VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INKREMENTÁLNÍ STAHOVÁNÍ WEBU POMOCÍ SYSTÉMU BUBING

INCREMENTAL WEB CRAWLING WITH BUBING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **KAREL ONDŘEJ**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. PETR ŠKODA**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Ondřej Karel**
Obor: Informační technologie
Téma: **Inkrementální stahování webu pomocí systému Bubing**
Incremental Web Crawling With Bubing System
Kategorie: Paralelní a distribuované výpočty

Pokyny:

1. Prostudujte existující využití systému Bubing a jeho případných mutací pro inkrementální stahování webu. Prostudujte jiné dnes používané systémy pro inkrementální stahování webu.
2. Propojte existující systém pro deduplikaci textů skupiny KNOT se systémem Bubing a navrhněte způsob pro vyhodnocení duplicity celých dokumentů stažených z webu.
3. Prostudujte interní struktury systému Bubing a navrhněte možnosti využití systému k uchovávání informací o poslední návštěvě dané URL a plánu další návštěvy.
4. Navrhněte úpravy Bubingu, které umožní opakované procházení URL a tedy inkrementální stahování změněných webových stránek.

Literatura:

1. Dle zadání vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Škoda Petr, RNDr.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá úpravou systému BUbiNG pro takzvané inkrementální stahování. V práci jsou dále popsány hlavní problémy spojené s inkrementálním stahováním internetu a využití dalších open-source systémů pro inkrementální stahování. Upravený systém podporuje opětovné navštěvování stránek pomocí dvou běžně používaných strategií. První ze strategií opětovně navštěvuje stránku vždy po stejném intervalu. Druhá strategie přizpůsobuje interval mezi návštěvami podle frekvence změn stránky.

Abstract

This bachelor thesis deals with modification of BUbiNG system for incremental crawling. The paper describes the main problems related to incremental Internet crawling and the use of other open-source systems for incremental crawling. As a result, BUbiNG system supports re-visiting pages using two commonly used strategies. The first strategy always re-visits page after the same interval. The second strategy adjusts the interval between visits according to the frequency of page changes.

Klíčová slova

BUbiNG, webový prohledávací modul, inkrementální stahování, duplicita textu, politika opětovných návštěv

Keywords

BUbiNG, web crawler, incremental crawling, duplicity detection, re-visit policy

Citace

ONDŘEJ, Karel. *Inkrementální stahování webu pomocí systému Bubing*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Petr Škoda

Inkrementální stahování webu pomocí systému Bubing

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Petra Škody. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Karel Ondřej
9. května 2018

Poděkování

Tímto chci poděkovat mému vedoucímu práce panu RNDr. Petru Škodovi za užitečné rady při vývoji a psaní této práce. Dále bych rád poděkoval Veronice Sýkorové za jazykovou korekci textu a celé mojí rodině za projevenou podporu.

Obsah

Kapitola 1

Úvod

Díky decentralizaci internetu je potřeba najít vhodný způsob pro jeho prohledávání a centralizované zpracování. Jedna z možností je využít inkrementální stahování, které nám vytvoří aktuální vstupní databázi pro následné operace.

Cílem této bakalářské práce je úprava již existujícího systému BUbiNG pro inkrementální stahování. Pro určení, jak často se má stránka navštěvovat, bylo zadáno využít systém pro deduplikaci výzkumné skupiny KNOT.

Kapitola 2 uvádí čtenáře do problematiky stahování webových stránek z internetu. Postupně je rozebráno několik druhů webových prohledávacích modulů a základní problémy, se kterými se potýkají.

V kapitole 3 je popsáno několik existujících open-source řešení a jejich možné použití pro inkrementální stahování. Tyto systémy jsou následně srovnány se systémem BUbiNG. Jelikož se jedná o komplexní systém, je pro lepší představu fungování stručně popsána i jeho architektura.

Implementaci provedených změn popisuje kapitola 4, která je rozdělena do dvou částí. První část se zabývá propojením nástroje pro deduplikaci výzkumné skupiny KNOT se systémem BUbiNG a jeho využití pro určování duplicity webových stránek. Druhá část kapitoly pojednává o potřebných úpravách systému BUbiNG pro potřeby inkrementálního stahování.

Vyhodnocení provedených změn se nachází v kapitole 5 a poslední kapitola 6 shrnuje výsledky práce a další možnosti vývoje systému.

Kapitola 2

Inkrementální stahování webu

Kapitola pojednává o důvodech stahování obsahu z internetu, principech aplikací stahujících webové stránky a problémech, se kterými se potýkají. Hlavně se zaměří na inkrementální stahování, kdy se stránky navštěvují opakovaně. S inkrementálním stahováním je spojeno určování, zda se stránka od posledního navštívení změnila, které slouží k naplánování další návštěvy.

2.1 Úvod do problematiky

Dokumenty na internetu nejsou centrálně uloženy, ale nachází se na mnoha různých webových serverech vlastněných různými majiteli. Z tohoto důvodu nemá nikdo přímý přístup ke všem aktuálním dokumentům sdílených na internetu.

Decentralizace internetu je problém pro operace, které se provádí nad celým jeho obsahem. Existuje proto snaha o vytvoření centrální databáze webových stránek, která by byla optimalizovaná pro tyto operace.

Jelikož se obsah webových stránek neustále mění, vznikají nové stránky a jiné zanikají, nebo na stejnou stránku může odkazovat více odkazů. Tím dochází k tomu, že je databáze neaktuální, nebo má duplicitní záznamy. Stránka, která se změnila, může taky obsahovat nové odkazy na ještě nestažené stránky.

Využitím inkrementálního stahování (*incremental web crawling*), kdy se periodicky navštěvují také již jednou stažené stránky a zjišťuje se změna od poslední návštěvy, případně se získají nové odkazy, se předejde neaktuálnosti databáze.

Se zvětšujícím se obsahem na internetu vzrůstá problém s nalezením umístění hledaných dokumentů. Vznikly tedy aplikace indexující internet, které poskytují uživateli odkaz na hledaný obsah. Tyto aplikace využívající právě inkrementálního stahování. Inkrementální stahování je využíváno také ve výzkumu například pro zaznamenání vývoje obsahu na internetu, vytváření statistik nebo získávání a zpracovávání informací. Dalším využitím může být i odhalování plagiátorství. [18]

2.2 Webový prohledávací modul

Webový prohledávací modul (*web crawler*) je program sloužící ke stahovaní webových stránek.

Stahování pomocí webových prohledávacích modulů probíhá v několika krocích. Na začátek potřebujeme prvotní množinu URL („seed“) sloužící pro inicializaci fronty stránek ke stažení. Z fronty se vybere URL webové stránky, která se stáhne. Z obsahu stránky se zpracují odkazy na další webové stránky. Pokud se jedná o ještě neznámé odkazy, tak se uloží do fronty a pokračuje se výběrem další URL z fronty. Postup se opakuje dokud není fronta prázdná. Ostatní užitečné data, jako obsah nebo datum stažení stránky, se uloží na disk.

Běžně se používá několik druhů webových prohledávacích modulů pro různé specifické účely, které jsou podrobně popsány v následujících podkapitolách.

2.2.1 Webový prohledávací modul se zaměřením

Webový prohledávací modul se zaměřením (*focused crawler*) se orientuje pouze na stahování stránek, které jsou v určitém vzájemném vztahu. Příkladem může být sbírání dat ohledně politických voleb, veřejného mínění nebo archivace určité části internetu. Důležitými prvky strategie jsou relevance stažené stránky a určení dalšího postupu. Proto se musí zvolit vhodná metrika pro určení relevance stažené stránky a predikce, které stránky se mají následně navštívit. [22]

2.2.2 Inkrementální prohledávací modul

Inkrementální prohledávací modul (*incremental crawler*) navštěvuje stránky periodicky a pokud se stránka změnila, uloží do databáze pouze aktuální verzi stránky. Frekvence, podle které navštěvuje jednotlivé stránky, se odvíjí od toho, jak často se stránka mění. Prohledávací modul také může ukládat pouze unikátní stránky, kdy tedy nahradí v databázi duplicitní méně podstatné stránky jinými podstatnějšími stránkami. Významnost dané stránky lze například určit podle počtu jiných stránek, které na ní odkazují. Základní metriky pro určení, které stránky se mají znovu navštívit, jsou stáří (*age*) a aktuálnost (*freshness*) stránky popsané v kapitole 2.4.1. [22]

2.2.3 Distribuovaný a paralelní prohledávací modul

Distribuovaný a paralelní prohledávací modul (*distributed and parallel crawler*) rozděluje práci mezi více strojů. Výhodou rozdělení je pokrytí větší části internetu a v případě poruchy jednoho stroje nedojde k přerušení stahování. Distribuované stahování se potýká s několika problémy. Každá stránka může být v dané chvíli stahovaná pouze jedním strojem a všichni agenti musí dodržovat politiku zdvořilosti, aby nedocházelo k přetěžování serverů.

Hranice mezi distribuovaným a paralelním systémem není pevně daná a jeden prohledávací modul může být nazýván oběma způsoby. Hlavním rozdílem distribuovaného systému je absence sdílené paměti a komunikace mezi agenty pomocí zpráv. Díky tomu můžou být jednotliví agenti na geograficky různých místech.

Rozdělování práce mezi jednotlivé stroje probíhá buď centralizovaně, kdy jeden ze strojů rozděluje ostatním stránky k návštěvě, nebo decentralizovaně, kdy každý z agentů musí zajistit, aby se stránky nestahovaly zbytečně [9]. Centralizovaný přístup je snazší k řízení, ale v případě poruchy řídícího stroje dojde bezprostředně k zastavení celého stahování.

2.3 Vyhledávací politika

Vyhledávací politika určuje, které stránky se budou stahovat a v jakých intervalech se budou ověřovat změny. Také určuje zásady, aby nedošlo k přetížení serveru nebo sítě a koordinaci distribuovaného stahování. V následujících podkapitolách jsou podrobně rozebrány jednotlivé aspekty vyhledávací politiky.

2.3.1 Výběrová politika

Stáhnout všechen obsah internetu již není v dnešní době efektivně možné, případně je cílem stáhnout pouze určité stránky. Na základě výběrové politiky je proto potřeba vybrat nejvíce relevantní stránky pro dané použití a také určit pořadí, ve kterém se mají stáhnout.

Možným kritériem pro stažení stránky může být její obsah. Příkladem lze uvést sbírání dat ohledně voleb, kdy se při navštívení stránky vyhledávají určitá klíčová slova. Na základě počtu a druhu klíčových slov se stránka ohodnotí, a pokud je shledána zajímavou, tak se uloží. Dalším možným kritériem může být stažení pouze HTML stránek. V takovém případě se stahují stránky, jejichž URL končí sufixem .htm nebo .html nebo se stáhne HTTP hlavička a na základě typu internetového média[1] (tzv. „typ MIME“) zjistí, jaká data se nacházejí v těle stránky. Jinými hledanými prvky mohou být jazyk dané stránky nebo její struktura.

Pro zamezení, aby se některé stránky navštěvovaly vícekrát, se používá normalizace URL. Normalizace upraví URL do určitého formátu, kdy více podobných URL budou mít po normalizaci stejný tvar. V rámci normalizace se převádí schéma a doména na malá písmena, převedení diakritiky na escape sekvence (tzv. „URL encoding“[2]) nebo naopak, odstranění fragmentů, upravení položek v dotazu, převedení relativních odkazů na absolutní a mnohé další.

V jakém pořadí se mají jednotlivé stránky stahovat, lze určit například pomocí *PageRank*. *PageRank* se využívá ve vyhledávači Google a vypočítá se na základě odkazů odkazujících na danou stránku [12].

2.3.2 Politika opětovných návštěv

Slouží k určení, kdy se mají znovu navštívit stránky již uložené v databázi. Interval se může určovat na základě několika kritérií tak, aby se efektivně udržovala aktuálnost stažených stránek v databázi. Jedním z kritérií může být četnost změn stránky. Stránky s neměnným obsahem není potřeba navštěvovat tak často jako stránky, které mění obsah každý den. Na druhou stranu stránky, které vygenerují při každé návštěvě jiný obsah, se musí penalizovat, aby nedošlo k přetížení serveru. Jinými kritérii jsou poslední čas navštívení nebo *PageRank*. Otázkou opětovného navštívení stránky se blíže zabývá kapitola 2.4.

2.3.3 Politika zdvořilosti

Pomáhá předejít přetížení sítě nebo serverů. Automatizované stahování znamená vždy zátěž pro danou síť a může vyústit k omezení či přetížení serveru nebo celé sítě.

Běžně webové prohledávací moduly běží po dlouhou dobu v řádech týdnů, až měsíců a paralelně stahují více stránek najednou. To se projeví na využití síťových zdrojů, zvláště

[1] Identifikátor formátu souboru na internetu, složený z typu, podtypu a nepovinných argumentů (např. `text/html; charset=utf-8`).

[2] Netisknuté, rezervované nebo znaky mimo tabulku ASCII jsou převedeny na sekvenci začínající znakem % a následující dvoumístným hexadecimálním číslem (např. převedení znaku ~ na %7E).

při velkém počtu stahování za krátký časový interval. Následkem toho může dojít k nedostatku šířky pásma pro další využití jak na straně sítě, kde je prohledávací modul spuštěn, tak na straně cílové sítě. Další zátěží je velká frekvence dotazů na jeden konkrétní server, kdy dochází ke snížení kvality poskytovaných služeb pro ostatní uživatele, až k samotnému selhání serveru. [15]

Servery mají ochranné mechanizmy, které dokáží zabránit častému dotazování z jednoho zdrojového bodu. V případě spuštění distribuovaných prohledávacích modulů jsou tyto ochranné mechanizmy neúčinné, jelikož nedokáží rozeznat, že se jedná o prohledávací modul spuštěný jedním uživatelem na více strojích současně. Toto chování se nápadně podobá tzv. „DDoS útokům“[3].

Dalším problémem je spuštění prohledávacích modulů více uživateli zaměřených na stejnou doménu bez rozumného omezení hloubky stahování. Jako příklad lze uvést zakázaní používání GNU Wget na stahování obsahu Wikipedie, jelikož v rekurzivním módu bez omezení hloubky stahování a bez omezení čekání mezi dalším navštívením stránky, se jedná o problém, kdy při spuštění více uživateli dojde k velké zátěži serverů.

Běžnými omezeními ze strany prohledávacích modulů jsou: dotazování na jednu doménu pouze jedním agentem, interval mezi stažením další stránky z jedné domény běžně v řádu několika sekund, interval mezi dotazy na jednu IP adresu a maximální počet stažených stránek z jedné domény.

Pro majitele webových stránek se může jednat o finanční ztrátu, jelikož pro potenciální zákazníky je jeho stránka nedostupná a ze stahování webovými prohledávacími moduly pravděpodobně žádný zisk neplyne. Možné řešení je zakázání stahování roboty pomocí *Robots Exlusion Protocol*[4], který by měly prohledávací moduly akceptovat. V kořenovém adresáři internetové stránky se nachází soubor robots.txt, který obsahuje pravidla zakazující nebo povolující přístup k určitým adresářům na webu. Příkladem může být zakázání přístupu na celou stránku, nebo jen do konkrétní části, například fotogalerie.

2.3.4 Politika paralelního zpracování

V případě, že stahování provádí více agentů, je třeba zajistit koordinaci mezi nimi, aby nedocházelo k přetížení nebo stahování jedné stránky vícekrát.

Koordinace mezi agenty může probíhat několika způsoby:

- Každý agent začíná stahování s různou počáteční množinou webových stránek. K dalšímu stahování stránek dochází úplně nezávisle na ostatních agentech. V takovém případě si nemusí agenti vyměňovat žádné informace o další návštěvě, tudíž dochází k minimální koordinační režii. Na druhou stranu mezi agenty jsou stránky stažené vícekrát, přestože při použití rozdílné počáteční množiny stránek by neměly být databáze stažených stránek totožné. [9]

- Další možností je dynamické přidělování stránek agentům. O to se obvykle stará centrální koordinátor, který logicky rozdělí web do menších částí a následně přerozděluje stránky mezi agenty. Web se může rozdělit na menší části například podle doménového jména stránky. Jednotlivým agentům jsou potom přiřazeny jednotlivé menší části, kdy jednu část vždy stahuje jeden agent. Když při zpracovávání stránky je nalezen odkaz

[3] DDoS (*distributed denial of service*) je útok na webové stránky, jehož cílem je znepřístupnit službu pomocí velkého množství dotazů z mnoha různých zdrojů.

[4] Seznam pravidel uložených v souboru robots.txt, které zakazují prohledávacím modulům přístup k jednotlivým částem stránky.

odkazující do přiřazené části, potom pokračuje agent v jeho stahování, v opačném případě je poslán centrálnímu koordinátorovi, který ho přiřadí jinému agentu. Čím je logické rozdělení webu jemnější, tím rostou režijní náklady na přerozdělování odkazů. [9]

- Pokud dojde k rozdělení webu do menších částí a jejich přiřazení jednotlivým agentům před začátkem stahování, jedná se o statické přidělení. Jelikož každý agent ví, které stránky stahují ostatní agenti, není potřeba centrální koordinace. [9]

2.4 Strategie inkrementálního stahování

Hlavním problémem inkrementálního stahování je udržení konzistentní databáze se zveřejněným obsahem na internetu. Podle aplikace pracující s databází se zvolí strategie, která bude nejlépe vyhovovat její efektivní funkčnosti.

Určení konzistence databáze není jednoduché a mění se v čase v závislosti na vytváření nového obsahu na internetu. Poukázat na nekonzistenci může existence stránek nenacházejících se v databázi, stránky jejichž obsah se od posledního stažení změnil, případně velikost změny, nebo doba od posledního stažení stránky. Tyto informace pomohou určit, kterou stránku je potřeba znovu navštívit a kdy. [18]

Pro vyhledávání je například důležité, aby uživatel dokázal vyhledat pro něj aktuální relevantní údaje. Proto je důležitější častěji navštěvovat stránky, které jsou pravidelně vyhledávané a naopak pro uživatele nezajímavé stránky nemusí být stažené vůbec.

2.4.1 Aktuálnost a stáří stránky

Jedním z nejjednodušších způsobů, jak měřit aktuálnost databáze, je stáří stránky (doba od změny stránky do jejího stažení) a zda se liší obsah stažené stránky v databázi s obsahem stránky na internetu.

Aktuálnost (*freshness*) lokální kopie webové stránky p_i v čase t je definována jako [8]

$$F(p_i, t) = \begin{cases} 1 & \text{pokud } p_i \text{ v databázi je aktuální v čase } t, \\ 0 & \text{v ostatních případech} \end{cases} \tag{2.1}$$

a potom aktuálnost celé databáze S obsahující N stránek: [8]

$$F(S, t) = \frac{1}{N} \sum_{i=1}^{N} F(p_i, t). \tag{2.2}$$

Aktuálnost databáze je prakticky velmi těžké měřit, jelikož by se musely neustále porovnávat všechny stránky stažené v databázi se stránkami na internetu.

Stáří (*age*) lokální kopie webové stránky p_i v čase t lze vyjádřit jako [8]

$$A(p_i, t) = \begin{cases} 0 & \text{pokud } p_i \text{ v databázi je aktuální v čase } t, \\ t - \text{čas změny } p_i & \text{v ostatních případech} \end{cases} \tag{2.3}$$

a stáří celé databáze S v čase t: [8]

$$A(S, t) = \frac{1}{N} \sum_{i=1}^{N} A(p_i, t). \tag{2.4}$$

Když bude databáze s 1000 záznamy obsahovat 600 aktuálních stránek a 400 stránek, jejichž obsah se již neshoduje s obsahem na internetu, potom bude celková aktuálnost databáze 0,6. Podobně pro stáří celé databáze, pokud by se před týdnem změnil obsah všech stránek v databázi, bude stáří celé databáze jeden týden.

2.4.2 Poměr změn stránek

Poměr změn stránek v databázi nám podává informaci o tom, kolik bylo staženo změněných stránek v cyklu. Jedná se o poměr mezi staženými a změněnými stránkami oproti celkovému počtu stažených stránek v jednom cyklu. Poměr se může v každém cyklu výrazně lišit, proto se používá průměr přes všechny cykly stahování. Jelikož některé stránky jsou zajímavější než jiné, přidává se ještě ke každé stránce váha například na základě *PageRank* [21]. Cílem každého cyklu stahování je nalézt co nejvíce změněných stránek.

Poměr změn stránek v jednom cyklu stahování C_i lze definovat jako [10]

$$C_i = \frac{1}{N} \sum_{i=1}^{N} w_i \cdot I_1(p_i), \tag{2.5}$$

kde průměr vah w_i musí být roven jedné a pro funkci I_1 platí: [10]

$$I_1(p_i) = \begin{cases} 1 & \text{pokud se stránka } p_i \text{ změnila,} \\ 0 & \text{v ostatních případech.} \end{cases} \tag{2.6}$$

Například když bylo v jednom cyklu staženo 1000 stránek stejné váhy a z toho 600 se změnilo oproti minulé návštěvě, poměr změn je roven 0,6. Metoda poměru změn stránek se využívá v projektech archivující internet, které si kladou za úkol stáhnout celou historii změn uložených dat z minulých procházení, proto je vhodné v jednom cyklu procházení odhalit co nejvíce změn.

2.4.3 Frekvence změn

Pro udržení aktuální databáze je potřeba rozhodnout, jak často a kolik stránek za určitý časový interval se aktualizuje. Čím častěji budeme stahovat již jednou stažené stránky, tím bude databáze aktuálnější.

Na základě dostupných informací o změnách stránek lze zvolit jednu z následujících možností:

- **Stejné frekvence návštěv:** V tomto případě se aktualizují všechny stránky po stejném intervalu. Tento způsob lze použít za předpokladu, že nevíme, jak často se jednotlivé stránky mění, ale víme, jak často se průměrně mění všechny stránky. Dalším důvodem pro použití tohoto přístupu je případ, kdy se stránky mění s podobnou frekvencí. Při použití této metody s vhodně zvoleným intervalem mezi návštěvami je podle experimentu z dizertační práce pana Junghoo Cho [7] celková aktuálnost a stáří databáze stažených stránek lepší než u použití různých frekvencí návštěv pro každou stránku. Použití stejných intervalů mezi návštěvami pro všechny stránky je tedy vhodné pro aplikace, které kladou větší důraz na udržení co nejvíce aktuální databáze, než na zachycení všech změn nebo minimalizaci zbytečných návštěv. Příkladem mohou být například webové vyhledávače.

- **Různé frekvence návštěv:** V tomto případě se předpokládá, že se stránky mění v různých intervalech. Proto se musí pro každou stránku určovat perioda další návštěvy. Oproti použití stejných frekvencí návštěv pro všechny stránky se zvyšuje počet zachycených změn u často se měnících stránek a snižuje se počet zbytečných návštěv u stránek s dlouhým intervalem mezi změnami. Ve výsledku může být v databázi zachyceno více změn, ale celková aktuálnost a stáří databáze se pravděpodobně zhorší oproti předchozímu způsobu. To je způsobeno právě tím, že zachycení změny u dlouho neměnících se stránek trvá výrazně déle, tudíž stránky zůstávají v databázi po delší dobu neaktuální. Využití různých intervalů mezi návštěvami je vhodné pro aplikace s důrazem na zachycení co nejvíce změn nebo dat, ale nepotřebují mít co nejaktuálnější databázi.

Jako příklad můžeme uvést databázi se třemi webovými stránkami, které se mění jednou, dvakrát a třikrát během dne. Při použití stejných frekvencí změn pro všechny stránky se stránky znovu navštíví s frekvencí $f = 2$. V případě různých frekvencí se každá stránka navštíví s frekvencí $f_1 = 1$, $f_2 = 2$ a $f_3 = 3$.

Metody pro učení frekvence návštěv

Další část kapitoly se bude zabývat určením intervalu mezi jednotlivými návštěvami stránky. To se týká převážně při použití proměnlivých frekvencí návštěv, kdy je potřeba určit interval změn pro každou stránku zvlášť. V případě stejných frekvencí návštěv by se musel interval pro další návštěvu určovat pro všechny stažené stránky společně, což je náročnější a bez většího přínosu. Proto se obvykle používá výchozí interval, který se během stahování nemění.

Pro určení intervalu pro další návštěvu je nejdříve potřeba definovat, co je považované za změnu webové stránky. Nejjednodušším způsobem je vypočtení a porovnání haše (*hash*) celé stránky, kde se projeví jakákoliv změna. V takovém případě může u HTML stránky dojít ke změně struktury stránky, ale obsah zůstane stejný. Dalším kritériem může být třeba změna 20 % stránky nebo změna textového obsahu.

Pro konkrétní využití prohledávacího modulu se musí definovat kritéria pro určování frekvence změn stránky, které mohou být: způsob jakým sledujeme změnu stránek, jaké informace o změnách máme a jak přesně je potřeba určit interval změn.

Změny stránek můžeme sledovat buď aktivně, nebo pasivně po určitém, nebo náhodném časovém intervalu. V případě inkrementálních prohledávacích modulů se jedná o aktivní sledování po určitém intervalu.

Další kritérium je, jaké informace máme o změnách stránek. Můžeme mít úplnou historii změn, při které se jednoduše dá určit frekvence změn stránky podílem počtu změn za určitý časový interval. Jedná se o nejpřesnější určení, ale k těmto informacím máme málokdy přístup, pokud nejsme vlastníky dané stránky. V dalším případě neznáme celou historii změn, ale známe datum poslední změny. Datum poslední změny lze získat z hlavičky protokolu HTTP pomocí pole *Last-Modified*. Nejedná se o povinnou položku a v případě dynamicky generovaných stránek se může obsah pole *Last-Modified* změnit, přitom obsah webu zůstat stejný. V posledním případě máme pouze informace o obsahu stránky při minulé návštěvě a aktuální obsah. Dokážeme tedy stránky porovnat a určit, že došlo ke změně, ale nedokážeme určit počet změn mezi jednotlivými návštěvami nebo datum, kdy k nim došlo.

V různých případech je kladen různý důraz na zaznamenání všech změn dané stránky. Při zpracovávání dat, kdy potřebujeme zachytit změnu co nejdříve, je potřeba odhadnout

interval pro další návštěvu co nejpřesněji. V jiných případech stačí rozdělit stránky do několika kategorií. Například můžeme rozdělit stránky do tří tříd, kdy se stránky v první třídě budou stahovat jednou denně, v druhé jednou týdně a v poslední jednou za půl roku. [7]

2.4.4 Detekce změny stránky

Při vzniku internetu byla většina obsahu statická a docházelo jenom k minimálním změnám. S jeho rozšířením a zpřístupněním běžným uživatelům se změnil také obsah stránek, kdy přibyly zpravodajské webové stránky, webové blogy nebo diskuzní fóra. Webové stránky tohoto typu přidávají často nový obsah a proto přibyla potřeba po efektivním rozpoznání změn, aby měli návštěvníci přehled o všech pro ně zajímavých příspěvcích.

Detekce změn stránek dále využívají inkrementální prohledávací moduly jako zpětnou vazbu, které stránky je potřeba navštěvovat častěji nebo při odstraňování redundantních záznamů HTML stránek v databázi, kdy není potřeba mít uloženou stejnou stránku vícekrát. Tyto dva způsoby využití bychom nalezli v rámci webových vyhledávačů.

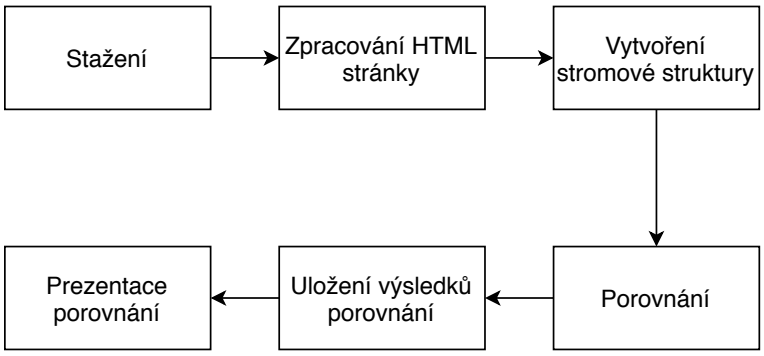
Změna stránky jde nejlépe vyhodnotit na základě:

- změny obsahu (přidání, odebrání části textu, ...),
- změny struktury (přidání, odebrání nebo změna pořadí prvků),
- změny atributů (změna vzhledu jednotlivých prvků).

Pro různé použití může být každý z parametrů jinak důležitý a všechny komplexní systémy by měly podporovat všechny tři zmíněné parametry a zároveň být schopny vyhodnotit, zda se jedná o podstatnou, nebo jenom minimální změnu. Při detekci je potřeba vzít v úvahu několik vlastností, jako například rychlost vyhodnocení, přesnost detekce změny nebo prostorovou náročnost.

Proces detekce změny HTML stránky

Obecně jde detekce změn stránky shrnout do několika kroků: stažení stránky, zpracování HTML značek a textu ze stránky, vytvoření stromové struktury ze získaných HTML značek, porovnání s referenční stránkou a vyhodnocení změny, uložení a prezentace výsledků. [19]



Obrázek 2.1: Proces detekce změny HTML stránky

Stažení stránky a její zpracování je součástí každého webového prohledávacího modulu. Při zpracování je potřeba provést vyčištění od neukončených značek a provést úpravy jako

převedení názvů značek, atributů a textu na malá, případně velká písmena, odstranění číslic, interpunkcí nebo přebytečných mezer.

Ze zpracované stránky je potřeba vytvořit stromovou strukturu, kde jednotlivé uzly reprezentují HTML značky a obsahují informace o jejich atributech a obsahu. Nad vytvořeným stromem se následně provádí porovnání s využitím různých technik a algoritmů založených na hašovací funkci, hodnoty signatury (podpisu) uzlů, hodnoty RMS nebo dalších kódů. Hodnota RMS se získá jako odmocnina součtu mocnin ASCII hodnot všech znaků v textu [23].

Algoritmy založené na hašovací funkci vytváří haš ke každému podstromu. Pokud se haš kořenového uzlu shoduje u obou iterací stahování, tak se stránka nezměnila. V jiném případě jde snadno procházením stromu najít všechny změny. Porovnávání hodnoty haše je oproti původnímu textu jednoduché a velmi rychlé. [5]

Dalším způsobem je porovnávání stromu na základě signatury jednotlivých uzlů. Signatura uzlu se vypočítá například součtem signatur všech jeho potomků, přičemž když uzel reprezentuje textový obsah, vypočítá se signatura na základě hašovací funkce nebo hodnoty RMS. Porovnávání jednotlivých uzlů může být pomalejší, jelikož oproti haši se nemusí jednat o čísla se stejnou délkou, naopak jejich výpočet není tak komplikovaný. [13]

Po určení změny stránky je potřeba uložit sestavený strom a výsledky porovnání na disk. Čím komplexnější porovnání, tím roste nejen časová náročnost porovnání, ale i množství potřebných dat k uložení.

Přístup webových prohledávacích modulů

Porovnávání změny stránky může být výrazně časově náročné, proto se webové prohledávací moduly omezují pouze na jednoduché a rychlé metody porovnávání struktury a obsahu, nebo určují změnu stránky pouze na základě vypočítaného haše celé stránky.

Jedním z dalších zajímavých přístupů je určit změnu stránky na základě frekvence výskytu slov v textu, který by neměl detekovat drobné změny stránky [11]. V tomto případě se odstraní značky HTML a slova, která nesplňují minimální délku (předložky, spojky, ...) nebo se jedná o často se vyskytující slova (tzv. *stop words*). Ke každému slovu se vypočte frekvence výskytu, kdy se odstraní málo frekventované slova, a slova se sestupně seřadí dle frekvence. Pro snadné porovnání a uložení se musí pro danou statistiku slov vypočítat haš. Jedním z možných způsobů je spojení seřazených slov do řetězce, ze kterého se vypočítá haš. Tento způsob, nebo haš obsahu celé stránky, se využívá v systému Nutch (viz kapitola 3.5) pro určení změny a duplicity stránky.

V předchozích způsobech lze zanedbání drobných změn stránky dosáhnout zvolením vhodné hašovací funkce. Příkladem je hašovací funkce *simhash*, která pro podobný vstup vypočítá podobný haš [17]. Zda se stránka změnila se rozhodne na základě *Hammingovy vzdálenosti*[5] mezi dvěma haši. Pokud je *Hammingova vzdálenost* mezi dvěma haši menší jak prahová hodnota, tak se stránka nezměnila. Algoritmus *simhash* využívá webový prohledávací modul od společnosti Google pro nalezení podobných stránek.

2.5 Shrnutí

V rámci kapitoly byl popsán princip webových prohledávacích modulů a jejich klasifikace. V podkapitole o vyhledávací politice byly popsány základní problémy, se kterými se

[5] Hammingova vzdálenost udává minimální počet rozdílných bitů mezi dvěma stejně dlouhými binárními čísly.

webové prohledávací moduly potýkají. Jedná se o určení, které stránky se budou stahovat, kdy se budou znovu navštěvovat, jakým způsobem předejít přetížení serverů, ze kterých se stahuje, a sdílení informací o stahování mezi více agenty.

Důležitou součástí každého inkrementálního webového modulu je inkrementální strategie. Pro její vyhodnocení slouží metriky stáří, aktuálnost stránky a poměr zachycených změn. Plánovat další návštěvu můžeme dvěma způsoby: pro všechny stránky společně, nebo se interval mezi návštěvami vypočítá pro každou stránku zvlášť. V druhém případě hraje velkou roli detekce změny stránky.

Kapitola 3

Systémy pro stahování webu

Ačkoliv jsou inkrementální webové prohledávací moduly jádrem každého internetového vyhledávače, open-source řešení si většinou kladou jiné cíle. Zaměřují se na archivaci internetu, ať v malém měřítku pro běžného uživatele, který si chce prohlížet webové stránky, i když nebude připojen k internetu, tak ve velkém, kdy se archivují stránky v lokální síti nebo celý internet. Jiným možným použitím je stahování a zpracovávání stránek pro další strojové vyhodnocení.

Tato kapitola se bude zabývat postupně systémy určenými pro běžného uživatele, a to přes příklad systému pro stahování strukturovaných dat, až po systémy používaných běžně při archivaci internetu. Posledním systémem popsaným v této kapitole je systém BUbiNG, který je v rámci této bakalářské práce upravován pro potřeby inkrementálního stahování.

3.1 GNU Wget

GNU Wget je balíček pro Linux distribuovaný pod GNU General Public Licence. Jedná se o neinteraktivní nástroj ovládaný z příkazové řádky sloužící ke stahování obsahu webu. Jeho hlavním cílem je archivace jednoho zadaného webu, který lze následně prohlížet offline. [1]

Wget poskytuje možnost rekurzivního stahování, takže se z každé stránky zpracují odkazy, které neodkazují mimo danou doménu, a následně se stáhnou. Díky velké možnosti nastavení lze vybrat do jaké hloubky zanoření se bude web stahovat, omezení pouze na podadresáře a mnoho dalšího.

Díky možnosti aktualizace již staženého webu lze použít GNU Wget pro inkrementální stahování. Při prvním stažení se uloží kromě obsahu stránky také časová značka poslední modifikace z hlavičky souboru. Po nějakém čase stačí znovu spustit GNU Wget v režimu pro aktualizaci, kdy se dotáže serveru na čas poslední modifikace. Na základě času poslední modifikace se určí, zda je lokální kopie stránky aktuální, nebo ji je potřeba opětovně stáhnout.

3.2 HTTrack

HTTrack je offline prohlížeč webových stránek napsaný v jazyce C/C++ pod GPL licencí. HTTrack je dostupný pro operační systémy Windows, Linux, OSX, BSD, UNIX nebo Android.

Po zadání domény HTTrack stáhne rekurzivně všechny složky a soubory ze serveru, uloží je do počítače a následně lze stránky prohlížet offline. Při stahování lze nastavit filtry pro stahování na základě typu, velikosti, názvu souboru nebo úrovni zanoření, URL nebo velikosti celé sítě. Podobně jako GNU Wget, podporuje aktualizaci již jednou stažených stránek a také pokračování v přerušeném stahování. [2]

3.3 Scrapy

Scrapy je *framework* pro extrakci dat z webových stránek pro python a běží na operačních systémech Linux, MS Windows, Mac OS a BSD. Hlavním cílem stahování pomocí Scrapy je stažení strukturovaných dat z webové stránky do určitého formátu, který je vhodný pro prohlížení různými tabulkovými procesory (např. MS Excel, LibreOffice Calc, ...) nebo provádění výpočtů nad staženými daty. *Framework* Scrapy je také vhodný pro zpracování dat z více různých zdrojů, kdy jejich získání nebo zpracování může být náročné.

Scrapy není běžný webový prohledávací modul jako Nutch nebo BUbiNG. Běžné webové prohledávací moduly stahují webové stránky a případně z nich zpracují nějaké informace jako například klíčová slova. Scrapy dokáže zpracovávat pouze data z webových stránek se známou strukturou, takže pokud navštíví stránku s neznámou strukturou, nedokáže vytvořit výstup v požadovaném formátu. Pomocí XPath[1] nebo CSS[2] výrazů se musí popsat, jak se budou zpracovávat data. [16]

Vývoj Scrapy je podporován několika organizacemi a sdružuje se kolem něho aktivní komunita. Z toho důvodu existuje spousta zásuvných modulů, mezi které patří i modul DeltaFetch[3] pro inkrementální stahování. Modul si při každé návštěvě stránky, kterou dokáže zpracovat, ukládá informaci o URL do databáze. Při dalším spuštění se navštíví počáteční množina stránek a z nich extrahované nenavštívené odkazy. Například mějme program, který jednou denně stahuje články ze zpravodajských portálů a z jednotlivých článků zpracovává titulek, autora a obsah. Úvodní stránka obsahující odkazy na jednotlivé články je zařazena do počáteční množiny stránek, a tudíž se vždy navštíví. Odkazy na každý článek se navštíví pouze jednou, jelikož z nich extrahovaná data se obvykle nemění.

3.4 Heritrix

Systém Heritrix začala vyvíjet instituce Internet Archive od roku 2003. Jedná se o open-source projekt distribuovaný pod licencí Apache 2.0 napsaný v Javě. Jeho primárním určením je archivace internetu, kdy každou stránku navštíví pouze jednou. Systém běží pouze na jednom počítači ve více vláknech.

Kvůli svému původnímu účelu použití lze znovu navštívit stránky pouze opětovným spuštěním systému se stejným počátečním vzorkem URL, jako v předchozím procházení. Takové stahování nebude mít žádné informace z předchozích spuštění, proto se stáhnou i stránky, které se nezměnily, a následně se musí vyřešit deduplikace externími nástroji.

Díky své modularitě lze po úpravě použít i na inkrementální stahování. Systém Heritrix používá pro uchovávání URL stránek ke stažení jednoduchou frontu typu FIFO. Pro

[1] XPath je programovací jazyk sloužící k navigaci v XML souborech. Příklad XPath výrazu: //base/@href

[2] CSS je programovací jazyk pro aplikovaní stylů na HTML souborech. Pomocí CSS výrazů lze vybrat určité značky HTML z dokumentu. Příklad CSS výrazu: base::attr(href)

[3] https://github.com/scrapy-plugins/scrapy-deltafetch

inkrementální stahování je potřeba ji upravit na prioritní frontu, aby se stránky určené k opětovnému navštívení, dostaly na začátek fronty v dobu, kdy se mají znovu navštívit.

Také je potřeba přidat rozhodovací proces, zda se stránka od posledního navštívení změnila, případně o jak výraznou změnu se jedná. Rozhodnout o tom lze na základě rozdílu haše uložené stránky a nově stažené. Po stažení stránky a zjištění změn nastává otázka naplánování příští návštěvy. Na základě změn, případně jiných informacích o stránce, se určí čas další návštěvy a vloží se do fronty stránek ke stažení. [20]

3.5 Nutch

Nutch je další z open-source projektů napsaných v Javě, který se zaměřuje na poskytnutí vyhledávání obsahu na internetu. Za tímto účelem podporuje inkrementální stahování, indexaci textu nebo deduplikaci stažené databáze. Hlavním cílem je flexibilita umožňující běh nejen na clusterech, ale také na osobním počítači. Mnoho malých firem, organizací nebo univerzit používá Nutch nebo některý z jeho klonů k vyhledávání na lokálních sítích, kde stačí k běhu pouze jeden server.

Nutch navštěvuje všechny stránky po stejné době, bez ohledu na to, zda se stránka od posledního stažení změnila. Neexistuje tedy žádná zpětná vazba říkající, které stránky by se měly navštěvovat častěji, a které méně často. Uživatel může manuálně nastavit pro každou doménu zvlášť interval, po kterém se má znovu navštívit. Po spuštění deduplikace se najdou duplicitní stránky a ponechá se pouze stránka s nejlepším ohodnocením. V případě stejného ohodnocení se ponechá nejnověji stažená stránka, případně s nejkratším URL. V rámci politiky slušnosti stahuje všechny stránky z jednoho serveru pouze jeden z počítačů, na kterých byl Nutch spuštěn, a zároveň dodržuje omezení cílové stránky, aby nedošlo k přetížení sítě. [14]

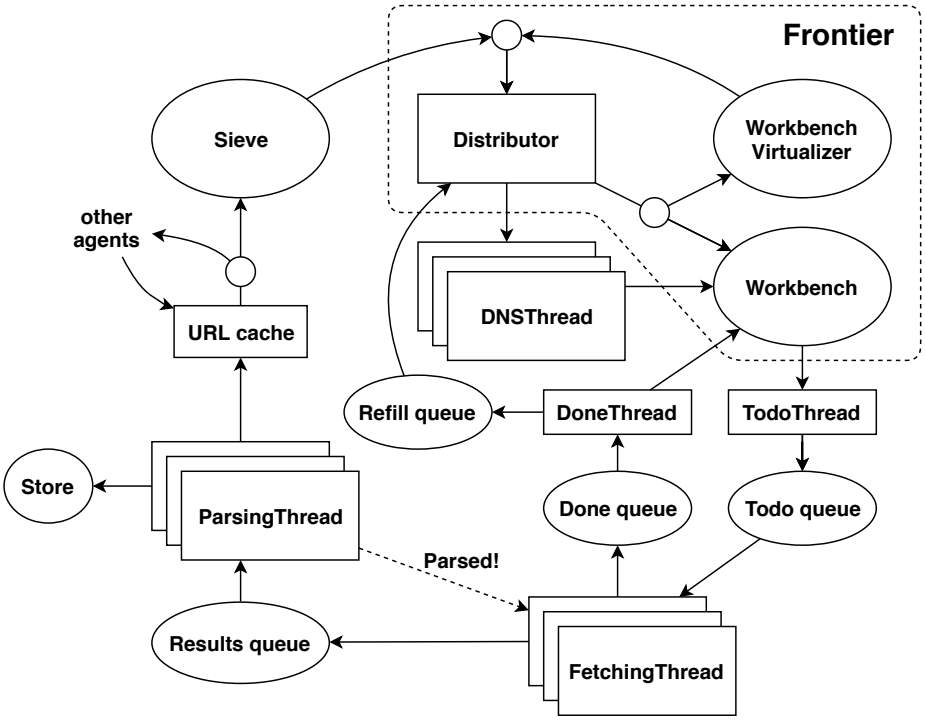
Nutch je používán od roku 2013 v projektu *Common Crawl*[4]. *Common Crawl* je nezisková organizace, která se zabývá stahováním webových stránek a jejich zveřejněním pro použití v jiných projektech. Od roku 2008 obsahují databáze miliardy webových stránek, které se stahují několikrát do roka.

3.6 BUbiNG

Systém BUbiNG byl vybrán pro využití v laboratoři KNOT především z důvodu výrazně vyššího výkonu oproti dříve zkoušeným řešením. Obdobně jako systém Heritrix je jeho hlavním cílem archivace internetu, tudíž nepodporuje inkrementální stahování. Předmětem dalších kapitol budou právě jeho úpravy pro inkrementální stahování.

Systém BUbiNG je open-source webový prohledávací modul vyvinutý na základě zkušeností z projektu UbiCrawler na Laboratoři webových algoritmů Katedry informačních věd Milánské university. Systém BUbiNG je napsán v jazyce Java pod licencí GNU GPLv3 a je plně distribuovaný bez potřeby centrálního řízení. Rychlost stahovaní roste lineárně s počtem agentů, kde na stanici s 64 jádrovým procesorem a 64 GB RAM dokáže stáhnout více jak 9000 stránek za sekundu. [6]

[4] http://commoncrawl.org/



Obrázek 3.1: Architektura systému BUbiNG [6]

3.6.1 Srovnání

Nejdříve srovnáme systém BUbiNG se systémy používanými běžnými uživateli, jejichž zástupci jsou Wget a HTTrack. Tyto systémy jsou navrženy, aby byly schopné fungovat i na méně výkonných strojích a jsou použitelné pro stažení maximálně pár tisíců odkazů. Oproti systému BUbiNG také běží pouze v jednom vlákně a na jednom stroji.

Dalším zástupcem je systém Scrapy, který není webový prohledávací modul v pravém slova smyslu. Jedná se o framework pro jazyk Python a zaměřuje se na zpracovávání strukturovaných dat. Výsledkem stahování Scrapy je databáze dat zpracovaných ze stažených stránek a proto se špatně porovnává se systémy BUbiNG, Heritrix, nebo Nutch, jejichž jsou místo databáze stažených stránek [16].

Systém Heritrix je už svým určením velmi blízký systému BUbiNG. V systému Heritrix byl poprvé použit formát WARC pro ukládání stažených stránek, který se následně stal standardem ISO a BUbiNG jej používá také. Podle provedených experimentů dosahuje BUbiNG několikanásobně větší rychlosti stahování než zmíněný Heritrix [6].

Nutch je považován za nejlepší open-source webový prohledávací modul a oproti systému BUbiNG je jeho cílem poskytnout plnohodnotné vyhledávání na internetu. K tomuto účelu podporuje kromě inkrementálního stahování taky indexaci textu nebo deduplikaci. Pro rozdělování práce mezi agenty využívá framework Apache Hadoop, který není tak účinný jako rozdělování práce v systému BUbiNG. Ve srovnání rychlosti stahování je Nutch pomalejší než BUbiNG, i než Heritrix [6].

3.6.2 Architektura

Systém BUbiNG se skládá z několika hlavních částí: *sieve*, *frontier*, *dnsThread*, *fetchingTread* a *parsingThread*, které jsou navzájem propojeny.

Sieve, jak už název napovídá, slouží jako „síto“, kterým projde každé URL pouze jednou. Následně se *Frontier* stará o dodržování politiky slušnosti, kdy seskupí URL na základě schématu[5] a doménového jména do záznamu o návštěvě a ke každému záznamu zjistí IP adresu. Na základě nastavených omezení pro přístup k doméně a IP adrese se předá záznam o návštěvě do jednoho z *fetchingTread*, které z tohoto záznamu stáhnou jednu URL. Stažená stránka je zpracována součástí *parsingThread* a nově získané odkazy se přerozdělí mezi ostatní agenty a přes *sieve* pokračují ve stahování. [6]

Sieve

Sieve je fronta s pamětí obsahující odkazy, které se mají teprve navštívit, a zároveň si uchovává informace o již navštívených odkazech. Pokud je odkaz vložen do *sieve* vícekrát, bude z ní odebrán pouze jednou. Tím je zaručeno, že se nebude jeden odkaz stahovat vícekrát. [6]

Frontier

Odkazy jsou ze *sieve* přesunuty do části nazvané *frontier*, která řídí další jejich zpracování. *Frontier* se skládá z jedné řídící komponenty *distributor* a dvou datových struktur *workbench* a *workbench virtualizer*.

Distributor běží jako vlákno s vysokou prioritou a řídí zpracovávání odkazů. *Distributor* nejdříve vybere odkaz ze *sieve*, a když už někdy v minulosti přistupoval k dané doméně, přidá odkaz k záznamu o návštěvě, který se nachází v komponentě *workbench*. V případě přeplnění *workbench* bude odkaz pomocí *workbench virtualizer* uložen na disk (viz níže). Záznam o návštěvě shlukuje odkazy pro danou doménu, které se mají navštívit, čas příští návštěvy a soubor robots.txt s časem, kdy se má aktualizovat. V případě, že daná doména ještě nebyla nikdy navštívena nebo se v komponentě *workbench* nenachází záznam, vytvoří *distributor* nový záznam a předá ho *dnsThread* pro získání adresy IP. *DnsThread* pak po určení adresy domény předá záznam o návštěvě do *workbench*.

Workbench uchovává záznamy o návštěvách a na základě politiky zdvořilosti určuje, kdy se může k dané stránce přistoupit. Jelikož může *workbench* obsahovat i několik stovek tisíc záznamů, musí být některé záznamy zapsány na disk, aby nedošlo místo v paměti. K tomu slouží *workbench virtualizer*. *Distributor* musí rozhodnout, které záznamy se zpracují, které se pomocí *workbench virtualizer* uloží na disk a které odkazy se přesunou z *workbench virtualizer* do *workbench*.

Vlákno *todoThread* v cyklu kontroluje *workbench*, zda lze již přistoupit k některému ze záznamů, a následně je záznam z *workbench* přesunut do fronty *todo*. Pro dosažení co nejefektivnějšího stahování by fronta neměla být nikdy prázdná, což ovlivňuje zvolená politika slušnosti. [6]

FetchingThread

FetchingThread je vlákno s nízkou prioritou sloužící ke stažení webové stránky, který většinu času stráví čekáním na příchozí data, proto jich je často několik tisíc. Po stažení stránky je na základě politiky slušnosti určen čas příštího přístupu k záznamu o návštěvě a ten je přes frontu *done* vrácen opět do komponenty *workbench*. [6]

[5] Schéma je část URI určující význam a syntaxi pro zbytek URI. Příkladem schématu může být http: nebo https: (viz https://en.wikipedia.org/wiki/Uniform_Resource_Identifier).

ParsingThread

FetchingThread je pomocí fronty *results* spojen s komponentou *ParsingThread*, pro kterou zpracovává obsah webové stránky. Počet vláken, ve kterých běží *ParsingThread*, je obvykle stejný jako počet jader procesoru, jelikož se jedná o proces náročný na CPU. *ParsingThread* zpracovává obsah webové stránky a získává nové odkazy, které přes *URL cache* předá do *sieve*. Hlavním cílem *URL cache* je zabránit vkládání často se vyskytujících odkazů do *sieve* a zároveň předejít jejich přidělení jinému agentovi. Nakonec je stažená stránka uložena na disk ve formátu WARC. [6]

Kapitola 4

Implementace

Tato kapitola se zabývá návrhem jednoduchého způsobu určování duplicity textu stažených stránek a jeho zapojení do sytému BUbiNG a potřebnými úpravami systému BUbiNG pro inkrementální stahování.

Systém BUbiNG sám o sobě umožňuje určit, zda je stránka duplicitní, ale s různými omezeními. Jedním z omezení je porovnávání stránek pouze s lokální databází, tudíž nesdílí informace s ostatními spuštěnými agenty. Dále se duplicita vyhodnocuje nad stránkou jako celkem, takže není oddělena struktura stránky od jejího obsahu. Pro potřeby výzkumné skupiny KNOT je potřeba vyhodnotit duplicitu stránky na základě jejího obsahu, kdy by byly zanedbány drobné změny například v datu, reklamě, struktuře nebo vzhledu stránky.

Jelikož původně byl systém BUbiNG určen pro archivaci internetu, kdy není potřeba navštěvovat stránky vícekrát, musely být provedeny změny, které by umožňovaly inkrementální stahování. Tyto změny se týkají uchovávání potřebných informací o další návštěvě pro každou stránku zvlášť, jejich uložení na disk, aby se uvolnilo místo v paměti, odhadnutí příští návštěvy stránky a implementace inkrementální strategie.

4.1 Vyhodnocení duplicity textu

Duplicitní a podobné webové stránky jsou jedním z důvodů rychle rostoucí velikosti obsahu internetu. Proč je tolik stránek podobných může mít několik důvodů: typografické chyby v textu, různé verze stránky, stránka s více různými URL, plagiátorství nebo automaticky generovaná struktura stránky. Vyhodnocení podobnosti stránek využívají webové vyhledávače pro zlepšení výsledků vyhledávání, webové prohledávací moduly pro efektivnější navštěvování stránek nebo aplikace, které například odhalují plagiátorství.

Detekce podobnosti webových stránek je časově náročná, jelikož se obvykle porovnávají velké databáze obsahující několik miliard záznamů. Dalším problémem je vyhodnotit, co je stejná, nebo podobná stránka. Nejjednodušším způsobem je porovnání dvojic stránek, kdy se obsah jedné stránky porovná s obsahem druhé. Takto jsme schopni porovnat, zda je stránka stejná, ale při sebemenší změně stránky se stránky vyhodnotí jako různé. Algoritmy vyhodnocující podobnost stránky jsou proto založené na porovnávání jednotlivých částí stránky a podle počtu nalezených shod se vyhodnocuje podobnost.

Pro vyhodnocení duplicity textu bylo zadáním určeno využít systém pro deduplikaci textu výzkumné skupiny KNOT. Systém zpracovává soubor obsahující odstavce z jednotlivých dokumentů ve vertikalizované podobě (viz kapitola 4.1.2) a vytvoří nový soubor

obsahující pouze unikátní odstavce. Na základě principu systému pro deduplikaci byl navržen způsob pro určování duplicity textů na základě počtu duplicitních odstavců.

4.1.1 Návrh vyhodnocení duplicity textu

Na základě principu systému pro deduplikaci výzkumné skupiny KNOT byl navržen způsob určování duplicity textu. Systém zpracovává dokumenty složené z odstavců textu ve vertikalizované podobě a odstraňuje z nich duplicitní odstavce, přičemž neprovádí deduplikaci kratších odstavců než 50 znaků.

Mějme webovou stránku P, která je složena z N odstavců p_i. Potom jde míra duplicity stránky vypočítat jako vážený průměr $\overline{D}$.

$$\overline{D} = \frac{\sum_{i=1}^{N} w_i \cdot I_1(p_i)}{\sum_{i=1}^{N} w_i} \tag{4.1}$$

Každému odstavci je přiřazena váha w_i (vztah 4.2) a funkce I_1 určuje, zda je odstavec duplicitní (vztah 4.3). Výsledná hodnota je v rozsahu $\overline{D} \in \langle 0, 1 \rangle$, kde při $\overline{D} = 0$ není žádný odstavec duplicitní a při $\overline{D} = 1$ jsou duplicitní všechny odstavce.

$$w_i = \begin{cases} |p_i] & |p_i| \geq 50 \\ 0 & |p_i| < 50 \end{cases} \tag{4.2}$$

Jednotlivé odstavce jsou rozděleny do dvou tříd podle své délky. Pokud je odstavec kratší než 50 znaků, je zařazen do třídy *krátkých odstavců*. Takové odstavce jsou zahozeny, jelikož se většinou jedná o položky menu nebo různé odkazy, které se nachází na více stránkách a tak by negativně ovlivňovaly vyhodnocení duplicity. Příklad může být položka v menu *Domů*, která odkazuje na úvodní stránku webové stránky. Odstavec *Domů* se nachází na velkém počtu různých stránek a je nepravděpodobné, že by se jeho obsah při novém navštívení stránky změnil. Proto ovlivňuje detekci duplicity stránky.

Druhou třídou je třída *dlouhých odstavců*. Do této třídy jsou zařazeny všechny odstavce delší než 50 znaků a váha odstavce se odvíjí od jeho délky.

Pro vyhodnocení duplicity odstavce se využívá systém pro deduplikaci textů výzkumné skupiny KNOT. Odstavec projde nejdřív normalizací, která zahrnuje převedení všech písmen na malé, bílých znaků na mezery, odstranění přebytečných mezer a všech znaků kromě písmen, číslic.

Následně je vypočítán haš takto normalizovaného odstavce, který je odeslán na server pro vyhodnocení duplicity.

$$I_1(p) = \begin{cases} 1 & \text{pokud je odstavec } p \text{ duplicitní} \\ 0 & \text{v ostatních případech} \end{cases} \tag{4.3}$$

Na základě konkrétního případu použití je potřeba určit prahová hodnota $\overline{D}$, kdy bude stránka považována za duplicitní. Tři základní případy jsou: odhalení plagiátů, podobných textů a změny v textu. V případě detekce změny textu by měla být hranice co nejvyšší, aby se zachytilo co nejvíce změn, naopak u rozpoznávání plagiátů je hranice výrazně nižší. Hodnota prahu pro podobné texty se nachází mezi těmito dvěma hodnotami.

Jako výchozí prahová hodnota byla zvolena hodnota $0,9$. Pro konkrétní použití jde následně zvolit jiná vhodná hodnota pomocí konfiguračního souboru systému BUbiNG.

4.1.2 Systém pro odstraňování duplicit z textů (corpproc_dedup)

Odstraňování duplicit z textů je jedním z kroků při zpracování a indexování rozsáhlých textových korpusů. K tomuto účelu byl vyvinut systém pro deduplikaci v rámci Výzkumné skupiny znalostních technologií (KNOT). Jedná se o distribuovaný systém implementovaný v jazyce C/C++, složený z klientů zpracovávajících soubory a databázových serverů.

Jelikož je systém BUbiNG napsán v jazyce Java, bylo nutné navrhnout řešení jak tyto dva systémy spojit. Z několika možných řešení byla nakonec vybrána reimplementace klienta systému pro deduplikaci výzkumné skupiny KNOT v jazyce Java a ponechání původní serverové části. V této části kapitoly bude popsána činnost serveru a původního klienta. V další části rozebereme možnosti pro spojení tohoto systému se systémem BUbiNG a podrobnosti implementace klienta v jazyce Java.

Server

Server obsahuje databázi uložených odstavců, které prošly deduplikací, ve formátu 64 bitového haše. Když dojde dotaz od klienta na duplicitu poslaného haše, server se podívá do databáze a pokud se v ní daný haš nachází, odpoví klientovi, že je odstavec duplicitní. Pokud se haš v databázi nenachází, pak je do ní vložen a klient dostane odpověď, že daný haš není duplicitní. Databáze může být při spuštění serveru načtena ze souboru a v případě jeho korektního ukončení také uložena.

Pro případ selhání serveru je také generován žurnálový soubor, který obsahuje ještě neuložené haše na disk. Při dalším spuštění serveru lze ze žurnálového souboru a zadaného vstupního souboru hašů vytvořit persistentní databázi všech hašů.

Jelikož se jedná o distribuovaný systém, každý server má uloženo ve své databázi jen určitou část hašů. Haše jsou rozděleny do bloků (výchozí počet bloků je 1999), které jsou před prvním spuštěním rovnoměrně rozděleny mezi servery. V případě vygenerování nového rozložení bloků hašů mezi servery, například kvůli optimalizaci, přidání nebo odebrání serveru, distribuovaný systém sám zajistí redistribuci uložených hašů podle nového rozložení.

Klient

Klient běží paralelně v několika vláknech a zpracovává vstupní vertikalizované soubory. Vertikalizovaný soubor je vytvořen ze stažených stránek ve formátu HTML, které jsou zbaveny HTML značek a dochází k normalizaci textu, kdy je každé slovo na samostatném řádku. Sama struktura vertikalizovaného souboru je podobná formátu XML, kdy záznam reprezentuje značka dokumentu $<\mathrm{doc}>$ a jednotlivé odstavce, neboli obsah mezi úvodní a koncovou značkou HTML, značka $<\mathrm{p}>$.

Při zpracovávání se nejdříve ověří duplicita URL a titulek stránky, které jsou uloženy jako atributy značky $<\mathrm{doc}>$. Pokud jsou unikátní, celý obsah záznamu se uloží do výstupního souboru a pokračuje se dalším dokumentem. Pokud by URL a titulní strana již byla uložena na serveru, spočítá se haš celého dokumentu a odešle se na server. Když bude celý dokument duplicitní, není už potřeba porovnávat jednotlivé odstavce, dokument je vyřazen a pokračuje se dalším. V druhém případě dochází k deduplikaci jednotlivých odstavců.

Odstavce jsou podle své délky klasifikované do dvou tříd. Třída *krátkých* odstavců obsahuje odstavce kratší 50 znaků a třída *dlouhých* odstavců ostatní odstavce. Jelikož krátké odstavce mohou být stejné na mnoha stránkách, nedochází k jejich deduplikaci (deduplikátor je ignoruje). V případě dlouhých odstavců se postupuje obdobně jako u celých dokumentů,

vypočítá se haš odstavce a odešle se na server. Pokud je odstavec duplicitní tak se zahodí, jinak se uloží do výstupního souboru.

Výsledkem zpracování je soubor *.dedup, který obsahuje pouze unikátní odstavce.

4.1.3 Spojení se systémem BUbiNG

Hlavní překážkou při spojení nástroje pro deduplikaci výzkumné skupiny KNOT se systémem BUbiNG je jejich implementace v různých programovacích jazycích. Spojit systémy lze několika způsoby, které jsou popsány ve zbytku kapitoly, kdy byla zvolena možnost reimplementace nového klienta v jazyce Java.

Vytvoření rozhraní JNI

První z možností pro spojení klienta se systémem BUbiNG je napsání rozhraní JNI (*Java Native Interface*). Rozhraní JNI slouží pro připojení nativních programů nebo knihoven napsaných v jazyku C/C++, jazyku symbolických adres, atd. s virtuálním strojem Java (dále JVM). Výsledkem je sestavená dynamická knihovna pro určitý hardware, která je následně volána z programů napsaných v jazyce Java. Rozhraní JNI může být použito nejen na spojení jazyka Java s nativními programy, ale také z jiných jazyků, jejichž interprety překládají skripty do *bytecode* a spouštějí ho pomocí JVM. Příkladem může být Jython, který je implementací interpretu jazyka Python v jazyce Java.

Tento přístup se hned potýká s několika problémy. Příprava rozhraní JNI pro klienta velmi znesnadňuje jeho nemodulárnost a použití velkého počtu globálních proměnných. Dalším problémem je formát vstupních dat. Každá stránka by musela projít „vertikalizátorem“, který by převedl HTML stránku na vertikalizovaný soubor. To by znamenalo použít další nástroj, který je napsaný v jazyce Python a spojit ho se systémem BUbiNG pomocí Jython. Takové řešení by ve výsledku bylo složité na spojení, nepřenositelné mezi stroji s různým hardwarem, při dalším vývoji jazyka Java by mohlo dojít k nekompatibilitě rozhraní mezi jazykem Python a jazykem Java. Nakonec by takové řešení bylo neefektivní, jelikož by se HTML stránka nejdříve zpracovala jedním nástrojem do vertikalizovaného formátu, který by následně druhý nástroj znovu zpracovával.

Tady se nabízí další možnost, a to využití původního klienta a napsání „vertikalizátoru“ v jazyce Java. Toto řešení znovu naráží na způsob implementace klienta a problémy s implementací rozhraní a také na mezistupeň převedení získaných odstavců z HTML stránky do vertikalizovaného formátu.

Reimplementace klienta v jazyce Java

Z dříve zmíněných důvodů byla zvolena reimplementace klienta čistě v jazyce Java. Tím odpadá nutnost využití externího „vertikalizátoru“ a s tím spojené problémy s implementací, jelikož BUbiNG obsahuje modul pro zpracování HTML stránek a získání jednotlivých odstavců textu. Výhodou implementace klienta v jazyce Java je absence nativního kódu, který by byl sestaven pro určitý hardware a zachovává tedy původní cíl přenositelnosti systému BUbiNG.

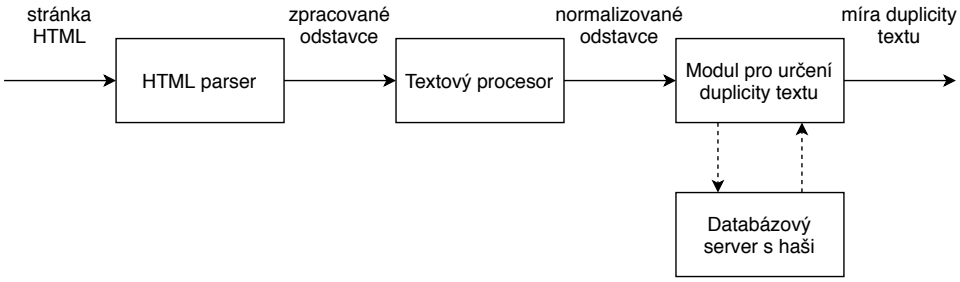
Zbytek kapitoly se bude dále zabývat podrobnostmi implementace klienta v jazyce Java a jeho spojením se systémem BUbiNG.

Klient se skládá z několika částí: zpracování souboru s rozložením bloků, odeslání haše a určení duplicity celého dokumentu.

Před spuštěním systému BUbiNG s námi navrženou detekcí duplicitních stránek je potřeba vygenerovat pomocí nástrojů deduplikátoru soubor s rozložením hašů. Tento soubor slouží k určení se kterými servery se má komunikovat a jaké haše se na ně mají posílat. V případě přidání nebo odebrání serveru je vždy potřeba vytvořit nový soubor s rozložením.

Soubor s rozložením bloků hašů lze zadat v konfiguračním souboru systému BUbiNG. Výchozím souborem je hashmap.conf uložený v pracovním adresáři. Jedná se o deserializaci C++ objektu `std::map<unsigned int, string>` za využití knihovny *boost*. Jelikož není serializace objektů jazyků C++ a Java kompatibilní, byla implementovaná nová metoda, která načte a zpracuje soubor s rozložením a vytvoří mapu, podle které se následně bude dotazovat serverů.

Při odesílání na server se nejdříve musí rozhodnout do jakého bloku haš patří. Zbytek po dělení hodnoty haše počtem bloků určuje index bloku, do kterého haš patří. Následně lze vybrat server, na který se odešle haš. Pokud dojde při odesílání k chybě, pokusí se klient znovu připojit k serveru a v případě opakovaného selhání připojení vrátí chybu. K selhání odeslání haše na server nedochází velmi často a nemá to výrazný vliv na rychlost stahování.



Obrázek 4.1: Proces vyhodnocení duplicity webové stránky

Prvním krokem pro výpočet míry duplicity stránky je získání odstavců textu (viz obrázek 4.1). Pro zpracovávání HTML stránek se využívá výchozí implementace HTML parseru v systému BUbiNG, kterému je předán jako parametr textový procesor. Parser předává textovému procesoru obsah mezi uvozovací a koncovou značkou k dalšímu zpracování. V momentální verzi systému BUbiNG je jediné využití textových procesorů pro detekci domén se stejným obsahem, přičemž výsledky detekce nejsou zatím nijak využity. Pro účely zjištění, zda se stránka změnila, byl vytvořen textový procesor, který jednotlivé odstavce normalizuje a ukládá. Po ukončení zpracovávání stránky je výsledná množina odstavců z textového procesoru předána klientovi.

BUbiNG vyhodnocuje duplicitu stránek na základě porovnání haše celé stránky. Pro optimalizaci využíváme interní porovnávání haše celé stránky s lokální databází. Toto porovnání využívá sám systém BUbiNG pro určení duplicity stránky a výsledek porovnání ukládá se staženou stránkou na disk. Toto porovnání neodhalí duplicitu stránek napříč všemi spuštěnými agenty. Pokud už je haš celé stránky duplicitní, nemá smysl porovnávat jednotlivé odstavce. Haš celé stránky se také použije pro určení duplicity v případě, kdy stránka neobsahuje žádný text, nebo se jedná o binární soubor a funkce pro výpočet duplicity vrací hodnotu `NaN`.

Když haš celé stránky není duplicitní, porovná se haš celého obsahu a pokud je duplicitní, již se neporovnávají jednotlivé odstavce. Pokud by celý obsah nebyl duplicitní a stránka obsahuje alespoň jeden odstavec ve třídě *dlouhých odstavců*, dojde k výpočtu duplicity stránky. Pokud je výsledek vyšší něž stanovená prahová hodnota, tak je stránka považována za duplicitní.

Zda se bude určovat duplicita textu lze nastavit v konfiguračním souboru systému BUbiNG, také s umístněním souboru s rozložením bloků hašů, portem serverů a hranicí pro určení duplicity textu. Soubor s rozložením bloků hašů musí být přítomný na všech spuštěných agentech.

4.2 Inkrementální stahování

Systém BUbiNG si klade za cíl archivaci internetu, kdy má dosahovat výrazně rychlejšího stahování než jiné open-source projekty. Při jeho přizpůsobení k inkrementálnímu stahování se musí vyřešit několik problémů, které kvůli svému zaměření BUbiNG sám neřeší, především s ohledem na co nejmenší zpomalení stahování.

Systém BUbiNG si neuchovává žádné informace o návštěvě jednotlivých odkazů. Z toho důvodu je potřeba navrhnout strukturu pro uchování informací o poslední návštěvě a plánu další návštěvy. Po stažení stránky je potřeba navrhnout způsob plánování další návštěvy a uložení informací o stažené stránce na disk, aby se uvolnilo místo v paměti pro další odkazy. V neposlední řadě musí být zajištěno, aby se stránka znovu navštívila podle plánu.

4.2.1 Struktura pro uchování informací o návštěvě stránky

Jednotlivé URL jsou v systému BUbiNG seskupeny do třídy `VisitState` na základě schématu[1] (http:, https:, ...) a doménového jména (např. example.com). Takové dělení na menší celky je výhodné, jelikož URL sdílí podobné vlastnosti, jako například již zmíněné schéma a doménové jméno a s tím spojenou IP adresu. Uchovávání těchto informací na jednom místě pomáhá snadnému dodržování politiky slušnosti a zmenšuje využité místo v paměti. Třída obsahuje frontu s cestou[2] a dotazem[3] jednotlivých URL uložených ve formě pole bytů.

Například mějme URL „http://example.com/“ a „http://example.com/index.html“. Třída `VisitState` si uloží informaci o schématu a doménovém jméně „http://example.com“ a fronta bude obsahovat dvě položky: „/“ a „/index.html“.

```
public class PathQueryState implements Delayed, Serializable {
    ...
    public byte[] pathQuery;
    public volatile long modified;
    public volatile long nextFetch;
    public volatile long fetchInterval;
    public volatile VisitState visitState;
    ...
    public int memoryUsage() { ... }
}
```

Výpis 4.1: Navržená struktura pro uchování informací o návštěvě stránky

Jelikož původní cíl systému BUbiNG nepředpokládá, že by stránky navštěvoval vícekrát, nepotřebuje si k jednotlivým URL uchovávat žádné další informace. V případě inkrementálního stahování přibývá nutnost plánování další návštěvy nejen podle politiky slušnosti,

[1] Schéma (*schema*) je část URI určující význam a syntaxi pro zbytek URI.
[2] Cesta (*path*) je hierarchickou částí URI, ke které jde přiřadit určitá cesta v souborovém systému.
[3] Dotaz (*query*) je nehierarchickou část URI, sloužící k bližšímu určení zdroje.

Typ	Místo v paměti [byte]
boolean	1
byte	1
char	2
short	2
int	4
float	4
long	8
double	8
reference na objekt	4[4] nebo 8[5]
hlavička objektu	8

Tabulka 4.1: Přehled využití místa v paměti primitivními datovými typy [3]

kdy postačuje naplánování návštěvy podle doménového jména a IP adresy, ale také na základě politiky opětovných návštěv, která se vztahuje ke každému URL zvlášť. Proto bylo uchovávání cesty a dotazu pouze v poli bytů nahrazeno novou třídou `PathQueryState` (viz výpis 4.1).

Třída musí obsahovat minimálně atributy `pathQuery`, `nextFetch` a `visitState`. V atributu `pathQuery` je uložena cesta a dotaz URL a atribut `nextFetch` obsahuje čas naplánované příští návštěvy v milisekundách. Atribut `visitState` pak obsahuje referenci na instanci třídy `VisitState`, ke které daná URL patří. Přes to, že informace o `VisitState` je obvykle k dispozici, tak při ukládání na disk k dispozici nebývá. Po ukončení stahování a zpracování stažené stránky, nemůže být `PathQueryState` vrácen do `VisitState`, který obsahuje pouze připravené URL ke stažení, a místo toho je vložen do fronty pro uložení na disk. Při ukládání na disk slouží atribut ke správné identifikaci, ke které instanci třídy `VisitState` daná instance `PathQueryState` patří.

Další dva atributy `modified` a `fetchInterval` uchovávají informace o poslední detekované změně stránky a vypočtenému intervalu mezi návštěvami. Tyto atributy jsou využity v rámci inkrementální strategie, která určuje interval mezi návštěvami pro každou stránku zvlášť.

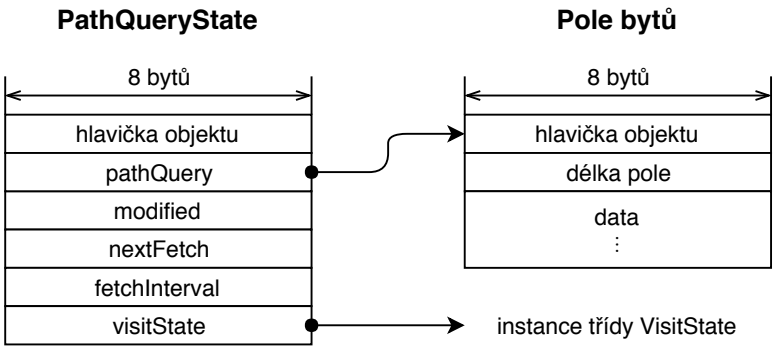
Velikost struktury v paměti

Velikost všech URL v paměti připravených ke stažení je konfiguračním souborem omezena, proto je potřeba odhadnout, kolik místa v paměti zabírá jedno URL. K tomuto účelu byla implementovaná metoda `memoryUsage`.

I když mají primitivní datové typy v jazyce Java pevnou velikost, je velikost instance třídy závislá na platformě a konkrétní implementaci JVM, proto je možné velikost pouze odhadnout. Odvození popsané v následujícím textu by mělo platit pro 64-bitové verze JVM na procesorech Intel.

Na obrázku 4.2 je znázorněno využití operační paměti instancí třídy `PathQueryState`. Každá instance třídy obsahuje hlavičku, která podle tabulky 4.1 má velikost 8 bytů [3]. Třída se skládá ze tří atributů s primitivním typem long o velikosti 8 bytů a dvou referencí na objekt, které na 64-bitové architektuře mají velikost 8 bytů. Když sečteme velikost všech atributů a hlavičky, dostaneme velikost 48 bytů.

[4] Velikost na 32-bitové architektuře dle [3]
[5] Velikost na 64-bitové architektuře dle nástroje *Classmexer*



Obrázek 4.2: Detail instance třídy `PathQueryState`

Dále je potřeba určit vztah pro výpočet velikosti pole bytů. Pole bytů se skládá z hlavičky o délce 8 bytů, atribut s informací o délce pole zabírající 8 bytů a dále za každou položku po jednom bytu (viz obrázek 4.2). Velikost všech objektů je obvykle zarovnána na násobek 8, proto prázdné pole bude mít velikost 16 bytů a pole s nejkratší cestou „/“ bude mít velikost 24 bytů. Ve výsledku odhad velikosti instance třídy `PathQueryState` s nejkratší cestou je 72 bytů, což je oproti původním 24 bytům třikrát více. Tento odhad byl porovnán s výstupy nástroje *Classmexer*[6], který slouží k měření, kolik zabírají objekty v jazyce Java místa v operační paměti.

4.2.2 Uložení informací o další návštěvě na disk

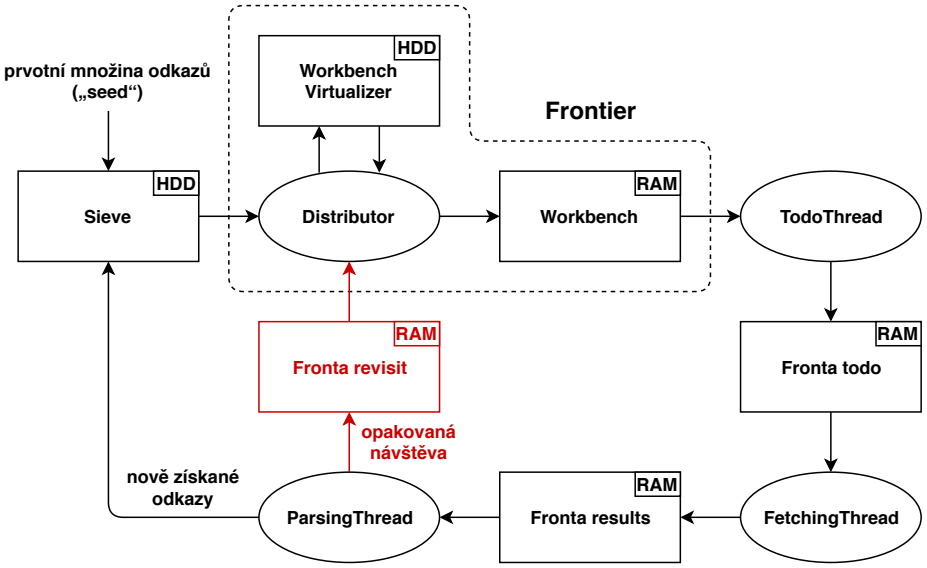
Po stažení stránky a naplánování další návštěvy je potřeba uložit informace o URL na disk a tím uvolnit místo v paměti. V systému BUbiNG existují dvě datové struktury ukládající URL na disk, které by k tomuto účelu bylo možné využít, a to *sieve* a *workbench virtualizer* (viz obrázek 4.3).

První možností je využití struktury *sieve*, která by se dala přirovnat k frontě obsahující jednotlivá URL a paměť s uloženými haši, které využívá při porovnávání jednotlivých URL, aby se nestahovaly vícekrát. Strukturu by bylo možné upravit, aby se kromě URL uchovávaly také s ním spojené informace o další návštěvě a odstraňovaly se pouze nově zpracované duplicitní URL. Problémem by bylo zaručit, že se stránka stáhne podle plánu. Jelikož se v *sieve* pracuje s URL v pořadí, ve kterém byly do něj vloženy, tedy když přijde na řadu URL, která ještě nikdy nebyla stažena nebo již může být stažena znovu, předala by se dál ke stažení. Pokud by ještě nenastal čas ke stažení, musela by se vložit zpět na konec fronty a doufat, že na ní v budoucnu přijde řada. Takový způsob by byl velmi neefektivní a proto byla vybrána pro další úpravy struktura *workbench virtualizer*.

Úprava datové struktury workbench virtualizer

Workbench virtualizer se dá přirovnat k asociativnímu poli, kde jako klíč slouží třída `VisitState`, tedy schéma a doménové jméno, a položkou pole je fronta URL uložených jako pole bytů. *Workbench virtualizer* původně slouží k uložení připravených URL ke stažení na disk, pokud není v komponentě *workbench* místo. Takové uspořádání dovoluje pracovat s menší částí uložených URL, jejichž počet je obvykle omezený konfiguračním souborem na několik tisíc.

[6] https://www.javamex.com/classmexer/



Obrázek 4.3: Životní cyklus URL a provedené úpravy v architektuře systému BUbiNG pro inkrementální stahování (červeně zvýrazněné)

Aby bylo možné uložit instanci třídy `PathQueryState` na disk, je potřebné ji převést na pole bytů. V každé z front se nachází vždy URL se stejným schématem a doménovým jménem, tudíž není potřeba ukládat atribut `visitState`. Výsledná posloupnost bytů je složena z naplánovaného času příští návštěvy, času poslední detekované změny, intervalu mezi návštěvami a částmi URL cestou a dotazem. Ve výsledku jedno URL zabírá na disku součet 24 bytů a délky cesty s dotazem.

Vybírání probíhá v blocích vždy pro jednu konkrétní instanci `VisitState`. Jednotlivé URL jsou uloženy na disku v pořadí, ve kterém byly vloženy do fronty, proto se musí načíst do paměti celá fronta pro danou instanci `VisitState` a seřadit podle plánovaného času návštěvy, přičemž ještě nikdy nenavštívené URL mají naplánovanou návštěvu na čas, kdy byly vybrány se *sieve*. Po seřazení pokračují připravené URL dále v cyklu stahování a ostatní se uloží zpátky na disk. Aby se minimalizoval počet přístupů na disk, uchovává se informace o nejbližším naplánovaném čase návštěvy ke každému `VisitState` uloženému na disku a vybírání proběhne jenom tehdy, když je ve frontě alespoň jedno URL připravené ke stažení. K jednotlivým frontám se obvykle přistupuje jednou za pět minut a provedené změny nemají výrazný vliv na výkon.

Uložení a zajištění opětovného navštívení odkazů

Ke struktuře *workbench virtualizer* může přistupovat pouze jedno vlákno, aby nedošlo k nekonzistenci uložených dat, ale stahování a zpracování stránek probíhá paralelně v několika vláknech. Z toho důvodu byla přidána nová fronta pro uložení informací o další návštěvě na disk, do které se vloží jednotlivé instance třídy `PathQueryState` a o ukládání na disk se postará komponenta *distributor*, která řídí přerozdělování odkazů v rámci komponenty *frontier* (viz obrázek 4.3).

Komponenta *distributor* také po uplynutí časového intervalu 15 minut provede kontrolu, zda se na disku nachází `VisitState` s připraveným odkazem ke stažení. Tento interval byl zvolen, jelikož čas této kontroly roste s počtem uložených `VisitState` na disku a může tr-

vat i několik vteřin. Interval kontroly lze v případě potřeby změnit pomocí konfiguračního souboru. Pokud nalezený `VisitState` neobsahuje v operační paměti žádný připravený odkaz ke stažení, provede načtení připravených odkazů z disku. Kdyby obsahoval ještě nějaké odkazy ke stažení, případně by probíhalo stahování, pak se načtení provede až po stažení všech odkazů nacházejících se v instanci třídy `VisitState`.

4.2.3 Inkrementální strategie

V předchozí části byly popsány změny v systému potřebné pro to, aby systém BUbiNG bylo možné využít pro inkrementální stahování. Pro správné fungování tedy stačí vyřešit poslední problém, a to plánování další návštěvy stránky. Pro různé potřeby mohou být vyžadovány různé strategie, proto bylo pro jednotlivé strategie navrženo jednotné rozhraní, aby se daly strategie jednoduše konfigurovat pomocí konfiguračního souboru. Stačí vytvořit třídu implementující dané rozhraní a zvolenou strategii a v konfiguračním souboru umístit její konstruktor jako hodnotu parametru `revisitSheduler`. Rozhraní obsahuje jednu metodu `schedule`, které je předána stažená stránka, struktura s informacemi o další návštěvě a příznak, zda se stránka změnila.

Samotné plánování probíhá po zpracování stránky ve vláknu *ParsingThread*. Systém BUbiNG podporuje možnost filtrování, kdy se dá jednoduše nastavit, které stránky se mají stahovat, ukládat a zpracovávat. Způsob filtrování implementovaných v systému BUbiNG byl rozšířen o možnost určit, které stránky se mají znovu navštěvovat. Tímto způsobem jde jednoduše zapnout, vypnout nebo omezit inkrementální stahování.

V rámci této bakalářské práce byly vytvořeny dvě běžně využívané strategie a to:

- `UniformRevisitScheduler`: Všechny stránky se navštěvují po stejném výchozím intervalu. Další návštěva stránky se jednoduše naplánuje za uplynutí nastaveného intervalu po dokončení stahování stránky. Jedná se například o výchozí strategii v systému Nutch.

- `NonuniformRevisitScheduler`: Intervaly mezi návštěvami se vypočítávají na základě změny stránky pro každý odkaz zvlášť. Pokud je stránka navštívena poprvé, potom se další návštěva naplánuje za výchozí interval. Při dalších návštěvách se předchozí interval mezi návštěvami zvětší, pokud se stránka nezměnila, nebo zmenší, když se stránka změnila. Jak se bude měnit interval ovlivňuje koeficient pro zvětšení, nebo zmenšení intervalu. Například při koeficientu 0,1 se interval změní o 10 %. Pro rychlejší určení intervalu pro málo se měnící stránky, ukládáme v rámci této strategie ke každému odkazu čas poslední detekované změny. Když je čas mezi poslední detekovanou změnou a stažením stránky větší než nově vypočtený interval, nastaví se tato doba jako nový interval mezi návštěvami. To se typicky děje, když nebyla detekována změna při dvou a více posledních návštěvách. Aby se penalizovaly často měnící se stránky a zachycení změn málo měnících se stránek netrvalo příliš dlouhou, lze nastavit minimální a maximální hodnotu intervalu mezi návštěvami. Tuto strategii nalezneme jako alternativní strategii v systému Nutch a také v inkrementální úpravě systému Heritrix.

Pro detekci, zda se stránka změnila, je využíváno určení duplicity stránky. Tento způsob byl zvolen, jelikož přidání dalšího systému pro detekci změny stránky by výrazně zpomalil stahování a zároveň detekce duplicity plní podobný účel. Tímto způsobem se také penalizují odkazy, které odkazují na stejnou stránku. Pokud by se totiž určovala přímo změna konkrétní stránky, pak by se všechny odkazy na jednu stránku navštěvovaly ve stejném

intervalu a rostl by počet stejných stránek uložených v databázi. Naopak bude-li se stránka považovat za změněnou, když není duplicitní, budou se jednotlivé odkazy na stejnou stránku navštěvovat méně často, ale počet návštěv stránky by měl být ve výsledku podobný, jako kdyby na stránku odkazoval pouze jenom jeden odkaz.

4.3 Shrnutí

Kapitola se zabývá implementací určování duplicity textu webové stránky a úpravami systému BUbiNG pro inkrementální stahování.

Navržený způsob určování duplicity textu určuje duplicitu jednotlivých odstavců, kdy každý odstavec má jinou váhu na základě své délky a celková duplicita stránky se určí jako vážený průměr. Pro porovnávání duplicity odstavců je využit distribuovaný systém pro odstraňování duplicit z textů výzkumné skupiny KNOT. Z tohoto systému je ponechán původní databázový server a klientská část je znovu implementovaná v jazyce Java, aby mohl být propojen se systémem BUbiNG.

Prvním krokem pro upravení systému BUbiNG pro inkrementální stahování bylo navržení způsobu uložení informací o další návštěvě. Původně byly jednotlivé cesty a dotazy odkazů uložené v poli bytů, které bylo nahrazeno objektem s informacemi potřebnými pro naplánování další návštěvy. Jelikož systém BUbiNG omezuje maximální velikost komponenty *workbench*, obsahující připravená URL ke stažení, bylo potřeba prozkoumat, kolik operační paměti zabírá nově vytvořená struktura.

Po stažení stránky je potřeba uložit informace o další návštěvě na disk, aby se uvolnilo místo v paměti pro další odkazy. Proto pro uložení na disk byla upravena již existující datová struktura *workbench virtualizer*. Ke každé doméně si struktura uchovává čas nejbližší návštěvy a ze struktury se pro danou doménu vyberou pouze připravené odkazy ke stažení.

A nakonec, jádrem každého inkrementálního prohledávacího modulu je plánování další návštěvy stránky. V rámci bakalářské práce bylo navrženo rozhraní pro plánování opakovaných návštěv, umožňující jednoduše vybírat strategie pomocí konfiguračního souboru. Zároveň byly implementované dvě nejpoužívanější strategie pro plánování opakovaných návštěv.

S těmito změnami souvisí drobné změny napříč celým systémem.

Kapitola 5

Vyhodnocení řešení

V rámci vyhodnocení implementace si nejdříve projdeme dopady provedených úprav na paměťovou náročnost a rychlost stahování. Dále si určíme vhodné koeficienty u strategie s proměnnými intervaly mezi návštěvami. Funkčnost celého systému otestujeme pomocí vytvořeného simulátoru. Na závěr zhodnotíme inkrementální stahování slovenského internetu pomocí systému BUbiNG.

5.1 Náročnost provedených úprav na zdroje

V zásadě můžeme uvažovat čtyři očekávané příčiny zhoršení výkonu a zvětšení nároků na využití operační paměti: zvětšení velikosti komponenty *workbench*, častější přístup na disk, normalizace textu a síťová komunikace systému pro deduplikaci.

Pro vyhodnocení rychlosti stahování a nároků na zdroje byl využit testovací nástroj, který je součástí systému BUbiNG. Jedná se o proxy server, který simuluje 100 miliónů hostů s průměrně 50 stránkami. Nástroj slouží pro otestování konfigurace, využití paměti a nastavení systému, aniž by byl omezován limity síťového připojení.

Systém BUbiNG byl spuštěn s nastaveným testovacím nástrojem jako *proxy server* a s úvodní webovou stránkou nástroje jako výchozí množinou odkazů.

Úpravy pro inkrementální stahování

Velikost komponenty *workbench* je obvykle omezena na 512 MB. Podle odhadu velikosti struktury pro uchování informací o další návštěvě z kapitoly 4.2.1 může velikost komponenty *workbench* narůst až třikrát. To znamená, že omezení velikosti *workbench* by mělo být navýšeno na 1,5 GB, aby komponenta mohla obsahovat stejný počet odkazů. Takový nárůst využití paměti by se však mohl výrazně odrazit na rychlosti stahování stránek u strojů s menší velikostí volné paměti.

Další zpomalení stahování by mohly způsobit změny spojené s úpravami struktury *workbench virtualizer*. V rámci úprav se musí načíst všechny odkazy uložené na disku k požadovanému doménovému jménu, seřadit podle naplánovaného času další návštěvy a nepřipravené odkazy znovu uložit zpět na disk, přičemž čtení a zápis na disk jsou relativně pomalé operace.

Při testu nároků na zdroje spojené s inkrementálním stahováním byl spuštěn jeden agent s 12 vlákny *ParsingThread* a 1000 vlákny *FetchingThread*. Výsledky testování jsou shrnuty v tabulce 5.1. Při testování se ukázalo, že po úpravách systému komponenta *workbench* zabírá v paměti přibližně dvakrát více místa. Velikost *workbench* se během testování ani

Parametr	Původní systém	Inkrementální systém
Odkazy ve frontách [-]	237307	238481
Velikost *workbench* [MiB]	10,41	20,47
Celkové využití paměti [GiB]	17-23	17-23
Rychlost stahování [stránek/s]	6 898,85	6 498,31

Tabulka 5.1: Test nároků na zdroje spojené s inkrementálním stahováním

zdaleka nepřiblížila k maximální velikosti a vzhledem k celkovému využití paměti systémem BUbiNG je nárůst využití paměti zanedbatelný. Provedené změny nemají také výrazný vliv na výslednou rychlost stahování, kdy pokles rychlosti je pouze 6 %.

Zpracování a určování duplicity textu

Dalším provedeným experimentem je dopad zpracovávání a určování duplicity textu na rychlost stahování. Při experimentu zpracovávalo text 12 vláken *ParsingThread* a haše se odesílaly na 3 databázové servery, každý na jiném stroji. V prvním testu se pouze normalizovaly zpracované odstavce a v druhém se také určovala duplicita stránky.

Zapnuté funkce	Rychlost stahování [stránek/s]
Původní systém	6 898,85
Normalizace textu	5 571,03
Normalizace textu a určování duplicity	3 302,17

Tabulka 5.2: Test nároků na zdroje spojené s určováním duplicity stránky

Z tabulky 5.2 vidíme výrazné zpomalení jak při normalizaci textu, tak při vytváření hašů z odstavců a jejich odesílání na databázový server. Tento výsledek není překvapivý, jelikož zpracovávání textu je náročná operace na výpočetní výkon. V případě určování duplicity textu se musí komunikovat s databázovým serverem, který je spuštěn na jiném počítači, a čekání na odpověď stojí za celkovým zpomalením stahování. Při stahování internetu není omezení rychlosti tak znatelné, jelikož kvůli omezení šířky pásma nikdy nedosáhneme takových rychlostí jako v testovacím prostředí (viz podkapitola 5.4).

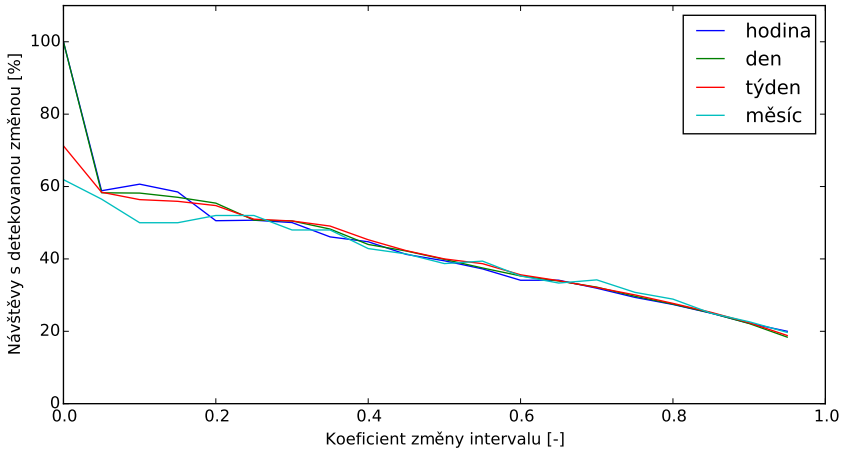
5.2 Vyhodnocení úspěšnosti inkrementální strategie

Pro vyhodnocení implementovaných strategií byly vytvořeny dva simulátory. První simulátor slouží pro určení vhodného koeficientu pro úpravu intervalů v rámci strategie, která navštěvuje jednotlivé stránky s jinou frekvencí. Druhý simulátor slouží k ověření funkčnosti jednotlivých strategií.

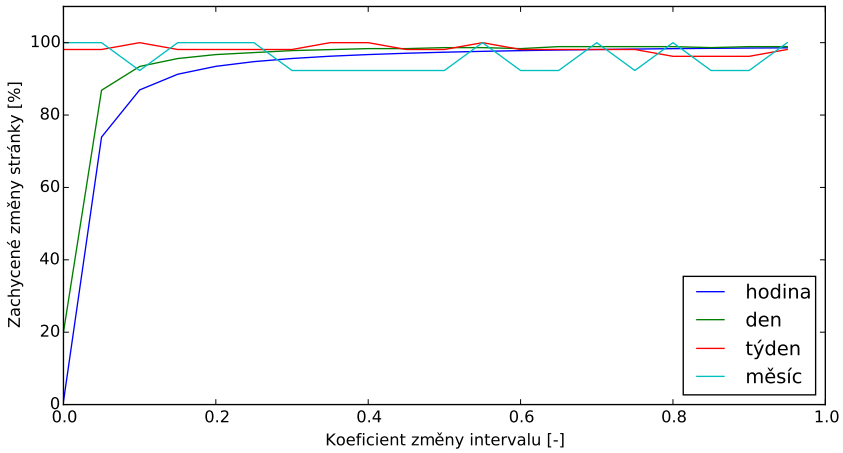
Určení vhodného koeficientu pro změnu intervalu mezi návštěvami

Simulace nevyužívá celý systém BUbiNG, ale jenom nezbytně nutné části pro plánování další návštěvy. Díky zjednodušení systému je možné provést simulaci běhu systému po delší časovou dobu a získat představu o počtu zachycených změn a počtu návštěv s detekovanou změnou. Na druhou stranu je při simulaci zanedbáno využití zdrojů a politika slušnosti.

V rámci simulace byl simulován běh systému BUbiNG po dobu jednoho roku s výchozím intervalem mezi návštěvami 5 dnů. Na stránkách, které se mění vždy po stejném intervalu,



Obrázek 5.1: Procentuální vyjádření počtu návštěv s detekovanou změnou

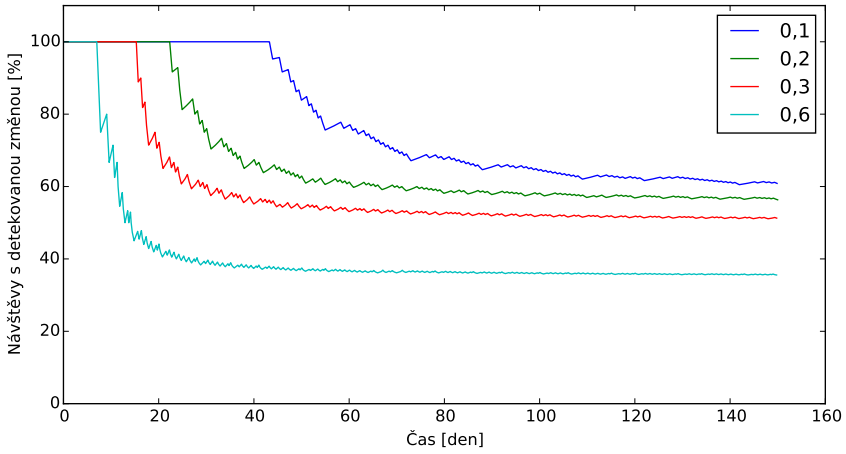


Obrázek 5.2: Procentuální vyjádření počtu zachycených změn stránky

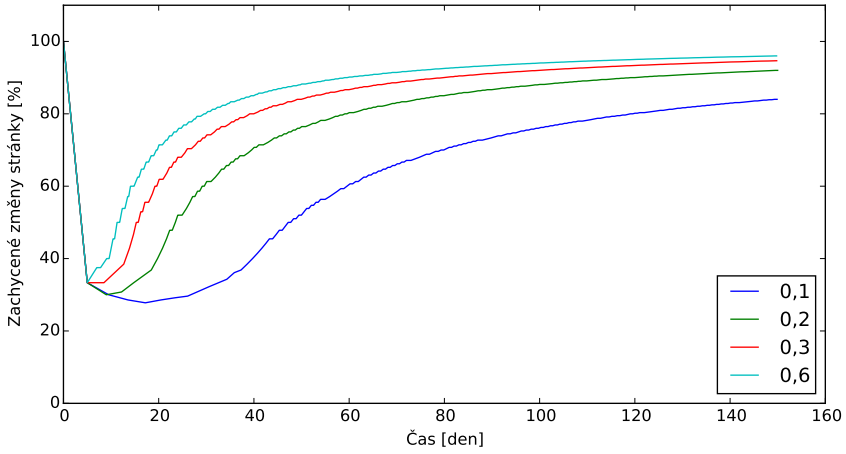
bylo otestováno několik hodnot koeficientu změny intervalu v rozmezí $\langle 0; 1)$ a s krokem 0,05. Během testování byly voleny koeficienty pro zvětšení a zmenšení intervalu vždy stejné. Například koeficient 0,5 změní interval mezi návštěvami o 50 % a při koeficientu 0 se budou stránky vždy navštěvovat po výchozím intervalu. Na grafech 5.1 a 5.2 jsou zachycené výsledky simulace, kdy je na vodorovné ose vynesen zvolený koeficient.

Graf 5.1 znázorňuje závislost počtu návštěv, kdy byla detekovaná změna, na hodnotě koeficientu změny intervalu mezi návštěvami. Z grafu vyplývá, že počet úspěšných návštěv s detekovanou změnou není závislý na frekvenci, s kterou se stránka mění, ale pouze na zvolených hodnotách koeficientů. Čím je zvolená hodnota menší, tím se zvětšuje počet návštěv, kdy byla detekována změna stránky. Naopak při zvolení větší hodnoty se stránky navštěvují častěji a narůstá počet předčasných návštěv, kdy se stránka ještě nezměnila.

V grafu 5.2 je vynesena závislost počtu zachycených změn na koeficientu změny intervalu mezi návštěvami. Čím je koeficient menší, tím klesá počet zachycených změn stránky u stránek s kratší periodou změny, než je výchozí interval pro další návštěvu. Aby byl



Obrázek 5.3: Procentuální vyjádření počtu návštěv s detekovanou změnou v čase (perioda změny stránky 1 den)



Obrázek 5.4: Procentuální vyjádření počtu zachycených změn stránky v čase (perioda změny stránky 1 den)

odhalen důvod menšího počtu zachycených změn, byla provedena simulace, kdy se zaznamenávala úspěšnost procházení v čase pro periodu změny stránky 1 den a výchozí interval mezi návštěvami 5 dnů. Pro simulaci byly zvoleny čtyři koeficienty pro změnu intervalu mezi návštěvami. Výsledky jsou zachyceny v grafech 5.3 a 5.4. Z grafů vyplývá, že čím je zvolený koeficient nižší, tím déle trvá nalézt vhodný interval pro opětovné navštívení stránky a to se odrazí na počtu zachycených změn.

U stránek měnících se méně často než výchozí interval se zachytí všechny změny stránky. Výjimkou je poslední změna, která nemusí být zachycena z důvodu ukončení simulace. Taky odpadá první fáze, kdy je při všech návštěvách detekovaná změna, a po prvotních výkyvech se ustálí poměr návštěv s detekovanou změnou, k předčasným návštěvám, na hodnoty z grafu 5.1.

Počet předčasných návštěv, kdy se stránka ještě nezměnila, by neměl být větší než počet návštěv s detekovanou změnou. Aby k tomu nedocházelo, neměl by být zvolený koeficient větší než 0,4. Naopak při zvolení nízkého koeficientu trvá nalezení vhodného intervalu pro další návštěvu delší dobu a to se projeví na menším počtu zachycených změn. Proto byly zvoleny výchozí koeficienty pro zmenšení a zvětšení intervalu mezi návštěvami 0,2. Jednotlivé koeficienty lze pro specifické případy upravit pomocí konfiguračního souboru.

Funkčnost jednotlivých strategií

Druhý simulátor testuje celý systém BUbiNG a slouží k ověření funkčnosti inkrementálního stahování. Jedná se o webový server implementovaný v jazyce Python, který simuluje 1000 webových stránek s různými intervaly změn. Intervaly jsou náhodně generovány v rozsahu od 2 do 80 minut podle rovnoměrného rozdělení pravděpodobnosti. U každé ze stránek byl sledován počet návštěv, zda při návštěvě byla detekovaná změna, kolik bylo zachyceno změn a aktuálnost a stáří stránky.

Systém BUbiNG byl spuštěn s oběma implementovanými strategiemi. První strategie navštěvuje stránky vždy po stejném intervalu a druhá strategie přizpůsobuje interval mezi návštěvami na základě frekvence změn stránky. Kvůli často se měnícím stránkám byl interval pro kontrolu připravených odkazů v komponentě *workbench virtualizer* nastaven pomocí konfiguračního souboru na 1 minutu. Průměrný interval mezi změnami jednotlivých stránek byl 40 minut, proto byl zvolen výchozí interval mezi návštěvami poloviční (20 minut). U strategie s různými intervaly návštěv byly nastaveny minimální a maximální intervaly tak, aby tyto hranice neomezovaly výpočet intervalu mezi návštěvami. Koeficienty pro zmenšení a zvětšení intervalu mezi návštěvami byly nastaveny na 0,2. Celkové výsledky obou stahování jsou shrnuty v tabulce 5.3.

Parametr	**Stejné intervaly**	**Proměnné intervaly**
Zachycených změn [%]	61,53	95,77
Návštěvy s detekovanou změnou [%]	54,40	57,55
Aktuálnost stažené databáze [%]	66,50	65,40
Stáří stažené databáze [s]	198	212

Tabulka 5.3: Vyhodnocení inkrementálních strategií

Z výsledků vyplývá, že pokud nebude strategie s různými intervaly mezi návštěvami nijak omezena, dosahuje výrazně lepších výsledků u počtu zachycených změn. U ostatních sledovaných parametrů si bylo chování obou strategií velmi podobné. Celková aktuálnost stažené databáze se u strategie se stejnými intervaly pohybovala v rozmezí od 63 do 70 %. K tomu docházelo, jelikož se všechny stránky stahovaly vždy v krátkém čase za sebou a potom čekaly na další návštěvu, naopak u druhé strategie bylo stahování více rozprostřeno, a proto zůstávala výsledná aktuálnost téměř neměnná. Při stahování internetu můžeme očekávat nižší úspěšnost při zachycení změn často měnících se stránek, způsobenou například politikou slušnosti.

5.3 Určování duplicity textu

Pro vyhodnocení určování duplicity textu byl systém BUbiNG spuštěn nad množinou slovenských stránek. Následně byla provedena analýza stažených stránek a byly prozkoumány důvody proč byly stránky vyhodnoceny jako duplicitní.

Pokud je stránka úplně totožná nebo obsahuje totožný obsah, pak je vždy správně vyhodnocena jako duplicitní. Na stránkách složených pouze z krátkých odstavců stačí změna v jediném odstavci a stránka je vyhodnocena jako neduplicitní. Stránky složené z většího počtu dlouhých odstavců jsou vyhodnoceny jako duplicitní, pokud neobsahují aspoň 10 % unikátních odstavců. Stránka je tedy vyhodnocena jako změněná, pokud se změní 10 % jejího obsahu.

Problém nastává u stránek složených z krátkých odstavců s pouze několika málo dlouhými odstavci. V těchto případech stránky často obsahují pouze jeden dlouhý odstavec, který se nachází na mnoha různých stránkách. Příkladem takových odstavců jsou například oznámení, že stránka pro správnou funkčnost požaduje povolení různých technologií (JavaScript, Flash Player, Cookies, ...). Takové stránky se mohou změnit, ale kvůli duplicitním dlouhým odstavcům jsou vždy vyhodnoceny jako nezměněné.

5.4 Stahování slovenského internetu

Systém byl po dobu 105 hodin spuštěn nad slovenským internetem se zapnutým inkrementálním stahováním a určováním duplicity textu. Jako inkrementální strategie byla vybraná strategie s proměnnými intervaly mezi návštěvami, kdy výchozí interval mezi návštěvami byl 24 hodin.

Parametr	**Hodnota**
Počet stažených stránek	91 902 513
Počet unikátních stránek	37 818 882
Počet opětovných návštěv	54 083 631
Rychlost stahování [stránek/s]	220,64
Počet *ParsingThread*	12
Počet *FetchingThread*	1000
Velikost fronty *results*	57
Velikost fronty *todo*	27 673
Počet odkazů ve frontách	810 997
Velikost *workbench* [MiB]	73,89
Celkové využití paměti [GiB]	22-27,4
Počet databázových serverů	3
Využití paměti databázovým serverem [GiB]	1,3

Tabulka 5.4: Výsledky stahování slovenského internetu

Výsledky pokusného stahování jsou zachyceny v tabulce 5.4. Rychlost stahování se po několika hodinách ustálila na 220 stránkách za sekundu. Tato malá rychlost, oproti testovacímu nástroji, je způsobena čekáním *FetchingThread* na stažení stránky z důvodu pomalé rychlosti stahování. To dokazují fronty *todo* a *results*.

Fronta *todo* obsahuje přes 27 tisíc připravených domén ke stažení, takže stahování není omezeno politikou slušnosti, kdy by se čekalo, až se může k nějaké doméně přistoupit.

Také zpracování obsahu stránky, včetně určování duplicity textu, není hlavním důvodem malé rychlosti stahování. Podle fronty *results* pouze 57 z 1000 vláken *FetchingThread* čeká na zpracování stažené stránky. Při předchozích pokusech na testovacím nástroji z podkapitoly 5.1 obvykle skoro všech 1000 vláken *FetchingThread* čekalo na zpracování stažené stránky.

Pro vyhodnocení paměťové náročnosti databázového serveru nemohl být použit původní testovací nástroj, jelikož generuje stránky se stejnými odstavci, tudíž by velikost databáze hašů neměla vypovídající hodnotu. Při stahování zpracovávalo text 12 vláken *ParsingThread* a byly spuštěny 3 databázové servery, každý na jiném stroji. Optimální poměr by měl být 4 klienti (vlákna) na jeden databázový server [4]. Po skončení stahování, kdy bylo staženo téměř 92 miliónů stránek, zabírala databáze hašů na každém serveru zvlášť 1,3 GiB místa (téměř 4 GiB dohromady).

Při stahování se ukázalo, že čím déle stahování trvá, tím méně se navštěvují nové stránky a místo toho se více navštěvují již navštívené stránky (viz tabulka 5.4 počet unikátních stránek a opětovných návštěv). To je způsobeno použitím struktury *workbench virtualizer* k uchování odkazů k opětovné návštěvě. Vybírání odkazů z *workbench virtualizer* musí mít přednost před novými odkazy z *sieve*, protože kromě odkazů k opětovné návštěvě obsahuje také dříve vybrané odkazy ze *sieve*, které kvůli omezením nemohly být v paměti. Ze *sieve* se následně vyberou odkazy jen pokud není *workbench* plný nebo podle počtu čekajících odkazů ke stažení. To může vést při dlouhodobém stahování ke stavu, kdy se již nebudou stahovat nové stránky, ale pouze opětovně navštěvovat již stažené stránky.

Tento problém nebyl v práci vyřešen a měl by být dále zkoumán. Pro jeho vyřešení by se měla navrhnout nová struktura pro uchování odkazů k opětovnému navštívení na disku, která by byla oddělená od *workbench virtualizer*. Dále by bylo vhodné navrhnout novou strategii pro přerozdělování odkazů v rámci *frontier* a případné další změny architektury části *frontier*.

Kapitola 6

Závěr

Práce se zabývá využitím systému BUbiNG pro inkrementální stahování. V první části práce jsou rozebrány problémy spojené se stahováním internetu a seznámení s jinými systémy a jejich využití pro inkrementální stahování webu.

Mezi první kroky patřilo prozkoumání architektury systému BUbiNG a navržení úprav potřebných pro inkrementální stahování. Původní systém BUbiNG uchovává k jednotlivým odkazům pouze informaci, zda byly navštíveny. Pro potřeby inkrementálního stahování byla navržena nová struktura, která si ke každému odkazu uchová informace o poslední návštěvě a plánu další návštěvy. Jelikož systém BUbiNG omezuje maximální velikost operační paměti, kterou mohou odkazy zabírat, bylo prozkoumáno, kolik místa zabírá nová struktura.

Po stažení stránky je potřeba uložit informace o další návštěvě na disk, aby se uvolnilo místo v paměti pro nové odkazy. Za tímto účelem lze využít dvě struktury nacházející se v systému BUbiNG: *sieve* a *workbench virtualizer*. U struktury *sieve* by bylo velmi obtížné zajistit, aby se stránky stahovaly podle plánu, proto byla k dalším úpravám vybraná struktura *workbench virtualizer*. Struktura byla upravena tak, aby si ke každé doméně pamatovala čas nejbližší budoucí návštěvy. Pokud se ve struktuře *workbench virtualizer* nachází odkaz připravený ke stažení, načtou se všechny odkazy do paměti, kde jsou seřazeny, stránky připravené ke stažení pokračují ve stahování, ostatní se pak uloží zpět na disk.

Jádrem každého webového prohledávacího modulu je plánování další návštěvy. V rámci této práce byly implementované dvě běžně používané strategie. První ze strategií plánuje další návštěvu vždy po stejném výchozím intervalu. Druhá ze strategií upravuje interval mezi návštěvami na základě změny stránky od poslední návštěvy.

Pro určení, zda se stránka od poslední návštěvy změnila, dle zadání práce byl využit deduplikátor výzkumné skupiny KNOT. Na základě principu deduplikátoru byl navržen způsob určování duplicity celých webových stránek. Duplicita webové stránky je určena na základě počtu duplicitních odstavců, kdy každý odstavec má jinou váhu podle své délky.

Při testování se ukázalo, že provedené úpravy pro inkrementální stahování nemají výrazný dopad na rychlost stahování nebo paměťovou náročnost systému. Naopak využití deduplikátoru výzkumné skupiny KNOT stahování dvojnásobně zpomaluje. Při dlouhodobém stahování stránek se častěji provádějí opětovné návštěvy než stahování nových stránek. To může vést do stavu, kdy budou existovat ještě nikdy nenavštívené odkazy, ale systém bude pouze navštěvovat již stažené stránky. Tento problém nebyl v práci vyřešen a měl by být dále zkoumán.

Implementované úpravy systému postupně nasazuje skupina KNOT ke svým experimentálním aktivitám. Systém BUbiNG je nadále vyvíjen na Milánské univerzitě a skupina

KNOT hodlá pokračovat v úpravách umožňujících používat systém BUbiNG pro inkrementální stahování.

Literatura

[1] *GNU wget*. [Online; navštíveno 11. 11. 2017].
URL https://www.gnu.org/software/wget/

[2] *HTTrack Website Copier*. [Online; navštíveno 30. 12. 2017].
URL https://www.httrack.com/

[3] *Memory usage of Java objects: general guide*. [Online; navštíveno 15. 3. 2018].
URL https://www.javamex.com/tutorials/memory/object_memory_usage.shtml

[4] *Odstraňování duplicit z textů (corpproc_dedup)*. [Online; navštíveno 28. 4. 2018].
URL http://knot.fit.vutbr.cz/wiki/index.php/Corpproc_dedup

[5] Artail, H.; Fawaz, K.: A fast HTML Web page change detection approach based on hashing and reducing the number of similarity computations. *Data & Knowledge Engineering*, ročník 66, č. 2, 2008: s. 326–337.

[6] Boldi, P.; Marino, A.; Santini, M.; aj.: BUbiNG: Massive Crawling for the Masses. *CoRR*, ročník abs/1601.06919, 2016.

[7] Cho, J.: *Crawling the Web: Discovery and Maintenance of Large-scale Web Data*. Dizertační práce, Department of computer science, Stanford University, 2001.

[8] Cho, J.; Garcia-Molina, H.: Synchronizing a Database to Improve Freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, ACM, 2000, s. 117–128.

[9] Cho, J.; Garcia-Molina, H.: Parallel Crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, ACM, 2002, s. 124–135.

[10] Cho, J.; Ntoulas, A.: Effective Change Detection Using Sampling. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, VLDB Endowment, 2002, s. 514–525.

[11] Chowdhury, A.; Frieder, O.; Grossman, D.; aj.: Collection Statistics for Fast Duplicate Document Detection. *Transactions on Information Systems*, ročník 20, č. 2, 2002: s. 171–191.

[12] Haveliwala, T. H.: Topic-sensitive PageRank. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, New York, NY, USA: ACM, 2002, s. 517–526.

[13] Khandagale, H. P.; Halkarnikar, P. P.: A Novel Approach for Web Page Change Detection System. *International Journal of Computer Theory and Engineering*, ročník 2, č. 3, 2010.

[14] Khare, R.; Cutting, D.; Sitaker, K.; aj.: Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, 2004.

[15] Koster, M.: Robots in the web: threat or treat? *ConneXions*, ročník 9, č. 4, 1995.

[16] Kouzis-Loukas, D.: *Learning Scrapy*. Packt Publishing, 2016, ISBN 978-1-78439-978-8.

[17] Manku, G. S.; Jain, A.; Das Sarma, A.: Detecting Near-duplicates for Web Crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, ACM, 2007, s. 141–150.

[18] McCurley, K. S.: Incremental Crawling. In *Encyclopedia of Database Systems*, Springer US, 2009, s. 1417–1421.

[19] Shobhna; Chaudhary, M.: A Survey on Web Page Change Detection System Using Different Approaches. *International Journal of Computer Science and Mobile Computing*, ročník 2, č. 6, 2013: s. 294–299.

[20] Sigurðsson, K.: Incremental crawling with Heritrix. In *Proceedings of the 5th International Web Archiving Workshop*, 2005.

[21] Tan, Q.; Mitra, P.: Clustering-based Incremental Web Crawling. *ACM Transactions on Information Systems*, ročník 2, č. 3, 2010.

[22] Udapure, T. V.; Kale, R. D.; Dharmik, R. C.: Study of Web Crawler and its Different Types. *International Organization of Scientific Research Journal of Computer Engineering*, ročník 16, č. 1, 2014.

[23] Yadav, D.; Sharma, A. K.; Gupta, J. P.: Parallel Crawler Architecture and Web Page Change Detection. *WSEARS Transactions on Computers*, ročník 7, č. 7, 2008.

Příloha A

Obsah přiloženého CD

- Tato práce ve formátu PDF
- Zdrojové soubory této práce v adresáři `thesis/`
- Zdrojové soubory systému BUbiNG včetně provedených úprav v adresáři `BUbiNG-BUT/`
- Zdrojové soubory systému pro deduplikaci v adresáři `dedup/`
- Zdrojové soubory webového simulátoru v adresáři `WebSimulator/`

Příloha B

Konfigurační soubor

1. Inkrementálního stahování s využitím strategie s proměnnými intervaly mezi návštěvami

```
revisitFilter=not IsProbablyBinary()
revisitScheduler=NonuniformScheduler(5d\, 12h\, 365d\, 0.2\, 0.2)
revisitCheckPeriodicity=15m
```

2. Určování duplicity stránek pomocí deduplikátoru výzkumné skupiny KNOT

```
knotDedup=true
parserSpec=HTMLParser(MurmurHash3\,
    it.unimi.di.law.bubing.parser.KnotDedupTextProcessor\, false)
duplicityThreshold=0.9
knotDedupHashMap=hashmap.conf
knotDedupPort=1234
```