoho paměti, či zvolit cestu úsporného indexu za cenu méně efektivního vyhledávání. Tyto vlastnosti lze ovlivnit pomocí předem definovaných kódů, které reprezentují poměr velikost/efektivita.

- **Otevřenost systému** – Systém poskytuje množství rozhraní, které usnadňují definování vlastní syntaxe dotazů či přizpůsobení datové reprezentace systému. Další výhodou je možnost ovlivnění jednotlivých kroků procesu indexace, tudíž je možné nahradit interní části MG4J částmi vlastními – například parseru, dotazovacího systému nebo vložení dalšího prvku do procesu indexace.

- **Distribuovanost** – Celý systém umožňuje efektivní rozložení indexu mezi několik uzlů, stejně jako v případě Elastic Search. Výhodou toho systému je však možnost

[8] http://www.unimi.it/
[9] http://www.gnu.org/copyleft/lesser.html
[10] https://groups.google.com/forum/#!forum/mg4j
[11] http://search.maven.org/

rozdělit i celé dokumenty bez ztráty jejich celistvosti. Jednotlivé uzly mohou být spuštěny v několika procesech, vláknech nebo na různých serverech, což je pro tuto práci velmi důležitý fakt, jelikož data, která jsou zpracovávána máme rozdělena na 33 serverů 5.1.

Kapitola 2

Dotazování dat

Tato kapitola popisuje nejčastější přístupy k dotazování strukturovaných/částečně strukturovaných textových kolekcí. Jednotlivé způsoby jsou uváděny na existujících, populárních řešení.

Z hlediska zaměření lze vyhledávání v dokumentech rozdělit na dva typy:

- *Dokumentově orientované* – Primárně sloužící pro širokou veřejnost, příkladem je Google či knihovna *Lucene*. Výsledkem je vyhledaný dokument.
- *Korpusově orientované* – Primárně sloužící pro výzkumnou činnost či jazykovědní disciplíny. Vyhledávání probíhá nad korpusy a výsledkem jsou často věty nebo fráze.

2.1 Google Advanced Search

Nejpopulárnějším vyhledávacím nástrojem je beze sporu Google. Google je uživateli používán jako obyčejný fulltextový vyhledávač, nicméně jeho možnosti jsou mnohem větší. Formát, který Google zpracovává, je krátce popsán v této kapitole.

Dotaz může být tvořen textem doplněným o symboly, jednoduché i složité operátory, modifikátory URL či vlastním vzorem [9].

2.1.1 Symboly

- $*$ – Symbol hvězdičky funguje jako "divoká karta", je používána pro nahrazení jakékoliv sekvence znaků
- $-$ – Minus slouží pro označení výrazu, který se má z vyhledávání vyřadit
- | – Tento oprátor funguje stejně jako textové *OR*
- .. – Dvě tečky mezi čísly reprezentují rozsah mezi nimi
- () – Závorkování ve vyhledávání modifikuje pořadí vyhodnocení

2.1.2 Operátory

Operátorů pro použití v *Google Advanced Search* je mnoho, proto jsou zde zmíněny jen některé.

Mimo klasické *AND* a *OR* operátory lze pracovat s následujícími

- *filetype:* – definování typu hledaného souboru (např jen `pdf`.
- *site:* – vyhledávání jen na definované stránce
- *related:* – hledání příbuzných stránek
- *cache:* – vyhledávání podle navštívených stránek či zaslaných dotazů

Mezi pokročilé operátory patří operátory popisující, ve které části stránky hledat, například `allintitle: Google Advanced` najde jen stránky, které mají v *title* obsažena uvedená slova.

2.1.3 Modifikátory URL

Modifikátory URL slouží pro definování typu stránky, kterou hledáme. Například přidáním `&tbs=blg` do URL získáme pouze výsledky obsažené pouze na stránkách typu blog.

Google mimo jiné podporuje i možnost sémantického vyhledávání, avšak stále na experimentální úrovni.

2.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) je sémantický dotazovací jazyk, pracující primárně s daty uchovávanými ve formátu *RDF* [4].

2.2.1 Jazyk

Jazyk pracuje s trojicemi z formátu *RDF*, kde jednotlivá pole reprezentují subjekt – predikát – objekt. Mezi základní typy pro čtení dat z databáze slouží `SELECT`, `CONSTRUCT`, `ASK` a `DESCRIBE`. Dále lze použít klíčová slova stejná jako v *SQL*, a to například `WHERE`, `FROM`, `DISTINCT LIMIT` atd.

- `SELECT` – vrací data ve formě tabulky
- `CONSTRUCT` – vrací data ve formě *RDF* formátu
- `ASK` – vrací `TRUE` nebo `FALSE`
- `DESCRIBE` – vrací data ve formátu *RDF*, která popisují výsledek

Tento jazyk je vhodný pro dotazování nad daty obohacenými o pojmenované entity díky své struktuře a variabilitě ve vytváření dotazů. Pro tuto práci je relevantním příkladem dotazování *DBpedie*, jelikož *DBpedia* poskytuje jeden z největším endpointů pro *SPARQL* [1]. V odkazu v poznámce pod čarou lze vyzkoušet různé dotazy pro získání dat z Wikipedie.

2.3 Dotazování MG4J

MG4J využívá svůj vlastní jazyk pro dotazování dat. Tento jazyk je velmi podobný ostatním jazyků pro dotazování částečně strukturovaných dat. Vstupem programu pro tuto práci je právě dotaz v popsaném formátu

[1] `http://dbpedia.org/sparql`

2.3.1 Operátory

- `AND` – standardní operace “a současně”.
- `OR` – standardní operace “nebo”.
- `NOT` – vyloučení výrazu z vyhledávání
- `"phrase"` – hledání fráze, výrazy v uvozovkách jsou hledány v daném pořadí
- `~` – znak tilda omezuje vzdálenost mezi slovy. Například `(George Washington)~3` vyhledá dokumenty, kde se vyskytují daná dvě slova do vzdálenosti tří slov od sebe
- `<` – použitím tohoto operátoru mezí více slovy definujeme pořadí, v jakém má být vyhledáváno
- `*` – Symbol hvězdičky funguje jako “divoká karta”, je používána pro nahrazení jakékoliv sekvence znaků na konci výrazu
- `( )` – Závorkování ve vyhledávání modifikuje pořadí vyhodnocení
- `:` – znak výraz před dvojtečkou definuje ve kterém indexu vyhledávat
- `..` – Dvě tečky mezi čísly reprezentují rozsah mezi nimi
- `^` – Tímto znakem lze specifikovat, jakého typu má být hledaný výraz. `Washington ^ nertag:(person)` vyhledá výraz “Washington” pouze v kontextu jména osoby

  Jednotlivé operátory lze kombinovat a dosáhnout tak velmi efektivního dotazu, pomocí kterého lze získat data v sémantickém kontextu tak, jak je požadoval uživatel.

  Mnou implementovaný systém využívá tohoto jazyka, tudíž jako příklady dotazu lze nahlédnout do kapitoly 5.

Kapitola 3

Návrh aplikace

V této kapitole jsou shrnuty všechny nástroje, které byly použity při implementaci, popsány jednotlivé technologie a jejich korespondence s výsledným systémem. Dále jsou zde popsány výhody či nevýhody daných technologií, jejich účel, proč byly použity a také problémy spojené s implementací.

3.1 Úvod

Cílem programové části této práce bylo vytvořit nástroj s rozhraním v příkazovém řádku, pomocí kterého je možné zasílat dotazy na běžící indexy. Indexy obsahují sémanticky obohacená textová data, která byla získána z anglické verze Wikipedie. Systém tak musí umožňovat nejen vyhledávání podle klíčových slov, ale především pomocí sémantických anotací daných slov. Dále systém umožňuje nastavení různých omezení pro množinu odpovědí a především flexibilitu ve volbě podoby výsledku či zvýraznění hledaných slov v dokumentu. Tento systém byl vytvořen na základě existujícího řešení, konkrétně systému MG4J popsaného v kapitole 1.4.3.

3.2 Vývojové prostředí

Jelikož se jedná o aplikaci, která ke správnému běhu potřebuje dostatečný výpočetní výkon a síťově přístupné indexy, je v této části práce zmínka o prostředí a postupech, jak byl systém vyvíjen.

3.2.1 Servery

Jelikož se veškerá indexovaná data nachází na školních serverech, přístup k nim je možný pouze ze serveru ve stejné síti (z bezpečnostních důvodů). To je jeden z důvodů, proč je téměř po celou dobu vývoje využíván server Athena1 [1] s operačním systémem Ubuntu 14.04.4 LTS[2]. Jako pracovní složka byl využit adresář projektu uložený na datovém serveru Minerva1[3]. Přístup k ostatním serverům nebyl během vývoje povolen, tudíž lze za hlavní pracovní stroj prohlásit server Athena1 s dostatečným výpočetním výkonem pro potřeby aplikace a osobní laptop s operačním systémem Fedora 22[4].

[1] athena1.fit.vutbr.cz
[2] http://releases.ubuntu.com/14.04/
[3] minerva1.fit.vutbr.cz
[4] https://www.getfedora.org/

3.2.2 Psaní zdrojových textů

Jak již bylo zmíněno, jedná se o aplikaci psanou v programovacím jazyce Java. Jelikož kompilace, sestavení a spuštění probíhalo na vzdáleném serveru, bylo výhodné psát samotný zdrojový text přímo na serveru pro rychlé otestování funkčnosti. Pro menší úpravy k tomu nejlépe slouží oblíbený textový editor *vim*, který oplývá řadou příkazů, klávesových zkratek a mnoha rozšířeními [14]. Pro psaní vysokoúrovňového kódu však není příliš vhodný, alternativou je použití IDE (Integrated Development Environment) jako Eclipse[5] v našem případě. Problém však nastává při přenosu dat na server, protože Remote Debugging[6] není na školních serverech přístupný a manuální upload souborů je pracný. Zvolil jsem proto využití nástroje *Incron*.

Incron[7] je nástroj velmi podobný nástroji *cron*, což je démon, který v definované časy spouští procesy nebo příkazy[11]. *Incron* se liší v tom, že definované příkazy spouští následkem nějaké události, která se odehrála na disku v dané lokalitě. Pro tuto práci jsem tedy využil sledování lokálního adresáře s projektem na veškeré modifikace obsažených souborů v adresáři. V případě, že byl změněn některý ze zdrojových kódů v prostředí Eclipse, *incron* provedl příkaz pro přenesení souborů na datový server Minerva1 do adresáře projektu, kde již bylo možné zdrojové texty přeložit a otestovat. Stejný postup jsem využil pro usnadnění práce s konfiguračními XML soubory.

3.3 Skriptovací a programovací jazyky

Během vytváření aplikace bylo využito možností dvou skriptovacích jazyků, a to *Bash* a *Python*. I přes jednoduchý zápis jazyka *Python* byl nakonec jazyk *Bash* vyhodnocen jako užitečnější, jelikož byl využit pouze jako podpůrný nástroj pro operace, které se netýkají samotného běhu programu. Jazyk *Python* zde tedy hraje pouze roli jazyka použitého pro shromáždění a porovnání statistik programu.

3.3.1 Bash

Pro usnadnění cesty k výsledku byl využit skriptovací jazyk Bash, což je shell využívaný k interpretaci příkazů pro komunikaci mezi uživatelem a operačním systémem[12]. Tento jazyk byl v průběhu vývoje využíván téměř pro každou operaci při testování programu. Velkou výhodou *Bashe* je jeho rychlost, což umožňuje efektivní psaní skriptů pro automatizaci některých operací nebo spouštění více příkazů v dávkách.

3.3.2 Java

Jak již bylo zmíněno výše, samotná aplikace je vytvořena v programovacím jazyce *Java*.

Jazyk Java je objektově orientovaný programovací jazyk, který byl nejdříve vyvíjen společností Sun Microsystems, později společností Oracle[8]. Jedná se o jeden z nejrozšířenějších programovacích jazyků dnešní doby, především kvůli své široké přenositelnosti a robustnosti [15].

[5] https://www.eclipse.org/

[6] http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Fconcepts%2Fcremdbug.htm

[7] http://www.inotify.aiken.cz/?section=incron&page=about&lang=en

[8] http://www.oracle.com/

Pro tento projekt byla Java zvolena jako hlavní programovací jazyk hned z několika důvodů. Prvním je jeho relativně vyšší rychlost vůči jazyku *Python*, který byl při návrhu také brán v úvahu. Dalším z aspektů, který byl brán v potaz, je využití frameworku *Restlet*, o kterém je pojednáno v kapitole 3.4.1.

Dalším z důvodů pro výběr Javy je efektivita využití vícevláknového programování. Jazyk Python je v tomto ohledu také dobrou volbou, nicméně rychlost zpracování pro naše použití je výrazně nižší než v případě Javy.

V neposlední řadě je pro Javu množství modulů, které v klientské části výrazně usnadňují zpracování parametrů, odesílání či zpracovávání dotazů.

3.4 Použité technologie

V této kapitole jsou shrnuty hlavní technologie, které byly využity při implementaci systému. Většinou se jedná o běžně používané technologie pro práci se systémem typu klient/server. Dále jsou shrnuty serializační formáty pro reprezentaci dat či využité knihovny pro jejich zpracování.

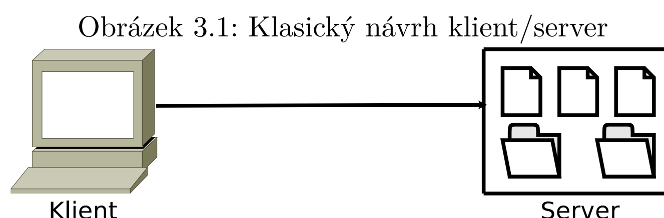
3.4.1 REST

REST (*Representational state transfer* styl architektury, který je využíván při komunikaci v aplikacích klient/server. Tato technologie má především výborné výsledky při práci s distribuovanými systémy, což je případ tohoto projektu. *REST* je využíván především pro vytváření, čtení, editování či smazání informací na serveru pomocí HTTP volání [6]. Jelikož systém implementovaný pro tuto práci komunikuje se serverem pomocí této architektury, jsou v této kapitole podrobněji rozebrány její vlastnosti. Mezi základní vlastnosti architektury *REST* patří následující.

Klient/Server

Klasický návrh klient/server architektury je stavebním kamenem technologie *REST*, jelikož umožňuje efektivní oddělení dat na serveru od uživatele. Pro klienta je tedy možné vytvořit portabilní prostředí, s implementovanými omezeními tak, aby mohl provádět pouze jemu přidělené operace.

Obrázek 3.1: Klasický návrh klient/server



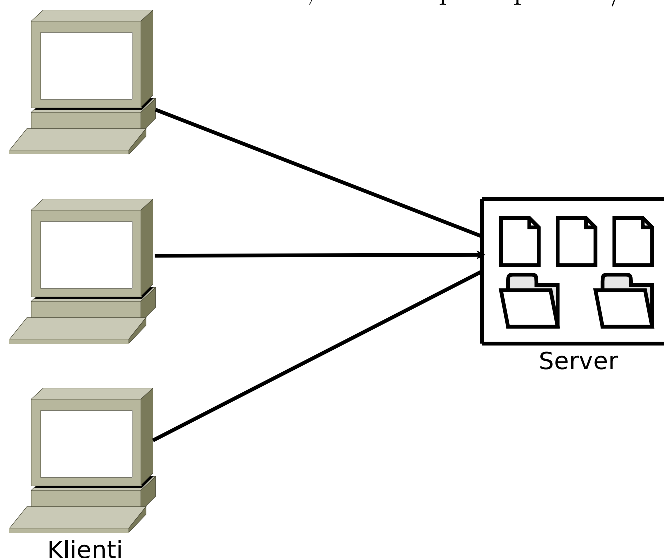
Bezestavovost

Bezestavovost vytváří nutnost, aby každý odeslaný požadavek od uživatele obsahoval všechny informace, které jsou potřebné pro jeho provedení. Nemůže se tedy stát, že by se klient dostal neprávem k datům na serveru, která nejsou žádána v požadavku(requestu). Výsledkem je

výborná konkurence schopnost serveru pro více klientů najednou, aniž by se navzájem ohrozili.

Jedinou nevýhodou této vlastnosti je vyšší zatížení síťového provozu, nicméně se nejedná o žádný extrém.

Obrázek 3.2: Více klientů, stateless přístup klient/server



Cache

Pro dosažení nižšího síťového zatížení lze stejně jako v mnohých síťových architekturách využít *cache*. *Cache* je pojem používaný pro *mezipamět*, což je vyrovnávací paměť uchovávající často používaná data pro určitý proces cache.

Pro použití *cache* v *REST* je nutné na straně serveru specifikovat, které atributy obsažené ve zprávě odpovědi mohou být na straně klienta uloženy do mezipaměti a při dalším ekvivalentním požadavku je použít. Toto označení lze provést buď implicitně či explicitně.

Využitím *cache* lze dosáhnout menšího zatížení serveru, snížení latence či snížení interakce mezi klientem a serverem.

Ačkoliv je použití *cache* velmi výhodné pro služby jako je webový vyhledávač, pro použití v této práci postrádá smysl, jelikož lze předpokládat, že dotaz bude často obměňován a data navrácená od serveru mohou být rozdílná.

Vrstvy architektury

REST využívá architektury několika vrstev, což umožňuje přístup k datům efektivním způsobem. Pomocí těchto vrstev lze oddělit jednotlivé části architektury podle frekvence použití dat či zatížení serveru. Důležitou vrstvou je často *load-balancing*, což je služba, která umožňuje rovnoměrné rozdělení zátěže mezi všechny uzly serveru a vyhnout se tak zahlcení [17].

Implementace REST

Pro tento projekt byla pro implementaci *REST* komunikace využita knihovna Javy *Restlet*. *Restlet* je open-source knihovna použitelná pro programování jak klientské tak serverové části systému. Jedná se o intuitivní framework pro vytvoření API na serveru a na klientské části vytvoření HTTP and HTTPS, SMTP, XML, JSON, Atom nebo WADL požadavku. Tento požadavek pro tuto práci obsahuje například hodnoty znázorněné v tabulce 3.1.

Tabulka 3.1: Příklad zprávy generované *Restlet* frameworkem

Element	**Hodnota**
method	POST
date	Fri, 06 May 2016 20:12:44 GMT
content-type	application/json; charset=UTF-8
accept	*/*
user-agent	Restlet-Framework/2.3.6
cache-control	no-cache
pragma	no-cache
host	athena1.fit.vutbr.cz:12000

Jak je z příkladu patrné, klient se serverem komunikuje prostřednictvím HTTP požadavků se serializovanými daty formátem JSON.

3.4.2 Serializační formáty

Serializací rozumíme obecně proces, při kterém je objekt převeden na řetězec reprezentující původní podobu objektu [16]. Serializací lze dosáhnout uložení a opětovné obnovení objektu do/z perzistentní databáze či souboru. V případě síťové komunikace jsou serializační formáty používány pro přenesení strukturovaných dat mezi zařízeními. Mezi nejrozšířenější lze zařadit XML, JSON, YAML či Protobuf vyvíjený společností Google.

Tato část textu shrnuje dva serializační formáty, které jsou využity pro tuto práci.

JSON

JSON (*JavaScript Object Notation*) je platformě nezávislý serializační formát, který je populární především kvůli svému jednoduchému zápisu a snadnou čitelností zdrojového kódu. Vstupem je datová struktura, výstupem je řetězec. *JSON* byl původně navržen jako podmnožina jazyka *JavaScript*, nicméně díky své flexibilitě se rozšířil do mnoha jiných jazyků.

JSON definuje množinu pravidel, která reprezentují strukturovaná data. *JSON* pracuje s datovými typy, které jsou popsány v následujícím seznamu.

- *Objekt* – JSONObject, množina dvojic název:hodnota, které jsou odděleny čárkou. Objekt je uzavřen ve složených závorkách. Na pořadí dvojic nezáleží.

- *Pole* – JSONArray, množina hodnot oddělených čárkou. Pole je uzavřeno v hranatých závorkách. Záleží na pořadí hodnot.

- *Číslo* – JSONNumber, zápis reálného čísla.

- *Řetězec* – JSONString, reprezentace libovolně dlouhého řetězce. Řetězec je uzavřen v uvozovkách. Lze použít prázdný řetězec "".
- *Boolean* – JSONBoolean – binární typ nabývajících hodnot *true* nebo *false*.
- *Null* – JSONNull – literál reprezentující *null*

Obrázek 3.3: Ukázka kódu v JSON, *zdroj:* www.zdrojak.cz

```
{"results":[
  "div":{
    "id":"twitter",
     "a":{
       "href":"http://twitter.com/zdrojak",
       "text":"Zdrojak"
     }
  }
]}
```

V této práci je serializační formát *JSON* stěžejní pro komunikaci mezi klientem a serverem. Klient zasílá dotazy v následující podobě.

Obrázek 3.4: Formát zprávy zaslané klientem

```
{
   "offset":"0",
   "query":"(lemma:(be){{lemma->token}})",
   "constraint":""
}
```

V ukázce 3.4.2 lze vidět tři hlavní položky dotazu, které jsou klientem odeslány na server. Nejdůležitější ze všech je položka *query*, která obsahuje celé znění dotazu bez dalších specifikovaných omezení. Tato omezení jsou uloženy v položce *constraint*. Poslední z položek je záznam *offset*, který definuje, kde hledaná data v indexu začínají. *Offset* 0 znamená hledání v celém indexu od začátku.

Takto zaslanou zprávu server deserializuje a provede požadované operace pro nalezení dat v indexu. Klient od serveru obdrží výsledek ve formě *JSON* zprávy. Tato zpráva má následující podobu.

Odpověď od serveru může být jakkoliv dlouhá, její délka závisí na počtu nalezených výsledků. První položkou odpovědi je *total*, což je hodnota udávající celkovou velikost odpovědi uvedenou v počtech znaků. Druhou položkou je *hostname*, což udává síťový název dotazovaného uzlu indexu. Záznam *documents* potom udává celkový počet naindexovaných dokumentu na daném uzlu. *Time* reprezentuje dobu v milisekundách od zpracování dotazu od klienta po odeslání zprávy s výsledky. *Error* popisuje, zda bylo při provádění dotazu nalezena chyba či nikoliv. Pro zpracování odpovědi je nejdůležitější pole hodnot *results*.

Obrázek 3.5: Formát zprávy zaslané serverem


```
{
   "total":"151846",
   "hostname":"knot31.fit.vutbr.cz:12000",
   "documents":"160199",
   "time":"78728564",
   "error":"false",
   "results":[
      {
         "data":"<block>Justices of the Rhode Island
                  Supreme Court This <a href=\...<\/block>",
         "doc":"0",
         "thread":"0",
         "title":"List of Justices of the Rhode Island
                   Supreme Court",
         "uri":"https%3A%2F%2Fen.wikipedia.org%2F
                wiki%2FList_of_Justices_of_the_
                Rhode_Island_Supreme_Court"
      },
      {
         "data":"<block>(<a href=\"https://google.com\"
                s...<\/block>",
         "doc":"11",
         "thread":"0",
         "title":"Maxwellton, Saskatchewan",
         "uri":"https%3A%2F%2Fen.wikipedia.org%2Fwiki%
                2FMaxwellton%2C_Saskatchewan"
      }
   ]
}
```


Results obsahuje neomezený počet položek reprezentující jednotlivé nálezy. Jednotlivé položky jsou složeny z atributů *data*, *doc*, *thread*, *title* a *uri*.

Řetězec uložený v položce *data* obsahuje textovou podobu dokumentu, ve kterém byl nalezen výskyt dotazu. Začátek i konec datové části odpovědi je uzavřen v párovém elementu *block*. Položka *doc* popisuje číslo dokumentu na daném uzlu a *thread* vlákno na kterém bylo hledání prováděno. *Title* slouží pro přenos informace o nadpisu dokumentu a *URI* obsahuje jeho cestu.

XML

XML (Extensible Markup Language) je značkovací jazyk, který je pro tuto práci důležitý především kvůli svému přehlednému kódu a možnosti snadné editace [10].

Jedná o jazyk poskytující velkou flexibilitu a mnoho možností. Popis jazyka není pro tuto práci důležitý kromě několika základních pravidel.

- Dokument musí mít alespoň jeden kořenový uzel
- Elementy musí být uzavřeny počáteční a koncovou značkou. Značka může nést libovolné jméno.
- Elementy mohou být zanořeny
- Každý element kromě kořenového musí být obsahem nadřazeného elementu.

Obrázek 3.6: Ukázka souboru *XML*, *zdroj:* *www.w3schools.com*

```
<?xml version="1.0" encoding="UTF-8"?>
  <note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Forget me this weekend!</body>
  </note>
```

Programová část této práce využívá jazyk *XML* pro zápis konfiguračního souboru. Konfigurační soubor obsahuje pouze jednoduché elementy bez atributů podobně jako ukázka v 3.4.2.

Kapitola 4

Implementace

Tato část práce popisuje způsoby, jakými byly použity technologie popsané v kapitole 3.4. Je zde popsáno jakým způsobem jsou získávána data od uživatele a jak jsou zpracovávána. Dále je zde úvod do problému paralelního zpracování dat z distribuovaného indexu a způsoby, jakými jsou výsledky předány uživateli.

4.1 Vstupní data

Pro správnou funkci celého systému je nutné správně definovat servery, které mají být dotazovány, celé znění dotazu a případné další omezení na filtrování výsledků. Jelikož se jedná o program s konzolovým prostředím, předání parametrů se provádí buď pomocí přepínačů při spuštění aplikace nebo pomocí konfiguračního souboru.

4.1.1 Přepínače

Zpracování parametrů definovaných pomocí přepínačů obsluhuje modul *JSAP (Java Simple Argument Parser)*. Tento "parser" umožňuje mimo jiné nastavení datového typu, výchozí hodnoty, krátké či dlouhé varianty dotazu či vytvoření krátké nápovědy k požadovaným argumentům programu. Vstupní parametry jsou zadávány standardním způsobem.

4.1.2 Konfigurační soubor

Další z možností, jak přizpůsobit nebo nastavit aplikaci je konfigurační soubor. Konfigurační soubor je zapsán v jazyce *XML*, popsáném v kapitole 3.4.2. Tento soubor je vhodný použít pro definování specifičtějších pravidel, než je tomu v případě řádkových přepínačů.

Konfigurační soubor je zpracováván jako strom *DOM (Document Object Model)* elementů. V souboru je tedy možné definovat vlastní značky bez nutnosti přidávat další zdrojový text pro jejich čtení.

V případě, že konfigurační soubor nebyl načten z důvodu nenalezení souboru či kvůli přístupovým právům, parametry nedefinované pomocí přepínačů jsou nastaveny na výchozí hodnoty.

4.1.3 Vnitřní reprezentace vstupu

Oba výše zmíněné způsoby definování parametrů lze kombinovat, přičemž parametry předané pomocí příkazového řádku mají vyšší prioritu. To je umožněno díky vnitřní reprezentaci,

která je obsluhována třídou *Parameters*. Prvním krokem je načtení a zpracování konfiguračního souboru, pokud jej bylo možné načíst. Všechny takto definované parametry jsou uloženy do objektu typu *Map<String, Object>*. *Map* je rozhraní umožnující mapování klíče na hodnotu, přičemž ke každému klíči přísluší nanejvýš jedna hodnota. Pro klíče je použit datový typ *String* a jako hodnota typ *Object*. Použití tohoto typu hodnoty umožňuje přiřazení jakéhokoliv datového typu odvozeného od třídy *Object*, což jsou v Javě všechny primitivní typy. Taková reprezentace dat potom umožňuje snadný přístup k hodnotám parametrů pomocí veřejných metod třídy *Parameters*.

Do zmíněného objektu *Map<String, Object>* jsou tedy první načteny všechny parametry z konfiguračního souboru a posléze, díky vlastnosti mapování klíče na jedinou hodnotu, jsou nastaveny parametry z příkazové řádky, čímž dojde k přepsání některých hodnot načtených z konfiguračního souboru.

4.2 Logování a chybové hlášky

Klientská část aplikace poskytuje vytváření logů o běhu aplikace. Logovací soubory jsou vytvářeny standardním způsobem pomocí modulu *Logger*. Do kterého souboru se má log zapisovat, lze zvolit pomocí konfiguračního souboru. Jedinou nutností jsou správně nastavená práva pro zápis pro cílovou složku.

Log je textový soubor formátovaný pomocí třídy *SimpleFormatter*, tudíž je snadno čitelný ve formátu znázorněném v 4.2.

Obrázek 4.1: Formát logovacího souboru

```
MONTH DAY, YEAR TIME PACKAGE.CLASS METHOD
```

message

Začátek logu informuje uživatele o stavu jednotlivých parametrů, se kterými byl program spuštěn. V dalších částech logu lze nalézt informační zprávy o právě prováděné operaci či chybách během vykonávání. Většina chyb je reprezentována pomocí *StackTrace*, jelikož má pro uživatele nejvíce informativní charakter a nalezení chyby značně zjednoduší.

Vypisování chyb na `STDOUT` není povoleno kvůli případům, kdy dojde k chybě během vypisování výsledků, což by negativně ovlivnilo další zpracovávání.

4.3 Nastavení a přístup k serverům

Povinným parametrem programu je definovat, jaké servery dotazovat. Jelikož se jedná o distribuovaný systém, je vhodné zadat více než jeden server. Kvůli této skutečnosti musí být servery zapsány do souboru, který je programu předán při spuštění. Programová část potom v očekává jeden server na jeden řádek následovaný dvojtečkou a číslem portu, na kterém daný uzel komunikuje s klientem. Počet specifikovaných serverů není omezen.

Uživateli je umožněno nastavit, jestli se mají dotazovat všechny servery, nebo pouze servery komunikující na specifikovaném portu. To značně zvýší rychlost celého zpracování, jelikož se často stává, že ne všechny uzly jsou zrovna dostupné. Klient potom musí počkat na timeout od serveru.

4.4 Dotazování serverů

Jelikož na dotazovaný index aktuálně běží na 32 uzlech, je výhodné, aby tyto uzly byly dotazovány paralelně. To je programově realizováno pomocí vláken, kdy je každý dotaz na jeden server spouštěn ve vlastním vlákně. Výhodou toho přístupu je především rychlost obdržení všech výsledků. Nevýhodou paralelního zpracování odpovědí od serveru je nekonzistentní pořadí doručených odpovědí. Jednotlivé odpovědi jsou vůči sobě výlučné, tudíž jedna do druhé nezasahuje, ale může se stát, že nebudou ve stejném pořadí jako byly servery zapsány do konfiguračního souboru.

4.5 Modifikace dotazu

Dotaz odeslaný na server lze zaslat buď v originálním znění, nebo lze povolit modifikaci dotazu, která přemapuje dotazy do stejného indexu. S touto modifikací tedy není třeba psát dotaz jak je uvedený v 4.5, ale stačí zkrácená podoba dotazu uvedená v 4.5.

Obrázek 4.2: Celé znění dotazu

```
"(nertag:personnertag-> token) killed"
```

Obrázek 4.3: Zkrácené znění dotazu

```
"nertag:person killed"
```

4.6 Personalizace odpovědi

Dotazovací systém umožňuje nastavení mnoha podob, jakým způsobem se mají vypisovat výsledky. Pro testování lze zapnout zobrazení neupravené zprávy od serveru, což je zobrazení celého výsledku ve svém originálním znění včetně tagů, definující sémantický význam jednotlivých slov. V případě, že je tato funkce vypnuta a žádné další modifikátory nejsou nastaveny, bude zobrazen pouze text odpovědi, bez jakýchkoliv zvýraznění či zobrazení sémantických anotací. Pro zobrazení sémantických anotací je nutné upravení konfiguračního souboru, kde lze nastavit, které "vlastnosti" výrazu mají být zobrazeny společně s původním textem. Takto obohacená data jsou zobrazena za daným výrazem uzavřena v hranatých závorkách a uvozovkách. Příkladem vhodného použití je povolení zobrazení entity *person* a sémantické anotace *data-nationality*. Výsledkem pak bude text, ve kterém budou všechny výrazy obsahující jméno osoby obohaceny o její národnost.

Důležitým prvkem při zobrazení odpovědi je zvýraznění určitých částí textu. Zvýraznění lze také definovat v konfiguračním souboru. Zde jsou zapsány datové typy entit, které korespondují s atributem *data-nertag*. Seznam těchto typů je uveden v 4.1. Pro zápis více typů lze jednotlivé typy oddělit znakem `|`.

Jelikož je často potřeba výstup z dotazovacího systému zpracovávat programově, je vhodné využít speciální značky pro oddělení hledaných slov v textu. To je také umožněno pomocí konfiguračního souboru, do kterého jsou zapsány počáteční a koncové značky hledaných slov. Tyto značky lze definovat libovolně, podle potřeby dalšího zpracovávání. V případě,

Tabulka 4.1: Seznam tagů, které lze zvýrazňovat

person	artist	location	artwork
event	museum	family	group
nationality	date	interval	form
medium	mythology	movement	genre

že jsou tyto značky definovány, veškeré zvýraznění pomocí barev je vypnuto. Barva použitá pro zvýraznění slov může být nastavena na jednu z hodnot uvedených v 4.2.

Tabulka 4.2: Barvy použitelné pro zvýraznění textu

red	green
yellow	blue
magenta	cyan
white	black

4.6.1 Proces zpracování zprávy

Jelikož je surová zpráva s výsledky obdržena v *JSON* formátu, je nutné ji správně zpracovat. Snadné zpracování je umožněno díky modulům přístupných v Javě. Formát zprávy je popsán v příkladě 3.4.2. Z celé zprávy je pro zobrazení výsledků klíčové pole *results*, konkrétně atributy *data*, *uri* a *title*.

Formát obsahu atributu *data* je zapsán v jazyce obdobném *HTML*, uzavřený v párovém elementu `<block>`. Obsah je tvořen především párovými značkami `<a>`, které vyjadřují ve svých parametrech sémantické anotace a ve svém těle samotné slovo nebo množinu slov, ke kterým se anotace vztahují. Pro primitivní zobrazení uživateli je použito regulárních výrazů, které odstraní všechny značky a zůstane tak pouze čistý text. Pokud uživatel zvolil zobrazení anotací, potom jsou jím zvolené atributy *parsovány* z tagů `<a>`. Postup zpracování probíhá v několika krocích, které jsou realizovány třídou `ResultProcessor`, konkrétně metodou `processDataChunk`.

V případě, že se nejedná o testovací výstup *raw*, potom je postup následující. Prvním krokem ve zpracování zprávy je odstranění nedělitelných mezer, tedy značek ` `. Následně, pokud je povoleno zvýrazňování či zobrazení anotací, následuje proces jejich získání. V případě zvýraznění jsou všechny výrazy, které mají být takto označeny, uzavřeny do značek `<strong>vyraz</strong>`. Následuje odstranění všech otevíracích značek `<a>` včetně jejich parametrů. Stejný proces platí pro uzavírací značky `<a>`. Následujícím krokem je proces "obarvení" výstupu, což obnáší nahrazení tagů `<strong>` zvolenými barvami či definovanými obalujícími tagy. Posledním krokem je nahrazení *HTML* entit jejich textovou alternativou. Setkáme se zde s entitami `>`, `<`, `"` a `&alt`, které jsou nahrazený znaky `>`, `<`, ¨ a `&`.

4.7 Postfiltering

Důležitou částí dotazovacího systému je možnost specifikovat filtry, které budou modifikovat výslednou množinu výsledků. Tyto omezení jsou serveru předány pomocí atributu *constraints*. Tyto omezení lze využít v případě, že jsme v dotazu označili token číselným identifikátorem. Označíme-li například token popisující osobu číslem `1` a další osobu číslem

`2`, postfilter v podobě `1.nerid != 2.nerid` nám zajistí zobrazení pouze vět, ve které se vyskytují na daných pozicích dvě různé osoby.

Kapitola 5

Statistiky a experimenty

Tato kapitola zobrazuje dosažené výsledky, jejich vizualizaci a zhodnocení. Experimentování bylo prováděno na školním serveru *athena1*. Na ostatních serverech zmíněných v 5.1 byly naindexovány texty z anglické Wikipedie, nad kterými byly prováděny dotazy. Všechny testy byly prováděny v průběhu několika dní, tudíž je možné, že počet serverů, které právě odpovídaly na dotazy se může lišit.

Tabulka 5.1: Seznam serverů pro experimenty

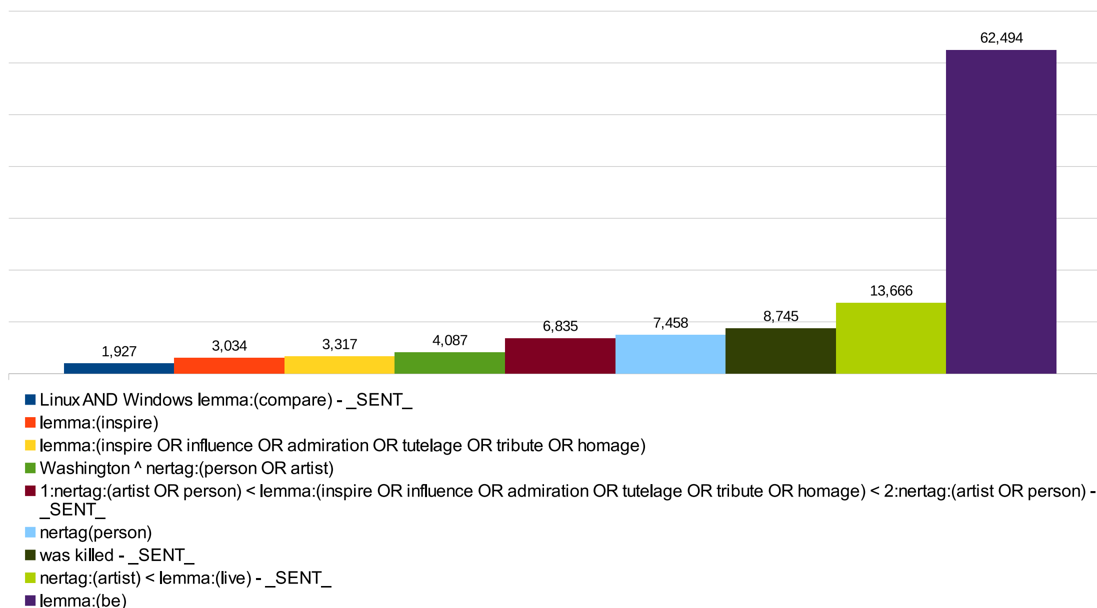
Hostname	**Port**	**Hostname**	**Port**	**Hostname**	**Port**
knot01.fit.vutbr.cz	12000	knot07.fit.vutbr.cz	12000	knot14.fit.vutbr.cz	12000
knot02.fit.vutbr.cz	12000	knot08.fit.vutbr.cz	12000	knot15.fit.vutbr.cz	12000
knot03.fit.vutbr.cz	12000	knot10.fit.vutbr.cz	12000	knot16.fit.vutbr.cz	12000
knot04.fit.vutbr.cz	12000	knot11.fit.vutbr.cz	12000	knot16.fit.vutbr.cz	12000
knot05.fit.vutbr.cz	12000	knot12.fit.vutbr.cz	12000	knot17.fit.vutbr.cz	12000
knot06.fit.vutbr.cz	12000	knot13.fit.vutbr.cz	12000	knot18.fit.vutbr.cz	12000
knot19.fit.vutbr.cz	12000	knot20.fit.vutbr.cz	12000	knot21.fit.vutbr.cz	12000
knot22.fit.vutbr.cz	12000	knot23.fit.vutbr.cz	12000	knot24.fit.vutbr.cz	12000
knot25.fit.vutbr.cz	12000	knot30.fit.vutbr.cz	12000	knot31.fit.vutbr.cz	12000
athena1.fit.vutbr.cz	12000	athena2.fit.vutbr.cz	12000	athena3.fit.vutbr.cz	12000
athena4.fit.vutbr.cz	12000	athena5.fit.vutbr.cz	12000	athena6.fit.vutbr.cz	12000

5.1 Časová odezva

Jedním z aspektů, který je pro vyhledávání v sémanticky anotovaných datech důležitý, je doba, kterou uživatel musí strávit čekáním na odpověď od serveru. Při návrhu a implementaci systému byl důraz kladen na minimalizování této doby. Tato část zobrazuje výsledky pro několik testovacích dotazů. Množina těchto dotazů byla sestavena tak, aby obsahovala různé typy dotazů, či aby počty potenciálních výsledků byly různé.

V 5.1 je přiložen graf zobrazující jednotlivé dotazy a čas ve vteřinách, potřebný pro zpracování. Uvedené hodnoty jsou průměrné a vychází z několika měření. Za uvedený čas je považována doba mezi odesláním dotazu a doručení celé odpovědi od serverů. Uvedený čas nezahrnuje dobu potřebnou pro zobrazení výsledků, jelikož tato doba může být ovlivněna například rychlostí připojení pomocí *ssh*. Data získaná v 5.1 obsahují data obohacená o sémantické informace, takže systém trávil nějaký čas získáním těchto anotací.

Obrázek 5.1: Časy jednotlivých dotazů



Jak je z grafu patrné, jednotlivé časy zpracovávání jsou velmi rozmanité. To je způsobeno hned několika příčinami. Nejdéle trvajícím dotazem je `lemma:(be)`, což je logický způsobeno obrovským počtem nalezených hodnot a jejich předáním uživateli. Čas je tedy ovlivněn především množstvím odpovědí, které server generuje a tudíž vzniká i vyšší síťové zatížení.

Opačným případem je dotaz `1:nertag:(artist OR person) < lemma:(inspire OR influence OR admiration OR tutelage OR tribute OR homage) < 2:nertag:(artist OR person) - _SENT_`. Jedná se o dotaz podstatně složitější než v předchozím případě a počet očekávaných výsledků je řádově menší. Nicméně doba potřebná pro zpracování tohoto dotazu je poměrně velká, takže i přes malý počet nálezů je čas poměrně dlouhý.

Ze zmíněných dvou příkladů lze vyvodit závěr, že čas potřebný pro získání výsledků závisí jak na složitosti dotazu, tak na množství nálezů na daný dotaz.

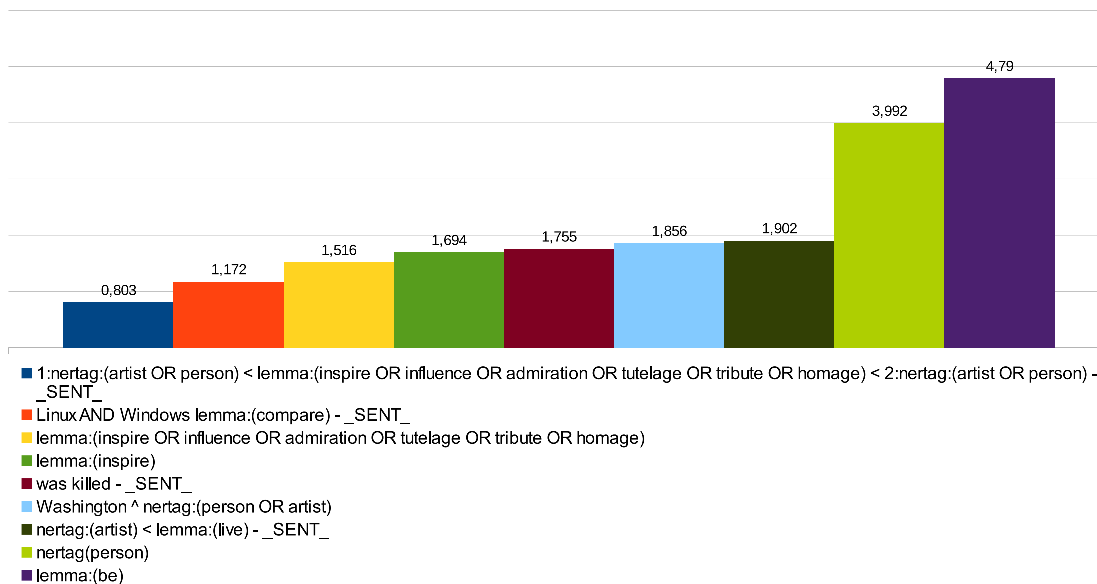
Pro srovnání výkonnosti je v 5.2 znázorněn graf jednotlivých časů zpracování bez dodatečného zvýraznění či získaní sémantických anotací.

Použití regulárních výrazů pro zvýrazňování výrazů je poměrně pomalý proces, a jak graf ukazuje, tak značně negativně ovlivňuje čas potřebný k získání takového výstupu. Na dotazu `lemma:(be)` lze pozorovat, že čas potřebný pro zpracování anotované, zvýrazněné verze je téměř 15krát větší než v případě čisté textové verze. Tento fakt je při práci se systémem nutné brát v potaz.

5.2 Počet výsledků

Testování systému na počet vrácených výsledků je prováděno se stejnou sadou dotazů jako tomu bylo v případě testování časové odezvy. Tento přístup byl zvolen především proto, aby si bylo možné vytvořit představu o korelaci mezi počtem výsledků a časovou odezvou. Počty výsledky v čistém textovém formátu se nijak neliší od počtu výsledků se sémantickými anotace. Graf reprezentující dosažené výsledky je zobrazen v 5.3. Dotazovány byly

Obrázek 5.2: Časy jednotlivých dotazů bez zvýraznění a anotací



všechny servery uvedené v tabulce 5.1. Během dotazování se nepodařilo spojit se serverem *knot05.fit.vutbr.cz* a *knot10.fit.vutbr.cz*, což má za následek nižší počet výsledků než v případě, kdy odpovídají všechny servery. Tento problém je popsán v kapitole 6.2.1.

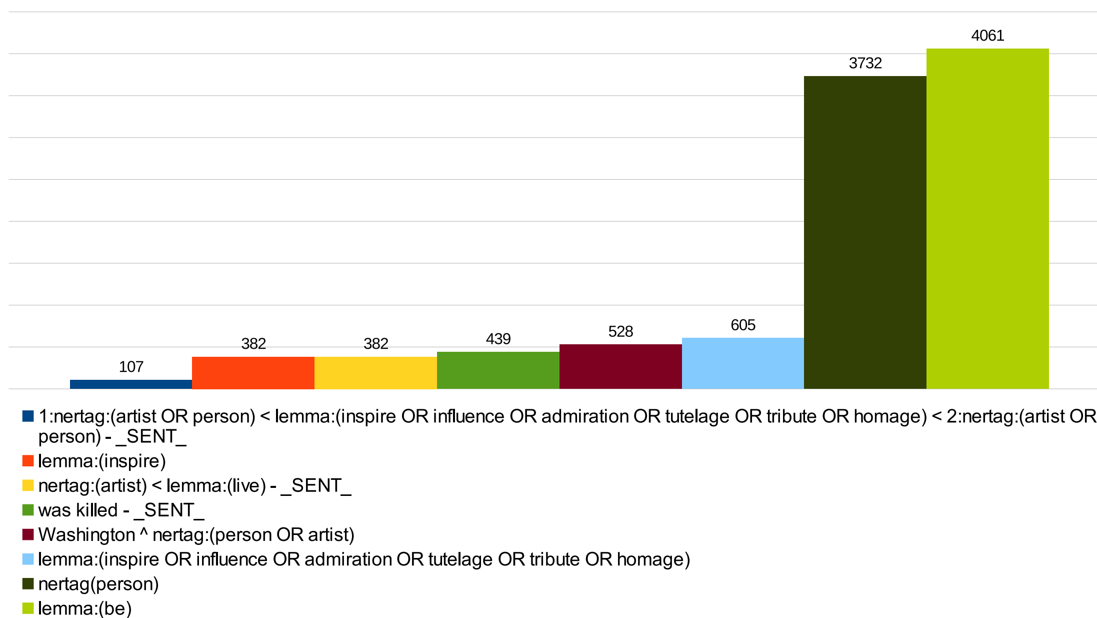
5.3 Porovnání dosavadních výsledků

Dotazovací systém pracuje s daty získanými z anglické verze Wikipedie. Extrahování informací z tohoto zdroje již bylo několikrát prováděno skupinou *KNOT*, tudíž lze dosavadní výsledky porovnat s dříve získanými. Výsledkem dříve prováděných metod jsou klíčová slova zapsaná v textových souborech. Tento systém však funguje podobně jako fulltextový vyhledávač, tudíž výstupem jsou celé věty nebo odstavce. Z tohoto důvodů je poměrně složité porovnávat tyto výsledky s dosavadními. Experimentálně jsem vytvořil několik skriptů, které se snaží o formátování výstupu pro dosažení stejného formátu dat. Tyto testy nemusí být naprosto přesné, avšak výsledek je následující.

Jako referenční soubor pro porovnávání jsem použil sadu textových souborů popisující vztahy ovlivnění mezi umělci. Jako vstup implementovaného systému jsem použil již zmíněný dotaz `1:nertag:(artist OR person) < lemma:(inspire OR influence OR admiration OR tutelage OR tribute OR homage) < 2:nertag:(artist OR person) - _SENT_` a několik dalších dotazů odvozených z výše zmíněného.

Tabulka 5.2 zobrazuje počet dříve získaných stavů a počet stavů získaných systémem pro tuto práci. Tabulka jasně vypovídá o tom, že výsledky jsou velmi slabé, jedná se téměř o 13% z dříve získaných výsledků. Tyto výsledky jsou do značné míry ovlivněny aktuálním stavem serverů a především hodnotícím skriptem, který porovnávání prováděl.

Obrázek 5.3: Počty výsledků na jednotlivé dotazy



Tabulka 5.2: Porovnání výsledků

	Dosavadní výsledky	**Výsledy systému MG4J**
Počet nálezů	2920	368

Kapitola 6

Komplikace, limity a rozšíření

V této kapitole jsou popsány komplikace spojené s implementací, jejich řešení a mimo jiné i limity této práce, které s sebou výsledný program nese. Tyto limity jsou vhodné návrhy na další rozšíření či doplnění funkcionality programu.

6.1 Komplikace při implementaci

6.1.1 Překlad zdrojových textů

Jelikož klientská část vychází z dříve implementovaného systému skupinou *KNOT*, s překladem této části systému nebyl problém. Nicméně problém nastal při pokusu o překlad serverové části, která je postavena na systému *MG4J* 1.4.3. Výsledné binární soubory již byly předpřipraveny a úprava zdrojových kódů nebyla nezbytně nutná, ačkoliv pro řešení některých problémů či pro odhalení chyb bylo vhodné provést drobné změny. Předpis pro správný překlad jsem nebyl schopen dohledat, takže poměrně velké množství času jsem strávil nad způsobem překladu. Tento problém byl však úspěšně vyřešen a nyní je možné serverovou část přeložit.

6.1.2 Spuštění vlastního serveru

Pro testovací účely bylo vhodné vytvořit kopii již běžícího serveru, případně serveru s upravenými zdrojovými texty. Příklad tohoto spuštění je uveden na Wiki stránkách *Corpora processing*. Tento postup se mi nepodařil reprodukovat, nicméně se jednalo pouze o drobné změny v příkazech. Nyní již je postup opraven. Samotné spuštění serverů na jiném portu však neprobíhalo dle mých představ, a to kvůli nedostatečným oprávněním. Démony obsluhující index jsou totiž spouštěny paralelně na všech dostupných serverech. Tento problém jsem však vyřešil spuštěním jediného démona na serveru, kde mám dostatečná oprávnění.

6.1.3 Paměťová omezení

Pro správný překlad klientské a serverové části je nutné stažení velkého množství modulů pro Javu, které je realizováno pomocí *Maven*. Takto vytvořené závislosti jsou uspořádány ve velké stromové struktuře, která přesahuje limity definované na serveru. Tato omezení však nejsou striktní, tudíž se překlad a práce s moduly dala realizovat i přes mnohá upozornění ze strany serveru. Po každém překladu tedy bylo nutné vyčistit adresáře obsahující závislosti a při dalším překladu je znovu stáhnout a nastavit.

6.1.4 Slabá komunita systému *MG4J*

Systém *MG4J* je sice dodáván jako open-source, nicméně vývojářská dokumentace není příliš rozsáhlá. V takovém případě je vhodné výhodné nahlédnout do *git* repozitáře, který bohužel není pro tento systém veřejně dostupný. Komunita využívající služeb *MG4J* není příliš aktivní, takže nalezení některých problémů online je značně komplikované.

6.1.5 Nedostupnost serverů

Bohužel se velmi často stávalo, že některý z démonů, který obsluhuje index na daném uzlu nebyl dostupný. To má za následek negativní vliv na počet získaných odpovědí a také na zaplnění logovacího souboru hláškami informujícími o problému. Také nastaly případy, že server nebyl vůbec dostupný na daném portu.

6.2 Limity práce

6.2.1 Nedoručené výsledky od serveru

Hlavním nedostatkem této práce je fakt, že v případě, kdy uživatel během provádění dotazu přeruší běh klienta, server se stane nedostupným. Tento problém omezuje množství serverů, na které lze přistupovat během testování. Důsledkem je pak nižší počet výsledků a nutnost restartovat servery, které jsou nedostupné. Nižší počet výsledků je důsledkem nedostupnosti démona, který obsluhuje část indexu s daty na nepřístupném serveru.

Tento nedostatek se mi bohužel nepodařilo vyřešit, jistě se však jedná o chybu na straně serveru.

6.3 Možná rozšíření

Systém v aktuálním stavu plní svou funkci, nicméně je zde i prostor pro možná rozšíření. Mezi prioritní lze zařadit prozkoumání serverové části systému a nalézt příčinu, proč se server stane nedostupným při nedokončení celého dotazu. Vyřešením tohoto problému lze dosáhnout podstatně vyššího počtu nálezů a tím i vyšší úspěšnosti při porovnání s dosavadními výsledky.

Prostor pro rozšíření se nabízí i v části modifikace výstupu programu, kde by bylo vhodné vytvořit mód, který vrací pouze klíčová slova k relevantnímu dotazu. Nebylo by potom nutné vytvářet další skripty pro zpracování celých vět.

Jelikož grafické prostředí pro dotazovací systém je již dostupné na školních serverech, pro konzolovou aplikaci by bylo přínosem vytvořit interaktivní mód, který by zpřehlednil a usnadnil práci s konfiguračním souborem a samotné zadávání dotazů, jelikož celý příkaz pro spuštění je často dlouhý a nepřehledný.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a realizovat dotazovací systém, který komunikuje s distribuovaným indexem obsahující sémanticky obohacená data z kopie anglické Wikipedie. Práce se zaměřuje především na práci s pojmenovanými entitami v textu. Klient komunikuje se servery pomocí *HTTP* zpráv, což zajišťuje přenositelnost aplikace, která je také podpořena implementací v programovacím jazyce *Java*. Systém je distribuován jako jediný `jar` archiv obsahující veškeré závislosti.

Jelikož se jedná o obdobu fulltextového vyhledávání, výstupem jsou celé věty nebo části textu, tudíž automatické zpracovávání výstupu není příliš pohodlné. Systém vykazuje dobré výsledky v oblasti časové odezvy, která je ovlivněna několika faktory, nicméně vzhledem k množství dat, které systém zpracovává, lze prohlásit, časová odezva je uspokojivá. Opačným případem je množství odpovědí od serveru, které je klientu doručeno. Ve statistice uvedené v 5.3 lze pozorovat, že počet vrácených výsledků je poměrně malý vůči článkům z Wikipedie. O tom vypovídá i srovnání výsledků s dříve aplikovanými metodami, které je uvedené v tabulce 5.2. Zhodnocením těchto omezení lze prohlásit, že automatizované zpracovávání výstupu dotazovacího systému není příliš vhodný přístup. Na druhou stranu systém dobře plní svou funkci pro ruční dohledání některých závislostí nebo ověření správnosti indexovaných dat.

Velká část práce byla "analytická", tudíž jsem si rozšířil vědomosti z oblasti indexace dat, dotazování či distribuovaných systémů. Během implementace jsem narazil na některé komplikace, které se mi podařilo úspěšně vyřešit. Jedna komplikace, která zůstala nevyřešená je popsaná v 6.2.1.

Systém byl vyvíjen na operačním systému Fedora a Ubuntu. Testování systému bylo prováděno na stejných operačních systémech, avšak díky běhu aplikace ve virtuálním prostředí je možné aplikaci libovolně přenášet.

Podrobný postup pro kompilaci, nastavení a spuštění aplikace je popsán v příloze B.

Literatura

[1] Abrahamsson, J.: *ElasticSearch 101 – a getting started tutorial [online]*. www.joelabrahamsson.com, 2.7.2013 [cit. 2016-4-24].
URL http://joelabrahamsson.com/elasticsearch-101/

[2] Azzopardi, L.; Kazai, G.; Robertson, S.; aj.: *Advances in Information Retrieval Theory: Second International Conference on the Theory of Information Retrieval, ICTIR 2009 Cambridge, UK, September 10-12, 2009 Proceedings*. Information Systems and Applications, incl. Internet/Web, and HCI, Springer, 2009, ISBN 9783642044168.

[3] Cellary, W.; Morzy, T.; Gelenbe, E.: *Concurrency Control in Distributed Database Systems*. Studies in Computer Science and Artificial Intelligence, Elsevier Science, 2014, ISBN 9781483294643.

[4] Curé, O.; Blin, G.: *RDF Database Systems: Triples Storage and SPARQL Query Processing*. Elsevier Science, 2014, ISBN 9780128004708.

[5] Foundation, T. A. S.: *Apache License [online]*. www.apache.org, leden 2004 [cit. 2016-4-23].
URL http://www.apache.org/licenses/LICENSE-2.0.html

[6] Hill, R.; Hirsch, L.; Lake, P.; aj.: *Guide to Cloud Computing: Principles and Practice*. Computer Communications and Networks, Springer London, 2012, ISBN 9781447146032.

[7] IBM; Zikopoulos, P.; Eaton, C.: *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, první vydání, 2011, ISBN 0071790535, 9780071790536.

[8] Manning, C.; Raghavan, P.; Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, 2008, ISBN 9781139472104.

[9] Miller, M.: *Using Google Advanced Search*. Using–, Pearson Technology Group, 2011, ISBN 9780789743657.

[10] Mlnkova, I.: *XML technologie*. Průvodce (Grada), Grada, 2008, ISBN 9788024727257.

[11] Nemeth, E.; Snyder, G.; Seebass, S.; aj.: *UNIX System Administration Handbook (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995, ISBN 0-13-151051-7.

[12] Newham, C.; Rosenblatt, B.: *Learning the Bash Shell*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., druhé vydání, 1998, ISBN 1565923472.

[13] Paulo Veríssimo, L. R.: *Distributed Systems for System Architects*. Springer US, první vydání, 2001, ISBN 978-0-7923-7266-0.

[14] Polzer, J.: *GNU Emacs a Vim: kapesní přehled*. CP Books, 2005, ISBN 9788025107829.

[15] Schildt, H.: *Java: The Complete Reference, Ninth Edition*. The Complete Reference, McGraw-Hill Education, 2014, ISBN 9780071808552.

[16] Soukup, J.; Macháček, P.: *Serialization and Persistent Objects: Turning Data Structures into Efficient Databases*. SpringerLink : Bücher, Springer Berlin Heidelberg, 2014, ISBN 9783642393235.

[17] Thomas, R.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, University of California, 2000.

Přílohy

Seznam příloh

A Obsah CD 38

B Překlad a spuštění 39

B.1 Překlad aplikace 39

B.2 Generování dokumentace 39

B.3 Příklady spuštění 39

C Konfigurační soubor 40

Příloha A

Obsah CD

```
/
├── MG4J/ - zdrojové texty dotazovacího systému
│   ├── src/
│   │   └── main/
│   │       └── java/
│   │           └── mg4jquery/
│   │               ├── Parameters.java
│   │               ├── Query.java
│   │               ├── ResultProcessor.java
│   │               └── config.xml
│   ├── target/
│   │   └── mg4jquery-0.0.1-SNAPSHOT-jar-with-dependencies.jar
│   ├── javadoc/ - vygenerovaný Java Doc
│   │   └── ...
│   ├── pom.xml
│   └── servers12000.txt
├── BP/ - zdrojové texty pro překlad textové části BP
│   └── ...
└── plakat
```

Příloha B

Překlad a spuštění

B.1 Překlad aplikace

Překlad se provádí pomocí *Maven* zadáním příkazu `mvn compile assembly:single` v adresáři obsahující soubor `pom.xml`

B.2 Generování dokumentace

Vygenerování dokumentace se provede příkazem `mvn javadoc:javadoc`

B.3 Příklady spuštění

```
java -jar target/mg4jquery-0.0.1-SNAPSHOT-jar-with-dependencies.jar -q "lemma:(be)"
-h servers.txt12000 -s mg4j/src/main/java/mg4jquery/mg4jquery/config.xml
```

```
java -jar target/mg4jquery-0.0.1-SNAPSHOT-jar-with-dependencies.jar
-q "lemma:(influence)"-h ../servers.txt12000
```

```
java -jar target/mg4jquery-0.0.1-SNAPSHOT-jar-with-dependencies.jar -q
"1:nertag:(artist OR person) < lemma:(inspire OR influence OR admiration
OR tutelage OR tribute OR homage) < 2:nertag:(artist OR person)"-h
../servers.txt12000 -c "1.nerid != 2.nerid"-s src/main/java/mg4jquery/config.xml
```

Příloha C

Konfigurační soubor

```
<?xml version="1.0" encoding="UTF-8"?>
<mg4jconfig>
  <!--  Configuration for list of servers -->
  <servers>
    <!-- Use this file as input  -->
        <file></file>
        <!-- Default port if port is not specified -->
        <port></port>
        <!-- Exclude unreachable servers -->
        <excludeUnreachable>true</excludeUnreachable>
        <!-- timeout -->
        <timeout>5</timeout>
        <offset>0</offset>
  </servers>
        <!-- Configuration for output formatting -->
        <output>
          <highlightOn>true</highlightOn>
          <color>blue</color>
          <!--  Please note that if opening/closing... -->
          <!-- openingTag <start> openingTag -->
          <!-- closingTag <end> closingTag -->

          <!-- Which tags to highlight -->
          <tagHighlight>person|artist</tagHighlight>
          <!--  highlight lemmas from query? -->
          <highlightLemmasFromQuery>true</highlightLemmasFromQuery>
          <displayAttribute>href</displayAttribute>
      <raw>false</raw>
        </output>
        <!-- other stuff -->
</mg4jconfig>
```