VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍŘENÍ SYSTÉMU PRO ZÍSKÁVÁNÍ, ZPRACOVÁNÍ A ANALÝZU ROZSÁHLÝCH KOLEKCÍ TEXTŮ Z WEBU

EXTENDING SYSTEM FOR ACQUIRING, PROCESSING, AND ANALYSING LARGE WEB TEXT COLLECTIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JIŘÍ MATĚJKA**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Cílem práce je rozšířit stávající systém pro kolekci, stahování, zpracování a analýzu webových stránek. Práce se zabývá automatizací veškeré prováděné činosti, přináší nové nástroje do stávajícího systému a nabízí nejen nové verze některých nástrojů zapojených do systému zpracování, ale nabízí i nové postupy a myšlenky.

Abstract

The aim of the thesis is to extend the existing system for collecting, downloading, processing and analyzing web pages. This work deals with the automation of all processes, brings new tools into the existing system and offers new versions of some tools involved in the processing system and also offers new procedures and ideas.

Klíčová slova

web, stahování webových stránek, korpus, vertikální text, extrakce textu z HTML kódu, analýza textu

Keywords

web, web pages downloading, corpus, text in vertical format, text extraction from HTML code, text analysis

Citace

MATĚJKA, Jiří. *Rozšíření systému pro získávání, zpracování a analýzu rozsáhlých kolekcí textů z webu*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Rozšíření systému pro získávání, zpracování a analýzu rozsáhlých kolekcí textů z webu

Prohlášení

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

.....
Jiří Matějka
26. dubna 2018

Poděkování

Tímto bych chtěl poděkovat všem, kteří se na mé práci podíleli nebo mi byly oporou, především panu docentu Pavlu Smržovi a panu doktoru Jaroslavu Dytrychovi nejen za cennou pomoc, ale i za možnost zapojit se do tohoto projektu.

Obsah

1 Úvod **4**
- 1.1 Volba tématu . . . 4
- 1.2 Úvod do problematiky . . . 4
- 1.3 Definice základních pojmů . . . 5
  - 1.3.1 Tokenizace . . . 5
  - 1.3.2 Tagging . . . 5
  - 1.3.3 Parsing . . . 5

2 Zpracování webových korpusů ve světě **6**
- 2.1 ClueWeb09 corpus . . . 6
- 2.2 ChatNoir . . . 6
- 2.3 Google Ngram corpus . . . 7

3 Kolekce odkazů a stahování webových stránek **8**
- 3.1 Zpracování RSS a ATOM souborů . . . 8
- 3.2 Použité nástroje . . . 8
  - 3.2.1 Urllib . . . 9
  - 3.2.2 GNU Wget . . . 9
  - 3.2.3 Warc . . . 9
  - 3.2.4 Lzma . . . 9
- 3.3 Paralelní přístup . . . 9
- 3.4 Deduplikace . . . 10
- 3.5 Ukládání stažených dat . . . 10
  - 3.5.1 Komprese dat . . . 10
  - 3.5.2 WARC . . . 10
- 3.6 Srovnání s původním systémem . . . 10
  - 3.6.1 Původní systém . . . 10
  - 3.6.2 Nový systém . . . 11

4 Získání čistého textu ze stažených stránek **12**
- 4.1 Problémy při zpracování webových stránek . . . 12
  - 4.1.1 Kódování obsahu . . . 12
  - 4.1.2 Reklamy . . . 12
- 4.2 Použité nástroje . . . 13
  - 4.2.1 Html . . . 13
  - 4.2.2 Gzip . . . 13
  - 4.2.3 Lzma . . . 14
  - 4.2.4 Warc . . . 14

4.2.5 Jednoduchý parser napsaný v jazyce C . . . 14
4.2.6 Beautiful soup . . . 14
4.2.7 Regulární výrazy . . . 14
4.2.8 MyHTML . . . 14
4.3 Původní systém . . . 14
4.4 Nový systém . . . 15

5 Převod textu do vertikálu **16**
5.1 Vertikál . . . 16
5.2 Použité nástroje . . . 16
5.2.1 UDpipe . . . 16
5.2.2 MorphoDiTa . . . 17
5.2.3 Regulární výrazy . . . 17
5.2.4 Subprocess . . . 17
5.2.5 Pexpect . . . 17
5.2.6 Langdetect . . . 17
5.3 Paralelní zpracování . . . 18
5.3.1 Rizika a problémy u paralelního zpracování . . . 18
5.3.2 Řešení rizik paralelního zpracování . . . 18
5.3.3 Ochrana proti deadlocku . . . 19
5.3.4 Vyřešení problému stárnutí . . . 19
5.3.5 Komunikace mezi procesy . . . 19
5.4 Vícejazyčnost obsahu . . . 20
5.4.1 Způsob detekce jazyka . . . 20
5.5 Původní systém . . . 20
5.5.1 Vertikalizátor . . . 20
5.6 Detekce jazyka . . . 22
5.7 Nový systém . . . 22
5.7.1 Nové tagy a atributy . . . 22

6 Tagging **23**
6.1 Použité nástroje . . . 23
6.1.1 MorphoDiTa . . . 23
6.2 Gzip . . . 23
6.3 Lzma . . . 23
6.4 Subprocess . . . 23
6.4.1 MorphoDiTa . . . 23
6.5 Paralelní zpracování . . . 24
6.6 Srovnání s aktuálním systémem . . . 24
6.7 Nový systém . . . 25

7 Spuštění zpracování a tvorba obslužných skriptů **26**
7.1 Instalace knihoven jazyka Python . . . 26
7.2 Instalace nástroje MorphoDiTa . . . 26
7.3 Automatické zpracování . . . 27
7.3.1 RSS zdroje . . . 27
7.3.2 Automatické vyhledávání nových zdrojů . . . 27
7.3.3 Manuální vložení nových zdrojů . . . 27

7.3.4 Soubory s odkazy určenými ke stažení . . . 27
7.3.5 Vertikalizace . . . 27
7.3.6 Tagging . . . 27
7.4 Manuální zpracování . . . 28
7.4.1 Kolekce odkazů . . . 28
7.4.2 Stahování stránek . . . 28
7.4.3 Převod textu do vertikálního formátu . . . 29
7.4.4 Tagging . . . 29
7.5 Změna použitých nástrojů . . . 30
7.5.1 Kolekce a stahování odkazů . . . 30
7.5.2 Převod textu do vertikální podoby . . . 30
7.5.3 Tagging . . . 31
7.6 Tvorba obslužných skriptů . . . 31
7.6.1 Funkce zjednodušující tvorbu skriptů . . . 31
7.6.2 Logování . . . 33

8 Měření výkonu zpracování **35**
8.1 Stahování stránek . . . 35
8.1.1 Původní systém . . . 35
8.1.2 Nový systém . . . 35
8.2 Nástroje pro kompresi dat . . . 36
8.2.1 Výsledky měření . . . 36
8.3 Získání čistého textu, tokenizace a vertikalizace . . . 38
8.3.1 Srovnání systémů . . . 38

9 Závěr **42**

Literatura **43**

Kapitola 1

Úvod

Na úvod mé práce uvedu několik důvodů, proč se zajímám o problematiku zpracování rozsáhlých kolekcí textů z webu a také uvedu čtenáře do problematiky.

V následující kapitole bude uvedeno několik zajímavých příkladů použití korpusů ve světě. Protože korpusů existuje obrovské množství, byly vybrány ty provedení, která mi přišla nejzajímavější.

Další kapitoly jsou věnovány mé práci. Kapitoly jsou rozděleny podle jednotlivých částí celkového systému zpracování. V každé z těchto kapitol se nachází podrobnější popis problematiky a jejího řešení, použité nástroje a na závěr je porovnání s původním systémem.

V přílohách je poté možné vyčíst různé údaje o výkonosti náročnosti projektu a srovnání s původním systémem, stručný postup jak tvořit obslužné skripty k vytvořeným knihovnám a samotné úkázky jednotlivých kroků zpracování.

1.1 Volba tématu

Téma Rozšíření systému pro získávání, zpracování a analýzu rozsáhlých kolekcí textů z webu jsem si zvolil hlavně kvůli své dvouleté spolupráci s výzkumnou skupinou KNOT [13], svému zájmu a svým zkušenostem v této problematice.

1.2 Úvod do problematiky

Cílem tohoto projektu není vytvořit nový proces zpracování webových korpusů, ale rozšířit stávající systém. Aktuálně dokážeme jednorázově vytvořit korpus a sémanticky ho označit. Problém tedy není v tom, že korpus nelze vytvořit, ale že to nelze provádět automaticky. To znamená, že do procesu musí vstoupit člověk, který je obeznámen s nástroji i problematikou, spustit mnoho nástrojů ve správném pořadí a sledovat, zda při zpracování nenastane chyba.

Dalšími problémemy jsou rozmanitost webových stránek, jak z hlediska vzhledu, jejich účelu, tak i způsobu, jak je zdrojový kód stránky napsán. Dalším faktorem ovlivňující obtížnost je tolerance jazyka HTML k chýbám Vícejazyčnost stránek nutí použití dalších nástrojů pro detekci jazyka, které zpomalují celkové zpracování a tyto nástroje jsou často mateny ne tolik gramatickými chybami, jako spíše příliš krátkými textovými úseky, jakým může být například komentář uživatele k jednomu ze článků na webové stránce.

Webových zdrojů je obrovské množství a je nemožné, nebo přinejmenším velice obtížné, napsat optimální program pro všechny webové stránky. Má práce se zaměřuje zejména na zpracování článků v českých mediích. Velké množství těchto zdrojů je logicky členěno do

bloků, ale je zde i mnoho reklam. HTML kód je často bez chyb a lze ho poměrně jednoduše zpracovat, ale jsou zde i skutečnosti, na které je potřeba si dát pozor.

1.3 Definice základních pojmů

Pro úplné porozumnění textu je potřeba znát spoustu odborných termínů. Většina termínů je vysvětlena v jednotlivých kapitolách, ale nachází se zde několik termínů, které je třeba vysvětlit předem.

1.3.1 Tokenizace

Tokenizace je proces, kde je sekvence znaků rozdělena na tzv. *tokeny* (žetony). Tyto *tokeny* jsou často označovány jako slova nebo výray. Jinými slovy, jedná se o proces, kde se užitečný text rozdělý na slova [22].

1.3.2 Tagging

Tagging je proces, ve kterém se pracuje s jednotlivými slovy. V tomto procesu se provádí morfologický rozbor daného slova, zařazuje se do gramatické kategorie a určuje se jeho základní tvar [5].

1.3.3 Parsing

Parsing je proces zpracování věty. Je to proces podobný větnému rozboru, kde se hledají vztahy mezi jednotlivými slovy ve větě.

Kapitola 2

Zpracování webových korpusů ve světě

Korpusy obsahují obrovské množství dat a jsou cenými informačními zdroji. Se samotnými korpusy lze provádět spoustu činností. S dostatečně velkou databází lze lépe porozumět jazyku (zjistit jak se jazyk v průběhu let mění, jaká slova jsou nejčastěji používaná, atd.), provádět výzkumy (jaké jsou trendy ve společnosti, jak dokáže určitá událost ovlivnit svět okolo sebe, ...), tvořit vyhledávače a spoustu dalších věcí. Z toho plyne, že se tvoří spousta korpusů.

Častým zdrojem dat pro tvorbu korpusů jsou webové stránky. Podle statistik každý týden se zaregistruje přibližně 800 000 – 1 100 000 nových domén [33], což je obrovské množství dat, která jsou potenciálně vhodná pro tvorbu webového korpusu.

V této kapitole popíšu některé ze zajímavých provedení tvorby a zpracování korpusových dat.

2.1 ClueWeb09 corpus

Jedná se o poměrně rozsáhlý webový korpus určený pro podporu výzkumu vyhledávácích informací a vývoj technologií zabývající se zpracováním lidských jazyků. Tento korpus obsahuje zhruba jednu miliardu webových stránek jeho komprimovaná velikost je 5 TB (po dekompresi velikost dosahuje až 25 TB). V korpusu je obsažen text v deseti různých jazycích.

Jednotlivé webové stránky jsou ukládány v souborech ve fromátu WARC 3.5.2. Každá z těchto kolekcí stránek obsahuje několik desítek tisíc stránek (většinou okolo 40 000 stránek) a komprimovaná nástrojem gzip.

Informace obsažené v této kapitole byla získána z webových stránek projektu [15].

2.2 ChatNoir

ChatNoir, francouzky černá kočka, je novým vyhledávačem, který pracuje nad korpusovými daty. Korpus, který byl k tvorbě tohoto vyhledávače použit, je anglickou podmnožinou dat korpusu *ClueWeb09 corpus* 2.1 (anglická podmnožina obsahuje 500 milionů stránek s celkovou veliksotí 12 TB) [17].

Projekt *ChatNoir* je psán převážně v jazyku Java a v procesu je mimojiné zapojeno získání textu z HTML stránek, detekcí jazyka a tvorba indexů pro vyhledávač [4].

2.3 Google Ngram corpus

Tento korpus obsahuje 500 miliard slov z 5,2 miliónu knih, které vyšly za posledních 500 let. Data byla získána z knih psaných v angličtině, francouštině, čínštině, ruštině španělštině a němčině. Nad tímto korpusem byl posléze postaven nástroj, který je schopen měřit frekvenci používání různých slov v průběhu let [24]

Kapitola 3

Kolekce odkazů a stahování webových stránek

Aby bylo možné zpracovat webové články, je nutné mít k dispozici alespoň minimum dat. Způsobů, pomocí kterých lze získávat pravidelně nové webové články, je více. Oblíbeným je například klasický webcrawling, kde se ze stažené stránky extrahují odkazy a získané adresy se použijí k dalšímu sběru dat. Má práce se zaměřuje hlavně na využití technologií RSS a ATOM.

RSS *Rich Site Summary* je technologie sloužící k syndikaci obsahu. Využívá XML formátu a uživatelé webových stránek si mohou díky této technologii nastavit odběr novinek z webu [30]. Zejména díky tomu, že technologie RSS neměla ve své době téměř žádnou konkurenci, rychle se rozšířila a ještě dnes je nejpoužívanějším nástrojem, přestože je vývoj od roku 2003 zastaven [18].

ATOM *Atom Syndication Format* je jedním z webových standardů. Podobně, jako RSS, se využívá XML formátu a slouží k informování uživatelů stránek o nových článcích [29] Na rozdíl od RSS se jedná o mladší formát a už od začátku se dbalo na modularitu, znovupoužitelnost a přehlednost kódu. Zatímco RSS není IETF standardem, ATOM sem byl zařazen v prosinci roku 2005 [18].

3.1 Zpracování RSS a ATOM souborů

Jak RSS, tak i ATOM jsou psané pomocí jazyka XML. Proto jsem použil modul *xml.etree.ElementTree* určený k práci nad jednotlivými elementy. Ovšem ne všechny zpracovávané dokumenty jsou bez chyb a právě na tyto dokumenty je výše zmíněný modul krátký. U poškozených dokumentů zpracování probíhá pomocí regulárních výrazů a díky tomu jsem schopen získat alespoň pár odkazů na nové články.

3.2 Použité nástroje

Kolekce odkazů a stahování stránek bylo naprogramováno v jazyce Python. Konkrétní verze jazyka je Python3.5, která je nainstalovaná na serverech, na nichž budou nástroje spuštěny. Jazyk Python jsem zvolil nejen kvůli popularitě a velkému množství knihoven, ale i kvůli nenáročnosti procesu kolekce a stahování stránek na procesor.

3.2.1 Urllib

Urllib je balíček několika modulů určených pro práci s url. V této práci bylo využito pouze modulu *urllib.request*, který je určený ke stahování webových zdrojů pomocí url odkazu [12]. Pomocí tohoto modulu se podařilo stáhnout většinu odkazů, ale stále zde bylo několik stránek, které se podařilo stáhnout až na pozdější pokus a také zde byly stránky, které se stáhnout nepodařilo, přestože přes prohlížeč na ně přistoupit bylo možné. Tento problém vyřešil nástroj *GNU Wget*.

3.2.2 GNU Wget

GNU Wget je aplikace pro stahování souborů pomocí protokolů HTTP, HTTPS, FTP a FTPS. Je to neinteraktivní nástroj spouštěný uvnitř příkazové řádky a je vhodný pro použití v automatických skriptech [16]. Pro spuštění této aplikace uvnitř skriptu využito knihovny *subprocess* 5.2.4. Bohužel úplné nahrazení *urrlib* aplikací *GNU Wget* poměrně výrazně zpomalilo proces kolekce odkazů. Bylo nutné každé stránce dát určitý časový limit na stažení. Většina stránek je stažena do 1 sekundy, ale pokud je zapotřebí stáhnout i zbylé stránky, které jsou umístěny na pomalejších, méně dostupných nebo vzdálených serverech, je nezbytné časový limit zvýšit alespoň na 3 sekundy.

3.2.3 Warc

Warc, je knihovna pro jazyk Python umožňující práci se souborem uloženým ve formátu WARC 3.5.2. Tato značně zjednodušuje získání jednotlivých záznamů (v tomto případě webových stránek) nebo tvorbu samotných souborů formátu WARC [7].

Pro účel stahování stránek zde byla tato knihovna použita právě za účelem tvorby souborů typu warc.

3.2.4 Lzma

Modul *lzma* nabízí třídy a funkce určené pro kompresi a dekompresi dat s použitím algoritmu LZMA. Součástí je také souborové rozhraní podporující souborové formáty *.xz* a *.lzma*, které používá aplikace xz [10].

Pro účely kolekce a stahování dat je tento modul použit ke kompresi souborů.

3.3 Paralelní přístup

Jak bylo zmíněno výše, aplikace *GNU Wget* nemohla úplně nahradit *urllib* z důvodu časové náročnosti. Proto byla zvolena zlatá střední cesta a v případě, že *urllib* selže, stránku se pokusila stáhnout aplikace *GNU Wget*. Sice bylo dosaženo rychlejšího stahování, ale stále to nestačilo. Protože se ale většinu času čeká na odpověď serveru a proces je uspaný, byl zvolen paralelní přístup. Místo toho, aby na odpověď čekal jeden proces, čekalo na ni hned několik procesů. K paralelizaci bylo využito modulu *concurrent.futures* a celý proces kolekce a stahování proces byl několikanásobně zrychlen. Veškerá rizika, která s sebou přináší paralelní zpracování, byla vyřešena přidělením maximální doby pro zpracování jednoho odkazu. Pokud rodič čekal příliš dlouho na potomka, potomek byl zabit a rodič pokračoval v práci.

3.4 Deduplikace

Při zpracování rozsáhlých webových kolekcích je potřeba provádět deduplikaci dat. Každou staženou webovou stránku je potřeba zpracovat dalšími nástroji a to znamená, že jim musíme věnovat určitý procesorový čas. Navíc je zbytečné archivovat jednu a tu samou stránku vícekrát, proto je potřeba už při stahování článku dát důraz na to, abychom opakovaně nestáhli tutéž stránku znovu. Je třeba si uvědomit, že v této fázi celého procesu zpracování máme pouze odkazy a jediný možný způsob deduplikace dat je zpracovat každý odkaz právě jednou. V novém systému se deduplikace řeší na úrovni kolekce odkazů. Kolektor odkazů si vede soubory, ve kterých jsou seznamy již zpracovaných odkazů. Klade se přitom důraz na to, aby s tímto souborem manipuloval vždy právě jeden kolektor a sám kolektor si tyto soubory u sebe držel minimální dobu a nedocházelo tak ke zbytečné ztrátě deduplikačních dat a poté i tvorbě následných duplicit.

3.5 Ukládání stažených dat

Stažené stránky se dlouhodobě ukládají, a proto je potřeba zvolit vhodnou formu komprese dat a formát uložení jednotlivých záznamů.

3.5.1 Komprese dat

Dnes existuje spousta nástrojů umožňující kompresi dat, k obecně nejznámějším patří asi WinRAR, 7-zip nebo gzip. Použití WinRAR a 7-zip je přinejmenším nevhodné v tomto projektu, proto jsem se rozhodoval mezi výše zmíněným nástrojem gzip a ne tolik známým xz. Oba nástroje mají své výhody. Mezi výhody formátu xz patří menší výsledná velikost souborů, ovšem za cenu rychlosti. Formát gzip má na druhou stanu relativně rychlou kompresi a dekompresi dat [14]. Protože, jak již bylo zmíněno, se stránky budou ukládat dlouhodobě, zvolil jsem formát xz.

3.5.2 WARC

Pro ukládání webových kolekcí je použit formát WARC (Web ARChive). Tento formát specifikuje metodu pro kombinaci několika digitálních zdrojů do jednoho archivu spolu se souvisejícími informacemi [21]. Tento formát se často používá pro webový crawling, a proto byl i zvolen v mé práci.

3.6 Srovnání s původním systémem

V této kapitole budete obeznámeni se základnímy prvky původního a nového systému. Dále budete obeznámeni s jejich zásadními rozdíly.

3.6.1 Původní systém

Původní systém využíval stahování stránek pouze pomocí *urllib*. Kolekci odkazů z RSS a Atom zdrojů zastávaly 2 skripty – *collect.py* a *download.py*. Skripty byly napsány v jazyce Python verze Python2.7 pro projekt Newsbrief eu [20].

Kolekce odkazů byla implementována skriptem *collect.py*. Samotná kolekce se spouštěla pomocí nástroje *cron* každé 2 hodiny. Později byl skript přepsán do jazyka Python verze

Python3.5 a optimalizován. To vystačilo až do doby, než se začaly hromadit RSS zdroje a skript přestával stíhat sbírat odkazy z RSS a ATOM zdrojů. Stávalo se běžně, že v jednu chvíli běželo najednou několik kolekcí odkazů a navzájem si přepisovaly deduplikační soubory a tak se jedna stránka mohla stáhnout vícekrát. Tehdy přestal být tento nástroj vhodný pro svůj účel.

O stahování stránek se staral skript *download.py*. Stejně jako *collect.py*, i tento nástroj se spouštěl pravidelně pomocí *cronu*, pouze se namísto každých dvou hodin spouštěl jednou za 24 hodin. Skript byl následně přepsán do jazyka Python3.6, z důvodu chyby ve vertikalizátoru 5.5.1.

3.6.2 Nový systém

Momentálně se, na rozdíl od předchozích nástrojů, stahují stránky paralelně. O spouštění kolekce, stahování stránek, deduplikaci odkazů a vedení statistik se stará skript *big_brother.py*, který pravidelně spouští 2 skripty. Prvním z nich je skript *link_collector.py*, využívající třídy *Link_collector*. Tato třída je schopna vyhledat RSS a ATOM zdroje na stažených stránkách a extrahovat z RSS a ATOM souborů odkazy na nové články. Navíc je tato třída přímým potomkem třídy *Page_downloader*, a díky dědičnosti je schopna stahovat stránky jak sekvenčně, tak paralelně.

Stahování je řešeno skriptem *page_downloader.py*, který využívá třídy *Page_generator*. Tato třída je schopna se dotazovat jednotlivých serverů s časovým odstupem mezi dvěma dotazy na tentýž server a stahovat z nich stránky. Navíc je, stejně jako *Link_collector.py*, přímým potomkem třídy *Page_downloader* a získává tak možnost paralelního i sekvenčního stahování souborů ze serverů.

Kapitola 4

Získání čistého textu ze stažených stránek

Ve chvíli, kdy jsou k dispozici data, lze započít další zpracování. Pro práci s html souborem existuje celá řada nástrojů programovaných v mnoha jazycích, přesto ale je zde velké množství překážek.

4.1 Problémy při zpracování webových stránek

Jak již bylo dříve zmíněno, internet je velice rozmanitý a existuje nespočet způsobů, jak lze vytvořit webovou stránku.

4.1.1 Kódování obsahu

Kódováním obsahu se zde nemyslí šifrování, ale způsob, jakým jsou datově reprezentovány jednotlivé znaky. Podle dostupných statistik se s drtivou převahou používá kódování UTF-8 – až 91 % stránek. Další 4 % využívají ISO-8859-1 a 1 % vývojářů zvolilo kódování Windows-1251 [32]. Zbylá procenta obsahují ostatní kódování, kterých je nepřeberné množství.

Pří použití nástroje *urllib* 3.2.1 není problém zjistit, jaké kódování stránka používá. Protože je ale použit i nástroj *GNU WGET* 3.2.2, je potřeba kódování vyčíst z hlavičky HTTP odpovědi serveru. Z vlastní zkušenosti však mohu říct, že ne vždy je možné tímto způsobem kódování zjistit. Tento problém se vyskytl zatím pouze při zpracování starších archivů a to pouze ojediněle.

V případě, že se nepovede kódování obsahu zjistit, je využito modulu *chardet*, který je schopen rozpoznat hned několik kódování. V případě, že i tento nástroj selže, aplikace není schopna stránku zpracovat a stránka je zahozena.

4.1.2 Reklamy

Spousta stránek obsahuje reklamy. Reklama nemusí vůbec rušit strukturu obsahu, ale také může být vložena do středu odstavce. Právě v této chvíli je schopná zmást jakýkoliv nástroj, který pracuje s obsahem stránky, například proces analýzy textu nebo vyhledávač.

Detekce reklam

V dnešní době je spousta nástrojů, které jsou schopny detekovat reklamy na stránkách a odstranit je. Mezi nejznámější patří například Adblock. Nejen Adblock, ale i jemu podobné nástroje pracují s množinou pravidel. Při zpracování HTML elementu, zejména *<img>* (obrázky) a * * (blokový element), zjistí identifikátor a třídu daného elementu a na základě pravidel prvek zobrazí či nikoli. V případě externích odkazů (element *<a>*) se zjišťuje adresa [1].

V této práci bylo hned několik pokusů o využití těchto nástrojů nebo jen použití samotných pravidel. Problémem byla náročnost procesu na výpočetní výkon. Celý proces zpracování se zpomalil a výsledky nebyly tak dobré, jak se očekávalo. Z těchto důvodů tyto nástroje nebyly vůbec použity.

Rozdělení obsahu do logických celků

Webové stránky, zejména zpravodajské deníky, mají text rozdělený do určitých bloků, za účelem znovupoužitelnosti a přehlednosti kódu, co se uživatelů týče, tak i přívětivějšího vzhledu a snadnějšího použití.

Nové nástroje jsou schopny takového členení využít a díky tomu označit pro nástroje, které s textem budou nadále pracovat, kdy text pokračuje a kdy se text týká už něčeho jiného. Toto řešení sice není tak dokonalé jako odstranění celé reklamy z textu, ale jeho náročnost na výpočetní výkon je přijatelná a svůj účel to dokáže splnit. HTML elementy, pomocí kterých se rozděluje text do logických bloků, jsou *<p>* (odstavce), * * (blokové elementy) *<h1>, <h2>, <h3>, <h4>* a *<h5>* (nadpisy).

HTML entity

Protože v jazyce HTML jsou některé znaky rezervovány nebo je potřeba vypsat znak, který se nenachází na klávesnici, existují tzv. HTML entity. HTML entita je zvláštní zápis právě těchto znaků [31]. Při Zpracování HTML stránky je třeba tyto entity detekovat a nahradit za skutečné znaky. K tomuto účelu byl použit modul *html* **??**.

4.2 Použité nástroje

Pro extrakci textu z webových stránek bylo vyzkoušeno poměrně několik postupů a nástrojů. Co se týče použitých programovacích jazyků, jendalo se o C a Python.

4.2.1 Html

Html je modul jazyka Python a zahrnuje nástroje pro práci s HTML kódem. V této práci byl využit jen a pouze pro převod HTML entit na znaky.

4.2.2 Gzip

Tento modul jazyka Python poskytuje rozhraní pro stejnou kompresi a dekompresi dat, jakou poskytují programy gzip a gunzip. Samotná komprese dat je ale uvnitř modulu řešena knihovnou *zlib* [9].

Modul *gzip* je zde použit pouze pro dekompresi starších archivů, které jsou ukládány ve formátu gzip.

4.2.3 Lzma

Modul *lzma* 3.2.4 v této části zpracování slouží pouze k dekompresi dat.

4.2.4 Warc

Knihovna *warc* 3.2.3 je zde použita pro čtení jednotlivých záznamů ze souborů typu WARC 3.5.2.

4.2.5 Jednoduchý parser napsaný v jazyce C

Vytvořit jednoduchý a rychlý program v jazyce C bylo první myšlenkou a pokusem. Samotná tvorba nástroje na extrakci textu z HTML kódu není složitá, dokud se nedetekují reklamy nebo není třeba členit text do logických bloků. Dalším problémem byly HTML entity, které je třeba nahradit za reálné znaky.

4.2.6 Beautiful soup

Beautiful soup je knihovna pro jazyk Python určená k extrakci dat z HTML a XML dokumentů [23]. Použití této knihovny je relativně jednoduché a dokumentace je výborná. Pokud by bylo v budoucnu potřeba použít pravidla pro blokování reklam, je ideální volbou. Tato knihovna plnila svůj účel dlouhou dobu, přestože její slabina byla už od samého začátku zřejmá – rychlost. Jakmile se projekt rozšířil o další zpracování, byl modul, používající tuto knihovnu, brzdou v celém procesu a bylo zřejmé, že je potřeba najít efektivnější řešení.

4.2.7 Regulární výrazy

Jednoduchou a relativně efektivní cestou se ukázalo použití regulárních výrazů. Sice se zkomplikovala detekce reklam na stránkách, ale v době, kdy byly regulární výrazy zapojeny do procesu zpracování, bylo jasné, že se žádné rozpoznávání reklam a jejich odstranění z důvodu výkonosti provádět nebude.

4.2.8 MyHTML

Pro zajímavost je zde uveden ještě modul *MyHTML* od Alexandera Borisova. Celý modul je psán v jazyce C, konkrétně standard C99 [6]. Tento modul do procesu nebyl nikdy zapojen, z důvodu nedostatku času. Navíc má velké nedostatky v dokumentaci a ke konfiguraci nástroje a samotného sestavení aplikace je potřeba oprávnění administrátora. Program byl proto sestaven na osobním počítači, kde byl i vyzkoušen. Modul je poměrně rychlý a je jednou z možných alternativ, které by mohly v budoucnu nahradit alespoň částečné použití regulárních výrazů.

4.3 Původní systém

V původním systému je k tomuto účelu použita aplikace zvaná vertikalizátor 5.5.1, který využívá mimo jiné nástroje *jusText*.

JusText je nástroj určený k odstranění boilerplate obsahu, například navigační odkazy, hlavička HTML stránky. Nástroj je vhodný pro použití při zpracování HTML stránek [3]. Potenciální nevýhodou tohoto nástroje je nutné mít tzv. *stoplist* na základě kterého se vyhodnocuje, zda je daný blok textu boilerplate – text je nechtěný v dalším zpracování.

Stoplist obsahuje slova (*stop words*), která se snaží nástroj najít v textovém bloku. Na základě množství těchto slov v textu, je danému bloku přidělena třída, pomocí které lze určit, zda text je vhodné pro další zpracování text zachovat nebo je lepší text odstranit [2]. Tento nebyl v této práci použit z důvodu, že takový nástroj už je ve zpracování využit a je vhodné mít i alternativní způsoby, jak řešit problém zpracování stránek.

4.4 Nový systém

Nový systém definuje třídu *Page*. Tato třída je schopna zpracovat HTML stránku a vrátit text rozdělený do logických bloků. Protože umí pracovat pouze s html kódem a ne komprimovanými WARC archivy, existuje ještě třída *Page_reader*. Tato třída je schopna číst soubory ve formátu WARC, xz a gzip. Třída má implementovaný iterátor, ve kterém postupně vytváří objekty typu *Page*. Žádná z těchto tříd nemá implementovanou obslužnou aplikaci. Třída *Page_reader* se používá během vertikalizace jako generátor stránek.

Kapitola 5

Převod textu do vertikálu

V momentě, kdy je zpracovaná HTML stránka, je nutno text převést do vertikální podoby.

5.1 Vertikál

Vertikál, popřípadě vertikální soubor, je textový soubor obsahující na každém řádku právě jednu pozici (ale je zde několik výjimek). Pozicí se zde rozumí slovo, XML tag, číslo, interpunkční znaménko nebo jiný oddělovač. Přestože se ve vertikálním souboru používají XML značky, samotný vertikál neodpovídá definici XML souboru [34]

5.2 Použité nástroje

Pro tvorbu vertikálu byly použity jak již existující nástroje, tak nově vyvinuté. V této kapitole budou podrobně popsány nástroje, které byly k vertikalizaci využity.

5.2.1 UDpipe

Prvním použitým nástrojem byl nástroj *UDpipe*. *UDpipe* je nástroj určený pro tokenizaci **??**, taggingu **??** a parsing **??**. *UDpipe* je schopen zpracovat texty v různých jazycích [27].

Tento nástroj byl použit pro vytvoření prototypu. Prototyp byl schopný zpracovat české i slovenské texty, ale musel řešit některé problémy, které vznikly při použití tohoto nástroje.

Tagging

K tagingu *UDpipe* používá nástroj *MorphoDiTa* 6.4.1.

Parsing

K parsingu je zde použit nástroj *Parsito*. Jedná se o parser napsaný v jazyce C++ s vysokou přesností a propustností až 30 000 slov za sekundu [26].

Vertikalizace

UDpipe neprovádí žádnou vertikalizaci, ale samotný proces vertikalizace je při zpracování dalšími nástroji nezbytný. Stažené HTML stránky byly nejdříve zpracovány vertikalizátorem 5.5.1, poté odstraněny značky a sestaven text. Výsledný text byl dán na zpracování

nástrojem *UDpipe* a poté do něj byly doplněny odstraněné značky vertikalizátoru. Doplnění probíhalo tak, že se z původních pozic jednotlivých značek ve vertikálním souboru vypočítaly nové pozice a posléze na tyto nové pozice byly zpětně vyjmuté značky doplněny. Naneštěstí tokenizace nástrojem *UDpipe* rozdělovala text jiným způsobem, než vertikalizátor, to způsobilo, že zpětně vypočítaná pozice nebyla korektní. Proto nakonec musel být proces tokenizace úplně nahrazen vertikalizátorem.

5.2.2 MorphoDiTa

MorphoDiTa je nástrojem určeným k morfologicé analýze textu. Jejím hlavním účelem v tomto projektu je provedení taggingu **??**, ale je zde použita i k tokenizaci **??**. Aby bylo možné text převést do vertikální podoby, je potřeba nejprve jednotlivé bloky textu rozdělit do vět a věty rozdělit na jednotlivá slova. Právě k tomuto účelu je zde použit tokenizer, který poskytuje nástroj *MorphoDiTa*. Podrobnější popis nástroje je v kapitole *Tagging* **??** [**?**].

5.2.3 Regulární výrazy

Pro vytvoření triviálního vertikalizátoru stačí i regulární výrazy. Při použití regulárních výrazů ale řešení postrádá eleganci a velice špatně se do procesu zanášejí výjimky z pravidel a to zejména, pokud je výjimek hodně. Regulární výrazy zde byly použity pouze jako dočasné řešení, než byl problém vyřešen lepším tokenizerem.

5.2.4 Subprocess

Modul *subprocess* umožňuje vytvoření nového procesu, připojení na jeho standardní vstup, standardní výstup, chybový výstup a zjištění jeho návratového kódu [11]. Modul *subprocess* není v tomto projektu použit poprvé. Na rozdíl od předchozího použití je zde i jistý způsob komunikace s vytvořeným procesem 5.3.

5.2.5 Pexpect

Modul *Pexpect* umožnujě vytvoření nového procesu a jeho kontrolu způsobem, jako by samotné příkazy psal sám člověk [25]. Použití tohoto nástroje vypadalo velice slibně, protože na rozdíl od nástroje *subprocess* umožňoval průbežně zapisovat na standardní vstup a číst ze standardního výstupu, aniž by hrozil deadlock 5.3.1. Naneštěstí je ale výkon tohoto nástroje značně omezen dobou zápisu na standardní vstup spuštěného procesu. Při každém zápisu na standardní vstup se čeká několik milisekund. Na standardní vstup se zapisuje právě tolikrát, kolik má zpracovaná HTML stránka řádků. Počet řádků se většinou pohybuje ve stovkách, ale v některých případech může být počet řádků i větší než tisíc. V takových případech nám vzniká i několikavteřinové zpoždění způsobené pasivním čekáním. Možné řešení se ukázalo nezapisovat na standardní vstup po řádcích, ale po jednotlivých záznamech. Tehdy se zpracování z neznámého důvodu zastavilo úplně, a proto byl nástroj *Pexpect* nahrazen nástrojem *subprocess*.

5.2.6 Langdetect

langdetect je jednou z řady knihoven jazyka Python určených pro detekci jazyka. Jedná se o import knihovny *langue-detection*, která byla vyvýjená společností Google pro jazyk Java

[8]. Tato knihovna je v projektu použita pro detekci jazyka, protože má poměrně dobré výsledky i na krátkých textových úsecích.

5.3 Paralelní zpracování

Protože doba zpracování rozsáhlých webových kolekcí je poměrně dlouhá, je lepší rozložit činnost mezi více procesů a tak zkrátit celkovou dobu zpracování.

Paralelní zpracování v tomto projektu funguje tak, že rodičovský proces zpracovává HTML stránky a jeho potomci provádějí tokenizaci, vertikalizaci a deteci jazyka. Činnost potomků je časově náročnější, z toho důvodu si na pomoc tvoří ještě proces, který provádí detekci jazyka, zatímco oni provádějí vertikalizaci. Potomků může být více nebo pouze jeden. Zajímavé je, že na každém ze strojů, na kterých probíhalo testování, je potřeba jiný počet potomků, aby stačilo zpracovávat stránky, které generuje rodičovský proces.

5.3.1 Rizika a problémy u paralelního zpracování

Už samotné použití více procesů s sebou ale nese velká rizika a problémy, keré je potřeba řešit.

Deadlock

Deadlock je situace, kdy každý proces z určité množiny procesů je pozastaven a čeká na uvolnění zdroje s výlučným přístupem vlastněného nějakým procesem z dané množiny, který jako jediný může tento zdroj uvolnit [28].

V tomto projektu se jedná zejména o situaci, kdy rodičovský proces dokončil zpracování HTML stránek, zpracovanou stránku předal k tokenizaci a čeká na výsledek. Při špatně zvolené synchronizaci procesů nemusí proces provádějící tokenizaci zjistit, že nějaká data ke zpracování dostal a čeká, než nějaká data dostane.

Stárnutí

Stárnutí nebo také hladovění, je situace, kdy proces čeká na podmínku, která nemusí nastat [28].

V případě tohoto projektu se jedná například o situaci, kdy rodičovský proces čeká na dokončení práce svého potomka, ale potomek již svou činnost dokončil nebo při zpracování dat potomkem nastala chyba a proces potomka byl ukončen.

Přepisování sdílené paměti

Dalším z vážných problémů je přepisování sdílené paměti. Při zpracování HTML stránek se postupně tvoří fronta dat, čekající na další zpracování. Je zde riziko, že si více potomků v jednu chvíli přečte totožný záznam z fronty a budou zbytečně zpracovávat stejnou stránku. Existuje zde i ještě větší riziko, že při uložení zpracováné stránky do fronty výsledků si procesy navzájem přepíšou paměť a dojde tak ke ztrátě dat – místo více záznamů se uloží jen jeden.

5.3.2 Řešení rizik paralelního zpracování

Protože zmíněná rizika nelze ignorovat, existuje již řada postupů, které lze využít k jejich řešení.

Sdílená paměť

Aby se zabránílo riziku ztrátě dat, je potřeba využít výlučný přístup k datům. Python poskytuje řadu možností, jak zajistit výlučný přístup.

Prvním a nejjednoduším ze způsobů jsou zámky. Zjednodušene řečeno, zámek je objekt, který má 2 stavy. Jeden ze stavů indikuje to, že se v kritické sekci, tj. části kódu, kde se pracuje se sdílenou pamětí, již jeden proces nachází. Druhý ze stavů identifikuje pravý opak – kritická sekce je volná. Aby zámek fungoval tak, jak má, je potřeba, aby každý z procesů při přístupu do kritické sekce zkontroloval stav zámku a pokud sekce volná není, vyjmul se z fronty aktivních procesů a zařadil se do fronty čekajících procesů na vstup do dané kritické sekce.

Dalším ze způsobů jsou semafory. Semafor je zámek rozšířený o počítatlo, které indentifikuje, kolik procesů může do kritické sekce vstoupit. Pracuje se s ním téměř stejně jako se zámkem.

Jazyk Python poskytuje i možnost využití fronty určené k paralelnímu zpracování. Jedná se o klasickou frontu s tím rozdílem, že manipulace dat je ošetřena několika zámky, aby nedošlo k žádným rizikům. Tato implementace fromty byla použita k řešení výlučného přístupu v této části projektu.

5.3.3 Ochrana proti deadlocku

S použitím front se vyřešil i problém s hrozícím deadlockem. Rodičovský proces postupně plnil frontu daty určenými k tokenizaci a potomci z ní postupně načítali data. V případě, že byla fronta prázdná při čtení, potomek byl uspán do doby, než se do fronty další data vložila. Podobným způsobem to fungovalo i při plnění fronty se zpracovanými daty. Jediným rozdílem bylo to, že pokud výstupní fronta při čtení rodičem byla prázdná, tak v případě, že existují ještě nezpracované HTML stránky, rodič nebyl uspán a zpracoval další z HTML stránek – tzn. byl uspán až tehdy, kdy byly všechny HTML stránky zpracovány a čekalo se jen na dokončení práce potomky.

5.3.4 Vyřešení problému stárnutí

Protože rodič ani potomek netuší, kolik mají přečíst dat z front, musí rodič po zpracování poslední z HTML stránek informovat potomky, že se jedná o poslední stránku. To je provedeno vložením speciálního objektu do fronty. Pokud potomek načte z fronty tento objekt, vloží ho zpětně do výstupní fronty, aby informoval rodiče o dokončení zpracování. To ovšem neřeší situace, kdy je potomek násilně ukon jiným procesem, uživatelem nebo z důsledku chyby při provádění požadované činnosti. Násilné ukončení procesu není nijak řešeno. Co se chyby při zpracování týče, jedním z možností je odchycení výjimky. Místa, kde se může chyba vyskytnout, je nutno identifikovat řádným a dlouhodobým testováním a řádně ošetřit.

5.3.5 Komunikace mezi procesy

V průběhu tokenizace a převodu textu do vertikální podoby existuje v jednu chvíli hned několik procesů, které musí mezi sebou komunikovat.

Rodičovský proces se stará o čtení dat z archivů, popřípadě i o samotnou dekompresi. Dále má na starost zpracování HTML kódu, čehož dosáhne pomocí již vytvořených postupů

posaných v kapitole 4. Poté zpracovanou stránku předá jednomu ze svých potomků a v případě, že jsou uspaní, je probudí, aby mohli pokračovat v činnosti.

Potomek, který má k dispocici zpracovanou stránku, zakóduje veškeré značky potřebné k vertikalizaci textu, takovým způsobem, aby každou z těchto značek tokenizer identifikoval jako právě jedno slovo ve větě. Poté potomek spustí samotný tokenizer, na standardní vstup mu zapíše výsledný text a počká na jeho zpracování. Po zpracování musí dekódovat zakódované značky a během dekódování je prováděna i samotná vertikalizace. Paralelně s vertikalizací se také provádí detekce jazyka nad odstavci. Ta je prováděna dalším procesem, kdy rodičovským procesem je zde proces provádějící vertikalizaci a potomkem je proces provádějící překlad. Výsledek, tedy vertikalizovaný text s provedenou detekcí jazyka, je posléze předán rodičovskému procesu. Pokud je rodičovský proces uspán, je potomkem probuzen.

5.4 Vícejazyčnost obsahu

V dnešní době se na internetu objevují články z celého světa, a proto je nutné rozpoznávat jazyk, v jakém je text na webových stránkách psaný. Navíc se zde objevují i stránky, na kterých je obsah tvořen hned v několika jazycích, typickým příkladem může být záznam rozhovoru českého novináře se slovenským politikem. Právě z těchto důvodů je pro správné rozpoznání jazyka nutné detekovat jazyk pro každý odstavec zvlášť.

Problémy detekce

Detekce jazyka je časově náročný proces a jeho použití je citelně znát na celkové době zpracování. U velmi krátkých úsecích je detekce jazyka nepřesná a při jednoslovných úsecích je schopna dojít k úplně zcestným výsledkům. Na druhou stranu, čím větší úsek je detekcí zpracován, tím je větší pravděpodobnost, že daný text se bude skládat z více jazyků.

5.4.1 Způsob detekce jazyka

Protože je detekce prováděna po odstavcích, musel být vybrán nástroj vhodný k tomuto účelu. Detekci jazyka provádí modul *langdetect* 5.2.6 a kvůli časové náročnosti a možnosti ji prováděd odděleně od zbylého zpracování textu, je prováděna paralelně s vertikalizací.

5.5 Původní systém

Původni systém je reprezentován jediným nástrojem zvaným *vertikalizátor*. Tento nástroj je rozsáhlý a zastává mnoho činností.

5.5.1 Vertikalizátor

Vertikalizátor je nástroj, který čte jednotlivé záznamy ve warc archivu, odstraňuje HTML značky a převádí text do vertikální podoby. K odstranění HTML značek využívá dříve zmíněného nástroje *jusText* 4.3.

Problémy vertikalizátoru

Samotný vertikalizátor je poměrně vhodný nástroj pro zpracování HTML stránek a jejich následný převod do vertikální podoby, ale i tak se zde vyskytlo několik problémů.

Jedním z problémů je, že při zpracování některých archivů uživatel dostane prázdný výstup a není vůbec informován, proč se tomu tak stalo. Samotný vertikalizátor využíval paměť, procesor a délka zpracování byla přiměřená, ale výsledný výstup byl pouze prázdný soubor a hlášení o chybě také nebylo nikde dostupné.

Dalším z problémů bylo špatné čtení WAR souborů. Vertikalizátor při čtení těchto souborů očekává hlavičky v dopředu daném pořadí, pokud se tak nestane, není schopen z hlavičky vyčíst potřebné informace a často zpracování končí s chybou. V rámci tohoto projektu byla tato chyba alespoň částečně opravena.

Výhody vertikalizátoru

Největší výhodou vertikalizátoru je, že už je začleněn do procesu zpracování poměrně dlouhou dobu v procesu zpracování, tzn. je řádně otestován. Co se výkonu týče, vertikalizátor klade menší zátěž na procesor.

Podporované XML tagy

Vertikalizátor podporuje následující XML tagy:

- *<doc>*
- *<head>*
- *<p>*
- *<s>*
- *<g/>*
- *<link="URL">*
- *<lenght=N>*

<doc> je párová značka a značí začátek a konec jednoho ze záznamů ve vertikálním souboru. Atributy, které jsou podporovány touto značkou, jsou *title* (titulek stránky), *id* (unikádní identifikátor stránky) a *url* (url adresa stránky, ze které pochází text).

<head> je párovou značkou, značící titulek stránky. Obsahuje ten samý text, který je obsažen v atributu *title* u tagu *<doc>*. Tentokrát je ovšem obsažený text ve vertikálním formátu. XML tag *<head>* je kompletně bez atributů.

<p> je další z párových značek. Její význam je podobný jako v HTML – značí právě jeden odstavec. Tato značka neobsahuje žádné atributy.

Párová značka *<s>* identifukuje věty v textu. Podobně jako značky *<p>* nebo *<s>* neobsahuje žádné atributy.

<g/> je nepárovou značkou, která se vkládá mezi 2 pozice, které nebyly rozděleny v původní větě mezerou.

<link="URL"> je další z nepárových značek. Používá se k identifikaci odkazů v textu. Za touto značkou je tabulátorem oddělena další z nepárovýzh značek – *<lenght=N>*. Tato značka udává, kolik předcházejích pozic bylo součástí odkazu.

Informace o používaných XML značkách byly získány z popisu vertikálního souboru [34].

5.6 Detekce jazyka

Detekce jazyka u vertikalizátoru je provedena pro každou stránku zvlášť a to až po extrakci čistého textu z webové stránky. Samozřejmně je možná jazyk stránky rozpoznat ještě před extrakcí textu, ale je to zbytečné a časově náročnější, proto se volý výše zmíněný postup.

Při zpracování českých a slovenských článků nešlo se spuštěnou detekcí jazyka docílit požadovaným výsledkům – výstup byl prázdný, opět bez jakékoliv chybové hlášky nebo informace

5.7 Nový systém

Nový systém je reprezentován třídou *Page_tokenizer*. Tato třídá využívá již vytvořených tříd *Page_reader* a *Page*. Třídu *Page_reader* využívá ke generování HTML stránek z uložených archivů. Třídu *Page* využívá nejen ke zpracování HTML získání textu a užitečných značek z HTML stránky, ale i ke kódování značek potřebných k vertikalizaci, aby během procesu tokenizace nebyly ztraceny. Obslužný skript, který pracuje se třídou *Page_tokenizer*, se nazývá *html_to_vert.py*.

5.7.1 Nové tagy a atributy

Nový systém také přišel s novými tagy a atributy, které ale mohou být vypnuty, aby výsledný formát odpovídal starším formátům.

Nové tagy jsou:

- *<h1>*
- *<h2>*
- *<h3>*
- *<h4>*
- *<h5>*
- *<img="URL">*

Tagy *<h1–5>* jsou určené k označení nadpisu odstavců. V původním systémy byly tyto nadpisy součástí odstavce a nebylo možné je rozlišit od zbytku textu v odstavci. Tyto značky jsou párovými značkami.

Tag *<img="URL">* je určen k označení obrázku v textu. URL, které je součísti samotné značky je url adresou vedoucí na obrázek. Podobně jako u značky *<link="URL">* se jedná o nepárovou značku a i za touto značkou je tabulátorem oddělena značka *<lenght=1>*. Tato značka je tam uvedena pouze proto, aby odpovídala formátu zápisu rezervovaného slova ___*IMG*___ ve vertikalizátoru, kde toto slovo sloužilo stejnému účelu jako zde značka *<img="URL">*.

Novým atribudem je zde atribut *lang*. Tento atribud může být (není to ale pravidlem), obsažen ve značce *<p>* a udává jazyk, ve kterým je odstavec napsán. Jazyk je zapsán ve kódech standardu ISO 639-1.

Kapitola 6

Tagging

Posledním procesem této práca je tagging 1.3.2. Jedná se jak o časově, tak i implementačně nejnáročnější proces.

6.1 Použité nástroje

Protože všechny nástroje zde použité již byly popsány v předchozích kapitolách, je zde podrobněji rozveden pouze program

6.1.1 MorphoDiTa

.

6.2 Gzip

Nástroj *gzip* 4.2.2 je zde použit pouze kvůli možnosti čtení dat z archivů.

6.3 Lzma

Lzma 3.2.4 je zde použita k dekompresi dat ze zdrojových archivů a ke kompresy výstupních dat.

6.4 Subprocess

Modul *subprocess* 5.2.4 je zde pužit téměř totožným způsobem jako je popsáno v kapitole zabývající se převodem textu do vertikálního formátu 5.

6.4.1 MorphoDiTa

Stěžejním nástrojem této části je nástroj *MorphoDiTa*, tento nástroj je schopen provádět několik činností – tokenizace 1.3.1, tagging 1.3.2, morfologická analýza a morfologická tvorba. V tomto projektu byl nástroj již použit k tokenizaci. Zde je použit pro provádění taggingu.

MorphoDiTa je psaná v jazyce C++, ale jsou existují i jeji verze v jazyce C#, Java, Python a Perl. V tomto projektu byla použita verze psaná v jazyce C++, nejen kvůli doporučení od autorů, ale i kvůli rychlosti.

Pro provedení taggingu je potřeba mít k dispozici jazykový model pro ten jazyk, ve kterém je psán text. Tento jazykový model obsahuje dlouhý seznam slov s pravděpodobností jejich výskytu v textu [19].

Protože je na vstupu tohoto procesu soubor ve formátu vertikálu, je potřeba nejdříve veškeré vložené značky dočasně odstranit a teprve až po jejich odstranení je text předán na zpracování. Zpracovaný text je následně doplněn o původní značky.

Úprava zdrojových kódu nástroje MorphoDiTa

Jedním z možných způsobů, jak zefektivnit proces taggingu a tokenizace je upravit zdrojový kód nástroje *MorphoDiTa*. Pokud by se podařilo nástroj přinutit ignorovat značky vertikalizátoru, odpadl mohl by se do nástroje vložit samotný vertikální soubor a nemuseli se tagy složitě dekódovat, odebírat a poté vracet zpět na své místo. Nicméně samotní autoři nástroje *MorphoDiTa* slíbili, že se tímto problémem budou zabívat a je to jedna z věcí, které plánují v tomto nástroji implementovat.

6.5 Paralelní zpracování

Způsob paralelního zpracování zde funguje na podobném principu (včetně použitých postupů k vyřešení problémů paralelního zpracování 5.3.2), jako u převodu textu do vertikální podoby.

Hlavní rozdíl je při samotném spuštění nástroje *MorphoDiTa*. Protože, jak již bylo zmíněno dříve, je zde potřeba jazykového modelu, musí se tento model také před samotným zpracováním načíst do paměti nástoje *MorphoDiTa*. Zpracování jazykového modelu klade na procesor zátěž a z časového hlediska je zpracování dlouhé 1 – 5 sekund, v závislosti na aktuální procesorové zátěži stroje a na samotném stroji, kde je proces spouštěn. Tento časový úsek byl zjištěn pomocí měření prováděného na serverech *athena* a *knot*. Právě kvůli nutnosti zpracování jazykového modelu při každém spuštění aplikace je ideální zařídit, aby aplikace byla spuštěna co nejméněkrát, ideálně jednou. Na druhou stranu, při nízkém počtu spuštění se zde objevuje problém dlouhé doby zpracování. Relativně dlouhou dobu zpracování lze řešit spuštěním nástroje na více procesech, tehdy ale rapidně narůstá paměťová náročnost.

Pro maximální využití výkonu stroje je zde i možnost spouštět několik procesů provádějící tagging zároveň. Rodič zde načítá vertikální soubor do paměti a při načtení určitého počtu záznamů (tento počet lze změnit i během procesu zpracování) předá načtená data do fronty, ze které si je vyjmou potomci. Potomek poté spustí tagger a zatímco tagger zpracovává jazykový model, potomek zbavý text značek, které vytvořil vertikalizátor a tokenizovaný text předají taggeru. Potomek počká na provedení taggingu, vrátí odebrané značky zpět do textu a výsledek uloží do fronty, ze které si to následně přečte rodičovský proces.

6.6 Srovnání s aktuálním systémem

Původní systém je zaměřený na zpracování anglických textů, ale tato práce je zaměřena zejména na zpracování vícejazyčných textů. V současném systému lze tedy teoreticky zpracovat text psaný v jakémkoliv jazyce, pokud je k dispozici jazykový model pro daný jazyk.

6.7 Nový systém

Tagger je reprezentován třídou *Page_tagger* implementovanou v souboru *Page_tagger.py*. Obsulžný skript pro tuto se jmenuje *tagger.py*.

Kapitola 7

Spuštění zpracování a tvorba obslužných skriptů

Pro vytvoření pracovní složky projektu je napsán shellovský skript *configure.sh*, který zkontroluje existenci všech potřebných částí zpracování. K některým z vytvořených knihoven také existuje obslužný skript. Tyto skripty jsou součástí tohoto projektu, ale uživatel je nemusí nutně použít. V této kapitole bude popsána i tvorba těchto obslužných skriptů.

7.1 Instalace knihoven jazyka Python

Pro instalaci knihoven jazyka Python se doporučuje použití konfiguračního skriptu. Pokud ale není použití tohoto skriptu chtěné k instalaci knihoven je možné provést pomocí nástroje *pip3*. Tato práce je implementovaná a určená pro linuxová zařízení.

Programy, které je potřeba nainstalovat:

- python3
- GNU WGET

Knihovny jazyka Python, které je potřeba nainstalovat:

- warc
- warc3
- langdetect

7.2 Instalace nástroje MorphoDiTa

Nástroj *MorphoDiTa* lze stáhtou z stáhnout pomocí nástroje *git* nebo ze stránek GitHubu a to ve dvou podobách.

První z možností je stáhnout zdrojové kódy nástroje a ty na samotném stroji přeložit. Druhou z možností je stažení samotných binárních souborů.

Ke spuštění taggingu je také potřeba mít jazykový model. Jazykový model lze stáhnout ze stránek repozitáře LINDAT/CLARIN. Momentálně jsou k dispozici 2 jazykové modely – Český a Anglický. Jazykový model pro slovenštinu

Při instalaci nástroje pomocí konfiguračního skriptu budou staženy již přeložené binární soubory.

7.3 Automatické zpracování

7.3.1 RSS zdroje

Pro kolekci webových stránek je potřebné mít seznam RSS a ATOM zdrojů. V tomto projektu jsou připojeny seznamy RSS pro stahování českých a slovenských zpravodajských deníků, stahování blogů, stahování komentářů ke článkům, anglických zpravodajských deníků a ostatních stránek které využívají technologií ATOM nebo RSS. Tyto zdroje se nacházejí ve složce *rss_sources*.

Stahování čekých a slovenských deníků je uloženo v soubory *cs_media.txt*. V tomto souboru se také nachází zdroje pro stahování komentářů k těmto článků.

Anglické zpravodajské deníky jsou uloženy v souboru *newsbrief_eu*.

Blogy a komentáře k nim jsou uloženy v souboru *blogs.txt*.

A ostatní RSS a ATOM zdroje které byly nalezeny jsou uloženy v souboru *others.txt*.

7.3.2 Automatické vyhledávání nových zdrojů

TODO

7.3.3 Manuální vložení nových zdrojů

Pokud je třeba manuálně vložit nové zdroje, je možné je přidat do již existujících RSS zdrojů nebo vytvořit nový soubor a uložit je do něj. Soubor by měl mít vždy koncovku *.txt*. Tento nový soubor bude načten a zařazen do zpracování při dalším cyklu kolekce dat.

7.3.4 Soubory s odkazy určenými ke stažení

Ke kolekci odkazů z RSS zdrojů a jejich deduplikaci je určena složka *collected_links*. Pro každý zdrojový soubor ve seložce *rss_sources* je zde vytvořena stejnojmená složka (bez koncovky *.txt*). Do těchto složek se ukládají nalezené okdazy na nové články. Soubory s koncovkou *.downloaded* jsou soubory, u kterých již bylo provedeno stažení a soubory končící koncovkou *.collected* jsou ty souboty, které ještě nebyly zpracované nástrojem pro stahování.

Manuální vložení nových odkazů ke stažení

Pokud je potřeba do procesu manuálně vložit nové odkazy, rozhodně není dobrým nápadem je vkládat do již existujících souborů. Tyto soubory mohou být právě některým z programů zpracovávány. Nové odkazy ke stažení je nutné vložit do nového souboru umístěného v patřičné složce. Tyto soubory budou zpracovány během dalšího cyklu stahování odkazů.

7.3.5 Vertikalizace

TODO

7.3.6 Tagging

TODO

7.4 Manuální zpracování

Pokud není automatické zpracování vhodné nebo potřebné, zle data zpracovat i manuálně.

7.4.1 Kolekce odkazů

Kolekci odkazů lze provést pomocí připraveného skriptu *link_collector.py*. Práci tohoto skriptu lze nastavit pomocí relativně velkého množství parametrů. Všechny parametry jsou volitelné.

Argumenty programu:

- *−i, −−input* Cesta k souboru nebo souborům, které obsahují adresy na RSS nebo ATOM zdroje. Výchozí hodnotou je standartní vstup.
- *−o, −−output* Výstupní soubor s okazy ke stažení. Výchozí hodnotou je standrtní výstup
- *−e, −−errors* Logovací soubor, kde budou ukládány varování a chyby během zpracování. Výchozí hodnotou je chybový výstup.
- *−a, −−append* Výstupní soubor nebude při otevření smazán, ale nové odkazy se zapíší na jeho konec.
- *−h, −−help* Zobrazí nápovědu k programu.
- *−w, −−wait* Tento argument určuje maximální délku čekání na odpověď serveru. Výchozí hodnota je 1 sekunda.
- *−t, −−test* Namísto, aby proběhla kolekce odkazů, budou zdrojové odkazy ověřeny, zda opravdu vedou na RSS nebo ATOM zdroje.
- *−m, −−multiprocessing* Tento parametr udává, kolik dalších procesů bude provádět kolekci odkazů. Výchozí hodnotou je 0 – kolekce probíhá sekvenčně.

7.4.2 Stahování stránek

Ke stahování webových stránek je určen skript *page_downloader.py*. Program má pouze jeden povinný argument – výstupní soubor. Pokud bude vstupní soubor mít koncovku *.xz*, bude použit algoritmus *lzma* 3.2.4 ke kompresi dat.

Argumenty programu:

- *−i, −−input* Cesta k souboru nebo souborům, které obsahují adresy stránek ke stažení. Výchozí hodnotou je standartní vstup.
- *−o, −−output* Výstupní soubor, kam budou uloženy stažené stránky. Soubor by měl vždy mít koncovku .warc nebo .warc.xz.
- *−e, −−errors* Logovací soubor, kde budou ukládány varování a chyby během zpracování. Výchozí hodnotou je chybový výstup.
- *−h, −−help* Zobrazí nápovědu k programu.

- *−w, −−wait* Tento argument určuje maximální délku čekání na odpověď serveru. Výchozí hodnota je 1 sekunda.
- *−p, −−pause* Tento argument určuje minimální délku čekání na mezi dotazy na jednu doménu. Výchozí hodnota je 1 sekunda.
- *−m, −−multiprocessing* Tento parametr udává, kolik dalších procesů bude provádět kolekci odkazů. Výchozí hodnotou je 0 – kolekce probíhá sekvenčně.

7.4.3 Převod textu do vertikálního formátu

Pro tento účel je vytvořek skript *html_to_vert.py*. Ke spuštění tohoto programu je nutné mít k dispozici binární soubor tokenizeru. Tento binární soubor se jmenuje *run_tokenizer* a je součástí nástroje *MorphoDiTa*.

Argumenty programu:

- *−i, −−input* Cesta k archivům s webovými stránkami. Soubory musí být ve fromátu WARC a mohou být komptimovány nástroji gzip a xz.
- *−o, −−output* Výstupní soubor, kde budou uloženy zpracované stránky. Soubor by měl vždy mít koncovku .vert nebo .vert.xz.
- *−h, −−help* Zobrazí nápovědu k programu.
- *−t, −−tokenizer* Cesta k binárnímu souboru tokenizeru.
- *−f, −−format_new* Budou značeny nadpisi i obrázky.
- *−m, −−multiprocessing* Tento parametr udává, kolik dalších procesů bude provádět tokenizaci. Výchozí hodnotou je 10.

7.4.4 Tagging

Tagging je možné provést pomocí skriptu *tagger.py*. Pro použití toho programu je zapotřebí mít jazykový model a binární soubor taggeru. Potřebný binární soubor se jmenuje *run_tagger* a je součástí nástroje *MorphoDiTa*.

Argumenty programu:

- *−i, −−input* Cesta k archivům s webovými stránkami. Soubory musí být ve fromátu WARC a mohou být komptimovány nástroji gzip a xz.
- *−o, −−output* Výstupní soubor, kde budou uloženy zpracované stránky. Soubor by měl vždy mít koncovku .vert nebo .vert.xz.
- *−h, −−help* Zobrazí nápovědu k programu.
- *−t, −−tagger* Cesta k binárnímu souboru tokenizeru.
- *−l, −−languagemodel* Cesta k jazykovému modelu.
- *−m, −−multiprocessing* Tento parametr udává, kolik dalších procesů bude provádět kolekci odkazů. Výchozí hodnotou je 10.

7.5 Změna použitých nástrojů

V tomto projektu se použité nástroje často měnily. Aby výměna jednoho nástroje neznamenala přepracování všech zdrojových kódů, byl projekt rozdělen na dílčí části. Každá z těchto dílčích částí řeší různé problémy zpracování.

7.5.1 Kolekce a stahování odkazů

Pokud je třeba změnit nástroje pro kolekci a stahování je potřeba tyto nové nástroje umístit do modulu *Page_downloader.py*. V tomtu modulu se nachází implementace metody *get_page_from_url*, která se stará o stažení jedné stránky. V tomto modulu je dále implementováno víceprocesorové stahování a logování.

Zpracování RSS a ATOM souboru

Modul, který řeší extrakci odkazů z RSS a ATOM souborů se nazývá *Link_collector.py*. V tomto modulu se nachází implementace metod pro práci s RSS a ATOM soubory. Pokud bude třeba změnit způsob, jak se vyhledávají zdrojové odkazy v těchto souborech, implementace vyhledávání se na chází v metodě *find_links*. Pokud bude potřeba upravit hledání odkazů na RSS a ATOM zdroje uvnitř HTML stránky, je zapotřebí upravit metodu *find_rss_from_page*.

Stáhování webových stránek

Pro stahování stránek byl vytvořen modul *Page_generator.py*. Protože veškeré stahování je řešeno již v modulu *Page_downloader.py*, bylo zde potřeba pouze implementovat iterátor, který zajišťuje určitou časovou mezeru mezi dvěmi dotazy na jeden server.

7.5.2 Převod textu do vertikální podoby

V této části je poměrně pravděpodobné, že dojde k výměně některých nástrojů. Aby přidání nebo náhrada nástroje proběhla korektně, je poteba dodržet několik pravidel, které jsou popsané níže.

Extrakce textu a důležitých entit z HTML souboru

K extrakci textu a některých HTML entit z webové stránky slouží modul *Page.py*. Pokud je potřeba změnit způsob extrakce textu z HTML stránek, je potřeba změnit metodu *get_text* a odstranit nebo změnit metody k ní přidružené.

Převod textu do vertikálního formátu a rozpoznávání jazyka

V souboru *Page_tokenizer.py* se nachází implementace třídy *Page_tokenizer*. Tato třída má definovanou metodu *_tokenizer*, ve které je zdrojový kód který je použit k implementaci činnosti potomků. Mimo jiné se zde i nachází spuštění tokenizeru, který poskytuje nástroj *MorphoDiTa*. Pokud je potřeba změnit použitý nástroj, musí se zde nastavit paramentry pro nový nástroj. Je možné, že bude potřeba změnit i formátování vstupních a výstupních dat nového nástroje. Formátování těchto dat je také implemetováno v této metodě.

Při změne nástroje určeného k rozpoznávání jazyka je nutné změnit metodu *_translator*. Narozdíl od metody *_tokenizer* je tato metoda statická.

7.5.3 Tagging

Tagging je implementován v modulu *Page_tagger.py*. Podobně, jako v modulu *Page_tokenizer.py*, je i zde proces taggingu reprezentován jedinou metodou. Tato metoda nese název *_tagger_tokenized* a nachází se v ní formátování vstupních a výstupních dat pro tagger, který poskytuje nástroj *MorphoDiTa*. Samotný tagger je zde i spouštěn.

7.6 Tvorba obslužných skriptů

Protože každá z vytvořených tříd má předem daný účel a řeší malou část celkového problému, je tvorba skriptu, který s touto třídou schopen pracovat, poměrně jednoduchá. Účel obslužného skriptu je použít danou třídu takovým způsobem, aby vyřešila problém, ke kterému byla vytvořena. Je ale potřeba dbát na jednoduchost a víceúčelovost skriptu, aby vytvořený skript byl použitelný. Objekty těchto tříd lze vytvořit s různými parametry a tyto pramatery výrazně ovlivňují dobu zpracování, zátěž na procesor a paměťovou náročnost. Protože se předem neví, na jakém stroji bude skript spuštěn, měla by existovat možnost, jak ovlivnit celkovou náročnost programu. K tomuto účelu často slouží parametry programu.

Velkým problémem často bývá kopírování kódů mezi skripty nebo lenost programátora zpracovávat velké množství argumentů. To způsobuje, že vznikají skripty, kde záleží na pořadí parametrů a tím se zvyšuje náročnost použití programu s každým parametrem, které ten program je schopen použít. Kopírování kódu zase zůsobuje to, že vypsaná nápověda k programu bývá zcestná, pokud ji autor nového skriptu neupraví, nebo se zkopírují i komentáře, popřípadě i chyby.

Další důležitou částí je jednodný způsob logování chyb. Pokud každý ze skriptů bude dodržovat jistý způsob logování a zároveň použije vhodný formát, lze tyto chyby procházet i automatickými skripty a vyhodnotit, kterým chybám je třeba věnovat pozornost.

V neposlední řádě je třeba zmínit i fakt, že když bude existovat desítky různých obslužných skriptů a každý se bude používat jiným způsobem, bude to pro jejich uživatele přinejmenším matoucí.

7.6.1 Funkce zjednodušující tvorbu skriptů

Pro řešení výše popsaných problému bylo vytvořeno několik funkcí, které několikanásobně (v mém případě) zefektivnili tvorbu skriptů a to nejen pro práci s těmito moduly. Tyto funkce jsou umístěny v modulu *Functions.py*.

Zpracování argumentů

Pro zpracování argumentů byla vytvořena metoda *get_settings*. Tato metoda má 2 argumenty. Prvním je seznam slovníků a druhým je seznam argumentů programu. První argument, seznam slovníků, definuje všechny podporované argumenty. Každý ze slovníků v poli reprezentuje právě jeden z argumentů. Slovník má následující klíče:

- *names* – seznam jmen, pomocí kterých lze argument zadat
- *optional* – určuje, zda je argument volitelný nebo ne
- *has_tail* – určuje zda má argument 0, 1 nebo N parametrů
- *word_index* – index, pod kterým lze zadané hodny zpřístupni v hashi vráceném funkcí *get_settings*

- *prerequisite* – prerekvizity nebo podmínky, za kterých je možné tento argument zadat
- *description* – popis zadaného argumentu, který bude použit pro automatické generování nápovědy

Prerekvizitou mohou být hodnoty *word_index* ostatních argumentů. V takovém případě se očekává, že daný argument lze použít pouze tehdy, pokud je program spuštěn s argumentem zadaným v prerekvizitě. Další užitečnou podmínkou je řetězec *___alone___*, který značí, že pokud je argument zadán, tak musí být jediným zadaným argumentem.

Klíč *has_tail* může nabývat hodnot 0, 1 nebo 2. V ostatních případech chování není definováno. V případě hodnoty 0 se očekává, že argument nemže mít žádné parametry. Pokud je hodnotou 1, argument může nabývat právě jednoho parametru. V případě hodnoty 2 má argument minimálně minimálně 1 parametr.

Návratovou hodnotou funkce je slovník, kde klíči jsou hodnoty *word_index* u takových argumentů, se kterými byl program spuštěn. Hodnoty těchto klíčů je vždy seznam zadaných parametrů.

Pokud není program spuštěn s argumenty, které mají nastavené *optional* jako *False*, jsou zadány argumenty, které nejsou definovány, není splněn počet parametrů argumentu nebo nejsou splněny prerekvizity, je vyvolána výjimka.

Příklad užití V tomto příkladě je ukázka použití zpracování argumentů funkce. Jedná se o skript, má jeden poviný argument – $-i$. Tento argument lze zadat i jako $--input$. Tento argument má 1 – N parametrů. Program má nadále volitelný parametr $-o$, který je možné zadat jako $--output$. Výstupní soubor může být identifikován pouze jedním parametrem. Žádný z těchto argumentů nemá nastavené prerekvizity.

```
import sys
import lzma
from Functions import get_setting
possible_arguments = [
    {
        'names' : [ '--input', '-i' ],
        'optional' : False,
        'has_tail' : 2,
        'word_index' : 'input',
        'prerequisite' : None,
        'description' : 'Cesta ke vstupnim souborum'
    },
    {
        'names' : [ '--output', '-o' ],
        'optional' : True,
        'has_tail' : 1,
        'word_index' : 'output',
        'prerequisite' : None,
        'description' : 'Vystupni soubor. Pokud neni zadan, bude pouzit standardni vystu
    }
]
settings = get_setting( possible_arguments, sys.argv[1:] )
out = sys.stdout
```

```
if ( 'output' in settings ):
    if ( settings[ 'output' ][0].endswith('.vert') ):
        out = open( settings[ 'output' ][0], 'wb' )
    else:
        out = lzma.open( settings[ 'output' ][0], 'wb' )

input = settings[ 'input' ]
for f in input:
    pass # zpracovani jednotlivych souboru
```

7.6.2 Logování

Pokud nastane jakákoliv chyba, je potřebné znát o ní co nejvíce informací. K tomu slouží funkce *get_exception_info*. Tato funkce je umístěna v modulu *Functionst.py* a má jeden volitelný argument. Očekává se, že tento argument je řetězcem a je určen k bližšímu popisu chyby. Návratovou hodnotou funkce je řetězec, ve kterém je umístěna informace zadaná argumentem funkce a k ní připojen zásobník volání funkcí s čísly řádků kódu a jmen souborů, ve kterých chyba nastala.

```
try:
    text = text.decode( 'utf-8' )
except:
    error_info = Functions.get_exception_info( 'Nepodarilo se dekodovat text.' )
```

Formát logovacího souboru

Logovací soubory této práce jsou uzpůsobeny k automatickému zpracování. Logují se chyby, varování a ostatní výpisy (ladící výpisy, statistické údaje, apod.).

Každou nalezenou chybu je možné v souboru poznat tak, že řádek souboru začíná znaky `-ERR`[`mezera`]. Za tímto řetězcem na tom samém řádku následuje stručný popis chyby. Dalším řádkem může (ale není to pravidlem) být podrobnější popis chyby. Podrobnější popsi chyby začíná znaky `-ERR_STAR`[`LF`], kde LF značí konec řádku (ascii hodnota 10). Popis je ukončen znaky `-ERR_END`[`LF`].

Podobně jako chyby se zapisují i varování. Varování lze identifikovat pomocí znaků `-WARN`[`mezera`], za kterými následuje stručný popis. Podrobný popis je zde opět volitelný. Podrobný popis je uvozen řádkem `-WARN_STAR`[`LF`] a je ukončen řádkem `-WARN_END`[`LF`].

Úplně stejný princip je použit i u ostatních logovacích výpisů. Rozdíl spočívá jen v tom, že výpis je identifikován řádkem `+OK`[`mezera`], za kterým opět následuje stručný popis. Volitelný podrobný popis se nachází mezi řádky `+OK_STAR`[`LF`] a `+OK_END`[`LF`]

```
+OK spoustim prikaz "./page_downloader.py -i ./test/cs_media/2018-04-10.txt ./test/cs_m
-ERR Vlakno ( id 1, projekt cs_media) - Kriticka chyba behem stahovani stranek.
-ERR_START
Program "./page_downloader.py" byl spusten nekortnim zpusobem - chyba pri zpracovani ar
File "./page_downloader.py", line 92, in <module>
```

```
  settings = get_setting( possible_arguments, sys.argv[1:] )
File "/mnt/sdb1/FIT/bachelor-thesis/source_codes/Functions.py", line 99, in get_setting
  raise Exception( 'Argumenty: "' + error_str +'" jsou povinne. Spuste s argumentem "--
Exception: Argumenty: "--output" jsou povinne. Spuste program s argumentem "--help" pro
-ERR_END
+OK spoustim prikaz "./page_downloader.py -i ./test/newsbrief/2018-04-10.txt ./test/new
```

Kapitola 8

Měření výkonu zpracování

V této kapitole budou uvedena různá statistická data, která byla nasbírána v průběhu testování nástrojů projektu.

8.1 Stahování stránek

Pro měření rychlosti stahování stránek bylo náhodně vybráno 10 000 odkazů. Doba čekání na odpověd serveru byla nastavena na 5 sekund a minimální časová mezera mezi přístupy na jeden a ten samý server byla 1 sekunda. Komprese uložených dat byla do tohoto času také zahrnuta. Použitý modul pro kompresi byl *lzma* 3.2.4.

8.1.1 Původní systém

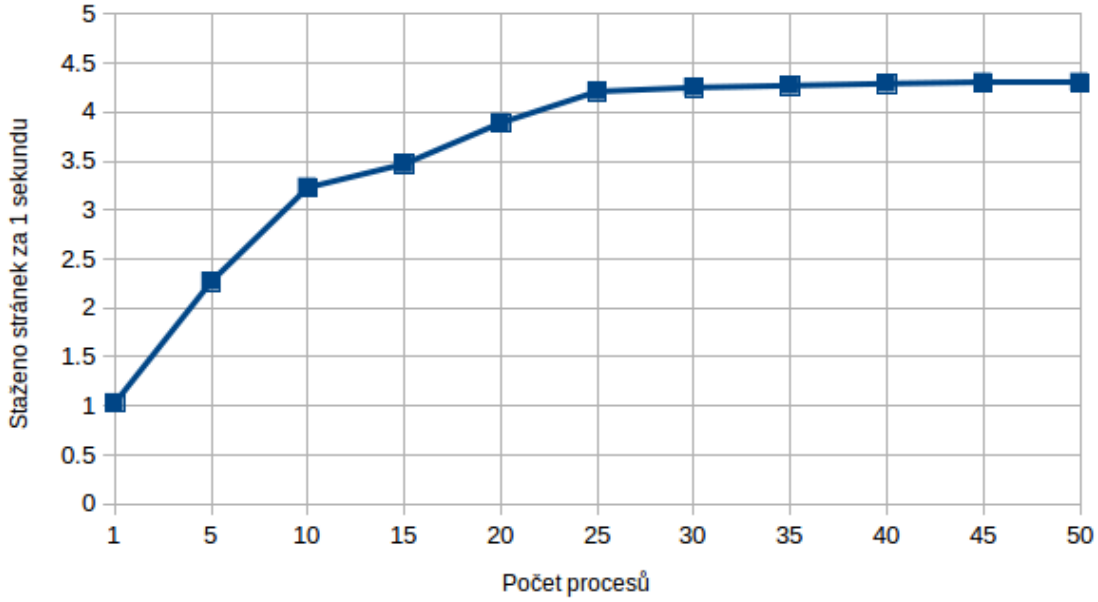
Původní systém ke stahování využívá pouze jednoho vlákná a nemá nastavenou dobu čekání na odpověď serveru, což vede k dlouhému a zbytečnému čekání na odpověď. Z těchto důvodů byla průměrná rychlost stahování 0.76 stránek za sekundu. Celková doba stahování byla 3 hodiny, 38 minut a 27 sekund. Protože během čekání na odpověď je proces uspán, bylo spotřebováno procesorového mnohem méně – 2 hodiny, 16 minut a 58 sekund. Celkem se pomocí tohoto nástroje podařilo stáhnout 8476 stránek.

8.1.2 Nový systém

Pro nový systém nebylo žádným problémem zpracovat takové množství odkazů. Stahování trvalo celkem 38 minut a 48 sekund. Protože stahování běželo vícevláknově, bylo spotřebovány 2 hodiny, 27 minut a 40 sekund procesorového času. Maximální počet procesů pro stahování byl nastaven na 50. Celkem se podařilo stáhnout 8 734 stránek.

Vliv počtu procesů na dobu zpracování

Vliv počtu procesů na stahování má obrovský vliv na celkou dobu zpracování. V ideálním by rychlost stahování stoupala lineárně spolu s počtem procesů. Protože rychlost je omezena i minimální dobou přístupu na danou doménu, lineárního nárustu na reálných datech nikdy nedosáhneme. Graf 8.1 znázorňuje výše zmíněnou skutečnost.



Obrázek 8.1: Vliv počtu procesů na počet stažených stránek za jednu sekundu

8.2 Nástroje pro kompresi dat

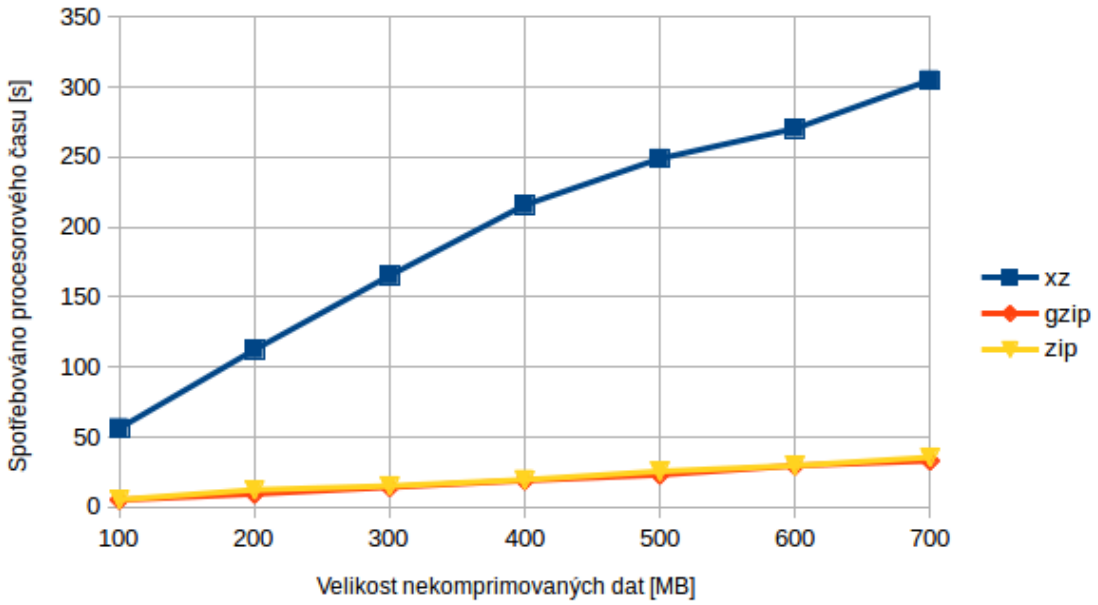
Během výběru vhodného nástroje pro kompresi bylo třeba rozhodnout který nástroj je lepší použít. Doposud byly používány 2 nástroje ke kompresi dat – *gzip* a *xz*, který využívá algoritmu *lzma*. Pro zajímavost byl ke srovnání přídán i populární nástroj *zip*. Všechny nástroje byly spuštěny s nejvyšší úrovní komprese dat.

Vstupnímy daty pro tyto nástroje bylo 7 souborů o velikosti 100 MB, 200 MB, 300 MB, 400 MB, 500 MB, 600 MB a 700 MB. Všechny tyto soubory obsahovaly stažené webové stránky ve formátu WARC 3.5.2. Měření probíhalo na stroji athena1.

Cílem měření výkonosti těchto nástrojů a efektivnosti algoritmů, které používají, bylo zvolit nejvhodnější nástroj pro kompresi. Měřenými daty byla náročnost na procesor během komprese a dekomprese dat a výsledná velikost archivu.

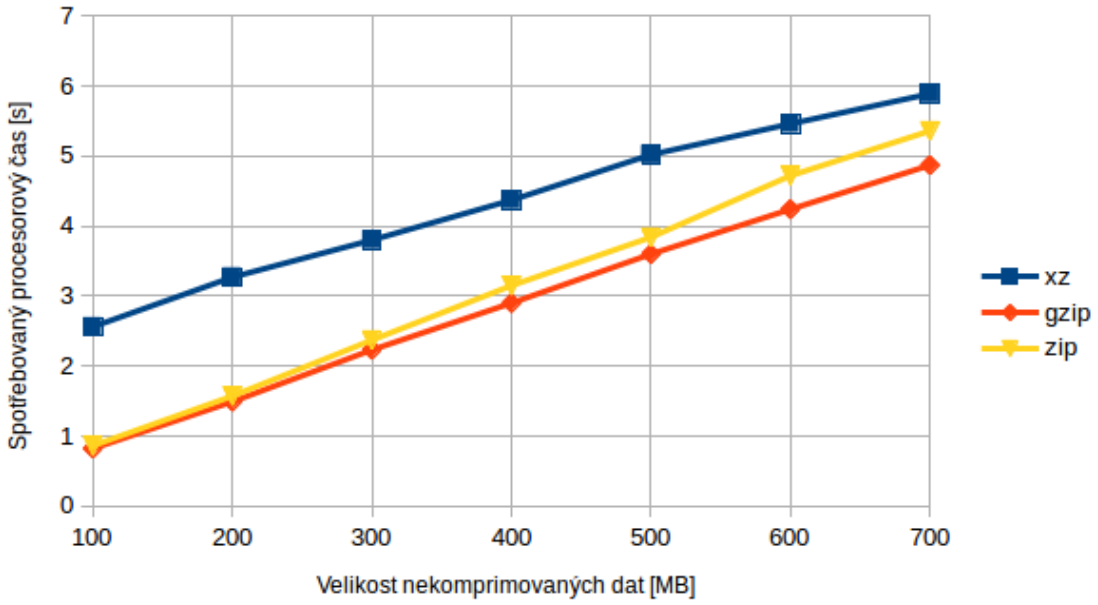
8.2.1 Výsledky měření

Na grafu 8.2 lze vidět zatížení procesorů během komprese dat. Nástroj *gzip* ve všech případech předčil nástroj *zip*, přestože oba nástroje měli velice podobné výsledky. Nástroj *xz* klade výrazně větší zátěž na procesor a zpracování tak trvá mnohem déle.



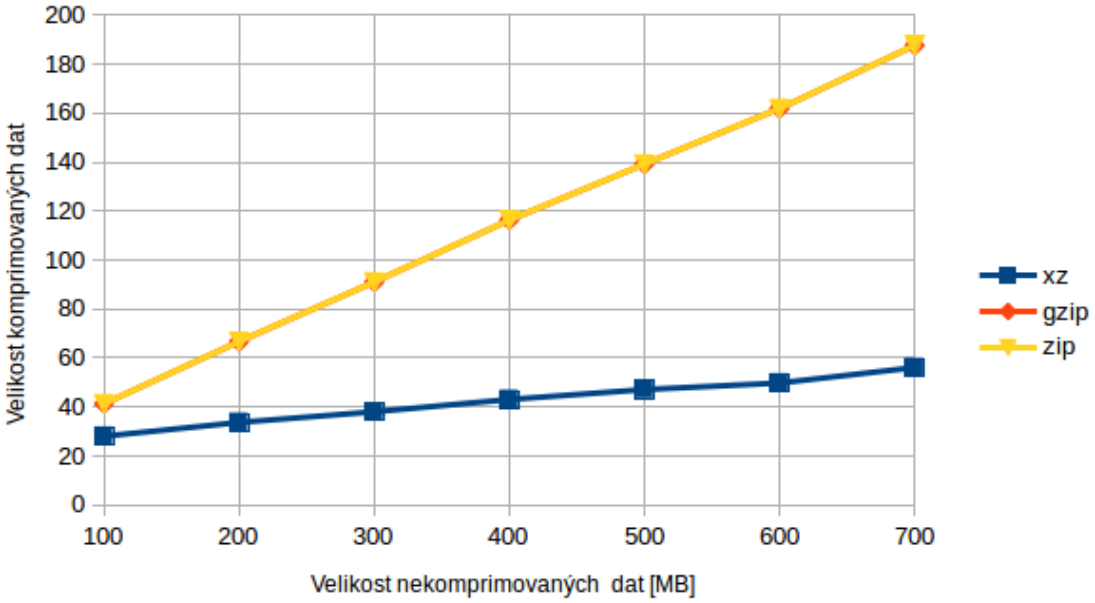
Obrázek 8.2: Procesorové zatížení během komprese dat

Graf 8.3 vizualizuje výsledky měření náročnosti dekomprese na procesor. Dekomprese je mnohem méně náročná a největší úspěch zde má opět nástroj *gzip*, přestože *zip* dosahuje podobných výsledků. Nástroj *xz* má opět nejhorší výsledek, ale faktem je, že čím větší je velikost dat, tím se jeho celková náročnost blíží k náročnosti ostatních nástrojů.



Obrázek 8.3: Procesorové zatížení během dekomprese dat

Důvodem, proč byl vybrán nástroj *xz* ke kompresi dat byla výsledná velikost archivů. V grafu 8.4 je vidět, jak moc v tomto ohledu předčil nástroje *gzip* a *zip*. Nástroje *gzip* a *zip* dosáhly v tomto měření téměř totožných výsledků.



Obrázek 8.4: Výsledná velikost komprimovaných dat

8.3 Získání čistého textu, tokenizace a vertikalizace

Cílem k efektivní práci těchto částí projektu je potřebna najít kompromis delší dobou zpracování nebo vysokou paměťovou náročností. S počtem procesů roste rychlost zpracování, ale i paměťová náročnost. Vše má své hranice a záleží hlavně na stroji, kde je process spuštěn, při kolika procesech začne být zpracování neefektivní z důvodu nedostatku paměti nebo neschopnosti efektivně obsluhovat velké množství procesů. Abych mohl rychlost zpracování porovnost s původním systémem, sáhl jsem po starších webových kolekcí, které byly přizpůsobeny pro nástroj vertikalizátor.

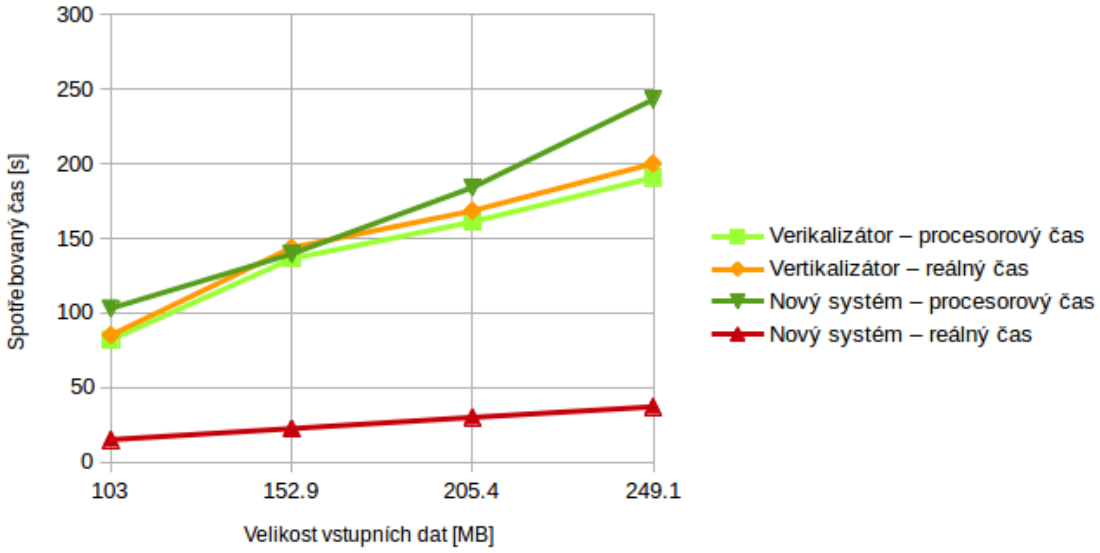
Vešekeré měření, včetně srovnání *vertikalizátoru* 5.5.1 a současného systému, probíhalo na serveru athena1.

8.3.1 Srovnání systémů

Pro srovnání systému byly vybrány balíčky o přibližné velikosti 100 MB, 150 MB, 200 MB a 250 MB. Oba nástroje byly spuštěny s vyplou detekci jazyka a nový systém byl spuštěn na 12 procesech (tento počet byl zvolen, protože byl v době tvorby této práce použit v automatickém zpracování. Měřenými hodnotami byla reálná doba zpracování a procesorová náročnost.

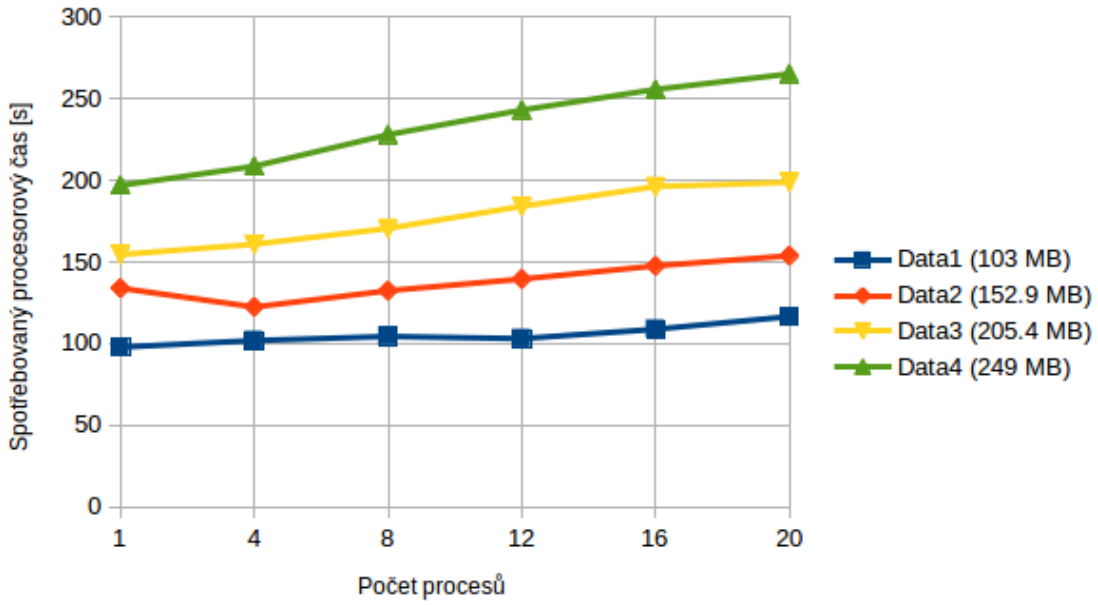
Graf 8.5 ukazuje výsledky srovnání obou nástrojů. Co se procesorové náročnosti týče, nový systém má vě většině případů horší výsledky než původní vertikalizátor. Nový systém ale nepochybně předčí ten starý v reálném čase (dokud má stroj dostatek paměti a více

jader na procesoru nebo samotných procesorů). Pokud je potřeba oba systémy při spuštění na jednom procesoru, lze z grafů 8.7 a 8.6 vyčíst, že nový systém lehce zaostává za tím původním.



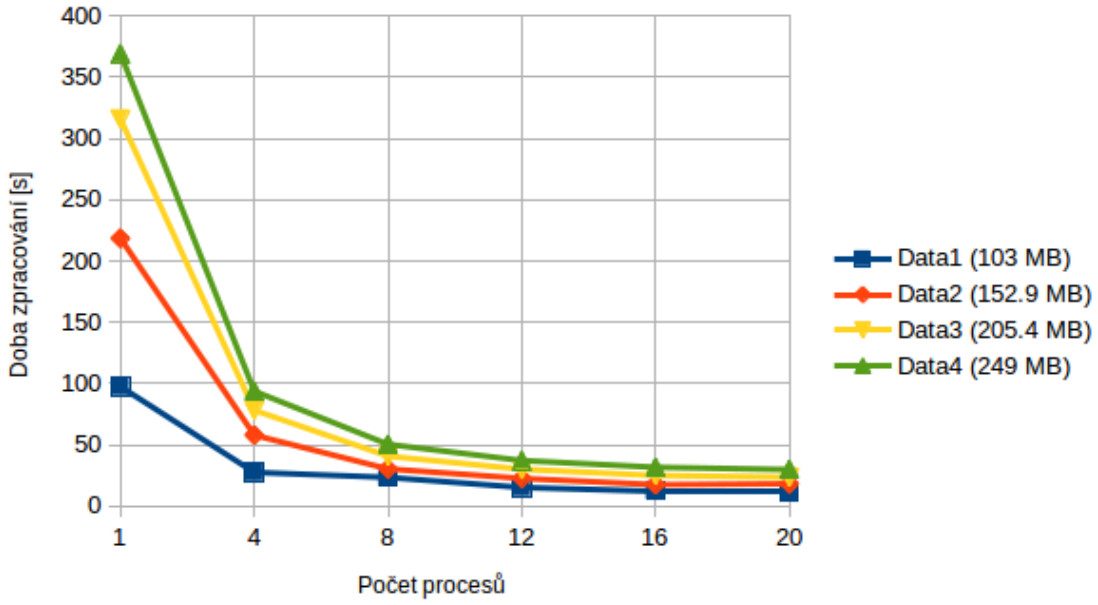
Obrázek 8.5: Srovnání původního vertikalizátoru a nového systému z hlediska časové a procoserové náročnosti.

Na grafu **??** je názorná ukázka toho, jak velký vliv má počet procesorů na celkovou rychlost zpracování. Z grafu také vyplívá, že čím více je použito procesů, tím méně naroste výkon s dalším přidaným procesem. Z toho důvodu je zbytečné spouštět program s přehnaně velkým počtem prosů, byť by na to stroj mohl mít výpočetní kapacitu.



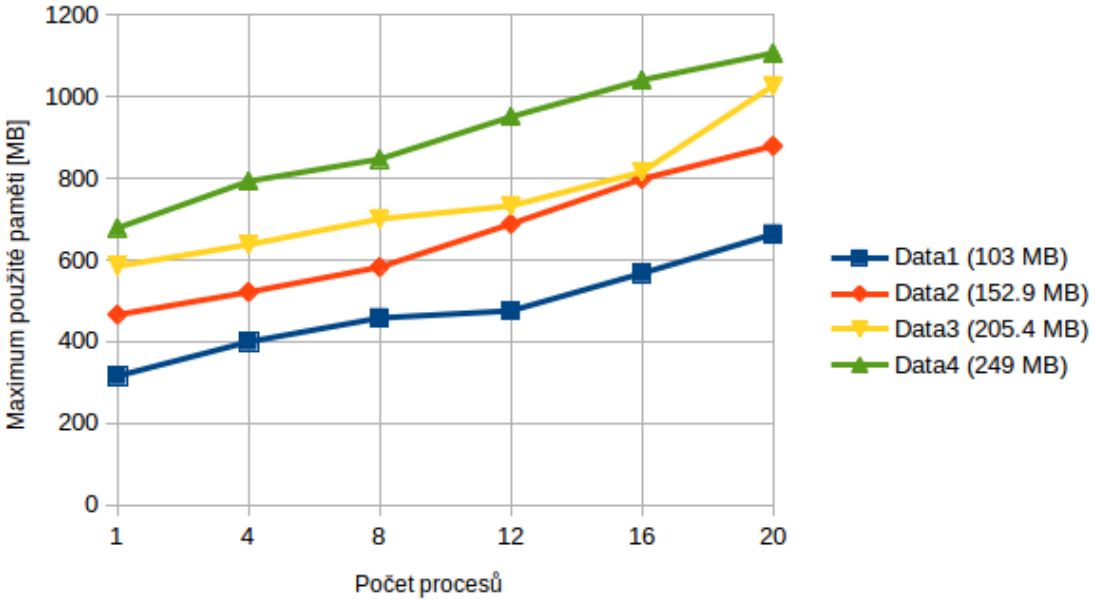
Obrázek 8.6: Vliv počtu procesů na celkovou procesorovou náročnost zpracování novým systémem.

Vliv počtu procesů na celkovou spotřebu procesorového času není vůbec drastický. Vypočítaná data jsou znázorněna grafem 8.6. Zajímavé je, že čím větší je velikost zdrojových dat, tím větší vliv má i počet procesů na celkovou náročnost výpočetu.



Obrázek 8.7: Vliv počtu procesů na celkovou časovou náročnost zpracování novým systémem.

Cenou za vyšší počet procesů je ale paměťová náročnost. Vztah počtu procesů na maximální využití paměti během zpracování dat je znázorněn na grafu 8.8. Na grafu je znázorněno maximální využití paměti během procesu zpracování.



Obrázek 8.8: Vliv počtu procesů na celkovou paměťovou náročnost zpracování novým systémem.

Kapitola 9

Závěr

Literatura

[1] adblockplus.org: *Writing Adblock Plus filters*. eyeo GmbH, [Online], [rev. 2017], [cit. 11. 4. 2018].
URL https://adblockplus.org/en/filters

[2] BELICA, M.: *jusText*. Faculty of Informatics Masaryk University, [Online], [cit. 13. 4. 2018].
URL http://corpus.tools/wiki/Justext/Algorithm

[3] BELICA, M.: *jusText algorithm*. Faculty of Informatics Masaryk University, [Online], [cit. 13. 4. 2018].
URL http://corpus.tools/wiki/Justext

[4] BEVENDORFF, J.: *ChatNoir*. GitHub, Inc, [Online], [rev. 16. 11. 2017], [cit. 20. 4. 2018].
URL https://github.com/chatnoir-eu

[5] BIRD, S.; KLEIN, E.; LOPER, E.: *Natural Language Processing with Python*. Sebastopol, USA: O'Reilly Media, Inc., 2009, ISBN 0596555717, 179 – 180 s.

[6] BORISOV, A.: *MyHTML – a pure C HTML parser*. Free Software Foundation, [Online], [rev. 23. 2. 2018], [cit. 11. 4. 2018].
URL https://github.com/lexborisov/myhtml

[7] CHITIPOTHU, A.; IBRAHIM, N.: *warc: Python library to work with ARC and WARC files*. Free Software Foundation, [Online], Verze 8f05a000, [rev. 2012], [cit. 13. 4. 2018].
URL http://warc.readthedocs.io/en/latest/

[8] DANILAK, M. M.: *langdetect 1.0.7*. Python Software Foundation, [Online], [rev. 3. 10. 2016], [cit. 15. 4. 2018].
URL https://pypi.python.org/pypi/langdetect

[9] docs.python.org: *gzip – Support for gzip files*. Python Software Foundation, [Online], [rev. 4. 2. 2018], [cit. 13. 4. 2018].
URL https://docs.python.org/3.5/library/gzip.html

[10] docs.python.org: *lzma – Compression using the LZMA algorithm*. Python Software Foundation, [Online], [rev. 4. 2. 2018], [cit. 13. 4. 2018].
URL https://docs.python.org/3.5/library/lzma.html

[11] docs.python.org: *subprocess – Subprocess management* . Python Software Foundation, [Online], [rev. 4. 2. 2018], [cit. 15. 4. 2018].
URL https://docs.python.org/3.5/library/subprocess.html

[12] docs.python.org: *urllib – URL handling modules*. Python Software Foundation, [Online], [rev. 4. 2. 2018], [cit. 9. 4. 2018].
URL https://docs.python.org/3.5/library/urllib.html

[13] DYTRYCH, J.: *KNOT Group Tasks*. FIT VUT v Brně, [Online], [rev. 7. 3. 2018], [cit. 9. 4. 2018].
URL https://merlin.fit.vutbr.cz/wiki/index.php/KNOT_Group_Tasks

[14] FARNCOMB, J.: *Gzip vs Bzip2 vs XZ Performance Comparison*. RootUsers, [Online], [rev. 17. 9. 2015], [cit. 11. 4. 2018].
URL
https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/

[15] FISHER, D.; HARDING, S.: *The ClueWeb09 Dataset*. The Lemur Project, [Online], [rev. 30. 3. 2018], [cit. 20. 4. 2018].
URL http://www.lemurproject.org/clueweb09/index.php

[16] gnu.org: *GNU Wget*. Free Software Foundation, Inc, [Online], [rev. 15. 9. 2017], [cit. 9. 4. 2018].
URL https://www.gnu.org/software/wget/

[17] GRASSEGGER, J.; HAGEN, M.; MICHEL, M.: *ChatNoir*. Webis Group Weimar, [Online], [rev. 2018], [cit. 20. 4. 2018].
URL http://webis15.medien.uni-weimar.de/

[18] JOAN, B.: *Difference Between RSS and ATOM*. DifferenceBetween.net, [Online], [rev. 21. 9. 2009], [cit. 9. 4. 2018].
URL http:
//www.differencebetween.net/technology/difference-between-rss-and-atom/

[19] JURAFSKY, D.: *Language Modeling*. Stanford University, [Online], [cit. 15. 4. 2018].
URL https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

[20] KARÁSEK, M.: *Newsbrief eu*. FIT VUT v Brně, [Online], [rev. 5. 7. 2016], [cit. 10. 4. 2018].
URL https://knot.fit.vutbr.cz/wiki/index.php/Newsbrief_eu

[21] loc.gov: *The WARC File Format*. Digital Preservation at the Library of Congress, [Online], Verze fdd000236, [rev. 31. 8. 2009], [cit. 13. 4. 2018].
URL https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml

[22] MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H.: *Introduction to Information Retrieval*. Cambridge, United Kingdom: Cambridge University Press, 2009, ISBN 0521865719, 23 s.

[23] RICHARDSON, L.: *Beautiful Soup Documentation*. Crummy, [Online], [rev. 2015], [cit. 11. 4. 2018].
URL https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[24] RUSSELL, M.: *Google Database Tracks Popularity of 500B Words*. Newser, LLC, [Online], [rev. 17. 12. 2010], [cit. 20. 4. 2018].
URL http://www.newser.com/story/107766/google-database-tracks-popularity-of-500b-words.html

[25] SPURRIER, N.: *Pexpect version 4.5*. Noah Spurrier, [Online], Verze 4507924a, [rev. 2013], [cit. 15. 4. 2018].
URL http://pexpect.readthedocs.io/en/stable/api/pexpect.html

[26] STRAKA, M.: *Parsito* . Institute of Formal and Applied Linguistics, [Online], [rev. 2015], [cit. 15. 4. 2018].
URL http://ufal.mff.cuni.cz/parsito

[27] STRAKA, M.; STRAKOVÁ, J.: Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Vancouver, Canada: Association for Computational Linguistics, srpen 2017, s. 88 – 99.
URL http://www.aclweb.org/anthology/K/K17/K17-3009.pdf

[28] VOJNAR, T.: *Synchronizace procesů*. FIT VUT v Brně, [Online], [rev. 3. 4. 2018], [cit. 15. 4. 2018].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIOS-IT%2Flectures%2Fios-prednaska-06.pdf

[29] w3.org: *Introduction to Atom*. W3C Software, [Online], Verze 1.10 (2005), [rev. 13. 10. 2007], [cit. 9. 4. 2018].
URL https://validator.w3.org/feed/docs/atom.html

[30] w3.org: *RSS 2.0 specification*. W3C Software, [Online], [cit. 9. 4. 2018].
URL https://validator.w3.org/feed/docs/rss2.html

[31] w3schools.com: *HTML Entities*. Refsnes Data, [Online], [cit. 13. 4. 2018].
URL https://www.w3schools.com/html/html_entities.asp

[32] W3Techs.com: *Usage of character encodings for websites*. W3Techs – World Wide Web Technology Surveys, [Online], [rev. 11. 4. 2018], [cit. 11. 4. 2018].
URL https://w3techs.com/technologies/overview/character_encoding/all

[33] webhosting.info: *DOMAIN STATISTICS*. WebHosting.info, [Online], [rev. 15. 4. 2018], [cit. 16. 4. 2018].
URL https://webhosting.info/domain-name-statistics

[34] ŠVAŇA, M.: *Vertikál*. FIT VUT v Brně, [Online], [rev. 11. 6. 2016], [cit. 13. 4. 2018].
URL https://knot.fit.vutbr.cz/wiki/index.php/Vertik%C3%A1l