TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Distribuovaný web crawler

Distributed web crawler

Diplomová práce

Autor: **Bc. Ondřej Novák**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 10. 05. 2011

Místo tohoto listu bude vloženo originální zadání s podpisy

Anotace

Webové prohlížeče, stejně jako další specializované nástroje pro získávání dat z WWW, používají web crawlery k vytváření rozsáhlých kolekcí webových stránek. Diplomová práce se zabývá vytvořením distribuovaného web crawleru. První částí práce je návrh architektury distribuovaného web crawleru. Důraz je kladen na problematiku tvorby distribuovaných aplikací a jejich řízení. Ve druhé části práce je popsán vytvořený distribuovaný web crawler a použité technologie. Základem aplikace je vícevláknový URL server řídící web crawlery distribuované na klientských počítačích. Klient / server komunikace je řešena pomocí SOAP protokolu a o přenos souborů se stará FTP server. V závěru práce jsou provedeny testy demonstrující schopnosti distribuovaného web crawleru a je vytvořen obslužný manuál.

Klíčová slova: distribuované programování, Python, SOAP, vícevláknové programování, web crawler, webové služby, World Wide Web

Annotation

Broad web search engines as well as other specialized tools used for data retrieval from the WWW use web crawlers to create large collections of web pages. This thesis deals with the creation of distributed web crawler. In the first part of the thesis the architecture of distributed Web crawler is created. Emphasis is placed on the issue of creating distributed applications and their management. The second part describes the developed distributed Web crawler and applied technologies. The basis of the application is multithreaded URL server that manages distributed web crawlers to client computers. Client / server communication is based on SOAP and file transfers provides an FTP server. Finally the possibilities of developed distributed web crawler are be presented in a few tests and the user manual is included.

Keywords: distributed computing, Python, SOAP, multithreading, web crawler, web services, World Wide Web

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **souhlasím** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 20. května 2011

Podpis:

Poděkování

Zaprvé bych chtěl poděkovat svým rodičům, že mě vždy podporovali a umožnili mi studovat. Všem lidem, kteří mi vědomě, či nevědomě pomáhali během vypracovávání diplomové práce. Kamarádům, kteří při mně vždy stáli, podporovali mě a snášeli mé nálady a ponocování. Chtěl bych poděkovat vedoucímu práce Mgr. Jiřímu Vranému, Ph.D. za dobré vedení a cenné rady týkající se vypracování práce.

Obsah

Seznam obrázků

Seznam ukázek z kódu

Seznam grafů

Seznam tabulek

Slovník použitých pojmů a zkratek

- **API** (Application Programming Interface) - rozhraní pro programování aplikací
- **ASCII** (American Standard Code for Information Interchange) – základní znaková sada v informatice pro reprezentaci textu
- **BFS** (breadth-first search) – algoritmus pro procházení grafů
- **crawling** – proces procházení WWW, stahování a zpracovávání dat
- **CORBA** (Common Object Request Broker Architecture) - distribuovaná objektová technologie vytvořená v rámci skupiny OMG
- **DCOM** (Distributed Component Object Model) - distribuovaná objektová technologie od Microsoftu
- **deque** (double oriented queue) – knihovna jazyka Python
- **distribuovaný systém –** seskupení nezávislých počítačů zapojených do téže sítě, které spolupracují na řešení nějakého úkolu
- **XML** (eXtensible Markup Language) – obecný značkovací jazyk
- **Firewall** - síťové zařízení sloužící zabezpečení síťového provozu
- **FTP** (File Transfer Protocol) – služba Internetu pro přenos souborů přes WWW
- **HTML** (Hypertext Markup Language) **–** jazyk pro vytváření WWW stránek
- **HTTP** (Hypertext Transfer Protocol) – základní přenosový protokol WWW
- **IDN** (Internationalized Domain Name) – označení domén obsahujících znaky národnostní abecedy
- **MIME** (Multipurpose Internet Mail Extensions) - Internetový standard rozšiřující formát elektronické pošty o možnost přenosu non-ASCII souborů
- **MOM** (Message-oriented middleware) - softwarová nebo hardwarová infrastruktura zaměřená na výměně zpráv mezi distribuovanými systémy

- **mutex** (Mutual exclusion) - synchronizační algoritmus vzájemného vyloučení
- **PageRank** – algoritmus umožňující ohodnocení důležitosti webových stránek
- **RPC** (Remote Procedure Call) - protokol pro vzdálené volání procedur
- **SOAP** (Simple Object Access Protocol) - protokol pro výměnu zpráv založený výměně XML zpráv pomocí HTTP
- **RMI** (Remote Method Invocation) – distribuovaná objektová technologie používané v Javě
- **TCP/IP** – Internetový komunikační protokol
- **UDDI** (Universal Description, Discovery and Integration) - platformně nezávislý na XML založený registr sloužící k prezentování a lokalizaci aplikací webových služeb
- **URI** (Uniform Resource Identifier) - jednotný identifikátor zdroje informací v Internetu
- **URL** (Uniform Resource Locator) - lokalizátor objektů v Internetu
- **web crawler** – automatizovaný program procházející WWW
- **webová služba** – metoda pro komunikaci strojů přes Internet
- **WSDL** (Web Services Description Language) – XML orientovaný jazyk umožňující popis webových služeb
- **WWW** (World Wide Web) – služba Internetu umožňující zpřístupnění informací

1 Úvod

Web crawlery (zkráceně crawlery) tvoří v informatice specifický druh programů sloužící k automatickému procházení World Wide Webu[1] (dále WWW nebo Web). Jsou podkladem existence internetových vyhledávačů typu Google, Yahoo, Bing, které je mimo jiné využívají ke sběru, aktualizaci a zálohování dat. Pro svoje „putování“ crawlery sledují grafovou strukturou Webu, která jim umožňuje navštěvovat další stránky. Jak tomu bývá, toto označení není jediným, které tyto programy definuje. V literatuře se lze setkat s anglickými názvy wanderers, robots, spiders, worms – pro lepší porozumění „poutníci“, „roboti“, „pavouci“, „červi“. Nejčastěji používaným označením je ale Web crawler, jehož nejbližší český ekvivalent je „procházeč“, „prohledávač“.

Sít' WWW lze označit za jeden z nejvýznamnějších informačních zdrojů dnešní doby. Kromě toho patří mezi největší a nejrychleji se rozvíjející zdroje. Na počátku vzniku crawlerů stály dva faktické požadavky a to zjistit, co vše se na webu nachází a jelikož tyto informace rychle zastarávaly, udržet je aktuální. Z historického hlediska je známo, že ještě ke konci roku 1993 Web obsahoval „pouhých“ 623 webových stránek.[2] Přesto, že by se toto číslo mohlo dnes zdát směšné, na tehdejší dobu to byl nevídaný úspěch, zejména přihlédneme-li k tomu, že během jediného roku se více než zdesetinásobilo. V současnosti indexovatelný Web, podle zprávy *worldwidewebsize.com*[3] publikované 14. dubna 2011, obsahuje nejméně 14,6 miliardy stránek a s rychlostí rozpínání WWW již za týden toto číslo nemusí být aktuální. Proto vytváření a udržování rozsáhlých kolekcí se záznamy o webových stránkách je nutností pro zajištění přístupu k aktuálním informacím. Kolekce vytvořené web crawlery mohou být například využity jako podklady pro vyhledávače, k výzkumným účelům, nebo mohou posloužit jako záloha obsahu Webu.

1 Dostupné na WWW: http://www.w3.org/WWW/ [cit. 2011-05-14].
2 Dostupné na WWW: http://www.mit.edu/people/mkgray/net/web-growth-summary.html [cit. 2011-05-14].
3 Dostupné na WWW: http://www.worldwidewebsize.com/ [cit. 2011-04-14].

Aby web crawlery dokázaly udržet krok s růstem WWW, bylo nutné zefektivnit celý proces crawlování. Sekvenční zpracovávání úloh u stávajících crawlerů bylo natolik pomalé, že data zastarávala rychleji, než mohla být aktualizována. Za tímto účelem byly vytvořeny distribuované a paralelní web crawlery, o kterých je tato práce.

1.1 Cíle

V zadání diplomové práce byly stanoveny dvě primární zásady pro její úspěšné vypracování:

- Seznámit se s problematikou vícevláknových a distribuovaných aplikací.
- Navrhnout a implementovat distribuovaný web crawler.

Navrhnout způsob distribuce web crawleru a vytvořit ucelený model pro realizaci distribuované aplikace, na jehož základu bude následně vytvořen distribuovaný web crawler. Aplikace bude vytvářena jako stabilní základ pro další rozvoj a možnosti výzkumu crawlování. Je velice důležité, aby distribuovaný web crawler byl dostatečně efektivním a robustním softwarem. Vyžaduje se od něj schopnost autonomního procházení Webu, stahování dat a jejich základního zpracovávání. V návrhu distribuovaného web crawleru by měly být zahrnuty veškeré požadavky kladené na politiku chování crawlerů pro přístup k webovým serverům. Neméně důležitým požadavkem je implementace vícevláknového programování do výsledné aplikace.

Programový kód musí být dobře strukturovaný a čitelný, aby ho bylo možné nadále rozvíjet a využívat k řešení sofistikovaných úloh.

1.2 Pojem web crawler

Podívejme se trochu blíže na to, jak crawler funguje. Cyklus běhu celého programu je nastartován listem URL adres (Uniform Resource Locator), který určí s jakými stránkami má web crawler započít svoji pouť. Stránky jsou postupně navštěvovány a vytěžovány pro požadované informace. Cyklus proto, že jeho práce nekončí zpracováním zadaných adres. Crawler využívá získaných hypertextových odkazů za

účelem navštívení dalších stránek, nalezení a zpracování informací z nich a tak pořád dokola.

Ač by tento popis mohl poukazovat na to, že crawlování je banálním procesem, opak je pravdou. Crawler by se dal přirovnat k automatizovanému, na uživateli nezávislému robotu putujícímu po Webu. Právě problematika spojená s WWW je pro crawler zásadní. Před programátora staví otázky týkající se způsobu komunikace, zpracovávání HTML (Hyper Text Markup Language) stránek a ukládání velikého množství dat. Web crawler může interagovat i s miliony počítačů v rozmezí několika týdnů. Jeho běh je tedy velice delikátní záležitost. Kromě otázek robustnosti, flexibility a správy samotného programu je nutné řešit ohleduplnost při přístupu k webovým serverům, aby nedocházelo k jejich přetěžování nebo získávání zakázaných zdrojů. Proto by se crawlery měly řídit kodexem definujícím podmínky bezproblémového pohybu po síti.[4] Příkladem nebezpečnosti crawlerů může být jejich využívání za účelem útoků na webové servery a služby.

1.3 Pojem Distribuovaný web crawler

Distribuované web crawlery jsou rozšířením klasických crawlerů. Nadstavba spočívá v možnosti masivního využití výpočetní síly propojených počítačů (pracovních stanic) za účelem zefektivnění celého procesu. Crawlery jsou v tomto případě rozmístěny na více stanic, které nemusí být centralizovány v jediném místě. Jelikož web crawlery operují na Webu, je nasnadě využít tuto síť za účelem jejich vzájemného propojení.

Zřetězení výpočetního výkonu počítačů za účelem crawlování je pouze jedním ze způsobů využití distribuovaných aplikací, tedy rozhraní vzniklého propojením desítek, stovek, tisíců autonomních počítačů z téže sítě. Počítačová síť představuje prvek umožňující propojení jednotlivých částí distribuovaného systému do jediného celku. Jedním z rysů distribuovaných systémů je schopnost využívání hardwarových (HW) zdrojů propojených stanic. Skutečnost, že počítače poskytují systému svoje procesory (CPU), paměti, představuje velice dostupný způsob pro vytvoření superpočítače. Důvod pro vytváření takto masivních systémů s obrovskou výpočetní silou leží v potřebě

4 Dostupné na WWW: http://www.robotstxt.org/guidelines.html [2011-05-14].

zpracovávání zvláště náročných problémů. Příkladem může být World Community Grid[5], kde v rámci celosvětově propojené sítě počítačů probíhá výzkumu genetických kódů, nebo seti@home[6] projekt zaměřený na hledání mimozemské inteligence.

Výhody využívání distribuovaných systémů:

- Navýšení výpočetního výkonu.
- Sdílení služeb (webové a aplikační servery).
- Přístup k datům (databáze, sběr dat).
- Decentralizované algoritmy (řízení, multi-agentní technologie).

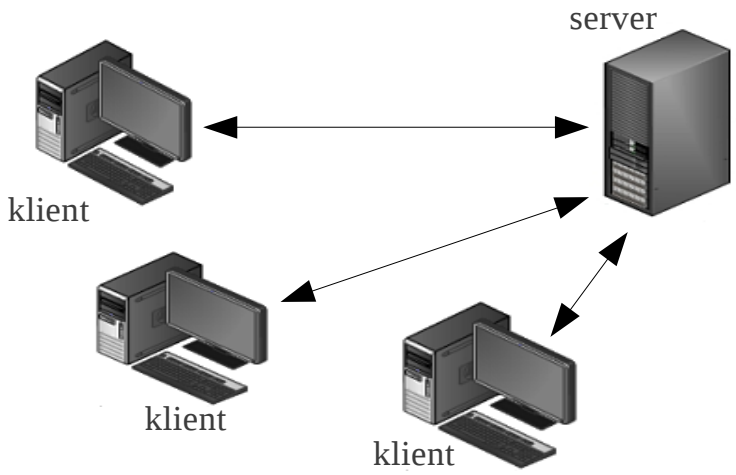
5 Dostupné na WWW: http://www.worldcommunitygrid.org [cit. 2011-03-28].
6 Dostupné na WWW: http://setiathome.berkeley.edu/ [cit. 2011-05-14].

2 Návrh počáteční architektury

Počátečním impulzem vedoucím k vytvoření této práce byla představa využití více počítačů, které by kooperovaly při procesu crawlování za účelem jeho celkového zefektivnění. Návrh aplikace je třeba směrovat k vysoce optimalizovanému systému, jak po stránce funkčnosti, stability, tak i rychlosti. Hlavním cílem práce je vytvoření distribuovaného crawleru schopného dlouhodobého samostatného běhu, bez zásahu administrátora či uživatele. Dlouhodobé crawlování znamená zpracovávání desítek až stovek milionů stránek v rozmezí několika týdnů, či měsíců. Interakce s natolik obrovským množstvím webových stránek bude výzvou systémového designu, správy I/O a efektivního využití sítě.

2.1 Distribuované aplikace

Vývoj distribuované aplikace je založen na technologii vícevrstvých aplikací.[7] Ve své nejjednodušší podobě (dvouvrstvé) je distribuovaná aplikace synonymem klient/server architektury, která využívá přesně danou množinu pravidel definujících chování mezi dvěma spolupracujícími procesy.



Obrázek 1: Klient server model diagram

7 Dostupné na WWW: http://java.sun.com/developer/Books/jdbc/ch07.pdf [cit. 2011-05-14].

Jedna z definic popisujících klient/server architekturu[8] ji vymezuje jako rozhraní pro možnost distribuování aplikace na dvou či více počítačích za účelem efektivnějšího využití.

2.2 Volba komunikačního modelu

Distribuovaná aplikace je založena na vztahu mezi klientem a serverem. Od toho se odvíjí i podoba celého komunikačního schématu. Propojení softwarových modulů distribuované aplikace lze realizovat řadou pro to vytvořených technologií a postupů, mezi které patří distribuované objektové technologie (CORBA, DCOM, RMI, atd.), MOM systémy, RPC, webové služby. Pro realizaci distribuovaného web crawleru byly zejména diskutovány dva poslední postupy. Komunikační model je založen na technologii webových služeb a protokolu SOAP.

2.2.1 Vzdálené volání procedur

RPC (Remote Procedure Call) je souborem pravidel a jejich implementací pro vzdálený přístup k procedurám přes Internet. Nabízí tedy mechanismus, jak z jednoho programu volat funkci umístěnou na vzdáleném systému. RPC protokol definuje metodu pro převod parametrů a výsledků volané funkce do formátu, ve kterém budou RPC požadavky předány po síti. Data jsou zapouzdřena pomocí značkovacího jazyka XML (eXtensible Markup Language) a přenášena http (HyperText Tranfer Protokolem). Tato koncepce umožňuje aplikacím komunikaci mezi různými počítačovými architekturami a jejich operačními systémy. Projekt je v současné době již dlouhou dobu ukončen, nicméně se stal předlohou pro rozšířený protokol SOAP (viz. kapitola 2.3).[9] [10]

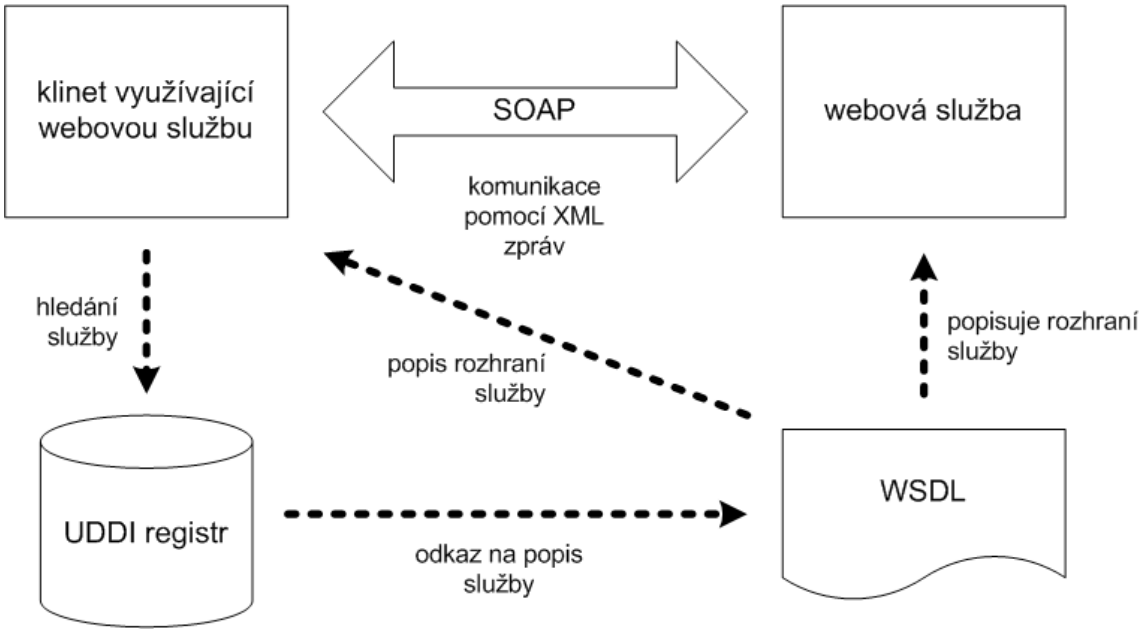
8 Dostupné na WWW: http://www.gantthead.com/content/processes/8615.cfm [cit. 2011-04-20].
9 Dostupné na WWW: http://en.wikipedia.org/wiki/Remote_procedure_call [cit. 2011-04-21].
10 Dostupné na WWW: http://www.kosek.cz/diplomka/html/ch04s02.html[cit. 2011-04-21].

2.2.2 Webové služby

Webové služby umožňují síťovou komunikaci mezi počítači nezávisle na platformě a formátu dat. Představují tedy společný předpis pro komunikaci cizích systémů. Webové služby bývají často řazeny pod hlavičkou messaging systémů pro přímou možnost emulace MOM a některé společné rysy. Stejně jako ony jsou založené na principu zasílání zpráv mezi aplikacemi. Velkou výhodu jim poskytuje schopnost levně integrovat aplikace naprogramované v různých jazycích, bežících na různých platformách. Aplikace spolu komunikují zasíláním XML zpráv pomocí HTTP protokolu. A právě využití těchto široce podporovaných standardů přispívá k tomu, že i webové služby jako celek jsou dostupné v podstatě všude. Základem webových služeb je SOAP(Simple Object Access Protocol), který definuje strukturu zpráv a způsob, jakým se do XML kódují běžné datové typy. Popis rozhraní webové služby (kde je služba k dispozici, jaké má vstupní/výstupní parametry) je možné formálně specifikovat s pomocí WSDL (Web Services Description Language).[10]

2.3 Využití webových služeb a protokolu SOAP při komunikaci

SOAP je základním protokolem pro komunikaci u webových služeb. Možnost použití webových služeb a SOAP napříč platformami společně s dostupností knihoven vedla k myšlence využít tento protokol pro návrh distribuovaného web crawleru. Hlavní důvod tohoto rozhodnutí spočíval ve způsobu, jakým SOAP komunikuje. Jak bylo naznačeno, webové služby umožňují jednoduchou komunikaci mezi aplikacemi v heterogenním prostředí. SOAP komunikace využívá platformě nezávislých standardů jazyka XML a protokolu HTTP. Aplikace si mezi sebou vyměňují XML zprávy (nesené pomocí HTTP) přenášející dotazy a odpovědi jednotlivých aplikací. HTTP je jeden z hlavních a nejpoužívanějších protokolů používaných na Webu pro přenos informací.



Obrázek 2: Komunikace webové služby[*]

Obrázek 2 zobrazuje interní komunikaci webových služeb. Hlavní částí je přenos zpráv pomocí SOAP. WSDL (Web Services Description Language) je navržený za účelem vytváření formálních popisů webových služeb. Lze tak přesně informovat o tom, co webová služba nabízí za funkce a určit způsob, jak se jí na ně ptát. WSDL, stejně jako SOAP, je tvořen XML dokumentem.[11] UDDI (Universal Description, Discovery and Integration) je platformně nezávislý, na XML založený, registr („telefonní seznam“) sloužící k prezentování a lokalizaci aplikací webových služeb. Převážně je využíván v komerční sféře. Umožňuje firmám jak uveřejňovat, tak naopak i získávat seznam existujících služeb. Samozřejmostí je parametrické vyhledávání v registru. Každá uveřejněná služba (aplikace) musí obsahovat předpis toho, jak s ní komunikovat, jak ji využít.[12]

SOAP používá pro přenos požadavku/odpovědi protokol HTTP. Jedním z důvodů je bezesporu široká podpora v aplikacích. Webová služba může být přímo nahrána na běžný server, jelikož klasická klient/server komunikace funguje na základě předávání zpráv pomocí HTTP. V tomto případě server slouží jako „dispečer“, který překládá komunikaci a jednotlivé požadavky předává odpovídající webové službě ke zpracování.

* Obrázek dostupný na WWW: http://www.kosek.cz/diplomka/html/websluzby.html [cit. 2011-04-21].
11 Dostupné na WWW: http://www.w3schools.com/wsdl/wsdl_intro.asp [cit. 2011-04-22].
12 Dostupné na WWW: http://www.w3schools.com/WSDL/wsdl_uddi.asp [cit. 2011-04-22].

Po zpracování požadavku službou je vrací jako výsledky tazateli. Další výhoda HTTP spočívá ve faktu, že síťová infrastruktura umožňuje komunikaci přes port vyhrazený pro HTTP (TCP port 80). Webové služby je tak možné používat bez nutnosti zásahu do konfigurace aktivních síťových prvků (firewally). Toto je jedna z hlavních výhod oproti ostatním distribuovaným protokolům (DCOM, CORBA, atd.), které budou na restriktivních firewallech zakázány.[10]

2.3.1 Struktura SOAP zprávy

SOAP požadavek je zasílán v těle HTTP dotazu. Využívanou metodou je POST (viz obrázek 3), která umožňuje přenos dat v těle HTTP. První částí zprávy je HTTP hlavička. V hlavičce se může definovat URI identifikující zdroj informací pomocí *SOAPAction*. Přítomnost URI v hlavičce bývá využívána servery a firewally k filtrování SOAP požadavků. SOAP zpráva samotná má formu jednoduchého XML dokumentu s kořenovým elementem *Envelope*. V této „obálce“ jsou uzavřeny dva elementy - *Header* (hlavička) a *Body* (tělo). Hlavička SOAP je nepovinná a používá se pro přenos pomocných informací důležitých ke zpracování zprávy (např. identifikaci uživatele, autentizační informace (jméno, heslo) apod). Tělo zprávy (*Body*) tvoří nosnou část dokumentu. Definuje informace identifikující požadovanou službu a předávané parametry, popřípadě návratové hodnoty služby.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

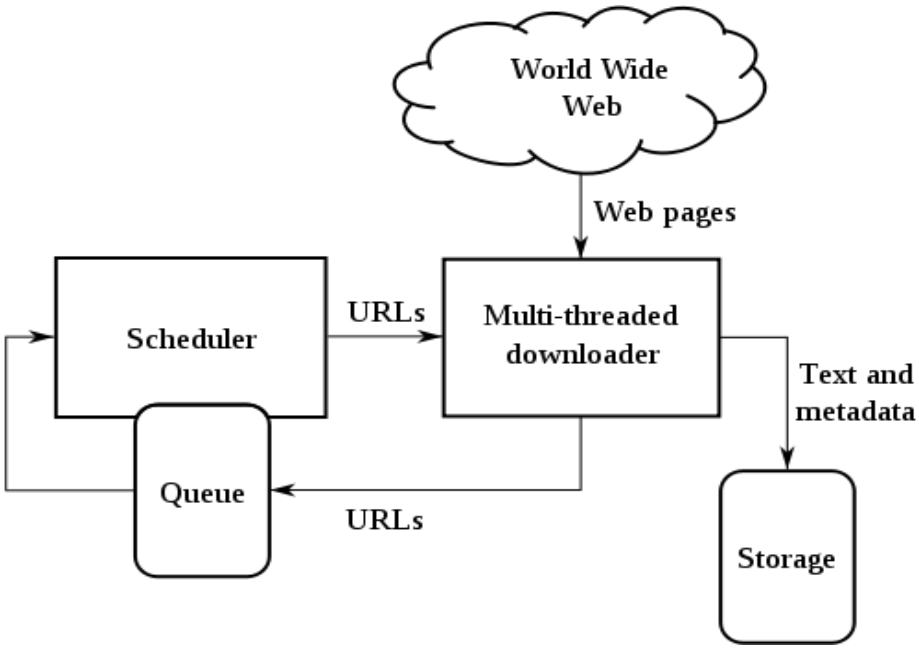
Obrázek 3: Příklad SOAP zprávy v HTTP požadavku[*]

10 Dostupné na WWW: http://www.kosek.cz/diplomka/html/ch04s02.html [cit. 2011-04-20].

* Obrázek dostupný na WWW: http://www.w3schools.com/soap/soap_example.asp [cit. 2011-05-15].

2.4 Obecná architektura Web Crawleru

Web crawlery se nezabývá mnoho prací. Částečně je to způsobeno tím, že tyto aplikace jsou součástí webových prohlížečů a podobných korporačních systémů založených na masivním stahování dat z WWW. Algoritmy těchto systémů jsou chráněny jako firemní tajemství společností a nejsou veřejně publikovány. Zaměřme se tedy na nějaký z dostupných crawlerů (např. UbiCrawler[13], Mercator[14]). Významem architektury je vytvoření schématického rámce celého projektu.



Obrázek 4: Typická high-level architektura Web crawleru[*]

Toto schéma (obrázek 4) vytvořené Carlosem Castillem v doktorandské práci Effective Web Crawling[15] názorně zachycuje datový tok příznačný pro crawlery. Centrální částí návrhu je *Multi-threaded downloader*. Přejmenujme ho na Obecný crawler pro potřeby tohoto projektu. Obecný crawler přijímá na vstupu seznam adres vybraných z fronty. Proces *Scheduler* naznačuje možnost řízeného přidělování adres odvozenou od strategie

13 Dostupné na WWW: http://ausweb.scu.edu.au/aw02/papers/refereed/vigna/paper.html [cit. 2011-05-15].
14 Dostupné na WWW: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.9096&rep=rep1&type=pdf [cit. 2011-05-15].
* Obrázek dostupný na WWW: http://www.chato.cl/papers/crawling_thesis/effective_web_crawling.pdf [cit. 2011-04-29].
15 Dostupné na WWW: http://www.chato.cl/papers/crawling_thesis/effective_web_crawling.pdf [cit. 2011-04-29].

procházení. Crawler stahuje WWW stánky, na které odkazují získané URL adresy. Hlavní datové struktury tvoří entity *URL queue* (fronta) a *Web page Storage* (úložiště). Crawler vyúsťuje ve dva výstupní proudy. Na jedné straně dochází k ukládání požadovaných dat do obecného úložiště. Základním úkolem crawleru je „vytěžovat“ URL odkazy z každé navštívené stránky a využít je pro další putování. To je znázorněno druhým výstupem v podobě URL adres, které jsou vkládány zpět do fronty. U crawlerů je pravidlem, že množství nalezených adres daleko převyšuje počet zpracovaných. Tento fakt umožňuje nepřetržitý chod crawleru až do vyčerpání všech URL adres.

Na úrovni obecného crawleru musí probíhat zpracovávání stránek. Tedy jenom v případě, jsou-li krom HTML stránek požadovány na výstupu dodatečné informace. Rozložme obrázek 4 na posloupnost jednotlivých operací, která bude více vypovídat o vnitřních procesech:

- Získej seznam URL z fronty a ulož si je do paměti.
- Stáhni HTML stránku/dokument odpovídající první URL.
- Je-li to HTML dokument, rozlož ho na jednotlivé části.
- Najdi veškeré hypertextové odkazy a vlož je do fronty.
- Získej ostatní požadované informace a ulož je do datového úložiště.
- Vrať se na první bod a opakuj.

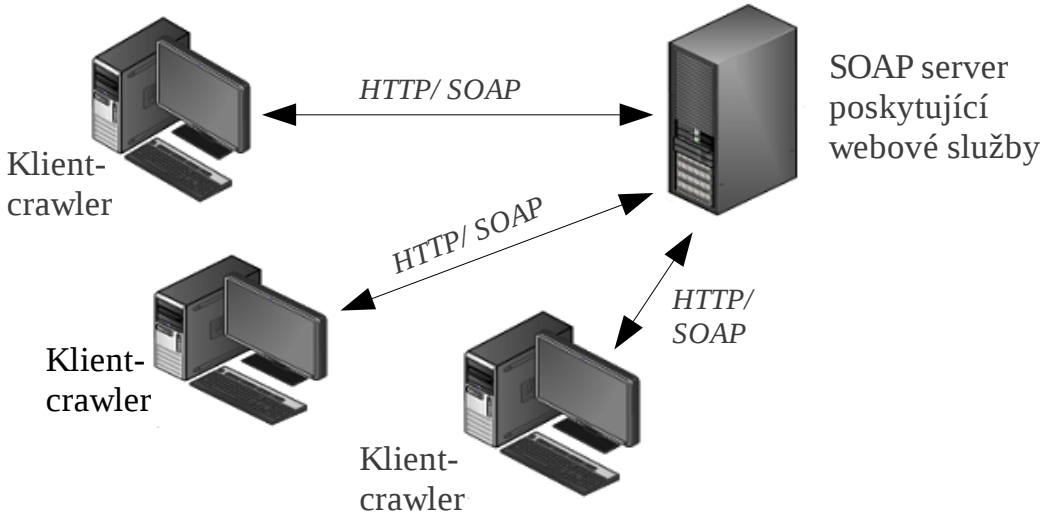
Tyto body představují popis primárních funkcí web crawleru, které jsou vždy neměnné. Další část tvoří rozšíření těchto bodů o požadavky kladené jak na funkčnost, tak povinnosti a náležitosti, které vytvářený distribuovaný web crawler musí splňovat:

- Crawler musí být vytvořen jako tzv. „fault tolerant“ systém. To znamená, že při výskytu chyby nesmí zhavarovat, ale naopak bude pokračovat dále v běhu.
- V případě zastavení crawleru nesmí dojít ke ztrátě dat.
- Každá získaná URL musí být navštívena právě jednou.
- Web crawler musí být schopen prohledávat webové stránky za účelem získávání požadovaných dat.

- Musí brát ohled na místo při ukládání velkého množství dat.
- Musí být vytvořen s ohledem na správnou politiku chování na Webu.[4]

2.5 Arhitektura Distribuovaného web crawleru

Distribuovaný web crawler bude tvořen nezávislými crawlery, jejichž práce bude synchronizována jedním řídícím serverem. Aplikaci budou tvořit dvě primární části, klient a server. Dále bylo definováno, že distribuce aplikace bude realizována pomocí technologie Webových služeb s použitím SOAP protokolu. Předpokladem realizace je tedy server a klient, kteří budou schopni s okolím komunikovat pomocí SOAP zpráv. Na serveru bude možné volat dvě metody. Jednu za účelem přidělení seznamu URL adres, kterými se bude řídit činnost crawlerů. Druhou pro zpětné získání a zpracování výsledků crawlerů. Ty budou umístěny na vzdálených klientech. Při každém spuštění klienta bude docházet k cyklickému volání obou metod na serveru.



Obrázek 5: Upravený klient/server model na distribuovaný systém

Tento model (obrázek 5) představuje základní kostru projektu zobrazující jeho nejprimitivnější podobu a funkce.

4 Dostupný na WWW: http://www.robotstxt.org/guidelines.html [cit. 2011-05-24].

2.6 Vícevláknové programování při návrhu distribuovaného web crawleru

Součástí zadání diplomové práce bylo seznámit se s teorií vláknového programování[16] a zahrnout ji do Distribuovaného web crawleru. Multithreading neboli paralelní běh dvou či více částí programu bude hrát důležitou roli jak při vytváření SOAP serveru, tak i zpracovávání URL adres crawlery. Každá aplikace obsahuje primárně jedno procesní vlákno (main thread), které obstarává celý chod. Server, myšleno single-threaded server, by nebyl schopen obsloužit více než jednoho klienta zároveň. V praxi by to vypadalo tak, že ve chvíli, kdy by byl zaneprázdněn jedním klientem, ostatní by nebyli obslouženi a museli by čekat na uvolnění serveru. Pro Distribuovaný Web crawler je tato situace nepřijatelná, jelikož by představovala znehodnocení celkové snahy o efektivní zapojování více počítačů do procesu crawlování. Je výhodné, aby vytvořený server využíval více vláken.

Použití vláken v procesu obecně přispívá ke zvýšení propustnosti celé aplikace. Tato vlastnost je důležitá u web crawleru. Během zpracování jediné URL bude vykonávat množství časově náročných operací. Navíc na jeho vstupu nebude pouze jedna URL adresa, ale celý seznam. Využití vláken urychlí běh crawleru paralelním zpracováváním přidělených URL adres.

16 Dostupné na WWW: http://en.wikipedia.org/wiki/Thread_(computer_science) [2011-05-16].

3 Výběr programovacího jazyka

Pro další vývoj distribuovaného web crawleru je třeba do celého návrhu zahrnout reálný programovací jazyk. Přejdeme od čistě teoretické formy popisu k praktickým ukázkám použitého kódu. Volbě jazyka muselo nutně předcházet popsání řešené problematiky s následným stanovením požadavků na vytvářený software (viz 2. kapitola) a určení nároků kladených na vlastnosti samotného programovacího jazyka:

- musí být zdarma;
- dobré vývojové prostředí;
- účinná práce s WWW a podpora návrhu webových aplikací;
- nástroje pro použití SOAP;
- multiplatformnost a široká uživatelská i vývojářská podpora;
- dobrá čitelnost kódu.

Pro realizaci distribuovaného web crawleru byl vybrán Python[17].

```
Import  BaseHTTPServer
def run(server_class=BaseHTTPServer.HTTPServer,
        handler_class=BaseHTTPServer.BaseHTTPRequestHandler):
    server_address = ('', 8000)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()
```

Obrázek 6: Využití standardní knihovny BaseHTTPServer

Obrázek 6 prezentuje vytvoření jednoduchého webového serveru v jazyce Python s použitím jedné z knihoven, které činí programování v tomto jazyce velice efektivním. Veškeré ukázky, knihovny a metody popisované v následujícím textu se budou přímo týkat programovacího jazyka Python, ač to nemusí být v daném místě uvedeno.

17 Dostupné na WWW: http://www.python.org/ [cit. 2011-05-16].

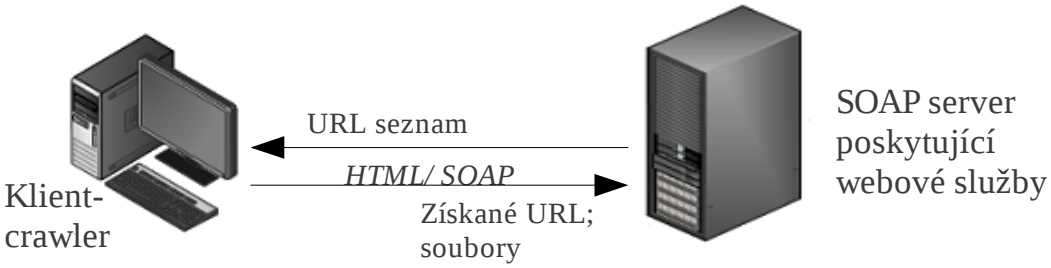
4 Distribuovaný web crawler

Předchozí kapitoly byly zaměřeny na vytvoření návrhu distribuovaného web crawleru, společně s definováním pilířů aplikace a výběrem programovacího jazyka pro její realizaci. Na těchto základech bude v následujících kapitolách vystavena aplikace distribuovaného web crawleru. Společně s tím bude popsáno množství metod a funkcí, které výsledný model musí obsahovat. Základní požadavky kladené na aplikaci:

- funkčnost;
- stabilita.

Neformálním požadavkem na kód navíc bude, aby veškeré komentáře byly psány v anglickém jazyce, za účelem publikování na Webu.

Prvním krokem přechodu k programu je upravení základního schématu (viz obrázek 5), který představoval pouze kostru pro vznikající aplikaci. Model demonstroval fyzické rozdělení klient/server architektury. Odmyslíme-li si všechny klienty kromě jediného, který nadále bude připojen k serveru, a vložíme rozšířenou podobu datových toků zmiňovanou v kapitole 2.4, získáme následující podobu:



Obrázek 7: Zjednodušený model distribuovaného web crawleru

Server odesílá klientovi seznam URL adres, o které žádá. Na oplátku klient vrací serveru vytěžené adresy ze zpracovaných URL a s tím i veškerá získaná data.

Změnou musí projít i nastíněná podoba komunikace webové služby z obrázku 2. Pro Distribuovaný web crawler nebudou WSDL ani UDDI využity. Cílem je klient/server systém, kde bude navržen specializovaný klient s maximální schopností interakce se

serverem. Složitost klienta je taková, že využití služby externími klienty není v současné době předpokládáno a tedy pro vytváření popisu by nebylo využití. Upravením modelu z obrázku 2 pro potřeby distribuovaného Web crawleru získáme následující komunikační schéma pro webové služby

Obrázek 8: Komunikační model distribuovaného Web crawleru

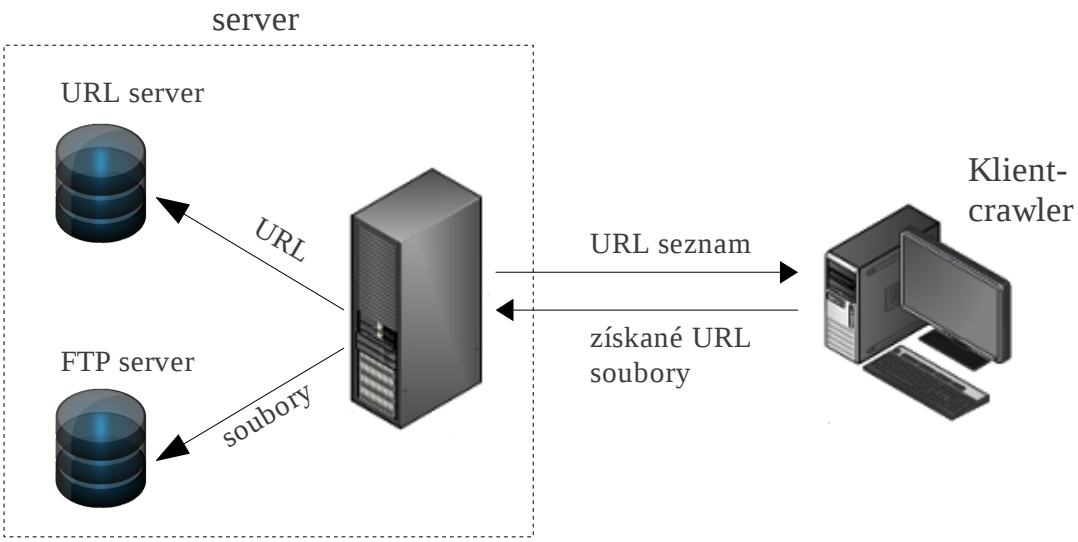
4.1 Rozšíření architektury o protokol pro přenos souborů

Doposud nebyly zmiňovány datové přenosy související s množstvím dat generovaných web crawlery. Z komunikační architektury (viz kapitola 2.3) je zřejmé, že SOAP je využíván hlavně pro systémovou komunikaci. Definice popisuje crawlery jako roboty stahující obsah webových stránek, mezi které patří HTML dokumenty a jiné druhy souborů podporovaných MIME (Multipurpose Internet Mail Extensions)[18]. Řídícím prvkem aplikace je server, který má také zálohovací funkci. Veškerá data, která crawlery získají, poputují na server do připraveného centrálního úložiště. Pomocí HTTP lze přenášet v podstatě všechny typy souborů podporovaných MIME přes TCP/IP. Aplikace přesto využívá pro přenos souborů FTP(File Transfer Protokol)[19]. Důvodem tohoto řešení, kromě vysoké přenosové rychlosti FTP a jednoduché implementaci jazykem Python, je soubor řídících příkazů jazyka FTP, které HTTP neobsahuje. Ty umožní jednoduché rozřazování přenášených dat do složek v pracovním adresáři. Kdyby nebyl prosazován tento návrh, přenos většího množství souborů by byl výhodnější pomocí HTTP.

18 Dostupné na WWW: http://www.w3schools.com/media/media_mimeref.asp [cit. 2011-04-25].
19 Dostupné na WWW: http://www.faqs.org/rfcs/rfc959.html [cit. 2011-04-25].

4.2 Aplikační model Distribuovaného web crawleru

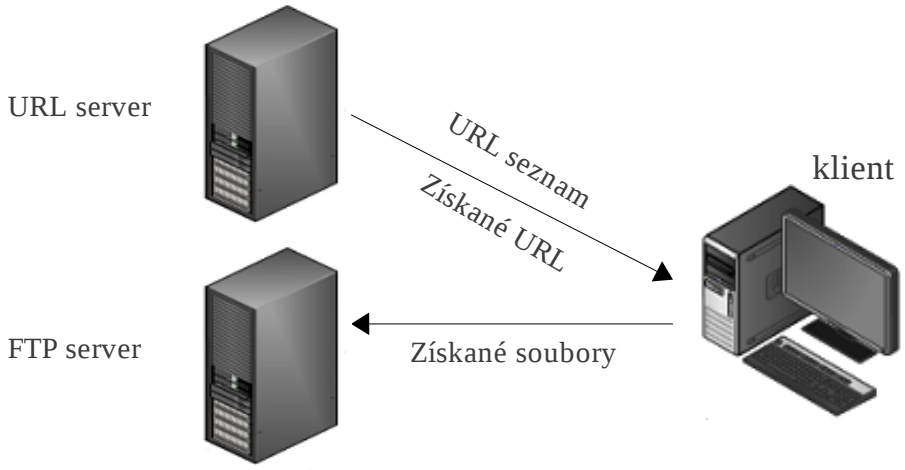
Přenos dat (HTTP i FTP) je založen na klient/server architektuře. Jeden server software nelze naneštěstí zároveň využívat pro poskytování webových služeb a pro přenos souborů pomocí FTP. Z toho vyplývá nutnost oddělení těchto dvou částí. Distribuovaná aplikace bude ve skutečnosti tvořena dvěma serverovými aplikacemi – URL serverem (řídí celou aplikaci a poskytuje webové služby), FTP serverem (obstarává přenos souborů od klienta do datového úložiště) a programem klient crawleru.



Obrázek 9: Upravené schéma distribuovaného web crawleru

Uploadování souborů na server zajišťuje FTP klient. V tomto případě není nutné vytvářet samostatnou aplikaci. FTP klient lze používat jako jednoduchou funkci volanou vně jiného kódu. Nosnou aplikací je tedy clientCrawler.

Obrázek 9 zobrazuje situaci, kdy software pro FTP i URL server je umístěn na jedné pracovní stanici. Toto řešení je plně funkční a použitelné. Nelze ho ale brát jako jediné možné. Jelikož tyto aplikace jsou na sobě naprosto nezávislé, lze docílit naprosté decentralizace jejich umístěním na oddělené počítače. Sníží se tak síťová zátěž serveru pomocí rozložení komunikace mezi dva body (viz obrázek 10).



Obrázek 10: Oddělení FTP a URL serveru

4.3 FTP server

FTP server tvoří jednodušší část práce, která vznikla čistě za účelem přenosu souborů od klienta do míněného úložiště na serveru. Bylo by samozřejmě možné využít již nějaký existující FTP server, ale vytvořením vlastního odpadá nutnost instalace a dodatečné konfigurace serveru. Uživateli stačí spustit přiložený ftpServer z konzole.

Jazyk Python je pro práci s FTP velice dobře vybavený. Nabízí hned několik knihoven umožňujících vývoj serveru i klienta. Naprogramování serveru zde bude řešeno za pomoci knihovny *pyftpdlib*[20]. Z ní stačí importovat *ftpServer* a třída pro vytvoření serveru je k dispozici. Na stránkách projektu je velice pěkně vytvořený tutorial[21] s komentovanými příklady.

K naprogramování FTP serveru je potřeba splnit tři věci:

- vytvořit instanci třídy *ftpServer;*
- vytvořit ftp uživatele s definovanými hodnotami pro připojení;
- určit adresu a port, kde bude server naslouchat.

20 Dostupné na WWW: http://code.google.com/p/pyftpdlib/ [cit. 2011-04-29].
21 Dostupné na WWW: http://code.google.com/p/pyftpdlib/wiki/Tutorial [cit. 2011-04-29].

```
authorizer.add_user('crawler', 'E%)c7h*o!', directory, perm='elamw')
```

Ukázka 1: Vytvoření uživatele FTP

Ukázka 1 zobrazuje kód FTP serveru definující uživatelské jméno (*crawler*) a heslo (E*%)c7h*o!*). Důvodem pro ověřování identity a přístupového hesla od klienta, je snaha o využití alespoň jednoho z minimálních bezpečnostních opatření FTP. Zajímavější částí je určení pracovního adresáře (*directory = './Pages'*, adresář umístěný na serveru) a přístupových práv (*perm='elamw'*).

Přístupová práva definovaná pro pracovní adresář:

- e (představuje možnost změnit aktuální pracovní adresář
- l (přístup k seznamu souborů)
- a (připisování dat existujícím souborům)
- m (vytvoření podadresáře)
- w (ukládat soubory na serveru)

```
address = (host, port)
ftpd = ftpserver.FTPServer(address, ftp_handler)
```

Ukázka 2: Inicializace FTP serveru s nastavením IP portu

Vytvoření instance serveru je jednoduché. Stačí mu předat IP adresu a port, na kterém má naslouchat společně s definovanou obslužnou rutinou. FTP primárně používá dva TCP porty - 20 (pro datové přenosy) a 21 (řídící příkazy v rámci FTP). Během testování několikrát nastal problém se spuštěním z důvodu obsazení portů jiným softwarem využívajícím FTP. Z toho důvodu byly defaultní porty přesměrovány na některé ze škály veřejných portů například 50003. Ač FTP používá pro komunikaci dva porty, definuje se pouze ten řídící. Přenos dat pak probíhá na tom číselně pod ním.

Při vytváření serveru by neměla být opomíjena možnost *pyftpdlib* omezit maximální počet připojení (ukázka 3).

```
ftpd.max_cons = 25
ftpd.max_cons_per_ip = 4
```

Ukázka 3: Omezení připojení k FTP serveru

Lze tak alespoň částečně zamezit nebezpečí útoků snažících se zaplavit server přílišným množstvím dat.

4.3.1 Datové úložiště

FTP serverem byl definován pracovní adresář. Datové úložiště je vytvořeno z důvodu potřeby existence jednotného úložiště získaných dat v rámci tohoto projektu. Použitým adresářem je *./Pages*, který je umístěn na serveru. Právě data v něm uložená tvoří výstup této aplikace. Vytvoření tohoto datového úložiště je jedním z důvodů existence crawlerů.

Předmětem práce je vytvoření Distribuovaného web crawleru, nikoli serach-enginu, který by si nárokoval konkrétní požadavky na podobu uložených dat. Přesto otázka vytvoření této struktury je velice důležitá. Ve chvíli, kdy jsou stránky připravené v předepsané formě, nebrání nic jejich dalšímu zpracování. Jak nastínili Sergey Brin a Lawrence Page ve své práci[22] optimálním řešením je ukládání stránek v čisté HTML podobě do jediného adresáře a teprve pomocí následného indexování probíhá rozřazování záznamů do barelů (specializovaný soubor, v podstatě databáze).

Distribuovaný web crawler využívá trochu jiné řešení. Jeho hlavním úkolem je prezentace možností této aplikace. Datové úložiště obsahuje obrazy všech URL adres, které kdy byly navštíveny crawlery. Obrazy jsou rozřazeny do složek podle URL domén.

22 Dostupné na WWW: http://infolab.stanford.edu/~backrub/google.html [cit. 2011-05-16].

```
~/Plocha/Distribuovany_webcrawler[2.3]/server/Pages$ tree
.
|-- ceskapojistovna.cz
|   |-- ceskapojistovna.cz.1303593991.1.tar.bz2
|   |-- ceskapojistovna.cz.1303595000.51.tar.bz2
|-- tul.cz
|   |-- tul.cz.1303593944.69.tar.bz2
|   |-- tul.cz.1303593967.28.tar.bz2
|   |-- tul.cz.1303593991.12.tar.bz2
```

Obrázek 11: Datové úložiště ./Pages

Podobu dat neovlivňuje server, nýbrž klient (viz. kapitola 4.5.3), který je uploaduje do příslušných složek v úložišti. V návrhu bylo zmíněno, že od aplikace se předpokládá schopnost zpracovávání desítek až stovek milionů webových stránek v rozmezí několika týdnů. Tato skutečnost znamená stahování obrovského množství dat, které bude potřeba nějak uskladnit. Ve složkách budou stránky součástí souborů komprimovaných pomocí tar.bz2. Jednotlivé stránky v tarfilech by měly být uloženy dle konfigurace clientCrawleru buď jako čité HTML dokumenty, nebo XML dokumenty obsahující výtahy informací ze stránek.

4.4 URL Server

Podle definice serverem se označuje program nebo počítač, na kterém je program spuštěn, který poskytuje určitou službu klientskému softwaru na stejném počítači, nebo počítači propojeném v síti.[23] Tato kapitola popíše naprogramování webových služeb a serveru, na němž budou umístěny. Nadpis kapitoly URL server představuje název, který v následujícím textu bude používán pro vytvářený server.

Hlavní objekty URL serveru:

- SOAP server;
- URL fronta *MainQueue* s URL adresami čekajícími na zpracování;
- kontrolní množiny CS a zpracS;

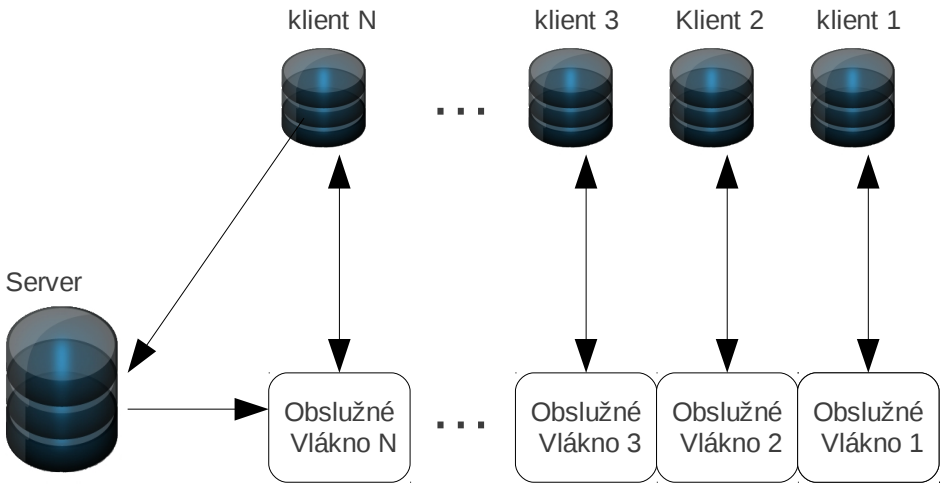
23 Dostupné na WWW: http://www.linfo.org/server.html [cit. 2011-04-30].

- funkce getJob;
- funkce sendAllData;
- funkce cExit.

4.4.1 Single vs. Multi-Threaded server

Než se pustíme do popisu vytvoření URL serveru, je třeba rozvážit jednu věc. Mezi serverem a klienty bude docházet k časté komunikaci. Společně se snahou o využití jejich maximálního potenciálu to znamená, že server musí být schopen komunikovat s více z nich zároveň. Řešením je vytvořit multi-threaded (vícevláknový) řídící server.

Obrázek 12 představuje klasickou realizaci moderních serverů. Nejlepším příkladem této architektury je Apache[24] server.



Obrázek 12: Základní architektura multi-threadového serveru

Každé z vytvořených obslužných vláken se stará jak o komunikaci s klientem, tak o vyřízení jeho požadavků. Na počátku spuštění serveru dojde k vytvoření jednoho hlavního vlákna (main Thread). Jeho úkolem je naslouchat na přiděleném portu na zavolání klienta. V tom okamžiku se vytvoří speciální obslužné vlákno, které převezme obsluhu klienta. Vytváření vláken lze realizovat pomocí dvou rozdílných metod. Při jednodušší se vytvoří nové vlákno pro každého klienta (využívá vytvořený

24 Dostupné na WWW: http://httpd.apache.org/ [cit. 2011-05-16].

distribuovaný web crawler). Toto řešení v sobě skýtá podstatný problém, který se projeví v případě, kdy se k serveru pokusí zároveň přihlásit větší množství klientů. V nejhorším možném scénáři může dojít k úplnému zahlcení serveru. Výhodou je ale celková jednoduchost implementace. Druhá metoda využívá naopak „recyklace“ vláken. Té je dosaženo využitím tzv. thread poolu. Výhoda spočívá v možnosti nastavení maximálního počtu vláken, které mohou být využity.

4.4.2 *Multi-threaded SOAP server*

Příklad vytvoření jednoduchého serveru byl prezentován ve 3. kapitole. Naprogramování serveru komunikujícího na bázi SOAP a schopného obsluhovat více klientů zároveň je obtížnějším úkolem. Python nabízí „silný“ nástroj pro práci se SOAP protokolem v podobě několika nestandardních knihoven. Server i klient jsou vytvořeni na podkladu knihovny SOAPpy[25]. Jejím největším přínosem je získání abstrakce nad celou komunikací a přístup ke vzdáleným službám pomocí standardního API jazyka Python. Programátor se při vytváření serveru nemusí přímo zabývat otázkou přenosu zpráv mezi klientem a serverem (tedy HTTP požadavků a obsažených XML dokumentů). Pod rouškou SOAPpy dochází ke zpracovávání XML zpráv, předávání parametrů určené funkci a odeslání návratových hodnot přes HTTP.

```
from SOAPpy import *

#create an instance of server
server = ThreadingSOAPServer((host,port))

#server aplications
server.registerFunction (getJob)
server.registerFunction (sendAllData)
server.registerFunction (cExit)

# Start the server
try:
    while run:
        server.handle_request()
```

Ukázka 4: Vytvoření SOAP serveru s definováním funkcí pro vzdálený přístup

Knihovna SOAPpy obsahuje dvě důležité třídy, *SOAPServer* a *ThreadingSOAPServer.*

25 Dostupné na WWW: http://soapy.sourceforge.net/ [cit. 2011-04-30].

Distribuovaný web crawler implementuje třídu *ThreadingSOAPServer,* která umožňuje serveru využívat řídící vlákna k obsluze volání. Instance třídy je volána s dvěma parametry: IP adresa (*host*) a TCP port (*port*). Hodnoty určují, kde má server naslouchat. Webovou službu kromě adresy serveru, kde je umístěna, definuje specifické jméno, pod kterým bude v serveru zaregistrována (*server.registerFunction(getJob)*).

4.4.3 Fronta URL

Fronta je jednou z hlavních částí práce. Představuje mechanismus pro přidělování URL klientům (crawlerům). Jejím úkolem je spravovat URL adresy získané crawlováním, které ještě nebyly zpracovány. Mechanismus přidělování a procházení Webu bude záviset na způsobu řazení adres ve frontě a jejich vybírání.

Crawlery používají pro svoje směrování grafovou strukturu webu. Stejně jako u grafů, i zde je zásadní teorie procházení grafů. Základní implementací fronty v Pythonu je modul *Queue*, který se využívá zejména u vláknově orientovaných aplikací. Modul umožňuje realizovat tři typy front: FIFO, LIFO a prioritně orientovanou. Využití FIFO, v porovnání s grafy, bude znamenat realizaci procházení webu do šířky (označováno jako BFS). Ve WWW se využívá s cílem získání co největšího množství stránek. To je také požadavkem této práce. Implementace BFS s sebou přináší řadu problémů pro distribuované a vícevláknové crawlery. Nalezené hypertextové odkazy často odkazují na jedinou doménu. Ve spojení s crawlováním to znamenat směrování velkého počtu HTTP požadavků na jeden server v relativně krátkém čase. Vzhledem k serverům to může být vyhodnoceno i jako útok. Distribuovaný web crawler musí používat sofistikovanou strategii přidělování URL adres crawlerům, aby byla rozložena zátěž na více serverů. Python nabízí metodu *deque* (Double ended queue). Stejně jako *Queue*, je považována za thread-save (díky mutexům) implementaci fronty. Hlavní výhodou oproti běžným frontám je netypická podpora přidávání a odebírání prvků z obou konců fronty.

```
from collections import deque
MainQueue = deque()

#pop URL from left side of queue
data = MainQueue.popleft()
#pop URL from right side of queue
data = MainQueue.pop()
```

Ukázka 5: Využití modulu deque

Jedno z řešení problémů FIFO front je založeno na náhodném přidělování URL adres crawlerům. Podmínkou fungování této metody je obrat velkého množství URL adres ve frontě. Crawlery před odesláním výsledků *urlServeru* náhodně promíchají získaný seznam URL adres. Na prvku 100 zpracovávaných URL lze získat více než deset tisíc odkazů, které jsou vloženy do fronty (v náhodném pořadí daném promícháním seznamů crawlery). Ve spojení s výběrem z obou stran fronty je zajištěno náhodné přidělování adres crawlerům. Významným doprovodným efektem fronty *deque* je obnovitelnost zdrojů, které v rámci nedokončeného cyklu klient navrací do fronty.

Mezi jiné možnosti řazení front by spadaly algoritmy nejrelevantnějšího výběru z fronty – PageRank, Naive Best - First, Fisch - search a další.

4.4.4 Kontrolní množiny

S ohledem na požadavky web crawleru URL nesmí být zpracováno více než jednou. Synchronizace souběžné činnosti více crawlerů je zabezpečena přidělováním URL adres serverem. Aby přidělování bylo efektivní, musí mít server strategii pro kontrolu předchozího zpracování URL odkazů. Distribuovaný web crawler využívá dvou kontrolních množin (sets v Pythonu):

- *CS* – množina obsahující veškeré již zpracované URL
- *zpracS* – množina obsahující pouze aktuálně zpracovávané URL

UrlServer přebírá jeden z formálních požadavků web crawleru (viz kapitola 2.4).

Server na jedné straně URL adresy odesílá, na druhé přijímá. Před odesláním URL klientovi jsou adresy vybírány z fronty a vkládány do seznamu pro odeslání. Seznamy musejí obsahovat pouze unikátní ještě nenavštívené adresy. Aby toho bylo dosaženo,

každá URL při výběru z fronty je porovnávána s oběma kontrolními množinami (*CS*, *zpracS*), jestli v nich není obsažena. Teprve potom dochází ke vložení adresy do seznamu a také do množiny *zpracS*. Ta zajišťuje, že URL nebude během zpracovávání jedním crawlerem přidělena nějakému jinému. Hlavní fronta pro přidělování adres může obsahovat redundantní záznamy. K aktualizaci kontrolních množin dochází v okamžiku přijetí výsledků crawlování URL serverem. Klienti společně se seznamem nalezených adres předávají i seznam původních URL, které měli za úkol zpracovat. Důvodem je potvrzení, že crawler příslušné URL opravdu zpracoval (byla z nich získána data) a lze je tedy vložit do kontrolní množiny CS a odebrat z množiny *zpracS*.

4.4.5 Funkce getJob

Funkce *getJob* je první webovou službou, o kterou klienti žádají. Jejím účelem je předat klientovy seznam URL adres, které bude mít za úkol zpracovat a předdefinované řídící příkazy. Pomocí příkazů server definuje způsob, jakým bude klient s adresami pracovat a kam odešle výsledky své práce.

Důležitou částí služby je kontrola přidělovaných URL adres, ke které bude docházet v rámci této funkce. Adresy jsou kontrolovány, nebyly-li již v minulosti zpracovány, popřípadě jestli zrovna nedochází k jejich zpracovávání jiným crawlerem.

```
if (not data in CS) and (not data in zpracS):
    consumer.append(data)
    #locking check set for adding another value
    lock.acquire()
    try:
        #adding value into set of send
        zpracS.add(data)
```

Ukázka 6: Kontrola zpracovaných URL adres

4.4.6 Funkce sendAllData

Služba SendAllData vyvolá přenos dat od klienta. Crawler během svého cyklu zpracoval veškeré adresy a před požadavkem o nové musí předat výsledky. Odesílá tedy serveru svoji IP adresu, seznam zpracovaných, nezpracovaných a získaných URL odkazů a informaci o celkovém množství získaných dat crawlerem. Úkolem

sendAllData je také zpracování těchto výsledků. V rámci funkce dochází ke vkládání URL do fronty. Na této úrovni není prováděna žádná kontrola vkládání. Ve frontě tedy budou jak totožné URL adresy, tak odkazy na již zpracované, či zpracovávané URL. Adresy, které se nepodařilo crawleru zpracovat, jsou ukládány do textového souboru, který představuje černou listinu URL. Přesto, že tyto adresy nebyly crawlovány, i tak musejí být obsaženy v seznamu zpracovaných adres.

```
#adding url to set for purpose of excluding processed URLs from zpracS
pomS =  set()
for url in zprac:
        pomS.add(url)
#it is also necessary to synchronize access to the check sets
lock.acquire()
try:
    #Remove the processed address from the set currently being processed (zpracS)
    zpracS = zpracS - pomS
    #sets's supervisory URLs
    #also shows how many URLs were sent to processing
    CS = CS|pomS
```

Ukázka 7: Zpracování URL adres serverem

Část kódu funkce *sendAllData* (ukázka 7) názorně zobrazuje vynětí zpracovaných URL z množiny *zpracS* a jejich přidání do množiny *CS*. Tento postup zajistí, že nedojde k dalším pokusům o jejich crawlování. Obě tyto operace jsou realizovány pomocí množinových operací rozdílu a sloučení. Velký důraz při kódování URL serveru byl kladen na synchronizaci kritických sekcí.

Posledním a neméně důležitým úkolem služby je prezentace výsledků.

4.4.7 Funkce cExit

Úkolem této služby je zajištění konzistence dat v případě předčasného ukončení clientCrawleru. Při volání klient předá URL serveru původní seznam zaslaných adres. Server je vyřadí z množiny *zpracS* (aktuálně zpracovávané adresy) a vloží je zpět do fronty *MainQueue*. Na tomto postupu závisí možnost opětovného crawlování těchto adres. Další z formálních požadavků na web crawler je zahrnut do serveru.

```
#in case of re-delivery address remove URLs from the set of processed
ret=set(urls)
zpracS = zpracS - ret
#return address back into the queue
 for url in urls:
    #exclude url from the set of processed
    MainQueue.put(url)
```

Ukázka 8: Navrácení adres do fronty

4.4.8 Další funkce serveru

Důležitou vlastností serveru je udržet konzistenci dat. V případě pádu, nebo ukončení uživatelem jsou uchovány doposud získané informace společně se seznamy zpracovaných adres a adres čekajících na zpracování. Pro tyto případy program využívá serializační modul *pickle* umožňující ukládání dat do souboru a jejich následné vyvolávání.

4.5 clientCrawler

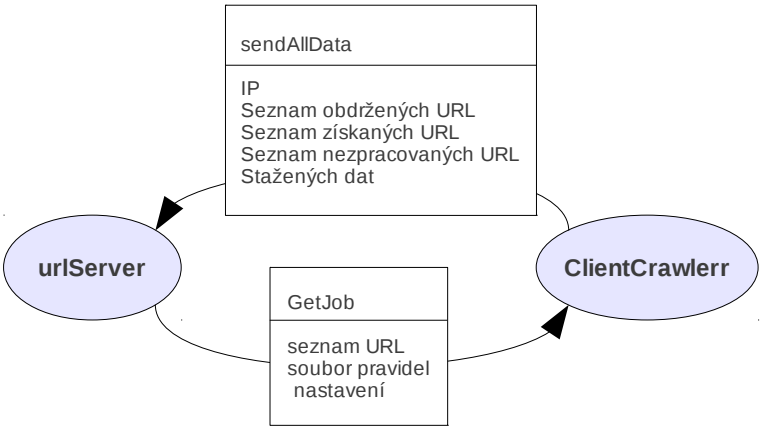
Popišme si nyní aplikaci zaměřenou na klientskou část architektury. Ta je v projektu zastoupena *clientCrawlerem*, neboli klientským softwarem zabezpečujícím komunikaci mezi serverem a crawlerem.

4.5.1 Klient

V Distribuovaném web crawleru se pod pojmem klient schovává web crawler, který je schopný komunikovat se serverem.

Požadavky:

- komunikace se serverem pomocí zasílání SOAP zpráv;
- bezeztrátovost v případě ukončení;
- klient bude spouštěn jako systémová služba (daemon).



Obrázek 13: Volání služeb clientCrawlerem se zobrazenými datovými toky

Klient v architektuře distribuovaných aplikací volá webové služby umístěné na serveru za účelem získávání dat pro svoji činnost. V distribuovaném web crawleru *clientCrawler* žádá server o přidělení URL adres a specifikací nastavení.

```
#connect to server and request getJob service
#need to send self IP identification
pocSeznam,config = server.getJob(getIdent())
ftpIP, ftpPort, agent_name, robots_control, sitemap, doc_save, page_parse = config
```

Ukázka 9: Volání webové služby getJob()

Aplikace je vytvořena s ohledem na možnou nedostupnost *urlServeru* i *ftpServeru*. Žádosti o přidělení služby jsou odesílány v pravidelném intervalu až do doby obdržení potvrzení.

Po dokončení crawlování klient žádá o službu předávající výsledky serveru. Klient serveru odešle:

- svoji identifikační IP;
- seznam obdržených URL;
- seznam získaných URL;
- seznam URL, které nebylo možné zpracovat
- celkovou velikost stažených dat z WWW.

```
#sending results of crowling
server.sendAllData(getIdent(),naZprac, getUrls, blacklist, size)
```

Ukázka 10: Volání služby pro předání dat sendAllData()

V případě nenadálého ukončení činnosti *clientCrawleru* je volána služba *cExit* a *URLserveru* jsou vráceny všechny zadané adresy.

4.5.2 clientCrawler jako systémová služba

Speciální část kódu tvoří metoda *daemon*. Umožňuje spuštění aplikace jako systémové služby (démona) v Unixových systémech. Démon je výraz označující typ procesu, který běží v pozadí operačního systému namísto pod přímou kontrolou. Jelikož *klientCrawler* tvoří autonomní část aplikace umístěné na klientském počítači, je vhodné ho spouštět tímto způsobem. Vytvoření démona v Linuxu podléhá specifickým pravidlům.[26]

4.5.3 Web crawler

Web crawler je vytvořen jako samostatný modul volaný pomocí funkce *crawl()*. Pro spuštění vyžaduje seznam URL adres a soubor konfiguračních pravidel (předáno jako parametry funkce) nastavujících politiku chování a funkčnost crawleru. Data na vstupu web crawleru určují:

- jaké adresy má crawler prohledávat;
- jestli se držet výhradně adres jedné domény, nebo procházet celý Web;
- adresu FTP serveru;
- pod jakým jménem se bude identifikovat serverům;
- má-li kontrolovat povolení přístupu k URL;
- mají-li ukládat nalezené dokumenty, či nikoliv;
- bude-li docházet k rozšířenému zpracovávání webových stránek, nebo budou

26 Dostupné na WWW: http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html [cit. 2011-04-30].

ukládány v původní HTML;

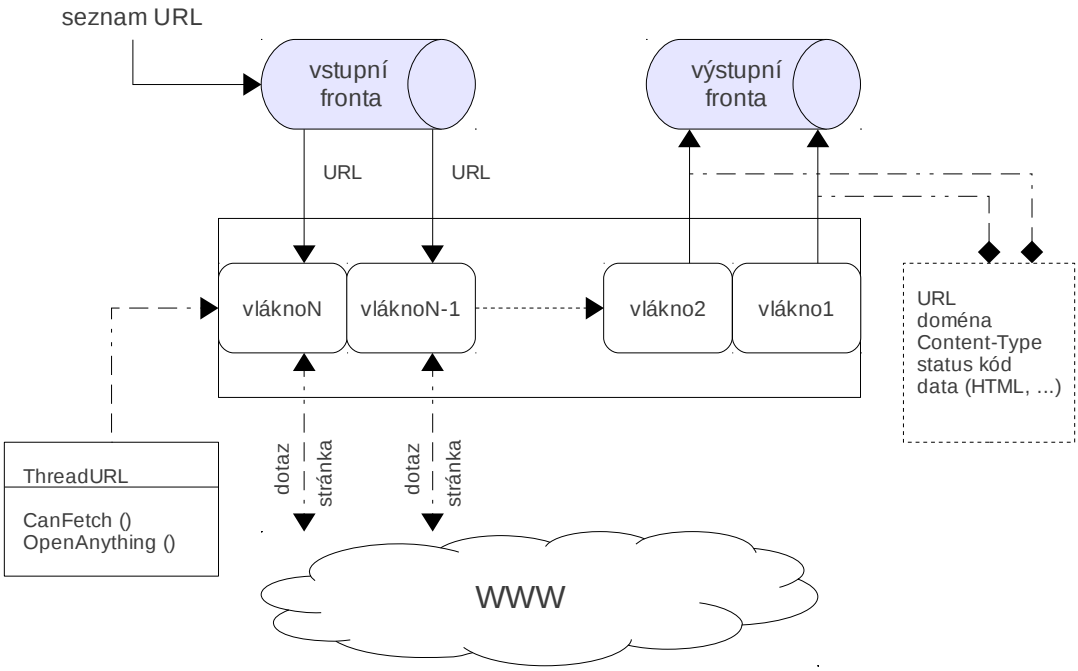
- kolik vláken může vytvořit.

Během vývoje a rostoucích požadavků na *clientCrawler* se sekvenční zpracování úloh stávalo příliš zdlouhavým a bylo nutné přejít k jejich paralelnímu zpracovávání (viz kapitola 2.6). Web crawler nejprve vytvoří dvě fronty, přičemž do jedné vloží získané URL od serveru. Tu nazveme vstupní frontou, druhá bude výstupní. Fronty zde představují důležitou část. Práce s vlákny za normálních okolností vyžaduje ošetření kritické sekce. Fronta je ale odvozena od modulu *Queue.Queue*, který toto defaultně řeší algoritmem vzájemného vyloučení (mutex) pro přístup.

Crawler vytvoří N obslužných vláken, která postupně vybírají adresy ze vstupní fronty (funkce *threadUrl*), dokud ji nevyprázdní. N (definováno v nastavení) značí maximální počet vláken, které mohou být spuštěny v crawleru v jednom okamžiku.

```
for i in range(self.num_threads):
        t = ThreadUrl(self.in_queue, self.out_queue, self.agent, self.blacklist, ...)
        t.setDaemon(True)
        t.start()
```

Ukázka 11: Vytvoření obslužného vlákna funkce threadUrl



Obrázek 14: funkce web crawleru threadUrl

Je důležité, aby veškeré adresy vně aplikace vyhovovaly určitému standardu (viz kapitola 4.5.3.1). URL adresy jsou porovnány s příslušným souborem robots.txt (viz kapitola 4.5.3.3). S ohledem na politiku chování crawleru na síti je potřeba kontrolovat, jestli administrátoři povolují přístup k danému zdroji na webovém serveru. Je-li to povoleno, crawler sestaví HTTP požadavek na danou URL a odešle ho webovému serveru.

```
request = urllib2.Request(source)
request.add_header('User-Agent', agent)
request.add_header('From', __mailto__ )
opener = urllib2.build_opener(SmartRedirectHandler(), DefaultErrorHandler())
return opener.open(request)
```

Ukázka 12: Sestavení HTTP požadavku

Jednou ze základních vlastností web crawlerů je sebe-identifikace. Crawlery se identifikují web serverům pomocí záznamu *User-agent* v těle HTTP požadavku. Další položkou, která by měla být v HTTP obsažena, je URL adresa poukazující na zdroj dalších informací o aplikaci, popřípadě e-mailová adresa vlastníka. Důvodem těchto informací je, aby administrátoři webových serverů měli možnost dohledat strůjce případných potíží a kontaktovat majitele crawleru.

V reakci na požadavek se může nastat několik scénářů. Nejčastěji požadovaný zdroj existuje a je vrácen serverem v podobě objektu. Jiným případem je, že zdroj existuje, ale byl přesměrován.

```
def http_error_301(self, req, fp, code, msg, headers):
    result = urllib2.HTTPRedirectHandler.http_error_301(
      self, req, fp, code, msg, headers)
    result.status = code
    return result
```

Ukázka 13: Ošetření přesměrování URL modulem openanything

V ukázce 13 je ošetřen případ přesměrování URL pomocí získávaného kódu (301) z odpovědi serveru. Je také nutné definovat maximální dobu, po kterou se má klient snažit získat nějaký zdroj ze serveru. Kdyby se tak nestalo, mohlo by dojít k zacyklení klienta při pokusu o připojení k serveru.

```
socket.setdefaulttimeout(10)
```

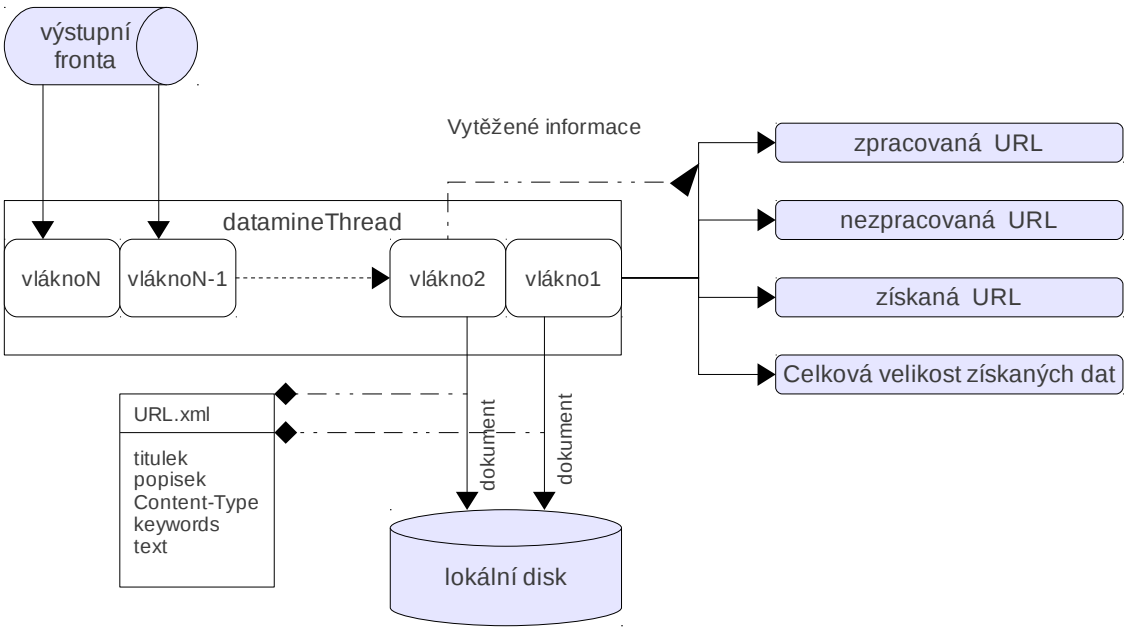
Ukázka 14: Definování doby před odpojením od web serveru

Získaná webová stránka je vložena ve formě slovníku (asociativní pole) do výstupní fronty. Slovník obsahuje:

- URL adresu (v případě přesměrování se bude lišit od adresy na vstupu);
- doménu URL adresy;
- datový typ (Content-Type);
- obraz stránky;
- status kód.

Důvodem existence výstupní fronty je oddělené zpracovávání získaných dat další sadou N vláken. Proces crawlování je tedy rozdělen na dvě části do dvou sad vláken. Jedna se stará o stahování souborů z WWW a druhá o jejich další zpracovávání.

Z výstupní fronty jsou obslužnými thready funkce *DatamineThread* vybírány výsledky předchozí činnosti (slovníky).



Obrázek 15: funkce web crawleru datamineThread

Důraz je kladen na celistvost slovníku, zejména na přítomnost obrazu stránky.

Nepřítomnost obrazu by značila, že URL odkazuje na neexistující stránku, nebo stránku, kterou nelze otevřít. Takováto URL se rovnou přesouvá do *blacklistu* a vlákno je ukončeno. Na této úrovni je kontrolován *Content-Type* získané stránky. Lze tak zajistit, že budou parsovány pouze HTML dokumenty (*text/html, text/plain)*, nikoliv soubory. Aniž by došlo ke zhoršení pokrytí webu, je možné částečně limitovat množství procházených dat pomocí kontrolování velikosti parsovaných stránek. Projde-li stránka testem na obsah a velikost, parser z HTML získá hypertextové odkazy, titulek, popis stránky, klíčová slova a text.

Lze použít omezení, kdy budou získávány pouze odkazy a nic jiného. Další možností je využití pouze odkazů vedoucích na doménu aktuálně zpracovávané URL, pomocí dělení odkazů na interní a externí. Dle nastavení jsou pak odkazy vkládány do sdílené množiny nalezených URL. Data z ní, po dokončení crawlování vstupních URL, jsou odesílána URL serveru. Úkolem vlákna kromě získání informací je také jejich uložení do souboru. Soubory jsou při ukládání rozřazovány do adresářů dle domén, kterým náleží. Striktně jsou odlišovány domény a subdomény. Výsledkem je stromová struktura uložených souborů. Podoba uloženého souboru závisí na jeho typu. Jinak bude uložen HTML dokument a jinak pdf, jpg, atd. Netextové soubory jsou ukládány v nativní formě. Podoba uložení HTML dokumentu je závislá na nastavení. Má-li crawler povoleno parsovat webové stránky, vytvoří se XML soubor, v opačném bude uložen čistý HTML text. Uložení je posledním úkolem *DatamineThreadu* a jeho dokonáním vlákno zaniká.

```
header = '<?xml version="1.0"?>'
xmlData = '%s<root URL="%s"><title>%s</title><description>...
dom = BeautifulStoneSoup(xmlData)
f.write(dom.prettify())
```

Ukázka 15: Uložení informací do XML souboru

BeautifulStoneSoup[27] je univerzální třída pro parsování XML dokumentů knihovny BeautfulSoup.

Poté, co jsou obě fronty vyprázdněny, web crawler předá výsledky *urlServeru* (viz obrázek 13). Každý z vytvořených adresářů (každý obsahuje stránky jedné domény) je

27 Dostupné na WWW: http://www.crummy.com/software/BeautifulSoup/documentation.html [cit. 2011-05-26].

zkomprimován do souboru pomocí tar.bz2 formátu. Zkomprimované soubory jsou následně uploadovány na FTP server do příslušných podadresářů v datovém úložišti. Původní adresáře jsou smazány, aby nedocházelo k hromadění dat na klientských počítačích.

4.5.3.1 urlnorm

Z obrázku 16 je patrné, že všechny URL adresy ukazují na tentýž zdroj:

http://website.com
http://www.website.com/index.html
http://www.website.com/default.aspx
http://website.com/default.aspx

Obrázek 16: Nenormalizované URL adresy[*]

Problémem je, že v očích robotů jsou tyto adresy rozdílné. Crawlery tedy zpracují všechny. To by vedlo k tomu, že se v systému objeví čtyři úplně identické soubory, zvýší se režie crawleru, zvýší se počet dotazů na server i zatížení systému. Aby crawler tyto URL adresy vyhodnotil správně, je nutné je upravit dle standardů pro normalizaci URL[28]. Normalizaci zajišťuje modul urlnorm 1.1.2[29]. Podporované normalizační kroky:

- převedení znaků URL na malá písmena;
- konverze adresy do IDN formátu;
- odebrání defaultního portu;
- odebrání operátorů definujících adresáře (./, ../, //, atd.);
- konvertování mezer na %20.

Pro potřeby práce je modul rozšířen o odstraňování defaultních indexů (*index.html, defaults.asp*) a celé části fragmentu z URL. Také není nezbytné prohledávat dotazy. Většinou obsahují URL adresy, které lze získat i z běžných zdrojů, proto jsou vyřazeny z dalšího prohledávání.

* Obrázek dostupný na WWW: http://cutewriting.blogspot.com/2008/12/url-canonicalization-duplicate-content.html [cit. 2011-05-25].
28 Dostupné na WWW: http://en.wikipedia.org/wiki/URL_normalization [cit. 2011-05-16].
29 Dostupné na WWW: http://pypi.python.org/pypi/urlnorm [cit. 2011-05-05].

Kanonizace URL adres je prováděna v celém distribuovaném web crawleru a to hned několikrát, aby bylo dosaženo jejího maximálního rozložení na všechny URL vně programu. Jelikož nelze zajistit, aby prvotní URL zadaná uživatelem vyhovovala standardům použitým v aplikaci, je nutné tyto počáteční adresy normalizovat ještě na straně serveru, než budou odeslány crawlerům. Následnou kanonizací získaných adres crawlery zajistíme jednotnou formu všech URL v celé aplikaci. Tedy až na drobnou výjimku, kterou mohou tvořit přesměrované adresy. Ty jsou normalizovány zvlášť.

4.5.3.2 Kódování dat

Kódování dat představuje jeden z největších problémů při zpracovávání dat z WWW. Pod pojmem kódování rozumíme způsob reprezentace znaků pomocí sekvence přirozených čísel nebo bajtů. Společně s odkazy jsou z webových stránek získávány titulek stránky, popis, klíčová slova a text. Pro zpracování těchto informací lze najít knihovny (např. *BeautfulSoup*), které se o kódování dat dokáží postarat samy. Hlavním problémem, co se týče kódování, bylo zpracování URL adres. Na tom má hlavní podíl používání IDN adres a automaticky generovaných adres PHP.

```
http://www.háčkyčárky.cz/
http://knihovna.tul.cz/Evropska-digitalni-knihovna-„Europeana“-je-nyni-pristupna-on-line-51719.php
```

Ukázka 16: IDN a automaticky generované adresy

Jak je vidět z ukázky 16, v obou adresách se nachází znaky, které překladač nedokáže správně přečíst. Původní internetová struktura nebyla totiž na IDN připravena. Využívaným řešením je překlad z Unicodu do speciálního ASCII kódu, který je následně možné šířit skrze internet. Veškeré adresy v Distribuovaném web crawleru jsou překódovány na jednoduchý ASCII Punycode.

```
url  = url.decode('idna')
```

Ukázka 17: dekódování URL

Ten zajistí, že adresa je reprezentována pomocí základních znaků a navíc použitá knihovna pro přístup ke zdrojům na WWW (*urllib2*), tento typ kódování URL přímo

podporuje bez nutnosti překódovávání. Veškeré adresy v aplikaci jsou kódovány do této podoby.

4.5.3.3 myRobotParser

MyRobotParser je modul vytvořený za účelem vyhovění standardům pro přístup ke zdrojům na WWW (The Robots Exclusion Protocolu)[30]. Ten je využíván správci webových serverů pro omezení přístupu robotů k datům na serveru. Vše funguje následujícím způsobem: Chce-li crawler navštívit webovou stránku (např. http://www.fm.tul.cz/cs/fakulta-mechatroniky), měl by před tím zkontrolovat soubor robots.txt definující pravidla přístupu. Soubor bývá umístěn v hlavním adresáři serveru (v tomto případě http://www.fm.tul.cz/robots.txt). Získávání informací o přístupu je totožné s vyhledáváním informací v textových řetězcích. Díky standardizaci obsahu těchto dokumentů lze jednoduše využít regulérních výrazů pro vyhledání příslušných informací.

Důležitou částí modulu je vytvořená vlastní omezená paměť (něco na způsob „cache“). Paměť je koncipována jako omezený asociativní slovník *[doména] = seznam zakázaných cest*. Omezení spočívá ve vytvoření horní hranice pro počet záznamů slovníku. Při překročení je nejstarší záznam automaticky zahazován. Není totiž nezbytné mít v paměti všechny robots.txt soubory, ale jenom ty nejčastěji dotazované. Crawler tedy nebude muset před každým pokusem o crawlování stránky stahovat a kontrolovat k ní příslušný /robots.txt dokument. Naopak bude mít k dispozici pravidla parsovaných robots.txt souborů v paměti. Toto opatření snižuje počet dotazů směrovaných na Web servery. V případě použití této kontroly s absencí paměti by došlo ke zdvojnásobení HTTP požadavků.

4.5.3.4 harvester

Úkolem tohoto modulu (je součástí funkce *DatamineThread*) je „vytěžování“ informací z webových stránek. Základem je knihovna Beautiful Soup, která obsahuje výkonný HTML/XML parser. Pomocí ní je text rozkouskován do stromové struktury. Z HTML

30 Dostupné na WWW: http://www.robotstxt.org/orig.html [cit. 2011-05-05].

je posléze možné přímo vybrat veškeré hypertextové odkazy. Po jejich převedení do absolutní podoby získáme seznam všech obsažených URL odkazů.

```
aTags = soup.findAll('a')
#iterative processing of found tags
for tag in aTags:
   #take just a href tag, other throw away
   if tag.has_key('href'):
         #complete the link
         link = urljoin(self.base_url, tag['href'])
```

Ukázka 18: Vytěžení hypertextových odkazů z HTML

Největší předností knihovny BeautifulSoup je schopnost pracovat i se špatně vytvořenými dokumenty, aniž by při tom zhavarovala. *Harvester*, kromě URL odkazů, ze stránky stahuje její titulek, popis, klíčová slova a samostatný text.

4.6 Odolnost systému

Pod pojmem odolnost systému rozumíme schopnost aplikace vypořádat se s překážkami a problémy, které před ní budou po dobu procházení Webu postaveny. Během celého vývoje Distribuovaného web crawleru bylo realizováno veliké množství testů zaměřených zejména na sledování vnitřních funkcí a chování aplikace. Jedním z primárních požadavků byla právě odolnost systému. Narazí-li jakákoliv část aplikace na problémovou URL, zahodí ji a pokračuje dále. Výjimkou je získávání informací z webových stránek. Namísto zahození dat je stránka uložena v nativním HTML, aby nedošlo ke ztrátě dat.

Aplikace je také navržena tak, aby odolala případným výpadkům sítě. Jednotlivé části neukončí svůj běh, ale v pravidelných intervalech se pokouší připojit k serveru.

5 Test distribuovaného web crawleru

Vytvořený Distribuovaný web crawler je plně funkční aplikací, lze ji tedy i testovat. U aplikace se bude zkoumat rychlost zpracovávání přidělených URL v závislosti na množství vláken a klientů a množství dotazů směrovaných na jeden webový server během jediného cyklu. Testy se zaměří na doménu http://www.tul.cz. Schopností crawleru je možnost prohledávat pouze určenou doménu, čehož je v testu také využito.

5.1 Test rychlosti zpracovávání přidělených URL

Hlavním zkoumaným prvkem je rychlost zpracovávání přidělených URL v závislosti na množství použitých vláken a klientů.

Testovaný vzorek: 10000 URL získaných postupným crawlováním http://www.tul.cz.

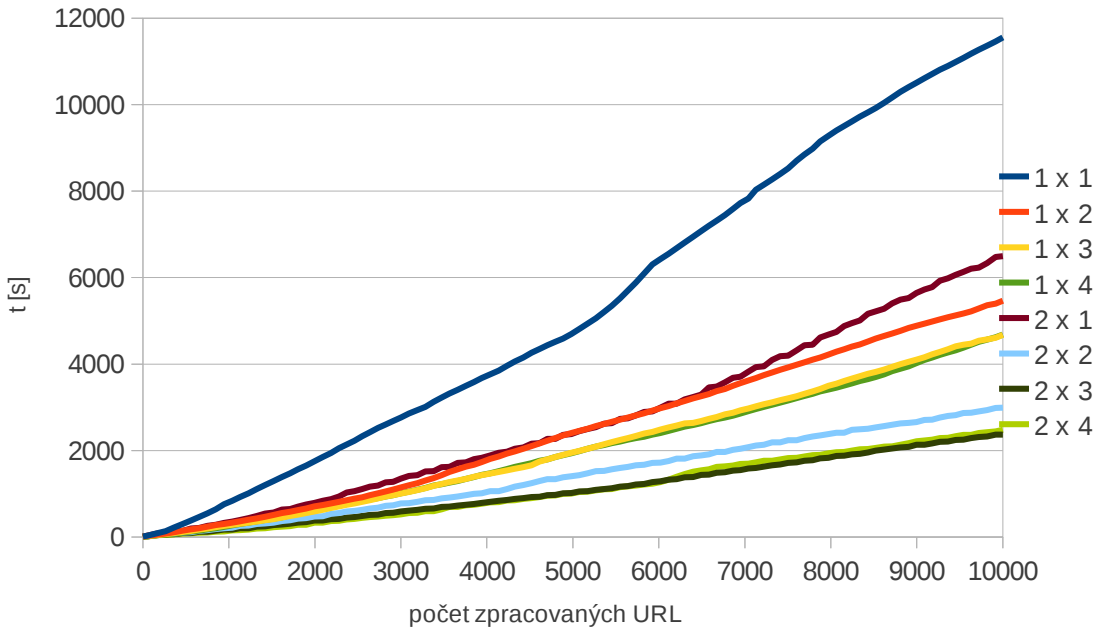
Použitá PC: DELL Latitude D630 (Intel Core2Duo, T8300, 2,4 Ghz) – server, klient

Acer Aspire 5100 (AMD Turion 64, 2 GHz) – klient

Rychlost připojení k Internetu: 1Gb/s (testováno mezi 12 – 24h)

Nastavení distribuovaného web crawleru: Crawleru bude vždy zasíláno 100 URL adres s tím, že při jejich zpracovávání bude testovat příslušné robots.txt soubory, bude parsovat stažené stránky a bude ukládat nalezené soubory.

Předpoklad: S každým použitým vláknem a klientem vzroste rychlost crawlování. Největší nárůst by měl nastat při použití 4 párů vláken a dvou klientů.



Graf 1: Rychlost zpracovávání URL v závislosti na počtu vláken a klientů

Každá z křivek grafu 1 představuje jednu konfiguraci clientCrawleru. První číslo udává počet klientů zařazených do výpočtů a druhé specifikuje nastavení množství vláken, které bude klient využívat ke crawlování. Nelze opomenout, že clientCrawler vždy vytváří vlákna v párech – jedno stahuje stránky, druhé z nich získává data. Opravdový počet vláken je tedy dvojnásobný.

		čas (s) nutný pro zpracování v závislosti na nastavení web crawleru							
		1 x 1	1 x 2	1 x 3	1 x 4	2 x 1	2 x 2	2 x 3	2 x 4
zpracované URL adresy	1	0,61	0,62	0,62	0,63	0,61	0,64	0,63	0,63
	1066	830,42	321,78	275,14	262,37	354,29	206,81	171,91	151,69
	2066	1739,88	694,76	585,28	612,64	781,18	468,29	375,80	327,64
	3066	2670,72	1085,65	963,16	962,16	1271,23	725,83	560,20	504,39
	4066	3572,02	1660,07	1383,93	1380,69	1796,89	992,32	765,07	745,29
	5066	4507,73	2279,40	1839,99	1844,25	2263,14	1331,04	972,17	958,22
	6066	5873,81	2803,78	2331,38	2277,99	2809,82	1652,22	1210,13	1188,29
	7066	7308,24	3360,47	2770,31	2717,23	3474,83	1957,08	1485,26	1618,67
	8066	8694,66	3973,68	3246,54	3207,87	4310,47	2231,80	1725,51	1830,42
	9066	9914,18	4577,52	3811,95	3695,64	5211,39	2586,89	1996,00	2059,61
	10066	10981,04	5114,16	4403,15	4313,82	6056,25	2925,81	2243,80	2329,16
	10666	11545,56	5453,81	4669,10	4677,44	6492,97	3132,64	2366,67	2478,24

Tabulka 1: Získané časy v závislosti na počtu zpracovaných URL adres (na tisíce)

Dle předpokladu 10000 URL adres nejpomaleji zpracoval jeden klient s jedinou dvojicí obslužných threadů. Trvalo mu to skoro tři hodiny. Test ale přinesl dvě veliká překvapení. Prvním byl až nečekaný nárůst rychlosti jak při zařazení dalšího klienta, který znamenal téměř zdvojnásobení rychlosti, tak použití druhého páru obslužných vláken. Tato konfigurace dokonce předčila dva klienty, což mohlo být způsobeno využitím nehomogenních PC. Další překvapení spočívalo ve vyvrácení druhé části předpokladu. ClientCrawler dosahuje nejvyšší rychlosti (10000 URL za necelých 40 minut) očekávaně při použití dvou klientů, ale ne již čtyřech párů threadů, nýbrž tří. Pravděpodobná příčina spočívá ve faktu, že systémové zatížení bylo natolik veliké, že zpomalilo crawlování. Lze tak usuzovat z druhého výsledku použití čtyřech párů vláken jediným klientem. I zde je rychlost zpracovávání URL adres nepatrně vyšší při využití pouze tří párů vláken.

Porovnáváním počtu odeslaných URL adres s počtem adres úspěšně zpracovaných byl získán údaj o úspěšnosti crawlování (stažení a parsování) aplikace při zpracovávání stránek.

		počet zpracovaných stránek vzhledem k nastavení (na tisíce)							
		1 x 1	1 x 2	1 x 3	1 x 4	2 x 1	2 x 2	2 x 3	2 x 4
Počet odeslaných URL	66	64	64	64	64	64	64	64	64
	966	945	948	952	952	956	950	951	955
	1066	1038	1046	1052	1049	1056	1048	1050	1054
	2066	1996	2030	2027	1998	2036	2032	2037	2036
	3066	2909	3016	3009	2944	3000	3017	3009	3006
	4066	3849	3965	3987	3884	3941	4008	3958	3985
	5066	4796	4902	4967	4805	4880	5002	4870	4959
	6066	5733	5815	5915	5737	5814	5998	5795	5936
	7066	6672	6735	6849	6663	6739	6993	6733	6903
	8066	7602	7653	7773	7590	7667	7965	7642	7871
	9066	8522	8550	8709	8498	8605	8930	8541	8838
	10066	9453	9441	9651	9435	9529	9882	9444	9816
efektivita		0,9391	0,9379	0,9588	0,9373	0,9467	0,9817	0,9382	0,9752
průměr		0,9519							

Tabulka 2: Počet odeslaných URL k počtu zpracovaných URL crawlery

Úspěšnost Distribuovaného web crawleru se pohybuje kolem 95 % zpracovaných URL na vzorku 8x10000 adres. Zbylých 6 % tvoří neexistující odkazy a stránky, které se z nespecifikovaných důvodů nepodařilo zpracovat.

5.2 Množství dotazů na webový server

Během vytvoření Distribuovaného web crawleru byl diskutován návrh zajišťující dostatečné rozložení URL adres mezi klienty, aby nedocházelo k přetěžování webových serverů. Na straně clientCrawleru jsou nalezené odkazy nejdříve náhodně promíchány a teprve poté odesílány URL serveru. Ten je zařadí do fronty ve stejném pořadí, v jakém je obdrží. Zpětný výběr adres probíhá z obou konců fronty, což jenom umocňuje náhodné přidělování URL klientům. Tento mechanismus by byl při cíleném vyhledávání neefektivní, ale pro potřebu mapování co největšího rozsahu webu je přijatelný.

Z přenosu souborů mezi klientem a serverem během crawlování byly vybrány čtyři náhodné vzorky odesílaných výsledků. Jelikož každý obsažený soubor představuje přesně jednu zpracovanou URL, lze z nich vyčíst i rozložení dotazů jednoho cyklu crawlování.

Cíl: Zjistit rozložení adres v dávce 100 URL a vypočítat největší množství dotazů směrovaných na webový server během jednoho cyklu crawleru.

Testovaný vzorek: 8 náhodně vybraných kompletních sad výsledků crawlování adresy http://www.tul.cz odebrané v různých stádiích běhu

Použitá PC: DELL Latitude D630 (Intel Core2Duo, T8300, 2,4 GHz) – server, klient

Nastavení distribuovaného web crawleru: Crawleru bude vždy zasíláno 100 URL adres s tím, že při jejich zpracovávání bude testovat příslušné robots.txt soubory, bude neparsovat stažené stránky a bude ukládat nalezené soubory. Použitá vlákna = 2

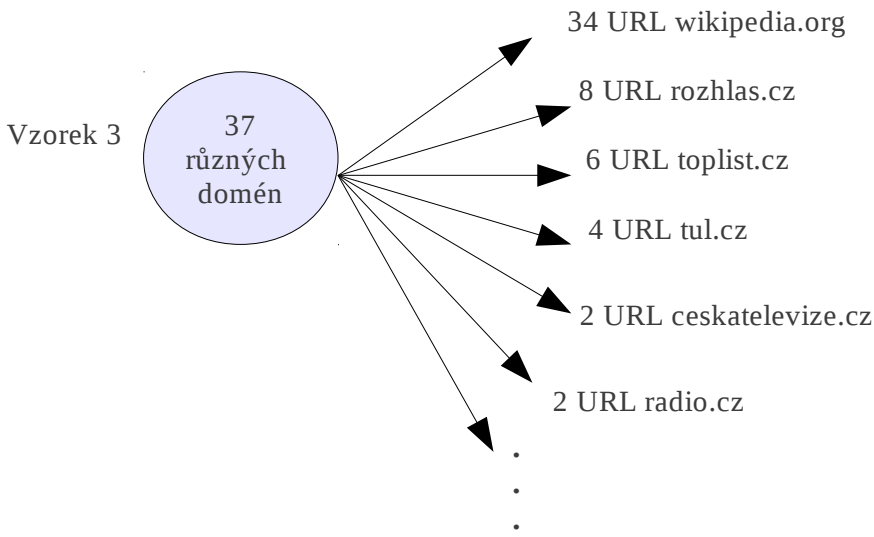
Rychlost připojení k Internetu: 1Gb/s (testováno mezi 12 – 24h)

	Rozložení 100 URL do x domén	adres ve frontě
Vzorek 1	44	145430
Vzorek 2	37	216665
Vzorek 3	37	279836
Vzorek 4	42	341017

Tabulka 3: Rozložení URL adres do domén

Tabulka 3 přesně zobrazuje, mezi kolik webových serverů bylo rozprostřeno 100 URL v

jednotlivých vzorcích. Podíváme-li se blíže na vzorek 3, zjistíme, že 100 URL adres se zde rozprostřelo mezi 37 domén. Zastoupení URL v doménách demonstruje následující obrázek.



Obrázek 17: Detailní rozložení URL adres ve vzorku 3

Z obrázku 14 je vidět, že 34 URL ze 100 přidělených odkazuje na doménu wikipedia.org. Při předpokladu, že jedna doména představuje fyzicky jeden webový server, bude na wikipedia.org směrováno 34 HTTP požadavků během jediného cyklu (35 připočteme-li dotaz na soubor robots.txt). Průměrná doba zpracování 100 URL v tomto testu byla 38,56 sekund. Z toho vyplývá, že v nejhorším případě bylo na server wikipedia.org směrováno 0,91 dotazu za sekundu. Toto číslo je pouze orientační, může výrazně lišit podle nastavení crawleru, množství použitých vláken a počítačů.

5.3 Crawlování jediné domény

Nejvyšší zátěž pro webové servery představuje crawlování jediné domény. Kdyby na tomto procesu spolupracovalo dostatečné množství počítačů, mohly by být schopny server přetížit a následně i „shodit“ (DoS útok). Vytvořený klientCrawler nedisponuje omezením rychlosti prohledávání Webu, je tedy potřeba v tomto případě využít minimálních zdrojů.

Úkol: Zjistit, kolik stránek a dat obsahuje jediná doména.

Testovaný vzorek: www.fm.tul.cz, www.tul.cz

Použitá PC: DELL Latitude D630 (Intel Core2Duo, T8300, 2,4 GHz) – server, klient

Rychlost připojení k Internetu: 1Gb/s (testováno mezi 12 – 24h)

Nastavení Distribuovaného web crawleru: Crawler bude zpracovávat adresy náležící pouze jediné doméně; dávka po 100 URL; bude testovat robots.txt soubor; nebude parsovat stažené stránky; bude ukládat nalezené soubory.

Výsledky: Počet nalezených stránek na doméně tul.cz je 661, z čehož bylo staženo 606 souborů o celkové velikosti 110,6 MB dat (78,4 MB po zkomprimování). Subdoména fm.tul.cz obsahuje 807 URL adres. Bylo staženo 778 souborů o celkové velikosti 172,6 MB.

Získaná data reprezentují veškerý obsah, který bylo možné z domén získat. Tímto způsobem lze zálohovat data jednoho webového serveru, nebo jenom testovat, co vše se na jednotlivých serverech může nacházet.

6 Manuál

Distribuovaný web crawler je volně šiřitelný software vytvořený za účelem možnosti autonomního stahování dat z WWW. Jádrem programu jsou kódy napsané v jazyce Python. Aplikace je vytvořená na základě technologie distribuovaných aplikací. Hlavní části tvoří urlServer, jehož úkolem je řídit distribuované klientCrawlery a ftpServer obstarávající datové přenosy. Předpokladem pro správnou funkci aplikace je existence HW serveru připojeného k Internetu, na němž bude spuštěn urlServer a ftpServer. Není nezbytně nutné, aby tyto dvě části byly umístěny na jediné stanici. Jsou na sobě nezávislé. Zbylou část představuje clientCrawler, který bude distribuován na uživatelských počítačích (nutné, aby měly přístup na Internet). Ten poběží jako služba systému. Klient nejdříve požádá server o přidělení úkolu, který poté zpracuje a odešle výsledky. Na ostatních klientech je nezávislý. Pro případ testování je možné umístit všechny tři části na jediný počítač a spouštět klientCrawler v testovacím módu.

6.1 Požadavky

Základními požadavky je OS Linux, aplikace pro něj byla primárně vytvářena a nainstalovaný balíček Python. Aby software bylo možné spouštět v OS Windows, je nutné provést dodatečné úpravy, které tento manuál nepokrývá.

Systémové požadavky:

- OS Linux 2.6
- Python 2.5 a vyšší

Aplikace je z větší části vytvořená na standardních knihovnách jazyka Python. Ostatní byly zahrnuty přímo do kódu aplikace, aby odpadla potřeba jejich dodatečné instalace. Veškeré části by měly být spustitelné. V případě problémů doporučuji doinstalovat moduly SOAPpy, pyXML.

6.2 Spuštění a konfigurace

Distribuovaný web crawler tvoří tři aplikace. Každá se pouští odděleně. Nejprve se zaměřme na možnosti konfigurace. Aby nebylo nutné hodnoty měnit a vyhledávat v kódu, jsou využity externí konfigurační soubory *config.cfg*.

```
~/Plocha/Distribuovany_web_crawler$ tree
|-- client
|   |-- clientCrawler.py
|   |-- config.cfg
`-- server
    |-- config.cfg
    |-- ftpServer.py
    |-- urlServer.py
```

Obrázek 18: Adresář distribuovaného web crawleru

Config.cfg v adresáři serveru je hlavním konfiguračním souborem.

```
[url_Server]
ip = 147.230.165.138
server_port = 50001
#start urls could contain more than one URL
#each URL is separated with ; to each other
#example: http://www.tul.cz/;http://www.fm.tul.cz;...
start_Urls = http://www.tul.cz
urls_to_crawl = 300
#client settings
robots_control = 1
page_parse = 1
sitemap = 0
document_save = 1

[ftp_Server]
ip = 147.230.165.138
ftp_port = 50003
#assign folder to download
directory = Pages
```

Ukázka 19: Config.cfg pro server

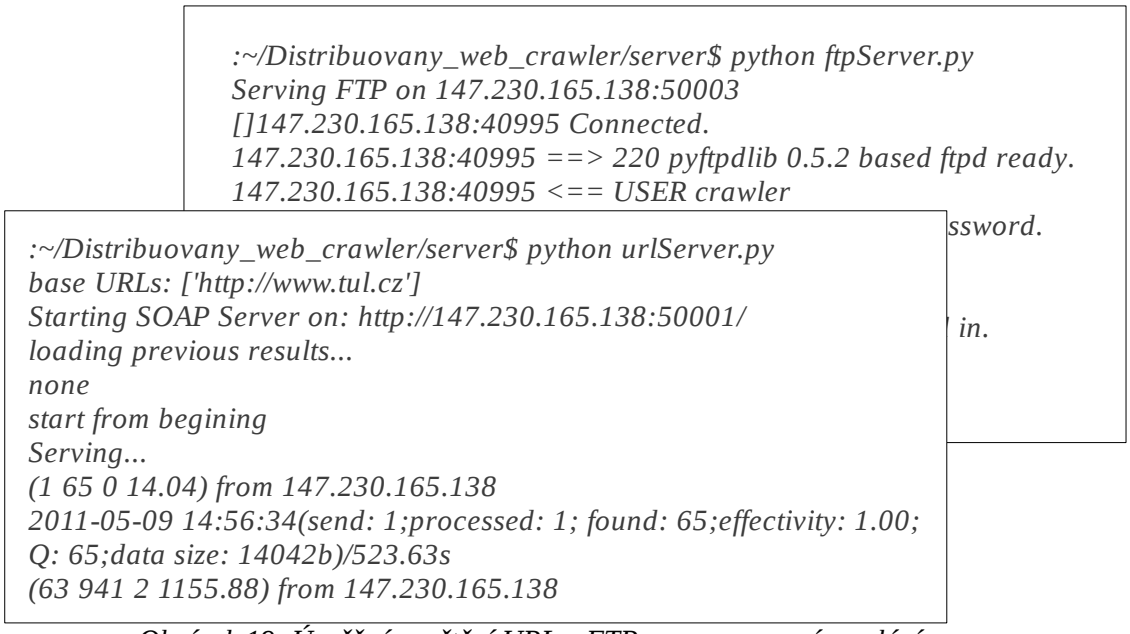
Jak je vidět z ukázky 19, config obsahuje oddělenou část pro URL a FTP server. Oběma nezávisle určí, na jaké adrese a portu mají naslouchat (pomocí hodnot IP a port). V případě oddělení serverů se bude jejich IP lišit. Zbytek nastavení pro URL server

tvoří konfigurační hodnoty pro řízení clientCrawlerů. Start_Urls definuje počáteční URL adresy a *urls_to_crawl* určí množství adres, které URL server odešle clientCrawleru. Nastavení *robots_control*, *page_parse*, *sitemap*, *document_save* server odesílá klientu, čímž definuje, jak má crawler pracovat:

- *robots_control = 1/0* (použije kontrolu souboru robots.txt)
- *page_parse = 1/0* (parsuje stránky, v případě 0 je ukládá jako HTML dokument)
- *sitemap = 1/0* (nastavení definuje, zda-li se povolí prohledávání pouze v rámci domény, nebo celého Webu)
- *document_save = 1/0* (určí, jestli se mají ukládat nalezené dokumenty)

Aplikace jsou spouštěny z příkazového řádku příkazy:

- *python urlServer.py* spustí URL server
- *python ftpServer.py* spustí FTP server



Obrázek 19: Úspěšné spuštění URL a FTP serveru s prvním voláním

Informace o běhu URL serveru jsou do konzole vypisovány při každém volání funkce *SendAllData* (přenos dat od klienta serveru) v programu. První řádek po výpisu o spuštění informuje o obdržených datech. V závorce je uveden v přesném pořadí: počet

úspěšně zpracovaných URL, počet nově nalezených URL, počet nezpracovaných URL, množství získaných dat. IP identifikuje klienta, který data zaslal. Druhý řádek vypovídá o aktuálním stavu serveru: datum a čas, kdy obdržel data, množství již odeslaných URL, celkový počet úspěšně zpracovaných URL, celkový počet nalezených URL, efektivita ve zpracování stránek, množství adres zařazených ve frontě a čekajících na zpracování, součet získaných dat, doba spuštění serveru v sekundách. Obsah kontrolních zprávách lze libovolně upravovat přímo v kódu URL serveru (změna výpisových hodnot funkce *SendAllData*).

Po spuštění URL serveru se v adresáři vytvoří složka *./mem* a soubor *test.txt*. Složka *./mem* uchovává záznam o průběhu crawlování. V případě ukončení serveru budou do složky uloženy veškeré výsledky. Při příštím spuštění si server tyto výsledky načte a bude pokračovat tam, kde minule skončil. V případě, že byste chtěli začít úplně nové crawlování, je potřeba tuto celou složku (nebo jenom soubory v ní) smazat. Kromě záznamů je zde i soubor *blacklist.txt* obsahující seznam nezpracovaných URL. Do souboru *test.txt* v kořenovém adresáři jsou nepřetržitě zaznamenávány hodnoty o době běhu serveru, počtu zpracovaných stránek a počtu nalezených URL. Spuštěním FTP serveru se v adresáři vytvoří složka *Pages*. Ta představuje pracovní adresář clientCrawleru. Obsahuje uložené stránky a dokumenty z WWW.

Po samotném spuštění serverů bychom nezískali žádné výpisy o běhu, pokud by na nějakém hostovi nebyl již spuštěný clientCrawler. Jeho možnosti konfigurace ze souboru *config.cfg* jsou značně omezené.

```
[settings]
server_ip = 147.230.165.138
server_port = 50001
number_of_threads = 2
```

Ukázka 20: Nastavení pro clientCrawler

Z obsahu souboru *config.cfg* (ukázka 20) je vidět, že klientovi je potřeba nastavit cestu k URL serveru *(server_ip, server_port)*. *Number_of_threads* umožňuje definovat, kolik vláken bude využito ke crawlování. Pozor na toto číslo! Podíváme-li se na návrh crawleru zjistíme, že k práci používá dvě skupiny vláken. Skutečné množství threadů v aplikaci bude rovno dvojnásobku definovaného čísla.

```
:~/Distribuovany_web_crawler/client$ python clientCrawler.py
usage: clientCrawler.py start|stop|restart|foreground
:~/Distribuovany_web_crawler/client$ python clientCrawler.py start
```

Obrázek 20: Spuštění clientCrawleru

ClientCrawler má několik možností spuštění. Pokud není žádná uvedena, jsou vypsána do příkazové řádku (viz obrázek 16). Spuštění s parametrem *start* spustí aplikace na pozadí (systémová služba unixových systémů). V tom případě nemá uživatel přístup k výpisům. Službu je možné restartovat, nebo úplně ukončit pomocí příslušných příkazů. Speciální možností spuštění je *foreground*, neboli spuštění do popředí. Ta je vytvořena za účelem testování a odlaďování aplikace. ClientCrawler je spuštěn v klasickém módu s výpisy do konzole. Lze tedy přesně sledovat jeho aktuální činnost z výpisů.

```
:~/Distribuovany_web_crawler/client$ python clientCrawler.py foreground
[2011-05-09 15:50:31] starting clientCrawler.py ...
[2011-05-09 15:50:31] get 300 URLs to crawl
Thread-3    http://fistfulofeuros.net/2004/06/
[CT: text/html; charset=UTF-8][size: 121347]
Thread-4    http://www.kpmg.com/cz/cs/whatwedo/Stranky/x
[CT: text/html; charset=utf-8][size: 88997]
Thread-3    http://www.maitre-eolas.fr/archive/2008/07/07
[CT: text/html; charset=UTF-8][size: 46570]
...
```

Ukázka 21: clientCrawler při spuštění do popředí

7 Závěr

Hlavním cílem diplomové práce bylo vytvoření distribuovaného web crawleru (viz kapitola 1.1). Ač některé návrhy distribuovaných web crawlerů spoléhají spíše na decentralizovanou architekturu (UbiCrawler[13], YaCi[31]), tato práce vychází z technologie webových služeb, tedy centralizované klient / server architektury. Nevýhodou je přítomnost centrálního řídícího prvku. Oproti tomu výhoda spočívá v nižších nárocích na chod a synchronizaci jednotlivých částí aplikace (centralizovanou architekturu využíval například Mercator AltaVista crawler[14]). Aplikace byla naprogramována v jazyce Python. Hlavní části tvoří server obsluhující webové služby a software web crawleru. Ten je možné libovolně distribuovat na klientské počítače. Volání vzdálených služeb je řešeno pomocí SOAP protokolu (viz kapitola 2.3). Přenos souborů zajišťuje FTP server. Systém řízení je založen na technologii náhodného přidělování URL adres crawlerům (viz kapitola 4.4.3).

7.1 Dosažené výsledky

Byl vytvořen funkční distribuovaný web crawler (viz kapitola 5). V práci jsou aplikovány principy vícevláknového programování. Aplikace je schopná paralelního zpracovávání úloh jak ze strany řídícího serveru (viz kapitola 4.4.1) tak crawleru (viz kapitola 4.5.3). Řídící server může zároveň obsluhovat více klientů. V experimentální části byl ověřen přínos spuštění klientů na více počítačích, stejně jako využití více vláken na klientu. S každým použitým klientem i vláknem dochází ke zrychlení aplikace (viz. kapitola 5.1).

13 Dostupné na WWW: http://ausweb.scu.edu.au/aw02/papers/refereed/vigna/paper.html [cit. 2011-05-15].

31 Dostupné na WWW: http://yacy.net/en/ [cit. 2011-05-18].

14 Dostupné na WWW: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.9096&rep=rep1&type=pdf [cit. 2011-05-15].

7.2 Možnosti dalšího vývoje

Možností navázání na tuto diplomovou práci je mnoho, ale mezi ty nejzajímavější by mohlo patřit:

- Vytvoření strategie pro opětovné navštěvování stránek (re-visit policy).
- Místo fronty *deque* použít jednu či více prioritně orientovaných front a upravit celý model procházení URL adres.
- Zahrnout metodu pro hodnocení významu stránek (např. PageRank).
- Zahrnout do kódu podporu pro sitemap a pomocí hodnocení generovat vlastní sitemapy.
- Přepracovat model datového úložiště a pro ukládání dat využívat některou z moderních nonSQL databází.
- Převést stávající kód do jiného jazyka (C, C#, JAVA) a porovnat přínosy.

Literatura

[1] *About The World Wide Web* [online]. Dostupný na WWW: http://www.w3.org/WWW/

[2] *Web Growth Summary* [online]. Dostupný na WWW: http://www.mit.edu/people/mkgray/net/web-growth-summary.html

[3] *The size of the World Wide Web* [online]. Dostupný na WWW: http://www.worldwidewebsize.com/

[4] Koster, M. *Guidelines for Robot Writers* [online]. Dostupný na WWW: http://www.robotstxt.org/guidelines.html

[5] *Worl community grid* [online]. Dostupný na WWW: http://www.worldcommunitygrid.org/

[6] *Seti@home* [online]. Dostupný na WWW: http://setiathome.berkeley.edu/

[7] *Distributed Application Architecture* [online]. Dostupný na WWW: http://java.sun.com/developer/Books/jdbc/ch07.pdf

[8] *Distributed Application Development Process* [online]. Dostupný na WWW: http://www.gantthead.com/content/processes/8615.cfm

[9] R*emote procedure call* [online]. Dostupný na WWW: http://en.wikipedia.org/wiki/Remote_procedure_call

[10] KOSEK, J. I*nteligentní podpora navigace na WWW s využitím XML* [online]. Dostupný na WWW: http://www.kosek.cz/diplomka/html/ch04s02.html

[11] *Introduction to WSDL* [online]. Dostupný na WWW:
http://www.w3schools.com/wsdl/wsdl_intro.asp

[12] *WSDL and UDDI* [online]. Dostupný na WWW:
http://www.w3schools.com/WSDL/wsdl_uddi.asp

[13] BOLDI, P., CODENOTTI, B., SANTINY, M., VIGNA, S. *UbiCrawler: A Scalable Fully Distributed Web Crawler* [online]. Dostupné na WWW:
http://ausweb.scu.edu.au/aw02/papers/refereed/vigna/paper.html

[14] SHKAPENYUK, V., SUEL,T. *Design and implementation of a high-performance distributed web crawler* [online]. Dostupný na WWW:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.9096&rep=rep1&type=pdf

[15] CASTILLO, C. *Effective Web Crawling* [online]. Dostupný na WWW:
http://www.chato.cl/papers/crawling_thesis/effective_web_crawling.pdf

[16] *Thread (computer science)* [online]. Dostupný na WWW:
http://en.wikipedia.org/wiki/Thread_(computer_science)

[17] *Python Programming Language – Official Website* [online].Dostupný na WWW:
http://www.python.org/

[18] *MIME Reference* [online]. Dostupný na WWW:
http://www.w3schools.com/media/media_mimeref.asp

[19] *RFC 959 - File Transfer Protocol* [online]. Dostupný na WWW:
http://www.faqs.org/rfcs/rfc959.html

[20] *Python FTP server library* [online]. Dostupný na WWW:
http://code.google.com/p/pyftpdlib/

[21] *Pyftpdlib tutorial* [online]. Dostupný na WWW: http://code.google.com/p/pyftpdlib/wiki/Tutorial

[22] BRIN, S., PAGE, L. *The Anatomy of a Large-Scale Hypertextual Web Search Engine* [online]. Dostupný na WWW: http://infolab.stanford.edu/~backrub/google.html

[23] *Server Definition* [online]. Dostupný na WWW: http://www.linfo.org/server.html

[24] *Apache HTTP server project* [online]. Dostupný na WWW: http://httpd.apache.org/

[25] *SOAPPy* [online]. Dostupný na WWW: http://soapy.sourceforge.net/

[26] *Linux Daemon Writing HOWTO* [online]. Dostupný na WWW: http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html

[27] *Beautiful Soup Documentation* [online]. Dostupný na WWW: http://www.crummy.com/software/BeautifulSoup/documentation.html

[28] *URL normalization* [online]. Dostupný na WWW: http://en.wikipedia.org/wiki/URL_normalization

[29] *urlnorm 1.1.2* [online]. Dostupný na WWW http://pypi.python.org/pypi/urlnorm

[30] *A Standard for Robot Exclusion* [online]. Dostupný na WWW: http://www.robotstxt.org/orig.html

[31] *YaCy, decentralized web search* [online]. Dostupný na WWW: http://yacy.net/en/

Seznam příloh

Příloha 1: CD

Příloha 1: CD

Elektronická příloha obsahuje:

- elektronickou verzi diplomové práce;
- aplikaci distribuovaného web crawleru.