Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Daniel Lessner

Webcrawler

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Pavel Pecina

Studijní program: Informatika, obecná informatika

2006

Na tomto místě bych chtěl poděkovat především Mgr. Pavlu Pecinovi za přívětivé vedení, dobré rady a zapůjčení literatury, dále Jonáši Fialovi za pomoc s prvními kroky v Pythonu a Petru Špačkovi za pomoc s překonáváním nástrah síťového provozu.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 30. 5. 2006

Obsah – DOPLN CISLA STRAN

Název práce: *Webcrawler*
Autor: *Daniel Lessner*
Katedra (ústav): *Ústav formální a aplikované lingvistiky*
Vedoucí bakalářské práce: *Mgr. Pavel Pecina*
e-mail vedoucího: *pecina@ufal.mff.cuni.cz*

Abstrakt: *Práce se zabývá tvorbou webového robota. Jeho úkolem je rekurzivně stahovat z internetu české stránky a čistit je na samotný prostý text (žádné HTML značky, styly nebo skripty). Ten potom bude využit pro tvorbu obrovského jazykového korpusu, užitečného pro další výzkum. Klíčovou vlastností robota je nenápadnost běhu, nezatěžování cizích prostředků a plné respektování nezávazného doporučení Robots Exclusion Standard. Robot je napsán v jazyce Python a intenzivně využívá jeho standardní knihovny a rychlou práci s textovými řetězci. Vzhledem k charakteru úlohy jsme se rozhodli pro paralelní implementaci, která by měla plně využít šířku pásma. S tímto záměrem jsme měli úspěch. Výsledkem práce je tedy robot připravený získat dostatek textů pro korpus. Samozřejmě je ale použitelný i pro jiné účely, zvlášť tam, kde je potřeba šetrnost k cizím prostředkům. Kromě jeho přínosu pro lingvistiku poskytuje i zajímavé informace o obsahu českého internetu.*

Klíčová slova: *robot, paralelní zpracování, HTML, jazykový korpus, Python*

Title: Webcrawler
Author: *Daniel Lessner*
Department: *Ústav formální a aplikované lingvistiky*
Supervisor: *Mgr. Pavel Pecina*
Supervisor's e-mail address: *pecina@ufal.mff.cuni.cz*

Abstract: This work? concerns? about developing a web robot. Its objective is to recursively download C?czech pages and clean them into plain text (no HTML tags, styles or scripts). ***These texts will be used ..... pro tvorbu obrovského jazykového korpusu, užitečného pro další research. Robot's key feature is nenápadnost běhu, nezatěžování cizích prostředků a plné respektování nezávazného doporučení Robots Exclusion Standard. R****obot je napsán v jazyce Python a intenzivně využívá jeho standardní knihovny a rychlou práci s textovými řetězci.* ***Vzhledem k charakteru úlohy jsme se rozhodli pro paralelní implementaci, která by měla plně využít šířku pásma. S tímto záměrem jsme měli úspěch. The result of this work is a robot, which is ready to získat enough textů for korpus. Of course, it can also be used for other purposes, especially when i pro jiné účely*** *zvlášť tam, kde je potřeba šetrnost k cizím prostředkům.* ***Except for its přínosu for linguistics it poskytuje i zajímavé informace o obsahu českého internetu.***

Keywords: *robot, parallel processing, HTML, language corpus, Python*

1. Úvod

Pro řadu metod z oblasti zpracování přirozeného jazyka a komputační lingvistiky je naprostou nezbytností dostatek vstupních textových dat. Ta je potřeba získat, základním způsobem zpracovat (rozdělit na slova atp.) a vhodně (aby byla použitelná k další práci) uložit. Jedním ze zdrojů dostatečného množství textu se jeví dokumenty dostupné na internetu. Tato práce je součástí pokusu ÚFALu o využití tohoto zdroje. Celý projekt spočívá na třech programových modulech – na robotovi, který automaticky stáhne relevantní soubory, další část (segmenter) má za úkol již zmíněné lingvistické předzpracování a poslední se stará o uložení získaných dat.

Předmětem této práce je první část – získání velkého množství surových textů (řádově o miliardy slov, desítky milionů stažených dokumentů). Nabízelo by se využít existující programy, umožňující rekurzivní stahování obsahu internetu (wget a další), ty ale v několika zásadních požadavcích nevyhoví. Jednak je třeba, aby robot stahoval především (zatím) české dokumenty. Nelze se spokojit s prostým omezením například na doménu .cz, bylo by to z obou stran nepřesné (existují cizojazyčné dokumenty v .cz i české mimo ni). Podrobnější úvahy vedou na individuální přístup k jednotlivým dokumentům a potřebu zpětné vazby pro robota. Druhým zásadním požadavkem je nenápadnost běhu. Je nutné zajistit, aby nedocházelo k přetížení cizích strojů nebo linek. Na druhou stranu bude zřejmě vhodné hledět na efektivitu využití času (zejména vzhledem k objemu získaných dat). Tyto nejzásadnější požadavky nejsou dostupnými programy dostatečně implementovány, zřejmě tedy nezbyde než napsat robota vlastního [**oReilly**].

Práce je členěna na 7 kapitol, lze je číst v různém pořadí, vyplatí se ale číst třetí před čtvrtou a druhou jako druhou - tato kapitola podrobně rozvádí východiska práce, požadavky na získávaná data a robota jak technické, tak i koncepční a diskutuje priority plnění daných požadavků. Uvádí motivaci dalšího postupu, od ní se všechno odvíjí. Třetí kapitola osvětluje použité technické prostředky, algoritmy a datové struktury, sloužící k naplnění požadavků z předchozí kapitoly. Nejde o podrobnou nebo jakkoliv formální "programátorskou dokumentaci", podle této kapitoly by jistě nešlo podobného robota bez dalšího rozmýšlení napsat. Cílem kapitoly je ozřejmit stěžejní myšlenky použitých postupů do takové hloubky, kdy je to ještě zajímavé bez čtení zdrojového kódu. Tato kapitola také zvolená řešení nijak hluboce neposuzuje – to je obsahem kapitoly následující (proto je vhodné je číst v tomto pořadí). V té je rozebráno, proč se zvolené postupy ukázaly jako vhodnější než jiné. Další kapitola se zabývá výsledky práce z různých pohledů. Stažené dokumenty jsou samozřejmě podstatné pro jazykový korpus, použití robota ale ukázalo zajímavé skutečnosti i o nich samotných. Dále jsou uvedeny údaje statistického charakteru získané při běhu robota.

Zdrojové kódy výsledného robota jsou přiloženy na CD. Jejich zprovozněním a použitím se zabývá poslední kapitola, obsah CD je rozveden v příloze.

2. Východiska, záměry, podrobné požadavky

2.1 Zamýšlená kostra pracovního postupu, technické požadavky, vstupy a výstupy

Základní datovou strukturou robota je fronta adres nestažených dokumentů. Stručně řečeno, práce robota probíhá v následující smyčce: vezme adresu z fronty, stáhne dokument, extrahuje z něj další adresy a zařadí je do fronty, dokument zredukuje na čistý text a pošle segmenteru k dalšímu zpracování, vezme další adresu z fronty a tak pořád dokola, dokud ve frontě něco je – tato kostra postupně nabude konkrétnějších rysů. Vstupními daty robota jsou adresy výchozích dokumentů, konfigurace a zpětná vazba od segmenteru, zejména ale samotné stažené dokumenty. Výstupem je polotovar korpusu - očištěné texty (od HTML značek atp.).

Aby mohl být robot plně využit a propojen s dalšími systémy, měl být napsán a odladěn na Linuxu (tak se také stalo). Předpokládal se silný stroj s desítkami GB paměti, rychlým přístupem na internet a velkým diskovým prostorem (to by se robota tolik týkat nemělo, stažené dokumenty by měl rovnou předávat dalším částem projektu). Protože má jít o produkt reálně nasazený, lze očekávat, že bude docházet k jeho úpravám a přizpůsobením, a je tedy zřejmě žádoucí brát na tuto skutečnost ohled při dodržování štábní kultury kódu a psaní dokumentace.

Komunikace robota a segmenteru by měla probíhat nejlépe síťově, aby mohly oba programy běžet na různých strojích. Konkrétní protokol měl vykrystalizovat během vývoje (to se také stalo). Jako preferované kódování bylo zvoleno ISO-8859-2.

V neposledné řadě je nutné, aby byl robot nastavitelný – jak z příkazové řádky při spouštění, nebo pomocí konfiguračního souboru. Navíc by měl obsahovat bezpečné základní nastavení. Konkrétní volby a možnosti konfigurace plynou z ostatních požadavků.

2.2 Požadavky na nenápadnost provozu

Vzhledem k očekávanému množství stahovaných dat je nutná jistá opatrnost vůči cizím prostředkům – jak linkám, tak i samotným strojům. Naprosto klíčovou vlastností robota je možnost účelně rozvrhnout jeho práci tak, aby jeho aktivitu pokud možno nikdo ani nezpozoroval. Rozhodli jsme se pro dodržování následujících čtyř parametrů:

1. Doba prodlevy mezi dvěma následujícími požadavky (na stažení souboru) na jednu doménu.
2. Nejvyšší počet "současně" kontaktovaných domén na jedné IP adrese (fyzickém stroji).
3. Nejvyšší počet požadavků na jednu doménu mezi generálními prodlevami.

4. Délka generální prodlevy. Tato prodleva následuje vždy po daném (předchozí parametr) počtu požadavků, dává serveru (ne jen doméně) čas na oddych, než dojde k další sérii požadavků.

Autoři stránek a správci serverů mají možnost se nežádoucím návštěvám robotů bránit. Na úrovni jednotlivých souborů jde o meta značky v HTML hlavičce, jejichž obsah udává, zda má být soubor indexován a jestli mají být následovány v něm obsažené odkazy. Na úrovni serveru (domény) jde o soubor robots.txt, kde je popsáno, kterými roboty smí (nebo častěji nesmí) být navštěvovány které adresáře (**[odkaz na RobExcStd]**). Všechny tyto pokyny musí robot samozřejmě a bezpodmínečně respektovat a dodržovat. Slušné chování robota je prioritou – ustupuje mu i rychlost samotného stahování. Je třeba raději práci robota zpomalit, než aby přetížil některé linky.

Ke slušnosti také patří identifikace robota s kontaktní informací v hlavičce každého HTTP požadavku. Kontaktem je informační stránka **[odkaz]** o robotovi, kde se správci serveru dočtou o tom, co je robot zač a jak mají postupovat v případě problémů. Pro ladění, pro zajímavost a v neposlední řadě pro ověřování správnosti chování, popřípadě řešení problémů správců serverů je nutné vést podrobné záznamy o činnosti robota.

2.3 Lingvistické požadavky

Robot má sloužit výzkumným účelům lingvistů, což by mohlo dělat dojem, že jsou na robota kladeny nějaké jazykovědné požadavky. Tak tomu ovšem není – veškerou jazykovou odbornost zajišťuje až segmenter (viz další podkapitolu). Úkolem robota sice je stažené texty pročistit, ale to je technická záležitost, kde se využije znalost HTML, ne žádného přirozeného jazyka. Požadavky na čistotu textů na výstupu jsou jednoduché – nesmí projít žádné HTML značky, skripty, styly ani cokoliv jiného. Navíc je třeba brát ohled na význam odstraňovaných značek – začátek odstavce například obvykle implikuje začátek nové věty, změna tloušťky písma nikoliv. Tyto informace by se po odstranění značek ztratily, je třeba je ovšem zachovat pro segmenter. Předpokládané konce vět mají proto být v očistěných souborech naznačeny odřádkováním. Cokoliv dalšího (nabízelo by se třeba odstranění kratičkých fragmentů textu, které pravděpodobně nebudou pro korpus použitelné) je už plně v režii segmenteru. Soubory jiných textových formátů (pdf, odt, rtf, doc a další) měly být podle původního návrhu rovnou čištěny a zpracovávány také, nakonec jsme se ale rozhodli pro jiné řešení. Snaha o implementaci čištění těchto formátů by vedla ke znepřehlednění kódu použitím nesourodých knihoven, přitom lze s výhodou využít řádkových konvertorů. Tyto soubory tedy bude robot ukládat stranou, a zpracovány budou až později jednoúčelovým skriptem. Odkazy v nich tedy budou také nalezeny a zpracovány (i když to bude později), což (proti alternativě zamžení činnosti robota) stojí za to.

2.4 Požadavky na vytyčování prostoru stahovaných souborů

Nejdůležitějším omezením prostoru stahovaných dokumentů (pro kvalitu výstupu) je zpětná vazba segmenteru. Jak už bylo napsáno, robot stažená data očistí a čistý text pošle segmenteru. Jeho úkolem je mimo jiné zjistit, zda je text vůbec použitelný (je česky, obsahuje rozumný počet odstavců s rozumně dlouhými větami apod.). Tuto použitelnost kvantifikuje a vrátí robotovi. Ten ji použije k rozhodnutí, zda z daného dokumentu následovat odkazy. Vycházíme z jednoduché

heuristické úvahy – anglické dokumenty se pravděpodobně odkazují na další anglické dokumenty. Proto robot nebude následovat odkazy z druhého dokumentu s nulovou použitelností. (Pokud se dokument odkazovaný nepoužitelným dokumentem ukáže jako taktéž nepožitelný, nebude robot prohledávat dál. Musí si tedy u odkazu pamatovat použitelnost rodiče.) Odhadujeme, že nenásledovat hned první nepoužitelný dokument by korpus mohlo ochudit a stažením ještě jedné úrovně neuděláme zase tolik práce navíc. Kromě toho lze vrácenou číselně vyjádřenou použitelnost využít k promyšlenějšímu stahování dokumentů, viz podkapitolu 7.1.

I přes existenci této zpětné vazby je ovšem třeba implementovat další nástroje k vytyčení prohledávaného prostoru. Jednak možnost nastavení maximální hloubky prohledávání (v pomyslném stromu, jinými slovy počet kliků od některého výchozího k cílovému dokumentu), dále možnost zadání seznamu zakázaných, resp. povolených domén (žádný dokument z nich stažen nebude, resp. žádný dokument nebude stažen odjinud).

3. Použité prostředky a postupy

V této kapitole je popis algoritmů a datových struktur, které robot využívá. Vzhledem k tomu, že spolu všechny dost úzce souvisí a dohromady se jedná o značně komplexní celek, rozhodli jsme se pro strategii postupného odhalování a nabalování. Ze začátku je tedy dost věcí záměrně zamlčováno, protože by ještě nedávaly dobrý smysl, ale ke konci už je vše podstatné řečeno a čtenář má jasnou představu, jak robot funguje. Tento způsob navíc dobře osvětluje postup návrhu.

3.1 Programovací jazyk, struktura programu, použité knihovny

Vzhledem ke hlavní náplni činnosti robota (práce s internetem a s textovými daty) se jevil jako velmi vhodný skriptovací jazyk Python. Navíc připravené standardní moduly (obdoba knihoven v jiných programovacích jazycích) a vysoká úroveň abstrakce umožnily plně se soustředit na řešenou úlohu a nerozptylovat se technickými detaily. (To může být vykoupeno rychlostí, což ale není zase až tak podstatné). Mezi použité moduly patří zejména `urllib2` pro komunikaci pomocí HTTP, `urlparse` pro práci s řetězci URL, `robotparser` pro čtení souborů /robots.txt, `socket` pro práci se sokety (zjišťování IP čísel serverů a komunikace s ostatními částmi projektu), `re` pro práci s regulárními výrazy, `cPickle` pro rychlou transformaci složitých datových struktur do textového formátu (který lze pohodlně poslat po síti), `time` pro měření času, `sys` a `os` pro přístup ke stadnardním systémovým funkcím, `getopt` pro zpracování parametrů příkazové řádky (a konfiguračního souboru) a v neposlední řadě modul `logging` pro zaznamenávání průběhu práce.

Program je rozdělen na dva skripty a několik modulů. Skripty queuekeeper a harvester (názvy vystihují jejich úlohu) budou popsány níže. Tyto skripty (kromě výše uvedených standardních) používají pro robota vyvinuté a algoritmicky nezajímavé moduly (v souborech se stejným názvem) `my_sock` pro pohodlnější komunikaci skriptů a segmenteru (nadstavba stan. modulu `socket`, pracuje s měřením délky zpráv), `timer` pro měření času (zase jde jen o zpřehlednění kódu), `common_conf` pro společné nastavení obou skriptů a `txtr_html` určený na extrakci textu z HTML (a XHTML) souborů a další dílčí úkoly související s HTML formátem.

O nastavení robota je namístě uvést, že probíhá ve třech fázích (všechny zajišťuje uvedený modul) – jako první se použije defaultní nastavení, to může být přepsáno nastavením uvedeným v konfiguračním souboru (to už může editovat uživatel) a to může být přepsáno argumenty předanými z příkazového řádku při spouštění skriptu (viz poslední kapitolu). Tyto tři fáze proběhnou v okamžiku importu modulu, kdy se do kontextu skriptu nahrají proměnné nastavené na správné hodnoty. Stejným způsobem probíhá i čtení konfiguračního souboru. Do modulu je importován "modul" `usr_conf.py`, který je ve skutečnosti linkem na textový soubor. (Jde o to, aby Python soubor našel, a zároveň aby uživatel neměl pocit že programuje.) Obsahem konfiguračního souboru jsou tedy přímo přiřazovací příkazy.

Přestože je čištění HTML souborů naprosto zásadní schopností robota, čištění probíhá velmi jednoduše a nechává většinu práce na Pythonu, přesněji na modulu pro regulární výrazy. Po podrobném prostudování specifikace (a zadání práce) bylo možné zkonstruovat několik regulárních výrazů na upravení bílých znaků (počítá se vždy jen první), postupné odstranění hlavičky, stylů a skrpitů, řádkových a blokových (nahrazeny odřádkováním) značek a nakonec odstranění nadbytečných odřádkování a mezer na krajích řádků. Korektní dokumenty jsou zpracovány správně a nekorektní v drtivé většině také, v principu ale vždy půjde najít (nekorektní) řetězec, který projde očistěn chybně (s kusy HTML značek). Kromě čištění se modul stará o hledání odkazů ve stažených dokumentech a další drobné úkoly související s HTML formátem.

3.2 Implementace fronty

Z přímočaré myšlenky o udržování lineární fronty požadavků v pořadí, v jakém mají být naplňovány, se vyvinul zcela jiný způsob implementace. Odkazy jsou evidovány obecně v nedefinovaném pořadí ve dvouúrovňovém slovníku (dictionary **[ODKAZ]**, asociativní pole v Pythonu) indexovaném na první úrovni doménami a na druhé cestami. Samotné záznamy obsahují hloubku, předpokládaný koeficient použitelnosti a počet pokusů o stažení (zvyšuje se při neúspěších způsobených chybou, která může být postupem času odstraněna. Toto by samo o sobě nestačilo, je třeba uchovávat informace o doménách jako takových (jakou mají IP adresu a obsah souboru robots.txt) i o IP adresách (které na nich robot eviduje domény a které z těch domén jsou zastopupeny ve struktuře požadavků). Obojí je opět řešeno slovníkem. Dále patří mezi základní datové struktury seznam stažených dokumentů, jde opět o dvojitý slovník (stejně jako u požadavků), obsahem záznamu je zde ovšem kromě hloubky a použitelnosti (teď už opravdu změřené segmenterem) záznam o délce doby platnosti dokumentu a okamžiku jeho poslední modifikace (pro účely pozdější aktualizace korpusu), času stažení dokumentu a titulek. Takto má robot zmapovanou prohledanou část internetu.

Následující řádky nejsou zcela přesné – vysvětluje se na nich způsob implementace samotné fronty tak, aby jej šlo snadno pochopit. Po přečtení dalších podkapitol bude ovšem dostatečně zřejmé, jak celý robot funguje.

Je zřejmé, že uvedené struktury žádné pořadí požadavků nenaznačují. S frontou v klasickém pojetí robot vůbec nepracuje, tento pojem je zde (i ve zdrojovém kódu) chápan ve svém abstraktním významu. S ohledem na jednoduchost plánování a snahu o dodržení pravidel z 2.2 pracuje robot ne po požadavcích, ale IP adresách, resp. jejich skupinách. IP adresa je základní jednotkou, se kterou fronta pracuje. Přesto ani IP adresy nejsou nijak časově rozvrhovány. Jsou pro ně definovány čtyři stavy – čekající, zpracovaná, aktivní a odpočívají (viz Schéma 1). Každá IP adresa je vždy právě v jednom z těchto stavů. Stavy jsou implementovány jako množiny (interní datová struktura Pythonu), obsahující IP adresy nacházející se v daném stavu. Z IP adres v aktivním stavu jsou právě stahovány dokumenty, odpočívající adresy jsou právě po takovémto “útoku” na generální pauze. Ostatní se dělí podle toho, jestli se k nim váží nějaké další požadavky na čekající (na jejich zpracování) a zpracované. S pojmem fronta se v práci lze setkat v několika významech, vždy je ovšem z kontextu jasné, který se myslí. Jednak jde o abstraktní pojem fronty (něco chodí dovnitř a něco odchází), potom o frontu na úrovni IP adres a jejich stavu a v neposlední řadě o frontu na úrovni požadavků (mluvíme o jejich braní z fronty nebo o zařazování) – v tom případě jde o strukturu (popsané dvojité asociativní pole) požadavků.

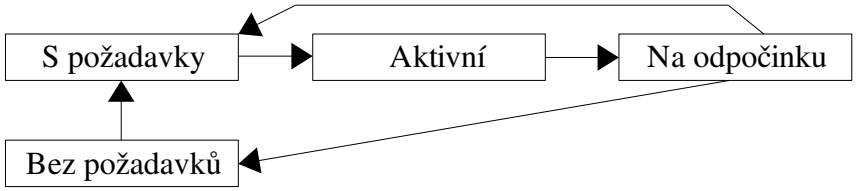

Schéma 1: Stavy IP adres

Práci robota s frontou lze popsat následujícím způsobem. S ohledem na průměrnou dobu splnění požadavku a daná pravidla je vytvořen balíček IP adres. V cyklu (omezeném příslušným parametrem) jsou pak z adres v balíčku brány domény a z výše popsané struktury vybírány konkrétní požadavky. Ty robot zpracovává (čistí text a posílá dál, zařazuje do struktury nové odkazy). Balíček je právě tak velký, aby požadavky v jednom průběhu cyklu naplnily dobu prodlevy mezi dvěma požadavky na jednu IP adresu. Požadavky nebudou kladeny příliš mnoho domén z jedné IP adresy a neproběhne jich na jednu doménu příliš mnoho – to přirozeně zajišťují meze příslušných cyklů. Po vyčerpání povoleného počtu požadavků z balíčku (nebo všech, pokud jich bylo méně než je ten povolený počet) se stav IP adres změní na odpočinek, kde zůstanou určený čas (implementace tohoto mechanismu je popsána v jedné z dalších podkapitol). Robot z čekajících IP adres připraví další balíček, opět ho zpracuje a takhle pokračuje, dokud eviduje nějaké požadavky. Pochopitelně se může stát, že ve struktuře není dost odpočatých IP adres, aby byl jeden cyklus balíkem dost dlouhý, popřípadě aby v balíku vůbec něco bylo. V takovém případě je nutné počkat. To se může zdát neefektivní, ale s ohledem na velikost internetu je zřejmé, že k těmto extrémním případům dochází jen na začátku a na konci stahování (a často při testovacím spouštění na malých datech). Navíc je jasné, že při dodržení pravidel nenápadnosti se čekání na konci stahování nelze vyhnout (na začátku pomůže zadat dost dlouhý seznam startovacích adres).

Dále je vhodné poznamenat, že se na robotovu činnost nedá pohlížet na prohledávání čistě do hloubky nebo do šířky (tomu se opět při daných pravidlech prohledávání a zachování efektivity stahvání nelze vyhnout). Jednak si proto lze činnost robota na pomyslném grafu stránek představit poněkud obtížně, podstatnější ale je, že evidovaná hloubka jednotlivých odkazů od startovací stránky není v běžném smyslu korektní, jde pouze o horní odhad. Robot při prohledávání může kratší cestu úplně vynechat. Pokud potom k danému souboru najde kratší cestu, záznam opraví, ne ovšem jeho potomky. Tento rys by šel sice snadno implementovat, ale není důležitý – parametr hlubky je k dispozici pro ladění a omezení shora, při ostrém provozu se s jeho použitím nepočítá. Spuštění robota s omezenou hloubkou tedy nevede nutně ke stažení všech stránek do dané hloubky. Je jenom jisté, že nestáhne žádnou stránku umístěnou hlouběji. Na první pohled to vypadá divně, ale ve skutečnosti je to správně (tj. dělá to to, co jsme chtěli).

3.3 Paralelní běh

Stáhnout český obsah internetu je dost masivní úloha. Nabízí se otázka, jestli by nebylo účelné ji řešit paralelně. Myšlenka spočívá ve skutečnosti, že než vzdálený počítač odpoví na požadavek, celý náš stroj čeká, přitom by ale mohl třeba teprve teď zpracovávat naposledy stažený dokument. Měření času spotřebovaného jednotlivými činnostmi robota naprogramovaného jednoprocesově ukázalo, že se tato cesta jeví jako nadějná – na síťovou komunikaci se spotřebovalo kolem poloviny celkové doby běhu a procesorový čas byl jen zlomkem skutečné doby běhu.

Rozhodli jsme se naznačené možnosti využít. Jak už bylo napsáno výše, fronta není implementována přesně v uvedené podobě. Její předchozí popis ale dává možnost snadno vysvětlit způsob rozdělení úlohy na menší části a jejich společný běh. Procesy robota se dělí na dva druhy – klienty a server (všechny mohou běžet na jediném stroji). Klientů je obecně několik a komunikují se serverem (zpravidla jen jedním) pomocí soketů. Úkolem serveru je spravovat frontu požadavků, seznam stažených dokumentů a struktury uchovávající informace o IP adresách a doménách. Klienti pracují s balíky. Fronta popsaná v předchozím odstavci se trochu změní – zejména se nebude jednat o balíčky IP adres, ale požadavků. Obsahem balíčku požadavků jsou ovšem tytéž požadavky, které by stáhla zjednodušená fronta. V předchozím popisu se také nemluvilo o tom, kdy probíhá aktualizace struktur. Následuje popis činnosti klienta a pak popis činnosti serveru.

Práce klienta spočívá ve zpracovávání balíků požadavků. V těch je dáno pořadí a mají strukturu, která umožňuje dodržet pravidla – podrobnější popis je v příslušné podkapitole. Klient se připojí k serveru, pošle výsledky získané zpracováním posledního balíku (napoprvé prázdné) a přijme nový balík požadavků. Z něj bere jeden za druhým, stahuje, čistí a posílá dál (segmenteru) soubory. Eviduje si obsažené odkazy a vytváří strukturu stažených a strukturu nestažených (chybou spojení) dokumentů. Po vyčerpání balíku pošle nové odkazy a struktury s dokumenty serveru a vše začne znovu.

Server, jak už bylo řečeno, veškerou práci koordinuje. S pomocí seznamu startovacích URL vytvoří struktury fronty popsané v předchozí podkapitole. Potom čeká na připojení klienta – přijme od něj (napoprvé žádné) změny, vytvoří balík požadavků a odešle ho klientovi. Ten se za nějaký čas opět připojí a server přijme nová data k zapracování do struktur fronty. Stažené dokumenty prostě jednoduše zařadí do existujícího slovníku, stejně jako nezpracované požadavky (došlo k pokusu o stažení, ale z nějakých důvodů to nevyšlo a lze očekávat, že to příště bude lepší). Server je také ten, kdo se stará o stavy IP adres – jedna IP adresa se současně účastní vždy nejvýše na jednom balíku (v takovém případě je v aktivním stavu), a po každém dokončení stahování balíku přechází příslušné adresy do stavu odpočinku. Z něj potom pokračují buď do stavu hotových, nebo čekajících adres. Práce se stavy se tedy paralelizací příliš nezměnila. Klient na stavy vůbec nevidí a nestará se o ně, neví nic ani o IP adresách.

Implementace v této podobě se ukázala být poněkud naivní. Zařazování nových domén (a čtení souboru robots.txt) měl na starost server, protože domén je jistě o mnoho méně než dokumentů ke stažení. Na začátku běhu ovšem domény přibývají příliš rychle – nakonec docházelo k tomu, že všichni klienti čekali (aniž by cokoliv stahovali) i několik hodin, než si server aktualizuje struktury, zařadí nové domény a vydá nové balíky požadavků ke stahování – přitom ve frontě bylo požadavků dost, stačilo tedy zařazování domén o něco odložit. Rozhodli jsme se proto i tuto práci nechat na klienty. Na začátku každého spojení klienta a serveru dojde k předání zprávy, co klient zpracovával, a podle toho se rozliší, jak má server výsledky zpracovat a naopak, server posílá před odesláním balíku informaci o tom, zda jde o balík požadavků, nebo balík nových domén.

Celkem tedy dochází k následujícímu zpřesnění. Při zpracovávání nových odkazů server rozliší odkazy na domény už známé, a na domény nenavštívené (nepovolené domény jsou vyloučeny už při nalezení odkazu). Odkazy na známé domény jednoduše zařadí do fronty požadavků. Vedle toho má ovšem frontu požadavků na neznámé domény. Pokud se připojí klient a server mu nemůže nabídnout balík požadavků, vezme z této fronty několik domén a předá je k předzpracování. Klient je po jedné projde, u každé zjistí IP adresu a informace ze souboru robots.txt. Takto naplněný seznam pošle zpět na server, který přesune příslušné požadavky z jedné fronty (neznámé domény)

do druhé (požadavky). Tím vznikne materiál pro balík požadavků a dokud zase nedojdou, běží robot výše popsaným způsobem.

Server běží, dokud je něco ke zpracování ve frontě nebo je něco právě stahováno klientem. Když je hotovo, uloží struktury do souboru (pro možnost opětovného spuštění apod. - nic takového ještě nebylo implementováno, ale zřejmě to bude potřeba) a stráví dobu odpovídající generální pauze v závěrečné fázi, kdy klientům snažícím se o připojení posílá zprávu, že mají skončit. Pokud klient něco stahoval, tak o tom server věděl a neskončil – klient tedy v okamžiku ukončení práce serveru čeká, jestli bude k dispozici nějaký balík požadavků. To ověřuje právě v časovém horizontu generální pauzy.

3.4 Konstrukce balíku požadavků

Zde přiblížíme strukturu a konstrukci balíku požadavků. Vyžadujeme od něj několik podstatných vlastností. Neměl by samozřejmě být nepřiměřeně velký, má se přenášet mezi procesy, a měl by jít postavit rychle. Hlavně ale má umožnit přehledný způsob dodržování pravidel – v balíku nesmí být víc než daný počet požadavků na jednu doménu ani víc než daný počet domén na jednu IP adresu. Navíc musí umožňovat hlídání odstupu dvou následujících požadavků na jednu doménu. Jak bylo naznačeno v podkapitole o frontě, při dostatku adres lze pravidla dodržovat přirozeně mezemi cyklů a podobně. Bylo by vhodné využít podobného mechanismu i při konstrukci balíku. Přitom do něj chceme dát co nejméně strukturálních informací, aby zůstal malý. Zřejmě se tedy vyplatí postavit balík podle pravidel už na straně serveru – na pravidla se tam musí myslet tak jako tak, stačí celou věc jen dotáhnout.

Balík je seznam seznamů požadavků. Není problém na něj nahlížet jako na tabulku (byť okousanou) se sloupci a řádky. Základní myšlenka je v tom, že balík je plněn po pomyslných řádcích, a vyprazdňován po sloupcích. Obojí lze dělat v konstantním čase (na jeden požadavek). Plnění potřebujeme dělat systematicky, aby bylo jasné, že neporušujeme pravidla. Vezmeme tedy vhodnou IP adresu (z čekajících, nebo zkusíme, jestli už není nějaká odpočatá), z ní doménu a pustíme se do plnění řádku, další postup je zřejmý (myšlenkově, ze zdrojového kódu moc ne) – hlídáme plnost řádku, plnost slovníku požadavků na doménu a kvótu požadavků na doménu. Při překročení bereme další doménu od stejné IP adresy (a pokud jsme jich už vzali víc než povolený počet, nebo už na dané adrese jiné domény nejsou, vezmeme další vhodnou adresu). Je tedy zajištěno, že v jednom sloupci je maximálně jeden požadavek na jednu doménu a obdobně s doménami a IP adresami. Řádků se snažíme naplnit dost na to, aby zpracování jednoho trvalo alespoň požadovný odstup dvou následujících požadavků na doménu (to se zjistí pomocí průměrného času zpracování jednoho požadavku, předávaného pokaždé klientem – čas se může různit mezi klienty i v čase).

Jak bylo řečeno, balík je seznam seznamů. Ty jsou chápány řádkově. Klient tedy v cyklu přes všechny řádky zpracuje první z prvků každého řádku. Měří si, jak dlouho mu to trvalo, a po skončení tohoto cyklu se pozastaví na dobu určenou rozdílem doby jeho trvání požadovaným odstupem dvou požadavků.

Všimněme si, že v jednom řádku za sebou následují řádově desítky až stovky URL se stejným prefixem tvořeným doménou. Dá se výrazně ušetřit, když doménové jméno nebudeme pokaždé opakovat. Navíc bude ještě lépe vidět, že dodržujeme pravidla. Řádek balíku tedy nebude seznam prostých požadavků, ale seznam dvouprvkových seznamů, kde prvním prvkem je doménové jméno

a druhým seznam cest. Nevolili jsme slovník, protože indexovat nic nepotřebujeme, naopak jde o pořadí – i domén. Může se totiž stát, že požadavky na jednu doménu jsou na dvou řádcích a my potřebujeme, aby se tyto neproťaly v žádném sloupci.

Schéma 2: Balík požadavků

Posledním pěkným zlepšovákem je rovnoměrné rozložení řádků s doménami z jedné IP adresy mezi ostatními. Dosahujeme toho jednoduchým ani ne trikem, řádky při konstrukci balíku ukládáme cyklicky do několika různých seznamů (počet je dán maximálním počtem stahovaných domén na IP adresu) a v závěru konstrukce je zřetězíme.

3.5 Zajištění pravidel – odpočívárna

Implementace zajištění pravidel pro dodržení nenápadnosti provozu už byla popsána (jde nejčastěji o přirozené meze cyklů, které se případně promítnou do obsahu balíku), kromě pravidla o dodržení délky generální pauzy, které si zaslouží bližší popis. Stejně jako v případě fronty, i zde jsme se vyhnuli nutnosti zaznamenávat si ke každé IP adrese čas přechodu do stavu odpočinku. Místo toho se tento stav odpočinku na tři podstavy, lze si to představit jako odpočívárnu rozdělenou dvěma přepážkami. Procházet lze vždy jen z prvního oddílu do druhého, ze druhého do třetího, přijít se dá pouze do prvního oddílu a odejít se dá jedině ze třetího oddílu. IP adresa tedy postupně projde celou odpočívárnou v daném pořadí. (Ve skutečnosti se cyklicky mění první oddíl na druhý atd.) Vtip je v tom, že je zajištěno, že ke změně dojde vždy po době delší, než je polovina požadované délky generální pauzy, neboli každá adresa zůstane ve druhém a ve třetím oddílu dohromady dostatečně dlouho. Druhou podmínkou správnosti je, že každá IP adresa nejpozději po povoleném počtu požadavků odpočívárnou projde (i kdyby už na ní žádné požadavky směřovány být neměly – to se může během odpočinku změnit).

S pomocí jediného časového záznamu (okamžik poslední změny místností) se tedy daří pravidlo dodržet. Za toto pohodlí platíme jistou ztrátou přesnosti čekání IP adres, která se projeví zajména při malém počtu požadavků ve frontě (typicky začátek a konec běhu robota). Většinu času je ale IP adres dost na to, aby ke zpomalení nedošlo.

4. Diskuse k použitým postupům

4.1 Implementace fronty

Zcela naivní (klasické) řešení fronty samozřejmě nevyhovuje požadavkům na nepřetěžování linek a nenápadnost běhu. Od fronty potřebujeme možnost nějakého plánování a kontroly, jestli to se stahováním nepřeháníme. Rozlišování zařazení požadavků na začátek nebo konec fronty korektnost provozu (přijatelnou míru stahování z jednotlivých vzdálených serverů) nezaručí. Další podstatnou skutečností je spojovanost HTTP – vyplatilo by se asi stáhnout víc souborů v jednom spojení (na druhou stranu, nesmí se to přehnat).

V první řadě bylo třeba rozmyslet, co má vlastně být obsahem fronty. Samozřejmě požadavky, chceme ovšem mít přehled i v plnění pravidel – ta jsou ovšem formulována s pomocí domén a IP adres. Kdyby obsahem fronty byly prosté URL, museli bychom vždy znovu a znovu zjistit doménu i IP adresu a podívat se v nějakých dalších strukturách (evidujících počty a časy stažení), jestli můžeme stahovat. Navíc bez jakékoliv vazby na další požadavky z dané domény, natož IP adresy. Taková implementace by zřejmě fungovala, ale hezké by to nebylo. Požadavek hlídání počtu a odstupu požadavku na doménu a počtu domén na IP adresu vede ke strukturalizaci fronty tak, aby byla jejich vazba patrná. Pak už je nasnadě, že je-li IP adresa na nejvyšší úrovni, je vhodným kandidátem na atomický prvek fronty (atomický z hlediska fronty – ta IP adresou točí bez pohledu a ohledu na nějakou strukturu pod ní).

Vnucuje se řešení: záznam o čase poslední návštěvy pro každou IP adresu, resp. čas nejbližší příští, a podle nich vybírat příští požadavek, datovou strukturou by byla pravděpodobně nějaká variace haldy. Potřebujeme od seznamu požadavků několik věcí:

- možnost ověřit, zda už daný požadavek evidujeme (aby se nestahovalo něco zbytečně dvakrát, nebo spíš mnohem víckrát)
- vybírat požadavky, které nic nepřetíží (neboli servery, kde robot dlouho nebyl)
- aktualizovat časovou značku, neboli zatřiďovat požadavek do fronty

Zejména poslední dva požadavky vedou na použití binomické nebo Fibonacciho haldy. To by zabralo jednak víc práce, druhak to přímo není nejlepší řešení – šlo by ještě vylepšovat (a to by zabralo ještě víc práce). Při předpokladu stability kvality spojení totiž lze dobře předpokládat, jak hluboko do fronty požadavků by měla být IP adresa zařazena. Rozmyslíme-li pohyb serverů ve frontě ještě lépe, dostaneme se téměř přímo na výše uvedené řešení (bez zdvojení některých struktur, ty jsou výsledkem plánování), ze kterého haldy úplně vypadly. Na vrcholu fronty by se točilo několik IP adres, dost aby zajistily časový odstup dvou požadavků a akorát tak dlouho, aby využili povolený počet stažení před generální pauzou. Se zbytkem fronty by se nedělo nic, jen by narůstala o nové požadavky. IP adresy opouštějící vrchol by byly zatřiďovány hlouběji do fronty, aby nepřišly na řadu dřív, než si odbudou generální pauzu. V souladu s tímto chováním fronty jsme se rozhodli ji upravit – rozdělění na několik stavů ji činí přehlednější.

Robot také na některých úrovních hlídá duplicity – obsahově stejné stránky nerozliší, stará se ale, aby nebyla zbytečně několikrát stažena jedna stránka. Vzhledem k počtu odkazů na běžné stránce je zřejmé, že jeden odkaz najdeme vícekrát. Kontrola probíhá ve chvíli zařazování požadavku do fronty – nesmí existovat záznam o jeho stažení. Může se ovšem stát, že už ho stahuje některý klient. Hledali jsme postup, jak tomu optimálně zabránit (je to jednoduché, ale při uvažovaném počtu kontrolovaných odkazů velmi záleží na tom, jestli se to udělá rychle). Buď by bylo třeba projít záznamy zpracovávaných balíků (ty už ovšem nemají dobře prohledavatelnou strukturu, pokud bychom ji pro tento účel neudrželi). Naopak přímočará je kontrola oprávněnosti požadavku v okamžiku zařazování do balíku. V tu chvíli požadavek jiným klientem jistě zpracováván není, protože by používal IP adresu dané domény, která by tak nebyla dostupná k přechodu do aktivního stavu (protože už by v něm byla). Kontrola na takto poslední chvíli je tedy spolehlivá. Ovšem provádět kontrolu až při řazení požadavku do balíku vede obrovskému nárůstu fronty, požadavky by se nám do ní stále vracely a o eleganci není vůbec řeč. Zvolili jsme kompromisní řešení – požadavek prověříme jednou při vstupu do fronty, vyhneme se nepříjemně zatěžujícímu prověření případných aktivních balíků a zkontrolujeme (jednoduchý dotaz na přítomnost v asociativním poli) jej ještě jednou při vkládání do balíku. Ztráty na rychlosti ani prostoru tedy nebudou veliké, což jsme také kontrolou duplicit chtěli zajistit (kromě efektivního využití síťových prostředků). Robot navíc eviduje i přesměrování – je-li skutečná adresa souboru vráceného na nějaký požadavek odlišná od požadované, zaznamenají se jako stažené obě.

Je třeba připustit, že jsme v naší implemantaci fronty ztratili možnost jemného plánování pořadí požadavků. Tvrdíme, že ji nepotřebujeme. Velmi hrubě už z toho, že souborů je obrovské množství a rozlišovat pořadí každého z nich jednotlivě ani nemá dobrý smysl. Nakonec bude robot muset stejně stáhnout všechno. Jemněji - stačí plánování přibližné, kdy požadavek zatřídíme s přesností třeba na stovky požadavků. Hlavním cílem plánování je snesitelné zatížení obou stran, a toho je již takto dosaženo. Dalším cílem může být přednostní a rychlé získání kvalitních výsledků, neboli chceme dříve stahovat z kvalitních (co se týče jazyka, rychlosti odezvy vzdáleného stroje i dalších parametrů) zdrojů – myšlenka je podrobněji popsána v podkapitole 7.1. Tady by zřejmě pomohlo, vágně řečeno, dávat lepší požadavky víc dopředu. Má to ovšem smysl jen u čekajících IP adres – zpracovat balík trvá řádově desítky minut, takové zpoždění v našich měřítcích nijak nevadí. V ostatních stavech zase pořadí podle kvality žádný význam nemá. V tomto ohledu je fronta otevřená budoucím úpravám – nic nebrání přidat do struktur položku s vyjádřením kvality a brát na ni ohled při vybírání IP adres, domén i jednotlivých požadavků ke zpracování. Tím by se mohly ke slovu vrátit haldy. Jinou možností by bylo rozmyšlení přesnosti odhadování kvality v budoucnu staženého dokumentu. Pokud by se ovšem ukázalo, že lze požadavky (a domény a IP adresy) rozdělit podle kvality do několika málo skupin, bylo by možná vhodnější je na tyto skupiny rozdělit. Dosahovali bychom tak stejných výsledků jako s haldou, jenže rychleji a přehledněji. Toto se nicméně netýká současné implementace, není tedy asi na místě věc rozvádět do podrobností.

Hlavním rysem zvolené implementace fronty je, že přirozeně umožňuje dodržet požadovaná pravidla k zajištění nenápadnosti provozu. Navíc nebrání dalším nadstavbám a vylepšením.

4.2 Paralelní běh

Při navrhování paralelizace úlohy jsme řešili tři hlavní otázky. První z nich se týkala jejího vhodného rozdělení. Zejména pokud se na úlohu díváme jako na prohledávání grafu, jeví se jako

vhodný postup rozdělení prostoru stránek, pravděpodobně podle IP adres, takže každý proces by zpracovával stránky na jisté skupině serverů. Tento zdánlivě přímočarý postup má několik úskalí, a jejich řešení vede na postup popsaný výše.

Problém naznačeného řešení spočívá prostě v tom, že nejde o vhodné rozdělení problému. Jednotlivé samostatně zpracovávané části prohledávaného prostoru jsou souvislé (dokonce velmi silně), je proto třeba řešit situaci, kdy jeden z "podrobotů" objeví odkaz do skupiny adres, které má zpracovat některý další. Samozřejně správný postup je příslušnému procesu požadavek poslat – otázkou zůstává určení správného adresáta. Samozřejmě může každý robot znát klíč, podle kterého jej určí. V takovém případě si bude postupně vytvářet vlastní strukturu přiřazující doménám IP adresy a těm pak procesy. Tyto informace jsou však nápadně duplicitní, nejen co týče uchovávání, byly by i vícenásobně získávány. Nabízí se řešení – vytvořit ještě jeden, zvláštní proces, který se bude odpovídat na otázky, kam který požadavek patří. Znatelně už se blížíme k nakonec použitému způsobu. Procesy už tedy nejsou rovnocenné a vtírá se otázka, jestli má vůbec dobrý smysl původní rozdělení úlohy na skupiny IP adres. Aby tenhle postup zafungoval, potřebovali bychom práci jednotlivých instancí robota rozdělit zhruba rovnoměrně. To buď znamená mít odhad, jak bude zpracování těch kterých IP adres náročné (takový odhad jednoduše nemáme), nebo prostor rozdělit náhodně (nebo alespoň hodně nezávisle, tedy ne na jednotlivé intervaly, ale třeba si pomoci modulovaným ciferným součtem číselného vyjádření adresy) a věřit, že se dost dobře trefíme (vzhledem k počtu IP adres máme celkem slušnou šanci). Neshledali jsme ale, že by pevné rozdělení prohledávaného prostoru přinášelo něco pozitivního. Teď stojí za povšimnutí, že na řešení popsané v předchozí kapitole lze nahlížet jako na zobecnění tohoto postupu s rozdělením prostoru IP adres – k rozdělení dochází dynamicky, s každým balíkem je jiné. Teď už jsme u řešení, jehož dotažená verze byla implementována. Zbývá přesunout datové struktury fronty do serveru, tedy procesu který se staral o koordinaci výpočtu. Mohlo by se zdát, že si pohoršíme, že spravováním velké fronty strávíme více času, než spravováním několika malých. Zkušební (neparalelní, tedy vše bylo v jedné frontě) verze robota však ukázala, že na správu datových struktur se spotřebují řádově desetiny procent spotřebovaného času, nemáme se tedy čeho bát. Bylo by samozřejmě nešikovné, kdyby klienti pracovali s jednotlivými požadavky – to přímo vede k myšlence balíků a použití již dříve navržené implementace fronty. Použitím balíků se navíc naskýtá možnost jednoduše zajistit plnění pravidel nenápadnosti (popsáno v předchozí kapitole). Naznačili jsme zde, že použitý způsob paralelizace úlohy je přirozený a že je zásadní, ale vhodnou úpravou na první pohled postačujícího řešení.

Druhá zásadní otázka při návrhu paralelizace nějaké úlohy je otázka plánování výpočtu. Tady se téměř není o čem rozepisovat – tak jako vlastně v celé práci, opět zvítězila cesta nejmenšího odporu a využití již hotového. Ta dokonce (zase opět) dobře odpovídá charakteru úlohy. Byly v zásadě dvě možnosti – buď si vlastnoručně řídit nějaká vlákna, nebo tuto práci nechat na někom jiném. Šetření času nespočívá v jakkoli vychytralém rozvržení nebo plánování práce procesoru, ale v jeho prostém co nejúplnějším vytížení. Jde o to, aby se při čekání na odezvu kontaktovaného serveru mohl na chvilku dostat ke slovu jiný proces – ale tohle počítač sám od sebe umí. Nechali jsme tedy plánování naprosto na něm, prostou svépomocí zřejmě dosáhne lepších výsledků než bychom mohli dosáhnout my. Při rozhodování o tom, co nechat na serveru a co nechat řešit klienty jsme se proto také přikláněli k tomu, aby toho klienti řešili co nejvíc a při volné chvilce čekání jednoho z nich měl procesor koho nechat pracovat. Práce serveru vlastně zdržuje – klient musí vždy počkat, až server aktualizuje struktury a až postaví nový balík. Sice by to šlo řešit vícevláknově, ale rozhodli jsme se (i na úkor čekání klientů) pro zachování přehledu v konzistenci struktur.

S tímto také souvisí pokus o řešení zahlcení popsaného v podkapitole 3.3. Opět jsme narazili na to, že naprosto nemáme odhad, jaká vlastně je struktura obsahu internetu – mysleli jsme prostě, že nových domén (a souborů robots.txt) bude vždy příliš málo, něž abychom se tím měli zabývat. Opak se ukázal být pravdou. Snažíme se dosáhnout toho, aby byl robot co nejčastěji v optimalním stavu – to je stav, kdy je plně vytíženo připojení k internetu, neboli klienti stahují nové soubory. Před řešením problému zahlcení je dobré se smířit s tím, že možná neexistuje – je dost dobře možné, že odkazů je bude prostě příliš, než aby bylo možné jejich zpracování odkládat a stahování se prostě zpomalí. Snažíme se tedy vytvořit postup, který uspěje (robot se nezadře), jen pokud to jde. Ještě jinými slovy – problém v principu nelze vyřešit tím, že bychom odstranili příčinu (protože ji prostě potřebujeme). Pokoušíme se ho rozmělnit. To co zdržuje, je čtení souboru robots.txt. Mohli by ho číst klienti ještě před odesláním nových odkazů na server, to ale zjevně není šťastné řešení – došlo by k neskutečně mnoha zbytečným stažením. Další možností je nechat server, ať soubory stahuje během čekání na připojení. To by možná stačilo, ale riskujeme, že je to málo a klienti budou stejně čekat, protože server jim nebude mít co dát. Chceme-li plně využít potenciálu, který již máme k dispozici, necháme soubory stahovat klienty – koneckonců jsou k tomu určeni. Rádi bychom ale (čistě pro jednoduchost a pohodlí) zachovat úlohu serveru jako jediného správce určujícího, co je třeba stáhnout a co ne. Proto ten na první pohled neobratný postup posílání domén serveru a nazpět klientovi.

Další zásadní otázka při návrhu paralelizace byla komunikace mezi procesy. Ihned se nabízí možnosti jako sdílení paměti, semafory a podobné techniky, věc však stojí za hlubší promyšlení. Komunikace má probíhat nárazově, vždy po zpracovaných balících. Je-li odstup dvou požadavků na jednu doménu 5 sekund a jejich nejvyšší povolený počet 500, dostaneme, že zpracování jednoho balíku potrvá 2500 sekund, tedy něco přes čtyřicet minut. Zde opravdu nemá opodstatnění snaha o rychlost meziprocesové komunikace. Navíc, naše úloha není náročná na procesor, ale na vstupní a výstupní operace. To vše mluví pro volbu soketové komunikace mezi procesy. Navíc s sebou tento způsob přináší další výhody, stručně je zmiňme. Komunikační protokol je jednoduchý a umožňuje experimentovat jak se serverem, tak s klienty, například srovnáváním jejich výsledků při současném běhu. Pokud by měl některý z klientů spadnout, zdaleka to neohrozí celý výpočet, rozložením na části se aplikace stává robustnější. Navíc má server možnost si odeslané balíky zazálohovat, a pokud dlouho nedostane odpověď, zařadit požadavky z nich zpátky do fronty. Tento efekt lze ještě posílit spuštěním klientů na jiných fyzických strojích, tedy distribuovaným během robota. Příjemná je možnost volného přidávání a ubírání klientů podle potřeby. Je jen potřeba hlídat, aby server i klient dodržovali stejná pravidla a nemohlo tak dojít k jejich porušení nebo k nějakým jiným anomáliím.

5. Výsledky běhu

V této kapitole je shrnutí zajímavých výsledků, které zatím robot podal – do odevzdání práce nestihl stáhnout všechen požadovaný obsah. Uvedeme souhrnné orientační údaje z nejdelšího běhu, který byl před uzávěrkou práce uskutečněn. Robot byl nastaven na stažení stránek v doméně .cz do hloubky 5 od stránek www.centrum.cz, www.cuni.cz, www.uhk.cz, www.cvut.cz, www.muni.cz, www.tul.cz, www.vslib.cz a www.upol.cz. Robot se postupně rozbíhal – po hodině aktivně pracovali dva klienti, bylo získáno 8,8 MB očištěného textu v 1400 dokumentech. Po hodině a půl se již účastnili čtyři klienti, textu bylo 21 MB a dokumentů 3500. Po dvou hodinách už pracovalo všech dvanáct spuštěných klientů, získali 74 MB textů a 20 000 dokumentů. Po třetí hodině jsme se rozhodli přidat dalších pět klientů. Tehdy jsme měli 178 MB textu ve 28 500 souborech. V těchto číslech nejsou zahrnuty soubory v jiných formátech než HTML. Ráno (10,5 hodin běhu) jsme přečetli, že robot stáhnul 215 000 dokumentů, které již zabíraly 1,3 GB. Podívali jsme se i na ne-HTML dokumenty. Bylo jich 9000 a zabíraly (nijak nečištěné!) 3,0 GB. Stále běželo všech sedmnáct klientů. Chvíli po tom šla tato práce do tisku.

Pro srovnání – jako první velká pokusná verze byl robot naprogramován neparalelně, tedy podle odstavců 2.2 a 3.1. Tehdy za 63 hodin stáhnul 226 tisíc dokumentů, které (očištěné) zabíraly 2,7 GB. Dokumenty v jiném než HTML formátu ignoroval, rozdíl ve velikosti dat připisujeme rozdílně vymezenému prostoru stahovaných stránek – šlo o stejný seznam bez www.centrum.cz, tentokrát ovšem seznam tvořil i omezení doménových jmen (z jiných se nestahovalo) a nebyla omezena hloubka stahování. Dá se tedy říci, že se tehdy robot pravděpodobněji dostal k obsažnějším stránkám. Na druhou stranu tehdejší výsledky tak dobře neilustrovaly stav při spuštění na celý internet.

Uvedené údaje toho celkem moc nevypovídají, pokud je nevztáhneme k číslům, jichž chceme dosáhnout. Předností získaného korpusu má být jeho velikost (nic jiného ani nezbývá – vzhledem ke zdroji textů a automatickému zpracování nelze čekat vyšší kvalitu než mají klasické, ručně tvořené korpusy). Očekává se 1-5 miliard slov – tento odhad plyne z informací vyhledávacích robotů, které evidují zhruba 50 milionů českých stránek. Uvažuje se tedy 100 slov na dokument. Nakolik je ten odhad přesný se nedozvíme, dokud data nestáhneme. Drtivá většina stránek obsahuje slov jen několik, pokud už na nich ale je nějaký souvislý text, je zpravidla delší než 100 slov. Chystáme-li se uvedenou rychlostí stáhnout všech 50 milionů stránek, čeká nás (opravdu velmi zhruba) sto dní nepřetržité práce, naštěstí tedy robotovy.

Interpretace, že zpracováváme přes dvacet tisíc požadavků za hodinu, je velmi pěkná a lákavá, je ovšem třeba se mít na pozoru – testování ukázalo, že prostředí internetu je značně nevyzpytatelné a na vývoj rychlosti mohou mít vliv netušené skutečnosti. Ale i kdyby ne, nemáme žádný dobrý odhad, jak se začne projevovat náročnost práce s datovými strukturami, až pořádně narostou, jestli nápor nových domén vůbec někdy poklesne a podobně.

Dále je vhodné uvážit, že naměřená čísla jsou z relativně (vzhledem k předpokládaným měřítkům stahování) malých experimentů. Zejména se na nich projeví "náběh" robota – ze začátku je fronta malá, a většina klientů čeká na balík. Teprve po několika hodinách se využije potenciál všech

klientů. Navíc se ukazuje být jako velmi podstatný efekt způsobený zařazováním nových domén – na začátku jich robot zařazuje hodně, a to ho brzdí. Bez větších experimentů lze jen těžko soudit, nakolik efekt s délkou běhu odezní.

Zastavme se u množství získaných dat. Lingvisté zřejmě texty měří na slova a věty – kolik jich robot získal ale bez segmenteru nevíme, budeme proto počítat v bytech. Bez HTTP hlaviček a souborů robots.txt klienti stáhli téměř 10,9 GB. Po odečtení nečištěných dokumentů dostáváme 7,9 GB HTML souborů, ze kterých zbylo po očištění pouhých 1,3 GB, tedy 16%. Přitom očištění lze chápat i jako oddělení obsahu od formy – při tomto pohledu je zjištěný poměr přinejmenším zajímavý. Navíc stále nejde o text, který se objeví v korpusu – příliš krátké útržky budou odstraněny segmenterem. Výtěžnost metody proto ještě významně klesne. Zarážející je také množství chyb – v přepočtu na počet pokusů vychází, že téměř každý desátý odkaz je nějakým způsobem nefunkční.

Uveďme ještě některé další údaje, vztahující se k době mezi desátou a jedenáctou hodinou běhu. Došlo k 319 spojením serveru s klientem, které zabraly celkem 13 minut. Slívání struktur dokumentů a nestažených požadavků nezabralo za celých deset hodin ani jednu vteřinu, zapracovávání nových odkazů zabralo 12 minut a konstrukce balíků požadavků 4 sekundy. Bylo evidováno 1560 IP adres a 4138 domén. Ve frontě čekalo na zpracování víc než 1,2 milionu požadavků. Počty dokumentů stažených jednotlivými klienty byly poměrně vyrované, kolem 14 000. Ukládání souborů na disk zabralo v souhrnu přes 10 minut, čištění HTML souborů jen něco přes 100 minut. Je třeba si uvědomit, že časové údaje se překrývají, což je dobře vidět na posledním údaji, který zároveň potvrzuje vhodnost paralelizace a úspěch implementace – celková doba čekání na odezvu internetových serverů za všechny klienty činí 29 hodin.

6. Instalace a použití programu

Tato kapitola má napomoci vyzkoušení robota přiloženého na CD (předpokládá unixový operační systém). Hned na začátku je třeba upozornit, že je nutno postupovat s jistou obezřetností a alespoň ze začátku nemít velké oči. Nechat robota bez dozoru může znamenat riziko zahlcení disku nebo linky a při obvzlášť neobratném nastavení i riziko potíží pro jiné stroje.

6.1 Instalace a spouštění

Instalace není nutná, jde o skript. Adresář se skriptem je třeba nechat pohromadě a nic z něj neubírat. Kromě něj je potřeba jen interpret Pythonu ve verzi alespoň 2.4 (a standardní moduly). Ke spuštění robota je určen shell-script textractor.sh (pro jistotu nemá právo být přímo spuštěn), v adresáři se skriptem je tedy třeba zadat

```
sh textractor.sh n [parametry]
```

kde `n` je požadovaný počet klientů správce fronty (povinný parametr), další paramtery jsou popsány níže. Skript spustí server – správce fronty a daný počet klientů a vrátí konzoli uživateli. Jejich výstup přesměruje do souborů s příponou `.out`. Identifikaci instance klientů nastavuje skript automaticky, jinak je všechno nastavení normální (viz níže). Po odhlášení ze systému robot poběží dál. Vedlejším výstupem spouštěcího skriptu je seznam čísel robotových procesů v souboru PID.txt (uložený do pracovního adresáře), který lze v případě potřeby robota zneškodnit využít k zavolání

```
kill [-9] `cat PID.txt`                (apostrofy jsou opravdu zpětné)
```

Jiné ovlivňování (než rozumné spouštění a ukončování klientů) za běhu se nepředpokládá.

6.2 Obsah a čtení záznamů

Jak už bylo uvedeno, robot svou činnost zaznamenává. Jednak pro dohledání jeho chování v případě něčí stížnosti, jednak pro zjišťování jeho poslední činnosti před případným pádem. Podle toho se také liší úroveň (detailnost) záznamu – na informační a ladící. Každý proces vypisuje souhrnné informace na terminál, kdyby jej chtěl někdo průběžně sledovat (tento výstup lze samozřejmě přesměrovat do souboru). Druhým výstupním kanálem jsou soubory podrobných záznamů.

Souhrnný výstup na terminál obsahuje zejména statistické informace (počty stažených dokumentů, spotřeba času na jednotlivé činnosti) – ty jsou uvozeny znakem %, což lze využít při jejich oddělování od zbytku souboru. Ostatní vypisované údaje dokládají aktuální činnost robota.

Do podrobného souboru robot zaznamenává pokusy o stažení dokumentu, úspěchy, nové odkazy, nově zařazené domény, IP adresy i požadavky a zařazování požadavků do balíku. Dále ještě zaznamenává průběh komunikace serveru a klientů. Řádky záznamů jsou uvozeny časem, spojením všech souborů a setříděním lze proto získat náhled na celkovou činnost robota.

6.3 Nastavení

Robota lze konfigurovat (pomineme-li editaci přímo zdrojového kódu) dvěma způsoby – konfiguračním souborem `config.txt` a parametry při spuštění. Při upravování souboru nastavení dbejte v něm uvedených pokynů. Nastavovat v něm lze totéž co na příkazovém řádku. Program má jistá výchozí nastavení, vhodná pro testovací spouštění – tak pomalý běh, že by nemělo dojít ke škodám a lze stíhat číst výpisy (jednoho klienta). Pokud tedy nenastavíte nic, použije se toto základní nastavení. Při kolizi parametrů v souboru a na příkazové řádce se použijí nastavení z příkazové řádky.

Parametry příkazové řádky jsou přijímány pouze v dlouhé formě. Ne všechny parametry mají smysl pro server i klienty zároveň, ale v zájmu jednoduchosti použití jsme podle toho parametry nerozlišovali. Co kdo nebude potřebovat, to prostě nepoužije.

Základní parametry:

`--version`	Vytiskne informaci o verzi.
`--help`	Vypíše velmi stručnou nápovědu.
`--target=cesta`	Cesta, kam se budou ukládat stažené dokumenty (pokud se nebudou rovnou posílat k dalšímu zpracování – zatím ale není kam).
`--non-html-target=cesta`	Cesta, kam se budou ukládat neočištěné dokumenty.
`--disallowed-target=cesta`	Cesta k souboru, ve kterém vznikne seznam nepovolených URL (pro robota).
`--urls=cesta`	Cesta k souboru se seznamem výchozích adres. (kompletní URL, tj. včetně "http://", oddělené mezerami)
`--log-path=vzor`	Vzor cest, kde budou vznikat soubory pracovních záznamů.
`--instance=nazev`	Identifikace klienta (používá ji při komunikaci se serverem). Při použití spouštěcího skriptu nemá význam.
`--batch-count=cislo`	Počet balíků, které má klient zpracovat (potom skončí). Tento parametr lze s výhodou využít při experimentování s počtem robotů.

Komunikace procesů robota:

`--harv-host=nazev`	Název počítače, ze kterého se budou připojovat klienti. Prázdný řetězec znamená localhost, stejně jako neuvedení parametru.
`--qk-host=nazev`	Název počítače, na který se budou připojovat klienti (tzn. běží na něm server). Prázdný řetězec znamená localhost, stejně jako neuvedení parametru.

`--harv-host=nazev`	Název počítače, ze kterého se budou připojovat klienti. Prázdný řetězec znamená localhost, stejně jako neuvedení parametru.
`--port=cislo`	Číslo portu, na kterém bude server naslouchat. Volte větší než 65536, nebo se zeptejte správce počítače nebo sítě.
`--instance=nazev`	Identifikace klienta (používá ji při komunikaci se serverem). Při použití spouštěcího skriptu nemá význam.

Omezení stahovaného prostoru:

Není otestováno spojení se segmenterem, robot tedy nemá žádnou zpětnou vazbu a tím ani omezení stahovaného prostoru. Důrazně proto doporučujeme spouštět jej s omezenou hloubkou prohledávání nebo s omezením na doménu .cz (`--include="\.cz"`).

`--exclude=vyraz`	Regulární výraz specifikující zakázané servery (ten, který výraz splní, nebude zpracován). Žádné omezení: `^$` (splní jen prázdné doménové jméno).
`--include=vyraz`	Regulární výraz specifikující povolené servery (všechny zpracované servery ho musí splnit). Žádné omezení: `.*` (splní cokoliv).
`--depth=cislo`	Maximální hloubka stahování (počet potřebných odkazů k dosažení "poslední" prohledávané stránky). Pro neomezenou hloubku nastavte 0.

Omezení zátěže:

`--req-delay=sekundy`	Doba prodlevy mezi dvěma požadavky (na stažení souboru) na jednu doménu v sekundách.
`--max-hosts=pocet`	Nejvyšší povolený počet současně kontaktovaných serverů na jeden fyzický stroj.
`--max-reqs=pocet`	Nejvyšší počet požadavků na jeden server mezi generálními prodlevami.
`--general-pause=sekundy`	Délka generální prodlevy v sekundách.

Ladění:

`--wait=sekundy`	Bezpečnostní čekání po každém požadavku v sekundách (při reálném provozu nastavit na 0 – to je i implicitní hodnota).
`--log-level=uroven`	Úroveň záznamu - informační "INFO" (implicitní hodnota) nebo podrobná (pro ladění) "DEBUG".

7. Závěr

Cílem práce bylo zajistit prostředek, jehož pomocí by šlo získat velké množství textu dostupného na internetu pro účely lingvistického výzkumu. Rozhodli jsme se implementovat vlastního robota, který rekurzivně prochází internetové stránky, čistí je a ukládá, a následuje v nich obsažené odkazy. Hlavními kritérii bylo správné očištění textů a nenápadnost robotových aktivit, včetně dodržování příslušných standardů. Tento cíl se podařilo úspěšně naplnit, robot je využitelný pro daný účel a plní všechny uvedené požadavky. Během vývoje objevené komplikace jsme odstranili, požadavky na robota postupně zpřesnili a přidali některé nové rysy.

Dále je na místě poznamenat, že čištění a komunikace se segmenterem nejsou nedílnou součástí robota – ten je tedy využitelný mnohem obecněji (zejména tam, kde se potřebujeme vyhnout nepřijatelné zátěži síťového okolí), nejen pro získávaní dat pro jazykový korpus. Kromě jeho přínosu pro lingvistiku navíc poskytuje zajímavé informace o obsahu a struktuře českého internetu a českých internetových stránek (s trochou opatrnosti by navíc jistě šlo výsledky zobecnit).

7.1 Co zbývá dopracovat a možné směry dalšího vývoje

Samozřejmě zbývá implementovat utility pro extrakci textů ze souborů uložených bokem pro pozdější zpracování. Další vývoj ovšem bude záviset především na tom, jak se získané texty osvědčí při konstrukci výsledného korpusu. Pokud ano (měly by, když odpovídají zadání), bude třeba definitivně rozhodnout otázky spojené s dlouhodobým provozem – jestli bude korpus uzavřen jako hotový (asi sotva), jestli bude snaha udržovat aktuální zrcadlo stavu českého internetu, nebo jestli bude korpus růst o změněné dokumenty (nejpravděpodobnější), jak často má k případným aktualizacím docházet (nebo přesněji: jak rozhodnout jak často aktualizovat) a další koncepční otázky.

Specifickou otázkou je aktualizace souboru robots.txt – v současné době je stáhnut pro každou doménu jednou, ovšem pokud by mělo dojít ke stahování v delších časových horizontech, bylo by zřejmě vhodné jej aktualizovat. Implementačně se nejedná o problém, pokud server poskytuje v hlavičce odpovědi čas poslední modifikace, to ale není zcela běžné. V ostatních případech nezbývá než soubor pravidelně stahovat znovu a znovu – otevřenou otázkou zůstává, jak často. Navíc, pokud bude změněn obsah souboru, bude třeba znovu zkontrolovat všechny už zařazené požadavky, nejlépe těsně před posláním na server. K tomu nyní není důvod, požadavky kontrolují se jen při zařazování do fronty.

U každé paralelizované úlohy je podstatnou otázkou, jakou zvolíme míru paralelizace, abychom dosáhli nejlepšího výsledku. V současnosti je robot spouštěn s konstantním počtem klientů, který lze případně ručně upravovat (nástroji poskytnutými operačním systémem). Přitom je zřejmé, že v různých okamžicích je optimální jiný počet – mění se velikost fronty, propustnost sítě, zatížení procesoru a další parametry, které mohou způsobovat, že je klientů málo (škoda strojového času)

nebo moc (zbytečné ztráty na režii serveru i procesoru). Zřejmě by stálo za to situaci průběžně sledovat a počet klientů dynamicky přizpůsobovat aktuálním podmínkám.

V současnosti je pořadí zpracovávání jednotlivých IP adres a jejich požadavků nedefinované (záleží na tom, co vrací metoda pop příslušné struktury, což není definováno). Vzhledem k době stahování může být žádoucí, aby byly lepší dokumenty stahovány dříve. To znamená jednak vyšší prioritu domén, které mají kratší odezvu, druhak potřebu odhadnout kvalitu dokumentu před jeho stažením. Sem směřovala zmínka o dalším využití zpětné vazby segmenteru – ta by k tomuto účelu šla jistě využít. Přesto zůstává metoda odhadu otevřenou otázkou.

Nejmlžnějším směrem vývoje, který se ani tak netýká robota samotného, je úvaha o dalších potenciálně dostupných zdrojích textu – diskusních skupinách a konferencích a dalších. Čím specifičtější (a méně dobře reprezentující "češtinu", záměrně v uvozovkách) ovšem zdroj bude, tím více si bude třeba dávat pozor při interpretaci výsledků vzniklých zpracováním těchto textů.

Literatura

: nebude to vypadat blbě?

- oReilly
- python documentace
- Jen ta, na kterou se odkazuju!
- weby?
- rob. exc std

Do Literatury uvedte toho Oreilyho a klidne i internetove odkazy.

http://www.robotstxt.org/wc/norobots-rfc.txt??spec-rfc
http://www.gnu.org/software/wget/wget.html
http://www.robotstxt.org/wc/norobots.html??text

python.org/dictionary

Příloha A – Obsah přiloženého CD

Na CD je jediný adresář a PDF soubor s touto prací. Obsahem adresáře jsou sobory samotného skriptu, jejich použití je popsáno v kapitole 6:

`docs`	Prázdný adresář, je implicitní cestou pro ukládaní stažených dokumentů. Je tam proto, abyste ho nezapomněli vytvořit.
`nohtml`	Totéž pro soubory jiného než HTML typu (budou se ukládat neočištěné).
`textractor.sh`	Shell-script pro spouštění serveru seoučasně s několika klienty.
`queuekeeper.py`	Skript serverové části robota.
`harvester.py`	Skript klientské části robota.
`my_sock.py`	Knihovna pro snažší komunikaci pomocí soketů.
`txtr_html06.py`	Knihovna pro čištění HTML souborů.
`timer.py`	Knihovna pro měření spotřebovaného (reálného, ne strojového) času.
`common_conf.py`	Modul pro nastavení robota (obou částí), čte konfigurační soubor a parametry příkazového řádku.
`common_ds.py`	Popis společných datových struktur.
`usr_conf.py`	Link na soubor `config.txt`.
`config.txt`	Soubor s uživatelským nastavením robota.