Vyhledávání na webu a v multimediálních databázích (BI-VWM)
Dokumentace semestrální práce

Vyhledávání v hypertextu a PageRank

Ing. Jiří Novák, Ph.D. (vedoucí práce)
Filip Kaňák, Kevin Kroupa

Fakulta informačních technologií
České vysoké učení technické v Praze

LS 2021/22

Obsah

1 Zadání

Zadáním bylo implementovat vícevláknový crawler, který bude procházet vybranou podmnožinu webu. Tato podmnožina může být limitována jak počtem navštívených stránek, tak hloubkou zanoření jednotlivých odkazů. Nedílnou součástí PageRanku je také vyhledávání na základě textové podobnosti k zadanému dotazu, kterou musí aplikace zohlednit ve výpisu relevantních výsledků.

2 Architektura aplikace

Výchozí doménu pro naší aplikaci jsme zvolili Wikipedii. Fungovala by samozřejmě jakákoliv jiná doména za předpokladu, že by bylo ošetřeno, které odkazy z dané domény náš crawler nebude zahrnovat do svých výsledků, protože jejich frekventovaný výskyt (a tím i vysoké výsledné skóre) může značně ovlivnit výsledek přestože bychom si to nepřáli (např. odkazy obsaženy na téměř každé stránce - přihlašování, jazyk, navigace webů).

Aplikace se skládá ze tří částí:

Uživatelské rozhraní v podobě webové stránky, skrze které se uživatel systému dotazuje a v reálném čase získává výsledek v podobě výčtu nejrelevantnějších výsledků.

Webový server zpracovává uživatelský vstup a stará se o veškerou logiku - organizace crawleru, indexace obsahu stránek odkazů v grafu pro pozdější vyhodnocování textové podobnosti, komunikace s databází, výpočet PageRanku.

Databáze, která ukládá nestrukturovaná data v kolekcích - slouží pro ukládání dat z jednotlivých iterací PageRanku. Myšlenka tohoto postupu je taková, že tato data se vyplatí mít perzistentně např. pro velké grafy, pokud se při více spuštěních využívá ta stejná doména - drasticky se tím šetří čas nutný pro výpočet Google matice.

3 Implementace

3.1 Knihovny

Pro implementaci našeho řešení jsme zvolili programovací jazyk **Kotlin**. O asynchronní webovou komunikaci s uživatelem se stará framework **Ktor**. Kotlin je díky svému designu kompatibilní s knihovnami napsanými v Javě. Z nich jsme využili následující:

Crawler4j - poskytuje knihovnu pro vícevláknové crawlování webových stránek, kterou jsme ve vlastním modulu přizpůsobili potřebám naší aplikace, např. hlídání počtu navštívených stránek, maximální hloubky zanoření - každá rutina současně musí zapisovat parsovaná data do svého indexu.

Poznámka: Crawler4j má nastavenou tzv. *politeness delay*. Je to doba mezi jednotlivými requesty z důvodu nepřetěžování serverů. Defaultně **200ms**, přepsáno na **50ms**. Toto nastavení značně omezuje rychlost crawlování, nicméně i přesto aplikace zvládne nacrawlovat stovky stránek za desítky sekund.

Apache Lucene - tuto knihovnu jsme využili pro hledání výsledků na základě podobnosti se zadaným dotazem. Knihovna využívá inverted index a veškeré načtené odkazy si ukládá ve svých vlastních strukturách - opět jsme pro tuto funkcionalitu vytvořili vlastní modul, který poskytuje potřebné rozhraní a inicializuje servisní informace pro práci s indexem.

3.2 Životní cyklus aplikace

Aplikace při svém spuštění vždy čeká na požadavek od uživatele. Ve výchozí fázi má uživatel možnost spustit crawler, nebo rovnou zadávat dotazy - v tom případě budou výsledky vyhledávání závislé na již existujících datech, která jsou v databázi/indexu. Crawler přijme potřebné parametry a začne zpracovávat odkazy podle startovacího bodu (nějakého odkazu), dochází zde k parsování HTML, přidávání textu do indexu. Když je tato část hotová, začne se počítat PageRank pro každou z těchto stránek a také k aplikaci serií úprav jednotlivých hodnot pro zaručení vhodných vlastností naší matice. Pro rozumnou rychlost výpočtu je počet iterací fixní (konfigurovatelný parametry při spuštění aplikace).

3.3 Algoritmy

Crawler - běží paralelně na více vláknech (zajišťuje knihovna), jejich počet je konfigurovatetelný před startem aplikace. Každou URL testuje na výjimky. URL nesmí končit na css/js/gif/jpg/svg/png/mp3/mp4/zip/gz, musí začínat "https://cs.wikipedia.org/wiki/"a za tímto počátkem nesmí obsahovat dvojtečku (tímto se odfiltrují funkční stránky wikipedie jako například stránky profilu, obsah, FAQ a podobné odkazy přítomné na všech stránkách wikipedie). Crawler poté v nové korutině (asynchronně) ukládá stránku do indexu Lucene a do MongoDB databáze. Končí na základě zadaných parametrů v POST requestu p (počet stránek) a d (hloubka zanoření) podle toho, který z nich nastane dříve.

PageRank - pouští se automaticky po doběhnutí crawleru nebo lze pustit POST requestem. Zátěž se rozloží mezi konfigurovatelný počet korutin (korutina - podobný princip jako vlákno s hlavním rozdílem, že korutina nevytváří vlákno nové, ale běží na aplikací zvoleném počtu vláken, podle zařízení, na kterém aplikace běží a přepínání mezi nimi neřídí OS, ale aplikace), každá korutina poté prochází část kolekce a pro každý outlink každé stránky připočte hodnotu

$$\frac{pagerank[iterace - 1]}{početoutlinků}$$

Po dokončení se hodnota pageranku této iterace pro každou stránku ještě upraví vynásobením hodnoty 0.85 a přičtením hodnoty 0.15.

Získání výsledků - GET request s parametrem query vrací 10 "nejlepších výsledků". Tyto výsledky se získají kombinací 10 nejlepších výsledků z textové podobnosti průměrované s hodnotou pageranku pro konkrétní stránku. Je možnost také přejít na další stránky přidáním parametru $page = číslo$ (například kliknutím na odkaz - číslo stránky). Aplikace projde všechny výsledky textové podobnosti, a po zprůměrování hodnotou pageranku se uloží do seřazené mapy výsledků. Tato mapa si udržuje pouze požadovaný počet ($stránka * 10 + 10$) výsledků, ostatní zahazuje a vypsáním hodnot této mapy (a případným přeskočením prvních n hodnot) se přímo získá požadovaný seřazený výsledek.

4 Ukázka projektu

4.1 Ukázka na webu

Aplikaci si lze vyzkoušet přímo online na **této VPS**.

Připravili jsme vyhledávací doménu kořenící u nás na FITu, tj. její článek na Wikipedii. Pro demonstraci vyhledávání ve výchozím stavu je tedy vhodné vybírat termy v souvislosti s naší fakultou. Pro pestrost výsledků lze samozřejmě domény dodatečně přidávat k těm, co již v databázi existují.

Pokud si přejete vyzkoušet crawlování na nějaké jiné doméně, je potřeba na to server upozornit. Do

repozitáře jsme přiložili adresář **postman/**, ve kterém je vyexportovaná hotová kolekce requestů, kterou používáme. Stačí jej naimportovat do programu Postman (a nebo použít definici requestu v libovolném jiném programu) a poté ze zmíněné kolekce zavolat **clear** a **start_crawler** request. Změny se pro rozumný počet stránek projevují zpravidla do několika minut (viz Experimentální sekce).

4.2 Příklad použití

Příkladný postup použití v podobě screenshotů je nahrán na **na tomto adresáři**.

5 Lokální spuštění

5.1 Repozitář

Veškeré zdrojové kódy naší aplikace lze najít na fakultním GitLabu **zde**.

5.2 Prerekvizity

- Java 11
- Gradle
- MongoDB
- Docker na 64bitovém systému
- Docker-compose

5.3 Postup instalace

5.3.1 Zavedení MongoDB databáze

V projektovém kořenovém adresáři spusťte databázi na pozadí následovně:

```
cd ./docker/mongodb
docker-compose up
```

5.3.2 Zprovoznění aplikace

```
git clone git@gitlab.fit.cvut.cz:kanakfil/link-analysis-and-web-page-ranking.git
cd link-analysis-and-web-page-ranking       # naklonovani repozitare
./gradlew installDist                       # stazeni zavislosti + build
./build/install/pagerank/bin/pagerank       # spusteni aplikace
```

5.4 Konfigurační parametry

Parametry lze předat programu různými způsoby. Většina možností je popsána v **dokumentaci frameworku Ktor**.

5.4.1 Konfigurační soubor

Pro konfiguraci pomocí souboru je třeba vytvořit konfigurační soubor např. **application.conf** a předat cestu k tomuto souboru jako parametr při spouštění aplikace **-config=application.conf**.

Příklad struktury:

```
ktor {
    deployment {
        port = 80
    }
    mongo {
        name = "admin"
        password = "password123"
        host = "127.0.0.1"
        db = "pages"
    }
    pagerank {
        threads = 20
        iterations = 20
    }
}
```

5.4.2 Popis parameterů

- **ktor.deployment.port** - port, na kterém se pustí webový server
- **ktor.mongo.name** - jméno uživatele pro přístup do databáze
- **ktor.mongo.password** - heslo uživatele pro přístup do databáze
- **ktor.mongo.host** - url, kde běží single instance MongoDB databáze
- **ktor.mongo.db** - jméno databáze, kterou aplikace použije
- **ktor.pagerank.threads** - počet korutin použitých pro výpočet pageranku a zároveň počet vláken crawleru
- **ktor.pagerank.iterations** - počet iterací pageranku

6 Experimentální sekce

Pro tuto sekci jsme si připravili měření délek jednotlivých operací. Měří se doba crawlování stránek, výpočet PageRanku a délka vytvoření odpovědi pro dotaz. Testy rychlosti jsou pro přesnost prováděné na lokálním serveru, který byl provozován na zařízení s procesorem Intel Core i5-4210H, SSD diskem a 16 GB operační paměti.

Počet stránek na vstupu	Proces crawlování + indexace	Výpočet PageRanku	Délka odezvy na dotaz
50	$20s$	$20s$	$101ms$
100	$22s$	$22.5s$	$123ms$
500	$39s$	86s	$160ms$
1000	$1m6s$	$4m25s$	$199ms$
2000	$1m35s$	$6m49s$	$210ms$

7 Pozorování, závěr

Aplikace si prošla několika optimalizacemi, v prvotním prototypu bylo problematické pomalé indexování souborů, později i například výpočet matice a komunikace s databází - tuto slabinu jsme zaznamenali až ve chvíli, kdy jsme se snažili aplikaci spustit kompletně na vzdálené VPS. U kompletní inicializace grafu jsme zaznamenali mnohonásobné zrychlení. Současný stav projektu hodnotíme pozitivně.

Reference

[1] Přednášky předmětu Vyhledávání na webu a v multimediálních databázích (BI-VWM)

[2] Popis pagerank algoritmu na portálu Neo4j,
`https://neo4j.com/blog/graph-algorithms-neo4j-pagerank/`

[3] Dokumentace jazyka Kotlin,
`https://kotlinlang.org/docs/home.html`

[4] Framework Ktor,
`https://ktor.io/`

[5] Ukázka použití Apache Lucene,
`http://web.cs.ucla.edu/classes/winter15/cs144/projects/lucene/index.html`

[6] Dokumentace knihovny Apache Lucene,
`https://lucene.apache.org/core/8_8_2/core/index.html`

[7] Knihovna pro přístup k MongoDB databázi z jazyka Kotlin,
`https://litote.org/kmongo/`

[8] Knihovna Koin pro dependency management,
`https://insert-koin.io/`

[9] Knihovna pro crawlování Crawler4j,
`https://github.com/yasserg/crawler4j`