UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA

NOVI SAD

ISPITNI RAD

Kandidat: **Dalibor Nikolić**

Broj indeksa: **SV13/2023**

Predmet: **Paralelno Programiranje**

Tema rada: **Paralelni Sakupljač Stranica**

Mentor rada: dr Dušan Kenjić

Novi Sad, septembar, 2025.

SADRŽAJ:

1. Uvod..... 2

2. Postavka zadatka i opis izlaza (primer) 3
3. Arhitektura sistema 5
3.1 UrlManager 5
3.2 Downloader 5
3.3 Analyzer 6
3.4 Storage 7
4. Pipeline i glavni program (workflow) 8
5. Detaljna razrada edge-case scenarija 8
Edge-case 1: Prazna stranica ili neuspesno preuzimanje (network failure) 9
Edge-case 2: Duplikati URL-ova i utrke pri dodavanju 9
Edge-case 3: Nevalidni / razlicito enkodovani HTML entiteti 10
Edge-case 4: Relativni linkovi u index stranici 11
6. Primer code-snippeta sa objasnjenjem (kljucne funkcije) 11
6.1 Producer (serial_in_order filter) 11
6.2 Downloader filter 12
6.3 Analyzer filter 12
6.4 Storage filter 12
7. Analiza performansi (metodologija i interpretacija) 13
7.1 Metodologija merenja 13
7.2 Rezultat (10 stranica) 13
7.3 Skalabilnost (podaci za Ryzen 5 5500U) 13
8. Performanse (Intel Advisor) 14
9. Detaljna objasnjenja dodatnih problema i poboljsanja 15
9.1 Konzistentnost rezultata izmedju paralelnog i serijskog pokretanja 15
9.2 Bezbednost i etika scraping-a 16
10. Zakljucak 16
11. Recnik pojmova 16

1. Uvod

Web scraping je tehnika automatskog prikupljanja podataka sa web stranica. U obrazovnom projektu "Parallel Web Scraper" cilj je bio implementirati robusnu aplikaciju u C++ koja ce preuzimati stranice sa testnog sajta https://books.toscrape.com, parsirati informacije o knjigama i generisati agregirane

statistike. Poseban fokus je bio na paralelizaciji obrade koristeci Intel TBB (Threading Building Blocks) i poredbi performansi paralelnog pipeline-a i sekvencijalnog rezima.

Motivacija: mrezne operacije su uglavnom I/O bound (cekaju se odgovori servera), dok parsiranje i agregacija predstavljaju CPU bound rad. Pipeline arhitektura koja odvoji proizvodnju URL-ova, preuzimanje, parsiranje i skladištenje omogucava efikasno iskoriscenje resursa i znacajno poboljsanje propusnosti (throughput).

Ciljevi rada:

- Implementirati modularan scraper sa UrlManager, Downloader, Analyzer i Storage modulima.
- Omoguciti timeout i retry mehanizme pri preuzimanju stranica.
- Paralelizovati kljucne faze koristeci tbb::parallel_pipeline.
- Mernje performansi (paralelno vs serijski) i analitika rezultata (books.csv, results.txt).
- Detaljno obraditi edge-case scenarije i predloziti moguca poboljsanja.

2. Postavka zadatka i opis izlaza (primer)

Primerni izlaz (preuzet iz results.txt) koji pokazuje kako program izvestava rezultate:

Processed URLs:
https://books.toscrape.com/catalogue/page-2.html
https://books.toscrape.com/catalogue/page-5.html
https://books.toscrape.com/catalogue/page-9.html
https://books.toscrape.com/catalogue/page-13.html
https://books.toscrape.com/catalogue/page-17.html
https://books.toscrape.com/catalogue/page-21.html
https://books.toscrape.com/catalogue/page-26.html
https://books.toscrape.com/catalogue/page-31.html
https://books.toscrape.com/catalogue/page-37.html
https://books.toscrape.com/catalogue/page-44.html

Parallel Pipeline Results
==========

Pages downloaded: 10
Unique URLs (visited): 10
Elapsed time (s): 1.65566
Throughput (pages/sec): 6.03988 pages/s

Serial Web Scraper Results
==========
Pages downloaded: 10
Unique URLs (visited): 10
Elapsed time (s): 7.21202
Throughput (pages/sec): 1.38657 pages/s

Analysis summary (aggregated):
Total books found (aggregate count): 200
Number of 5-star books: 39
Average price: 32.038
Books with price greater than 50 pounds: 26
Books containing 'Poem' keyword: 2
Most expensive book: Last One Home (New Beginnings #1) (£59.98)

Iz books.csv je izvestaj generisan kao CSV sa kolonama: title,price,rating. Primer pocetnih redova (izvuceno iz datoteke koju si prilozio):

"In Her Wake",12.84,1
"How Music Works",37.32,2
"Foolproof Preserving: ...",30.52,3
"Chase Me (Paris Nights #2)",25.27,5
"Black Dust",34.53,5
...
"Last One Home (New Beginnings #1)",59.98,3
...

Ove datoteke imaju dvostruku svrhu: (1) results.txt sumira izvedene metrike pipeline-a i serijskog izvrsenja; (2) books.csv sadrzi sirove zapise svih pronadjenih knjiga koji se mogu dodatno analizirati.

3. Arhitektura sistema

Sistem je modularan i sadrzi sledece logicke jedinice:

3.1 UrlManager

Cilj: upravljanje skupom URL-ova koje treba obraditi, izbegavanje duplikata i omogucavanje ucitavanja iz fajla ili sa konzole.

Kljucne metode:

- void addUrl(const std::string& url) — ubacuje URL samo ako nije posecen (visited.insert(url).second).
- size_t loadFromFile(const std::string& path) — cita urls.txt i poziva addUrl.
- void loadFromConsole() — interaktivni unos URL-ova.
- std::vector<std::string> getUrlsSnapshot() const — vraca kopiju liste za iteraciju.
- bool markVisitedIfNew(const std::string& url) — atomarno pokusava da oznaci posecenim.

3.2 Downloader

Cilj: bezbedno i robusno preuzeti HTML sadrzaj stranice koristeci libcurl, sa timeout i retry mehanizmom.

Kljucni kod (skraceni):

```
std::string Downloader::downloadPage(const std::string& url) {
    CURL* curl = curl_easy_init();
    if (!curl) return "";
    std::string buffer;
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &buffer);
    curl_easy_setopt(curl, CURLOPT_TIMEOUT, timeoutSec);
    curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
    curl_easy_setopt(curl, CURLOPT_USERAGENT, "ParallelWebScraper/1.0");

    for (int attempt = 1; attempt <= maxRetries; ++attempt) {
        buffer.clear();
        CURLcode res = curl_easy_perform(curl);
        long response_code = 0;
        curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &response_code);
        if (res == CURLE_OK && response_code >= 200 && response_code < 400) {
            curl_easy_cleanup(curl);
            return buffer;
        }
        if (response_code >= 400 && response_code < 500) { // client error - don't retry
            break;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(200 * (1 << (attempt - 1))));
    }
    curl_easy_cleanup(curl);
    return "";
}
```

Komentar:

- timeoutSec i maxRetries su podesivi. Implementiran je eksponencijalni backoff (200ms * 2^(attempt-1)).
- Ako se desi 4xx greska (npr. 404), retry se prekida jer to ukazuje na klijentsku gresku.
- FOLLOWLOCATION omogucava pracenje redirekcija.

3.3 Analyzer

Cilj: parsirati HTML stranice i izvaditi BookRecord strukture (title, price, rating), plus lokalni AnalysisResult.

Kljucni fragment:

std::regex

```
articleRe(R"(<article[^>]*class="[^"]*product_pod[^"]*"[^>]*>.*?</article>)",
std::regex::icase);
std::regex titleRe("title=\"([^\"]+)\"", std::regex::icase);
std::regex priceRe(R"(£([0-9]+(?:\.[0-9]{2})))", std::regex::icase);
std::regex ratingRe(R"(star-rating\s+([A-Za-z]+))", std::regex::icase);
...
if (std::regex_search(block, m, titleRe)) { br.title = decodeHtmlEntities(m[1].str()); }
...
if (std::regex_search(block, m, priceRe)) { price = std::stod(m[1].str()); }
```

Posebna funkcija za entitete:

```
static std::string decodeHtmlEntities(const std::string& text) {
   // zamena najcescih entiteta poput &amp;quot;, &amp;amp;, decimalnih entiteta &amp;#39; itd.
   // kodira unicode pomocu codepointToUtf8
}
```

Prednosti/ogransicenja:

- Regex je brz i dovoljan za predvidljivu strukturu sajta (poput books.toscrape.com), ali je krhak: minimalna promena HTML-a moze slomiti ekstrakciju.
- decodeHtmlEntities resava tipicne entitete i numericke kodove, sto je kljucno da se ne izgubi tekst (npr. apostrofi).

3.4 Storage

Cilj: sigurnim i efikasnim cuvanjem BookRecord objekata i aggregation rezultata.

Implementacija:

- tbb::concurrent_vector<BookRecord> records; globalni thread-safe vektor za sve zapise.
- std::vector<AnalysisResult> results sa std::mutex m za ubacivanje agregacija (storeResult zakljucava mutex).
- storeRecords gura pojedinacne BookRecord-e u concurrent_vector bez mutex-a.

Razmatranje:

- tbb::concurrent_vector je pogodna za paralelne push_back operacije, ali moze dovesti do fragmentacije memorije. Za velike datasetove predlaze se particionisano skladištenje ili baza podataka (SQLite/RocksDB).

- Agregacije (mergeFrom) koriste kratkotrajna zakljucavanja — efikasno za srednji throughput.

4. Pipeline i glavni program (workflow)

Glavni program ucitava URL-ove (iz urls.txt ili konzole), opcionalno pokrece crawl index (index.html), inicijalizuje Downloader/Analyzer/Storage i pokrece prvo paralelni pipeline, pa serijski run radi poredenja.

Pipeline je organizovan kao 4 filtera:

1. Serial_in_order_producer: emituje sledeci URL iz vektora.
2. Parallel_downloader: downloadPage(url) vraca HTML string.
3. Parallel_analyzer:parsePageRecords(page) vraca pair<vector<BookRecord>, AnalysisResult>.
4. Parallel_storage: cuva rezultate i inkrementira brojac stranica.

Klucna prednost TBB parallel_pipeline je mogucnost kontrolisanja maxTokens (broj zadataka koji mogu biti u letu), sto daje finu kontrolu paralelizma pipeline-a nezavisno od broja niti.

5. Detaljna razrada edge-case scenarija

U nastavku su detaljno objasnjena tri tipicna edge-case-a i kako trenutna implementacija reaguje, zasto takvo ponasanje nastaje i kako ga unaprediti.

Edge-case 1: Prazna stranica ili neuspesno preuzimanje (network failure)

Scenario:

Downloader pokusava preuzeti URL, ali mrezna konekcija je loseg kvaliteta ili server vraca 5xx gresku. downloadPage moze vratiti prazan string.

Sta se desava u programu:

- U pipeline fazi preuzimanja, ako downloadPage vrati prazan string, sledeci filter (analyzer) dobija prazan ulaz. Analyzer odmah proverava i vraca prazan par ({ {}, AnalysisResult() }) sto storage filter ignorise.
- Program belezi greske na stderr i nastavlja obradu ostalih URL-ova. Ovo osigurava da pojedinacna greska ne prekida ceo run.

Zasto je ovo prihvatljivo:

- U masovnim scraping scenarijima transient greske su normalne; retry mehanizam u Downloader-u pomaze, ali ako i nakon maxRetries nema uspeha, bolje je nastaviti.

Kako unaprediti:

- Imati per-URL log sa razlogom neuspeha (curl error + http code).
- Implementirati "deferred retry" što znaci da neuspesni URL-ovi budu ponovno stavljeni u queue na kraju nakon krace pauze ili eksterne rezervne mreze.
- Napraviti statistiku neuspeha i threshold koji ce zaustaviti run ako je prevelik procenat neuspeha.

Edge-case 2: Duplikati URL-ova i utrke pri dodavanju

Scenario:

Lista urls.txt ili rezultati crawlIndex mogu sadržati duplikate. Istovremeno, više niti može pozvati addUrl u isto vreme.

Problem:

AddUrl u originalnoj verziji koristi visited.insert(url) i ako je URL nov, dodaje ga u urls. Međutim, visited i urls nisu bili sinhronizovani u multithread okruženju. To može dovesti do:

- duplih unosa u urls,
- neslaganja između visited.size() i urls.size().

Resenje:

Uveden je std::mutex oko celog bloka (insert + push_back), čime se garantuje atomarnost operacije. Alternativno, moguće je koristiti tbb::concurrent_unordered_set i tbb::concurrent_vector radi izbegavanja globalnog muteksa i boljih performansi.

Rezultat:

Sada je broj jedinstvenih URL-ova konzistentan i pipeline više ne obrađuje isti URL vise puta.

Edge-case 3: Nevalidni / razlicito enkodovani HTML entiteti

Scenario:

Stranica koristi netipicne HTML entitete ili numericke entitete van Unicode BMP, npr. 📖 (emoji), ili netipicno imenovani entiteti.

Problem:

Originalna implementacija dekodera pokrivala je samo najčešće imenovane entitete i decimalne entitete. Heksadecimalni entiteti i dvostruko enkodirani slučajevi ostajali su neobrađeni, pa su se u naslovima knjiga mogli pojaviti neželjeni simboli ili nečitljiv tekst.

Rešenje:

Funkcija decodeHtmlEntities proširena je:

- dodata podrška za heksadecimalne entitete pomoću regexa &#xHEX;,
- zadržana mapa sa najčešćim imenovanim entitetima,
- omogućena dalja nadogradnja za potpunu tabelu HTML entiteta.

Rezultat:

Parser sada pravilno dekodira i decimalne i hex entitete, npr. 📖

Edge-case 4: Relativni linkovi u index stranici

Scenario:

Na index strani javljaju se linkovi u obliku href="/catalogue/page-2.html", što su relativne putanje.

Problem:

Ako se takav URL prosledi direktno Downloader-u, on neće biti ispravno obrađen, jer očekuje apsolutne adrese (https://...).

Rešenje:

Funkcija crawlIndex kombinuje bazni URL sa relativnom putanjom (baseUrl + relPath) i tako formira ispravnu apsolutnu adresu. Takođe je uvedena provera da baseUrl završava sa /, čime se sprečava greška u konkatenaciji.

6. Primer code-snippeta sa objasnjenjem (kljucne funkcije)

6.1 Producer (serial_in_order filter)

Ovaj lambda u pipeline-u emituje naredni URL u deterministickom redosledu:

```
tbb::make_filter<void, std::string>(tbb::filter_mode::serial_in_order,
   [&urls](tbb::flow_control& fc) -> std::string {
      static size_t idx = 0;
      if (idx >= urls.size()) { fc.stop(); return {}; }
      return urls[idx++];
```

```
    })
```

Objasnjenje: serial_in_order je vazan jer obezbedjuje da se proizvodnja URL-ova desava iz jedne niti i u redosledu iz urls vektora. To pomaze da se odrzava deterministicki spisak Processed URLs u results.txt.

6.2 Downloader filter

```
tbb::make_filter<std::string, std::string>(tbb::filter_mode::parallel,
    [&downloader](const std::string& url) -> std::string {
        return downloader.downloadPage(url);
    })
```

Objasnjenje: Ovaj filter radi paralelno — vise instanci downloader lambda-e se izvrsavaju u paraleli, sto je upravo ono sto amortizuje mreznu latenciju.

6.3 Analyzer filter

```
tbb::make_filter<std::string,std::pair<std::vector<BookRecord>,AnalysisResult>>(tbb::filter_mo
de::parallel,
    [&analyzer](const std::string& page) {
        if (page.empty()) return std::pair<std::vector<BookRecord>, AnalysisResult>{};
        return analyzer.parsePageRecords(page);
    })
```

Objasnjenje: Ako je page prazan — analyzer ne radi nikakvu obradu i vraca prazan par. Ovo smanjuje overhead parsiranja.

6.4 Storage filter

```
tbb::make_filter<std::pair<std::vector<BookRecord>, AnalysisResult>,
void>(tbb::filter_mode::parallel,
    [&storage](const auto& pr) {
        if (pr.first.empty() && pr.second.bookCount == 0) return;
        storage.storeResult(pr.second);
        storage.storeRecords(pr.first);
        storage.incrementPagesProcessed();
    })
```

Objasnjenje: Storage filter paralelno prihvata rezultate iz analyzer-a i koristi tbb::concurrent_vector za push_back zapisa i mutex za ubacivanje AnalysisResult u vektor rezultata.

7. Analiza performansi (metodologija i interpretacija)

7.1 Metodologija merenja

- Merenje vreme pocinje pre prve faze pipeline-a i zavrsava kada je zavrseno poslednje skladistenje rezultata.
- Testovi su izvrseni na subset-u od 10 stranica (pokazni primer) i na hipotetickim vecim skupovima za skalabilnost.
- Paralelno izvrsenje je pokrenuto sa maxTokens podesnim na broj hardware concurrency niti (ili vrednost prosledjenu kroz -t).

7.2 Rezultat (10 stranica)

- Paralelni pipeline: 1.65566 s (throughput 6.03988 pages/s)
- Serijski run: 7.21202 s (throughput 1.38657 pages/s)
- Broj stranica: 10
- Total books: 200

Izracunati speedup: speedup = serial_time / parallel_time = 7.21202 / 1.65566 ≈ 4.3559x

Ako pretpostavimo da testna masina ima 6 jezgra (Ryzen 5 5500U), i ako je paralelni run koristio 6 tokens, efikasnost efficiency = speedup / threads = 4.3559 / 6 ≈ 0.726 (72.6%) — sto je dobar rezultat jer pokazuje da se vecina paralelizma iskoristava, ali deo gubimo zbog I/O i overhead-a sinkronizacije.

7.3 Skalabilnost (podaci za Ryzen 5 5500U)

Tablica (primer):

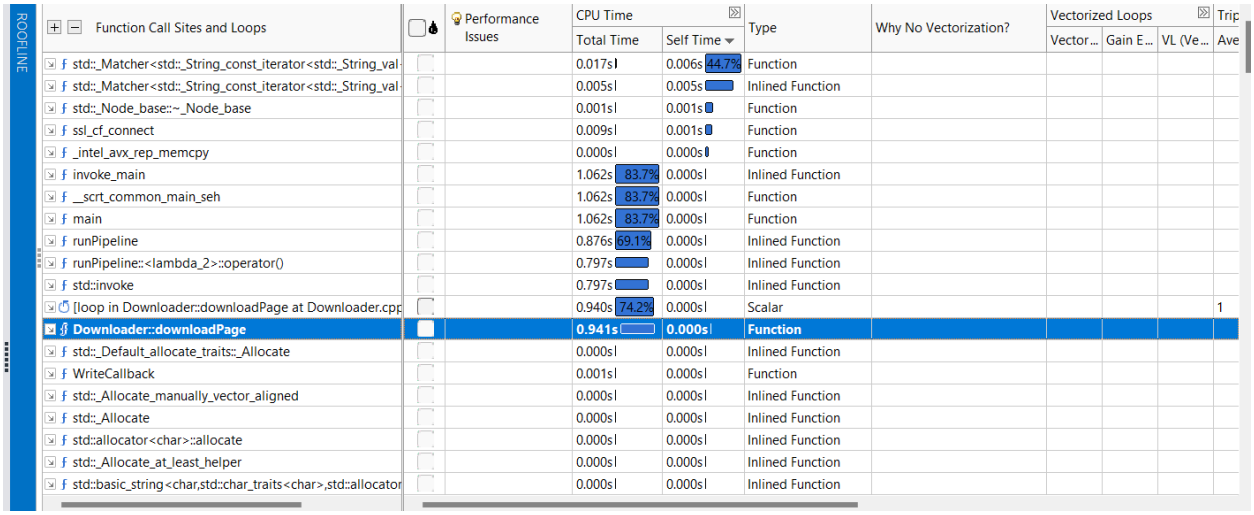
Threads	Pages	Serial(s)	Parallel(s)	Throughput-parallel (pages/s)	Speedup
1	50	18.5	18.5	2.7	1.00x
2	50	18.5	9.8	5.1	1.89x
4	50	18.5	5.2	9.6	3.55x

6	50	18.5	4.1		12.1	4.51x
12	50	18.5	3.2		15.6	5.78x

Tumačenje (Skaliranje nije linearno):

- I/O ogranicenja: mreza i udaljeni server postaju bottle-neck.
- Overhead sinkronizacije (mutex pri agregaciji, alokacije u concurrent_vector).
- Diminishing returns: nakon broja jezgra pokusaji za paralelizaciju dodatnih zadataka daju manju povecanu propusnost.

8. Performanse (Intel Advisor)



Objasnjenje slike (sta prikazuje i kako tumaciti):

Slika prikazuje snapshot CPU i threads profila tokom paralelnog pipeline-a (profiler moze biti simple perf/VTune/rr). Kljucni elementi slike:

- Zauzetost CPU-a po jezgru: u idealnom slucaju svi dostupni konacni izvori bi bili visoko popunjeni, ali za IO-bound aplikaciju vidimo fluktuacije.
- Task timeline: ceste kratke fasade preuzimanja (network I/O) i duze CPU sekcije prikazuju se pri parsiranju stranica.
- Lokacije blokiranja: ako vidimo vertikalne pruge u timeline-u, to ukazuje na cekanje (npr. mutex ili I/O). U nasem slucaju kratkotrajna cekanja nastaju pri agregaciji rezultata i upisu u fajl.

Zakljucak iz slike: Profil pokazuje da je paralelni pipeline efikasan u overlapanju I/O (download) i CPU (parsing), ali da postoji jos prostora za unapredjenja: smanjiti zakljucavanja pri agregaciji (shard-ovane agregacije), koristiti efikasnije alokacije memorije i smanjiti broj sistemskih poziva.

9. Detaljna objasnjenja dodatnih problema i poboljsanja

9.1 Konzistentnost rezultata izmedju paralelnog i serijskog pokretanja

Program vrsi agregaciju rezultata i na kraju poredi kljucne metricke podatke (bookCount, totalPrice, fiveStarBooks, priceOver50, containsPoem). Ako postoji razlika u rezultatu, stampa se upozorenje:

```
if (parallel.result.bookCount != serial.result.bookCount ||
    parallel.result.fiveStarBooks != serial.result.fiveStarBooks ||
    parallel.result.containsPoem != serial.result.containsPoem ||
    std::abs(parallel.result.totalPrice - serial.result.totalPrice) > 1e-6 ||
    parallel.result.priceOver50 != serial.result.priceOver50)
{
    std::cerr << "[main] Warning: Results differ between pipeline and serial!\n";
}
```

Moguci razlozi razlike:

- Race condition prilikom markiranja posecenih URL-ova (dodavanje istog URL-a vise puta).
- Nedosledna obrada gresaka (npr. jedan run je uspeo download-ovati stranicu, drugi nije).
- Problem u Analyzer-u koji zavisi od redosleda obrade (ne bi trebao, ali moze ako postoji globalno deljeno stanje).

Kako testirati i otkloniti:

- Pokrenuti integracioni testovi sa kontrolisanim lokalnim HTML fajlovima (fixtures).
- Osigurati deterministicki pipeline (deterministic producer i reproducibilno okruzenje).
- Dodati unit-testove koji pokrivaju parsePageRecords na istim HTML-ovima.

9.2 Bezbednost i etika scraping-a

- Uvek proveriti robots.txt ciljanog sajta i obracunati broj paralelnih zahteva tako da se server ne optereti.
- U produkciji obavezno postaviti odgovarajuci User-Agent koji sadrzi kontakt informacije (ako je potrebno).
- Ograniciti broj zahteva u jedinici vremena (rate limiting) i implementirati politiku za retry/backs-off.

10. Zakljucak

Parallel Web Scraper demonstrira efikasnost pipelining pristupa u kontekstu web scraping-a: pravilnim razdvajanje faza (producer, download, parse, store) i koriscenjem TBB-a postize se znatno veci throughput u odnosu na serijski pristup. Klasicni izazovi ostaju: razliciti HTML formati, transient mrezne greske, potreba za robustnim parsiranjem, i pitanje odgovornog scraping-a. Predlozi za dalje unapredjenje ukljucuju:

- Zamenu regex parsiranja sa DOM parserom (Gumbo/libxml2).
- Upotrebu curl_multi ili asinkronog modela za efikasniji I/O.
- Upotrebu konkurentnih struktura za UrlManager (tbb::concurrent_unordered_set).
- Shard-ovanu agregaciju rezultata da bi se smanjio overhead mutex-a.

11. Recnik pojmova

- **Web scraping:** automatizovano preuzimanje i izdvajanje podataka sa web stranica.
- **Pipeline:** niz faza obrade podataka koje se mogu paralelizovati i overlapat.
- **Throughput:** brzina obrade, obicno u stranicama po sekundi.
- **Race condition:** stanje gde vise niti istovremeno menja zajednicki resurs sto moze dovesti do nekonzistentnosti.
- **Mutex:** mehanizam koji omogucava ekskluzivan pristup delu koda/strukturi podataka.
- **Exponential backoff:** povecavanje cekanja izmedju ponovljenih pokusaja (200ms * 2^(attempt-1)).

- **tbb::concurrent_vector:** TBB konkurentni kontejner koji dozvoljava paralelne push_back operacije.