Programming Assignment 1 - Web Crawler

Luciano Belo de Alcântara Júnior*
luciano.alcantara@dcc.ufmg.br
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil

Abstract

Implementação de um web crawler utilizando políticas explícitas de seleção, revisitação, paralelismo, polidez e armazenamento.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Web Crawling, Crawler, WARC, Multiprocessamento, Politeness, Information Retrieval, Corpus Analysis

ACM Reference Format:
Luciano Belo de Alcântara Júnior. 2018. Programming Assignment 1 - Web Crawler. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

Introdução

Este projeto implementa um web crawler para coleta de páginas HTML. A estrutura de pastas segue padrões de Clean Architecture:

- **src/adapters**: entrada (argumentos, seeds) e saída (logs, WARC).
- **src/config**: configurações do sistema.
- **src/core**: lógica central (controller, fetcher, frontier, corpus).
- **src/domain**: tipos de dados (Page, PrioritizedURL).
- **src/infra**: acesso a infraestrutura externa (robots.txt).
- **src/shared**: funções auxiliares e utilitários (helpers, utils).

O arquivo principal *main.py* inicializa o Controller, que coordena todo o processo.

Estruturas de Dados, Algoritmos e Complexidade

O sistema é composto por quatro componentes centrais: Controller, Fetcher, Frontier e RobotsCache. Cada um adota estruturas de dados específicas conforme o que foi elucidado em sala de aula.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

Controller

O Controller é responsável por orquestrar o ciclo de coleta. Ele gerencia a inicialização das threads, o controle de concorrência (via `threading.Lock`), o armazenamento de páginas e a coordenação da frontier e do armazenamento WARC. Sua principal estrutura é a combinação da frontier (baseada em fila de prioridade) com contadores e estatísticas. As operações críticas de coleta e contagem possuem complexidade constante $O(1)$, enquanto a adição e remoção de URLs têm custo $O(\log n)$ devido à estrutura de heap utilizada na frontier.

Fetcher

O Fetcher realiza o download das páginas, aplicando a política de politeness (respeito ao `robots.txt` e ao `crawl-delay`). Para isso, mantém um cache de últimos acessos por domínio (`dict`), permitindo verificar rapidamente se um novo acesso é permitido. A busca no cache é feita em tempo constante $O(1)$. A complexidade principal do fetch é dominada pelo tempo de resposta da rede, não pelo processamento local.

Frontier

A Frontier gerencia as URLs pendentes de coleta. Ela é implementada como uma fila de prioridade (`heapq`), que garante que as URLs de maior prioridade sejam processadas primeiro, e um conjunto (`set`) para evitar revisitações. Inserções e remoções da fila custam $O(\log n)$, enquanto a verificação de duplicidade ocorre em $O(1)$.

RobotsCache

O RobotsCache é encarregado de armazenar os `robots.txt` previamente baixados, evitando downloads redundantes, utilizando a biblioteca *Protego*. Internamente, utiliza um dicionário (`dict`) para mapeamento domínio → parser. A busca no cache é $O(1)$, e o parsing do conteúdo do `robots.txt` é linear no tamanho do arquivo.

Funções Auxiliares

O sistema também conta com funções auxiliares importantes. A função `extract_outlinks` utiliza o BeautifulSoup para fazer parsing do HTML e extrair todos os links, de forma linear no tamanho do documento. Já a função `normalize_url` aplica normalizações padronizadas em URLs (remoção de inconsistências, adição de esquema padrão) a partir da biblioteca *url-normalize*, garantindo unicidade no conjunto de URLs, também com complexidade linear.

Gerenciamento de Armazenamento WARC

O envio das páginas do Controller para o processo gravador é realizado via `multiprocessing.Queue`, com custo amortizado de $O(1)$ por envio, salvo eventuais bloqueios caso a fila esteja cheia.

Dentro do processo *warc_writer_worker*, a gravação de cada página envolve:

- Conversão do HTML para bytes (codificação UTF-8) — custo $O(n)$, onde n é o tamanho do HTML.
- Criação do registro WARC usando a biblioteca `warcio` — custo $O(n)$.
- Escrita física no disco com compressão gzip — custo $O(n)$.

A rotação de arquivos, necessária a cada 1000 páginas gravadas, tem custo constante $O(1)$ amortizado.

Dessa forma, o custo computacional dominante do gerenciamento WARC é linear no tamanho da página HTML ($O(n)$). O uso de multiprocessamento garante que essas operações intensivas de I/O não bloqueiem a coleta de novas páginas, mantendo o throughput elevado.

Eficiência Empírica

Threads	Páginas/Segundo	Aceleração
2	1,78	1,00
6	3,51	1,97
12	1,80	1,01
24	3,19	1,79

O aumento de threads teve comportamento não linear:

- De 2 para 6 threads, houve ganho quase linear: a velocidade quase dobrou (1,78 → 3,51 páginas/s).
- Com 12 threads, o throughput permaneceu práticamente igual a 2 threads (1,80 páginas/s), indicando overhead por excesso de threads ou contenção no I/O.
- Com 24 threads, houve nova melhoria (3,19 páginas/s), equilibrando paralelismo e estabilidade.

Foi utilizado `os.cpu_count() * 2` (24) threads, padrão na aplicação. Apesar do ganho não ser estritamente proporcional, a escolha de utilizar os.cpu_count() * 2 (24 threads):

- Maximizaram o throughput de coleta.
- Mantiveram estabilidade de conexões sem aumento significativo de falhas.
- Aproveitaram eficientemente a capacidade do sistema para múltiplas requisições simultâneas.

Logo, os.cpu_count() * 2 representam um ponto de compromisso eficiente entre paralelismo e overhead de contexto, maximizando a taxa de download observada nos experimentos.

A viabilidade de criação de threads é validada em tempo de execução em `src/shared/utils/get_safe_thread_count.py`, com um valor seguro padrão caso o sistema não suporte o número sugerido.

Políticas de Crawling Implementadas

O sistema segue políticas de coleta explícitas conforme especificado:

- **Selection Policy**: Implementada no Fetcher, que filtra apenas documentos com `Content-Type: text/html`, ignorando arquivos binários ou não-HTML.
- **Revisitation Policy**: Gerenciada pela Frontier, que impede reprocessamento de URLs já coletadas usando um conjunto de URLs vistas.
- **Parallelization Policy**: Definida no Controller no método `run()`, onde o número de threads é controlado com base em `os.cpu_count()` e testes de viabilidade.
- **Politeness Policy**: Aplicada pelo Fetcher, que respeita *crawl-delay* extraído dinamicamente via RobotsCache. Se indisponível, um atraso mínimo de 100ms por padrão é aplicado.
- **Storage Policy**: Coordenada pelo Controller, utilizando multiprocessing para gravar páginas em WARC ou como HTML em disco, sem bloqueio das threads de coleta.

Caracterização do Corpus Rastreado

O corpus resultante da execução do crawler contém **100.000** páginas web provenientes de **1.510** domínios distintos. A coleta foi realizada totalizando um tempo de execução de aproximadamente **13,5 horas**, o que resultou em uma taxa média de coleta de **2,05 páginas por segundo**.

Distribuição de Páginas por Domínio

Observa-se uma distribuição altamente desbalanceada de páginas entre os domínios:

- O domínio `m.webnovel.com` concentra **60.485 páginas**, representando aproximadamente **60,48%** de todo o corpus.
- Em contraste, a maioria dos domínios possui apenas 1 ou 2 páginas coletadas (1240 domínios, cerca de 82%).
- A média de páginas por domínio é de **66,23**, mas a mediana é apenas **1**, indicando forte assimetria na distribuição.
- O desvio padrão de **1.677** reforça a alta dispersão no número de páginas por domínio.

Essa concentração de páginas em poucos domínios é consequência direta da estratégia de coleta adotada. Como o sistema não impõe limite de páginas por domínio e adiciona todos os outlinks válidos encontrados, domínios que possuem forte interconexão interna — como portais de leitura, fóruns ou redes sociais — acabam dominando a frontier. Dessa forma, o crawler tende a permanecer por longos períodos explorando intensamente um único domínio antes de alcançar outros, resultando na alta assimetria observada no corpus final.

Distribuição de Tokens por Página

A análise textual revelou:

- Total de **239.105.054 tokens** extraídos.
- Média de **2.391 tokens** por página.
- Mediana de **1.994 tokens**, com desvio padrão de **2.534,72 tokens**.

Esses valores indicam que, embora existam páginas muito extensas (com dezenas de milhares de tokens, sendo que o máximo foi 301068 tokens), a maioria das páginas tem um tamanho textual relativamente moderado. A alta variabilidade no número de tokens sugere a presença tanto de páginas curtas (como homepages ou páginas de erro) quanto de documentos longos (como capítulos de livros, artigos ou fóruns).

References

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009