Summary

Introduction

Crawler Architecture

Crawl Ordering Problem
Summary

Introduction

Crawler Architecture

Crawl Ordering Problem
Introduction

▶ A web crawler (also known as a robot or a spider) is a system for the downloading of web pages.

▶ They are used for a variety of purposes:
  ▶ Web Search Engines
  ▶ Web Archiving
  ▶ Web Data Mining
  ▶ Web Monitoring
Introduction

- The basic algorithm is simple. Given a set of seed URLs:
  1. downloads all the web pages addressed by the URLs;
  2. extracts the hyperlinks contained in the pages;
  3. iteratively downloads the web pages addressed by these hyperlinks.

- Despite the apparent simplicity of this basic algorithm, web crawling has many inherent challenges.
Challenges

- **Scale**
  - The web is very large and continually evolving.
  - Crawlers that seek coverage and freshness must achieve extremely high throughput, which poses difficult engineering problems.

- **Content selection tradeoffs**
  - Crawlers do not purport to crawl the whole web, or keep up with all the changes.
  - Goals: acquire high-value content quickly, ensure coverage of all reasonable content, and bypass low-quality, irrelevant, redundant, and malicious content.
  - The crawler must balance coverage and freshness, while obeying constraints such as per-site rate limitations.
Challenges

▶ **Social obligations**
  ▶ Crawlers should not impose too much of a burden on the web sites they crawl.
  ▶ Without safety mechanisms a high-throughput crawler can inadvertently carry out a *denial-of-service attack*.

▶ **Adversaries**
  ▶ Some content providers inject useless or misleading content into the corpus assembled by the crawler.
  ▶ Motivation: financial incentives, for example (mis)directing traffic to commercial web sites.
Summary

Introduction

Crawler Architecture

Crawl Ordering Problem
Architecture Overview

- Consists of multiple processes running on different machines connected by a high-speed network.
- Each process consists of multiple worker threads.
- Each worker thread performs repeated work cycles.
- At each work cycle, a worker obtains a URL from the **Scheduler** or **Frontier**, which organizes URLs according to priority and politeness policies.
- The worker thread invokes the **HTTP fetcher**:
  - It calls a DNS module to resolve the host component of the URL into the IP address of the corresponding web server
  - Connects to the web server and checks for robots exclusion files and attempts to download the web page.
Architecture Overview

- If the download succeeds the web page is stored in a repository.
- The Link extractor parses the HTML content of the page and extracts hyperlinks.
- The extracted URLs are passed to a URL distributor, which assigns each URL to a crawling process.
- This assignment is typically made by hashing the URLs host component, its domain, or its IP address (requires additional DNS resolutions).
- Most hyperlinks refer to pages on the same web site, thus assignment to the local crawling process is the common case.
Architecture Overview

- The URL passes through:
  - **Custom URL filter**: to exclude URLs belonging to "black-listed" sites and URLs with particular file extensions that are not of interest
  - **Duplicate URL eliminator**: maintains the set of all URLs discovered so far and passes on only never-before-seen URLs.
  - **URL prioritizer** selects a position for the URL in the Frontier or Scheduler, based on estimated page importance or rate of change.
Key Design Points

- Any web crawler needs to keep track the URLs.
- Operations that must be supported:
  - Adding a new URL
  - Retrieving a URL
  - Marking a URL as downloaded
  - Testing whether the set contains a URL
- In-memory data structures (e.g., trees or sorted lists) exceed the amount of memory available on a single machine.
Key Design Points

- Solution: maintain the data structure on disk.
- However, cost of accessing items in the set (e.g., set membership test) involves an expensive disk seek.
- Every modern web crawler splits the crawl state into:
  - **URL-seen test** or **Duplicate URL eliminator**: maintains the set of URLs that have been discovered (whether downloaded or not)
  - **Scheduler**: maintains the set of URLs that have yet to be downloaded.
Scheduler Data Structure and Politeness

- A straightforward implementation is a First-in-First-out (FIFO) queue, which results in a breadth-first traversal of the web graph.

- However, this simple approach has drawbacks:
  - Most hyperlinks refer to another page on the same web server, resulting in many consecutive HTTP requests to that server.
  - A barrage of requests is considered “impolite”, and may end in a denial-of-service attack.
Scheduler Data Structure and Politeness

- One solution: multithreaded or distributed crawler that issues many HTTP requests in parallel.
- Avoid issuing multiple overlapping requests to the same server.
  - Maintain a mapping from web servers to crawling threads, e.g., by hashing the host component of each URL.
  - Each crawling thread has a separate FIFO queue, and downloads only URLs obtained from that queue.
Scheduler Data Structure and Politeness

- Politeness policy: space out requests to each web server according to that server’s capabilities.
  - E.g., delay subsequent requests by a multiple (say $10 \times$) of the time it took to download the last page from that server.

- This policy ensures that:
  - The crawler consumes a bounded fraction of the web server’s resources.
  - Fewer pages will be downloaded from slow or poorly connected web servers than from fast, responsive web servers.
Scheduler Data Structure and Politeness

- It may be desirable to prioritize pages according to their estimated usefulness, based on:
  - PageRank
  - Traffic they receive
  - Reputation of the web site
  - Rate at which the page has been updated in the past
  - Etc.
URL Seen Test

- The second major data structure: **URL-seen test** (UST) or the **duplicate URL eliminator** (DUE)
- It keeps track of the set of URLs that have been previously discovered and added to frontier.
- UST needs to support:
  - Insertion
  - Set membership testing
- In a continuous crawling setting, it must also support deletion to cope with URLs that no longer point to a valid page.
In a disk-based hash table, each lookup requires a disk seek, severely limiting the throughput.

Caching popular URLs can increase the throughput.

However, the average web page contains close to hundred links and each link needs to be tested for novelty.

Implementations performing random file accesses perform poorly, but those that perform streaming sequential reads or writes achieves good throughput.
URL Seen Test

- The IRLbot crawler [84] uses:
  - Batch of URLs arriving at the DUE is written to disk, distributed over multiple files keyed by the prefix of each hash.
  - Once the size of the largest file exceeds a certain threshold, the files that together hold the batch are read back into memory one by one and merge-sorted into the main URL hash file on disk.
Auxiliary Data Structures: Robots File

- **Robots Exclusion Protocol**: allows an administrator to bar crawlers from crawling the site, or pages within the site.
- Provides a file at URL `/robots.txt` containing rules that specify which pages the crawler is allowed to download.
- A crawler should check whether the site supplies a `/robots.txt` file, and if so, adhere to its rules.
- To avoid repeatedly requesting `/robots.txt`, crawlers cache the results of previous requests of that file.
Auxiliary Data Structures: DNS

- URLs contain a host component (e.g., www.yahoo.com), which is “resolved” using the Domain Name Service (DNS) protocol.
- DNS requests can take quite a long time due to the request-forwarding nature of the protocol.
- Therefore, crawlers often maintain their own DNS caches.
Incremental Web Crawling

- Web crawlers can be used to:
  - Assemble one or more static snapshots of a web corpus (batch crawling)
  - Perform *incremental* or *continuous crawling*: download newly discovered pages and re-download previously crawled pages.

- Efficient incremental crawling requires a few changes to the major data structures of the crawler.
  - The DUE should support the deletion of URLs that are no longer valid (e.g., that result in a 404 HTTP return code).
  - URLs are retrieved from the scheduler and downloaded as in batch crawling, but they are subsequently reentered into the scheduler.
Summary

Introduction

Crawler Architecture

Crawl Ordering Problem
Crawl Ordering Problem

- The crawl order is extremely important:
  - due to the growth rate of new content
  - due to dynamically generated content

- A good crawler order must consider:
  - **Coverage**. The fraction of desired pages that the crawler acquires successfully.
  - **Freshness**. The degree to which the acquired page snapshots remain up-to-date, relative to the current “live” web copies.
Simplified Crawl Ordering Model

Diagram showing the crawl ordering model with a sequence of pages (P1 to P14) and their relationships with the web, crawl history, page downloader, crawl ordering policy, and page repository.
At a given point in time, some historical crawl order has already been executed (P1, P2, P3, P4, P5 in the diagram).

Some future crawl order has been planned (P6, P7, P4, P8, ...).
Model

- Pages downloaded by the crawler are stored in a repository.
- The future crawl order is determined, at least in part, by analyzing the repository.
- The content of a web page is subject to change over time.
- It is sometimes desirable to re-download a page that has already been downloaded, to obtain a more recent snapshot of its content.
Model: Two Approaches

- **Batch crawling:**
  - The crawl order does not contain duplicate occurrences of any page.
  - The entire crawling process is periodically halted and restarted as a way to obtain more recent snapshots of previously crawled pages.

- **Incremental crawling:**
  - Pages may appear multiple times in the crawl order.
  - Crawling is a continuous process that conceptually never terminates.
  - Most modern commercial crawlers perform incremental crawling, which is more powerful because it allows re-visititation of pages at different rates.
Web Characteristics

- One notable study is by Broder et al. [26], which uncovered a “bowtie” structure consisting of:
  - a central strongly connected component (the core).
  - a component that can reach the core but cannot be reached from the core.
  - a component that can be reached from the core but cannot reach the core.
- In addition, there are a number of small, irregular structures such as disconnected components and long “tendrils”.
In an earlier study, Broder et al. [28] showed that there is an abundance of near-duplicate content of the web.

Using a corpus of 30 million web pages collected by the AltaVista crawler, they found that:

- 29% of the pages were more than 50% similar to other pages in the corpus
- 11% of the pages were exact duplicates of other pages.

Sources of near-duplication include mirroring of sites (or portions of sites) and URL synonymy.
Taxonomy of Crawl Ordering Policies

- Next figure presents a high-level taxonomy of published crawl ordering techniques.
- The first group of techniques focuses exclusively on ordering pages for first-time downloading, which affects coverage.
  - Can be applied either in the batch crawling scenario, or in the incremental crawling scenario in conjunction with a separate policy of the second group.
- The second group of techniques governs re-downloading of pages to maintain freshness.
- Techniques in the third group consider the combined problem of interleaving first-time downloads with re-downloads, to balance coverage and freshness.
## Taxonomy of Crawl Ordering Policies

<table>
<thead>
<tr>
<th>Technique</th>
<th>Objectives</th>
<th>Factors considered</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coverage</td>
<td>Importance</td>
</tr>
<tr>
<td>Breadth-first search [43, 95, 108]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Prioritize by indegree [43]</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prioritize by PageRank [43, 45]</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prioritize by site size [9]</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prioritize by spawning rate [48]</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prioritize by search impact [104]</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Scoped crawling (Section 4.2)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Minimize obsolescence [41, 46]</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Minimize age [41]</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Minimize incorrect content [99]</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Minimize embarrassment [115]</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Maximize search impact [103]</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Update capture (Section 5.2)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>WebFountain [56]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OPIC [1]</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
References


References


References


References


References


References


References


References


References


References


References


References


References


