Modern Information Retrieval

Chapter 12

Web Crawling

with Carlos Castillo

Applications of a Web Crawler
Architecture and Implementation
Scheduling Algorithms
Crawling Evaluation
Extensions
Examples of Web Crawlers
Trends and Research Issues

Introduction and a Brief History

- A **Web Crawler** is a software for downloading pages from the Web
- Also known as Web Spider, Web Robot, or simply Bot
- Cycle of a **Web crawling process**
  - The crawler start downloading a set of **seed pages**, that are parsed and scanned for new links
  - The links to pages that have not yet been downloaded are added to a central queue for download later
  - Next, the crawler selects a new page for download and the process is repeated until a stop criterion is met

Introduction and a Brief History

- The first known web crawler was created in 1993 by Matthew Gray, an undergraduate student at MIT
- On June of that year, Gray sent the following message to the `www-talk` mailing list:

  "I have written a perl script that wanders the WWW collecting URLs, keeping tracking of where it's been and new hosts that it finds. Eventually, after hacking up the code to return some slightly more useful information (currently it just returns URLs), I will produce a searchable index of this."
- The project of this Web crawler was called WWWW (World Wide Web Wanderer)
- It was used mostly for Web characterization studies

Introduction and a Brief History

- On June 1994, Brian Pinkerton, a PhD student at the University of Washington, posted the following message to the `comp.infosystems.announce` news

  "The WebCrawler Index is now available for searching! The index is broad: it contains information from as many different servers as possible. It's a great tool for locating several different starting points for exploring by hand. The current index is based on the contents of documents located on nearly 4000 servers, world-wide."

- The WebCrawler become a commercial success
- Other search engines based on Web crawlers appeared: Lycos (1994), Excite (1995), Altavista (1995), and Hotbot (1996)
- Currently, all major search engines employ crawlers

Applications of a Web Crawler

Applications of a Web Crawler

- a Web Crawler can be used to
  - create an index covering broad topics (**general Web search**)
  - create an index covering specific topics (**vertical Web search**)
  - archive content (**Web archival**)
  - analyze Web sites for extracting aggregate statistics (**Web characterization**)
  - keep copies or replicate Web sites (**Web mirroring**)
  - Web site analysis

General Web Search

- Web search has driven Web crawling development during the last years
- Types of Web search
  - **General Web search:** done by large search engines
  - **Vertical Web search:** the set of target pages is delimited by a topic, a country or a language
- Crawler for general Web search must balance coverage and quality
  - **Coverage**: It must scan pages that can be used to answer many different queries
  - **Quality**: The pages should have high quality

Vertical Web Search

- **Vertical Crawler**: focus on a particular subset of the Web
- This subset may be defined geographically, linguistically, topically, etc.
- Examples of vertical crawlers
  - **Shopbot**: designed to download information from on-line shopping catalogs and provide an interface for comparing prices in a centralized way
  - **News crawler**: gathers news items from a set of pre-defined sources
  - **Spambot**: crawler aimed at harvesting e-mail addresses inserted on Web pages

Vertical Web Search

- Vertical search also includes segmentation by a data format
- In this case, the crawler is tuned to collect only objects of a specific type, as image, audio, or video objects
- Example
  - **Feed crawler**: checks for updates in RSS/RDF files in Web sites

Topical Crawling

- **Focused crawlers**: focus on a particular topic
- Provides a more efficient strategy to avoid collecting more pages than necessary
- A focused crawler receives as input the description of a topic, usually described by
  - a driving query
  - a set of example documents
- The crawler can operate in
  - **batch mode**, collecting pages about the topic periodically
  - **on-demand**, collecting pages driven by a user query

Web Characterization

- **Web characterization** includes all attempts to derive statistical properties of Web pages
- Difficult questions regarding Web characterization
  - what constitutes a representative sample of the Web?
  - what constitutes the Web of a country?
- Crawled pages deeply affect the results of the characterization
  - page-centered characterization are less affected than link-centered characterization efforts
- The seed URLs must be chosen carefully

Mirroring

- **Mirroring** is the act of keeping a partial or complete copy of a Web site
- Crawlers used for mirroring are usually simpler
- Mirroring policy includes:
  - The refreshing period, typically daily or weekly
  - The time of the day to do the mirroring

Web Archiving

- **Web archiving** is a mirroring without discarding the outdated copies
  - That is, the whole history of each page is recorded
- The largest project of Web archiving is the **Internet Archive**:
  - URL: http://www.archive.org/
  - Its main purpose is to preserve the state of the Web on each year
  - Collection of March 2006 consisted of 55 billion of pages

Web Site Analysis

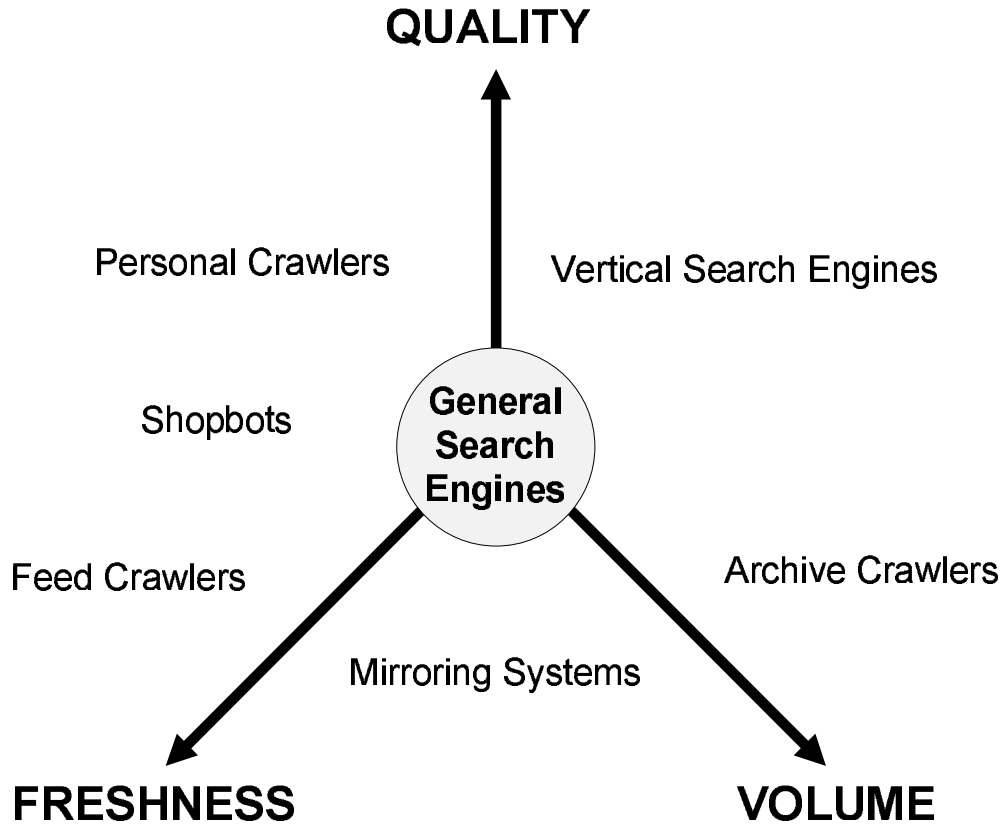
- A Web crawler can be used to analyze a Web site and even change it on the fly according to a predefined criteria
- Examples of automatic site analysis:
  - **Link validation**: scanning the pages of a site for broken links
  - **Code validation**: ensures that all pages of a site are well-formed
  - **Web directories analysis**: looking for sites that are no longer available
- A site analysis tool can also be used to find vulnerabilities in Web sites
  - Including to find older and unpatched versions of popular scripts

Web Site Analysis

- In large text repositories (such as Wikipedia), Web crawlers can be used to automate many tasks, including:
  - Categorization, to ensuring that all pages in a set conform to a standard
  - Detection of images with unknown copyright status
  - Detection of orphan (unlinked) pages

Taxonomy of Crawlers

- The crawlers assign different importance to issues such as **freshness**, **quality**, and **volume**
- The crawlers can be classified according to these three axes



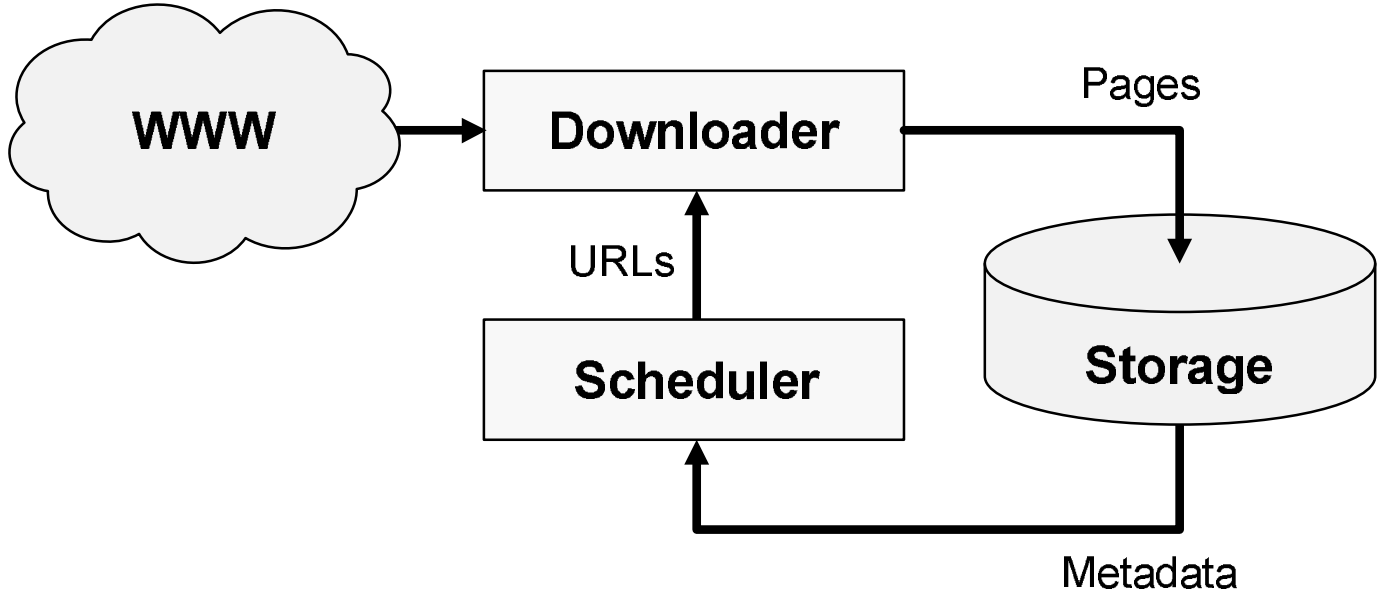
Politeness

- A crawler would like to use all the available resources as much as possible (crawling servers, Internet bandwidth)
- However, crawlers should also fulfill **politeness**
  - That is, a crawler cannot overload a Web site with HTTP requests
  - That implies that a crawler should wait a small delay between two requests to the same Web site
  - Later we will detail other aspects of politeness

Architecture and Implementation

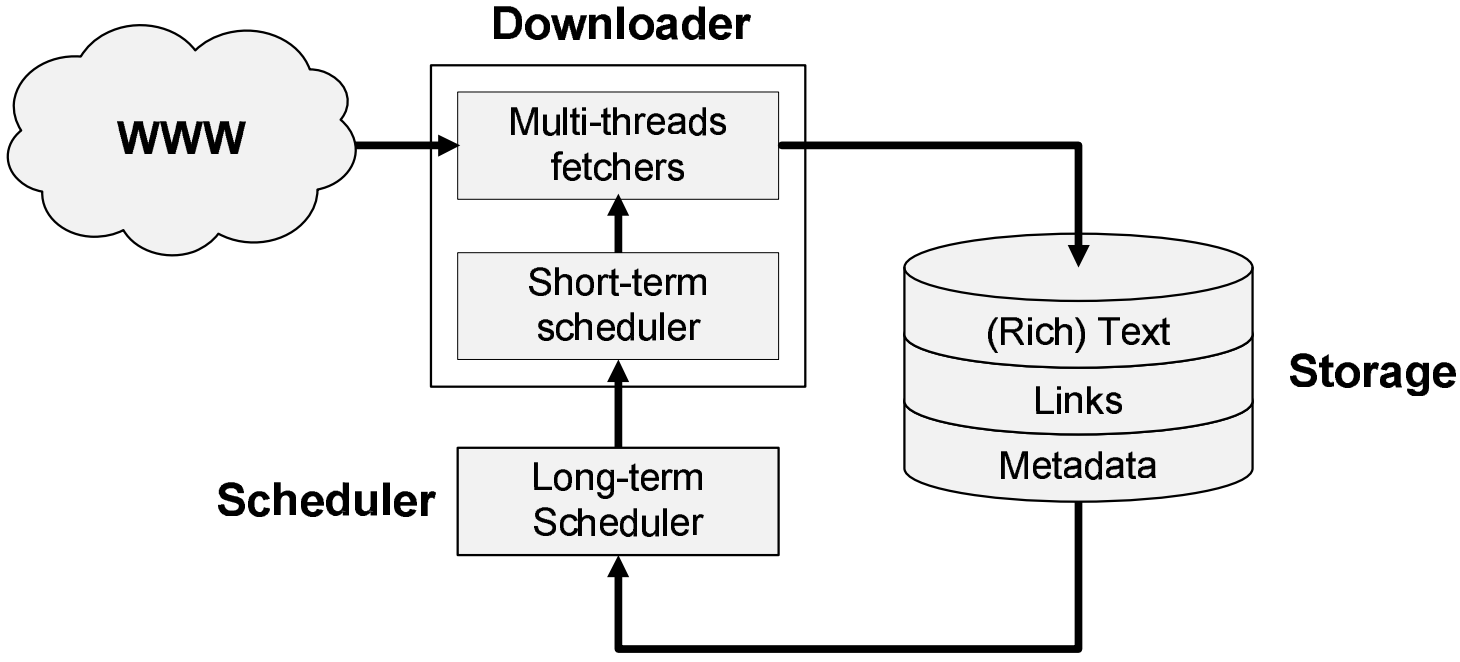
Architecture of the Crawlers

- The crawler is composed of three main modules: downloader, storage, and scheduler
  - **Scheduler**: maintains a queue of URLs to visit
  - **Downloader**: downloads the pages
  - **Storage**: makes the indexing of the pages, and provides the scheduler with metadata on the pages retrieved



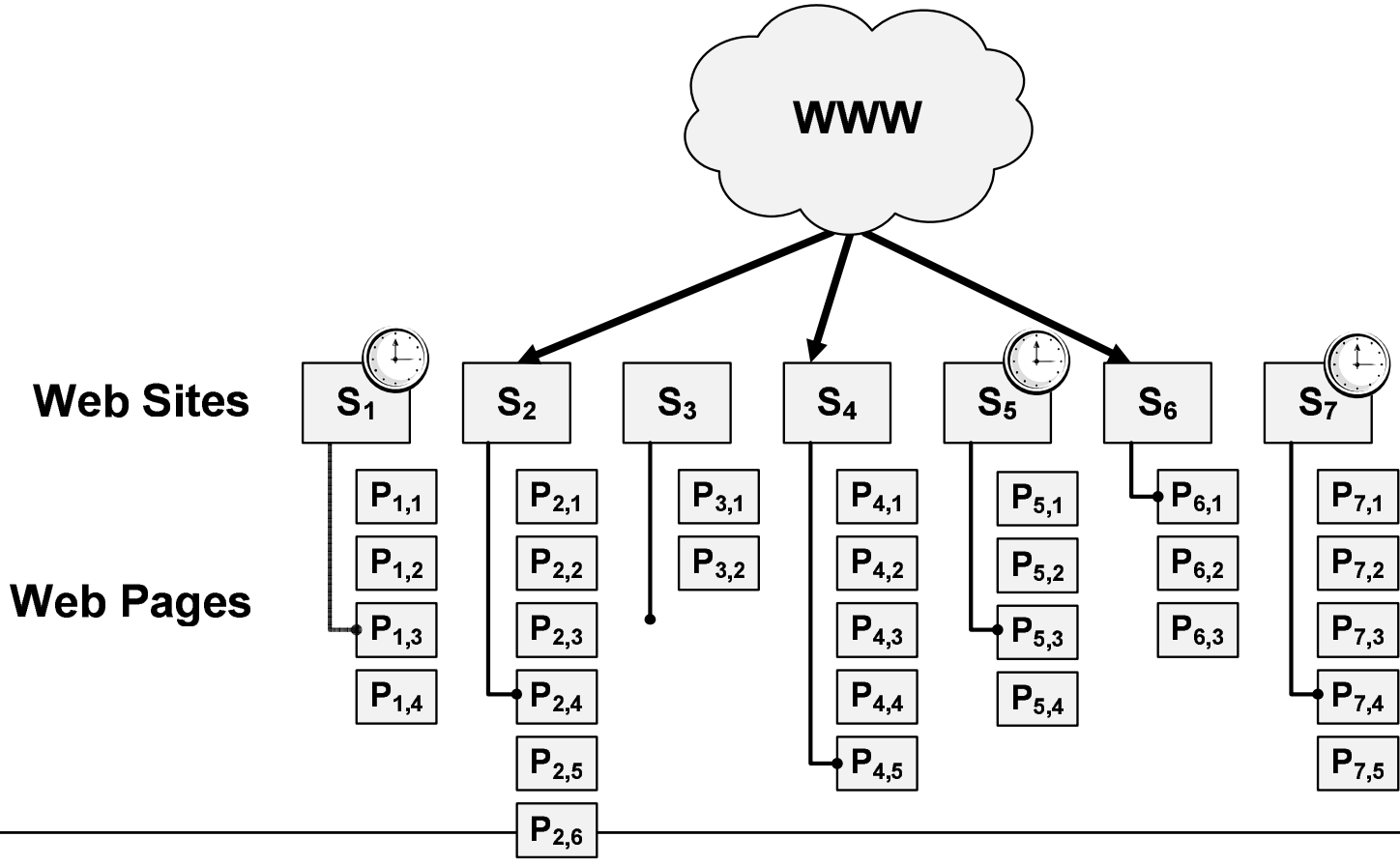
Architecture of the Crawlers

- The scheduling can be further divided into two parts:
  - **long-term scheduling**: decide which pages to visit next
  - **short-term scheduling**: re-arrange pages to fulfill politeness
- The storage can also be further subdivided into three parts: (rich) text, metadata, and links



Architecture of the Crawlers

- In the **short-term scheduler**, enforcement of the politeness policy requires maintaining several queues, one for each site, and a list of pages to download in each queue

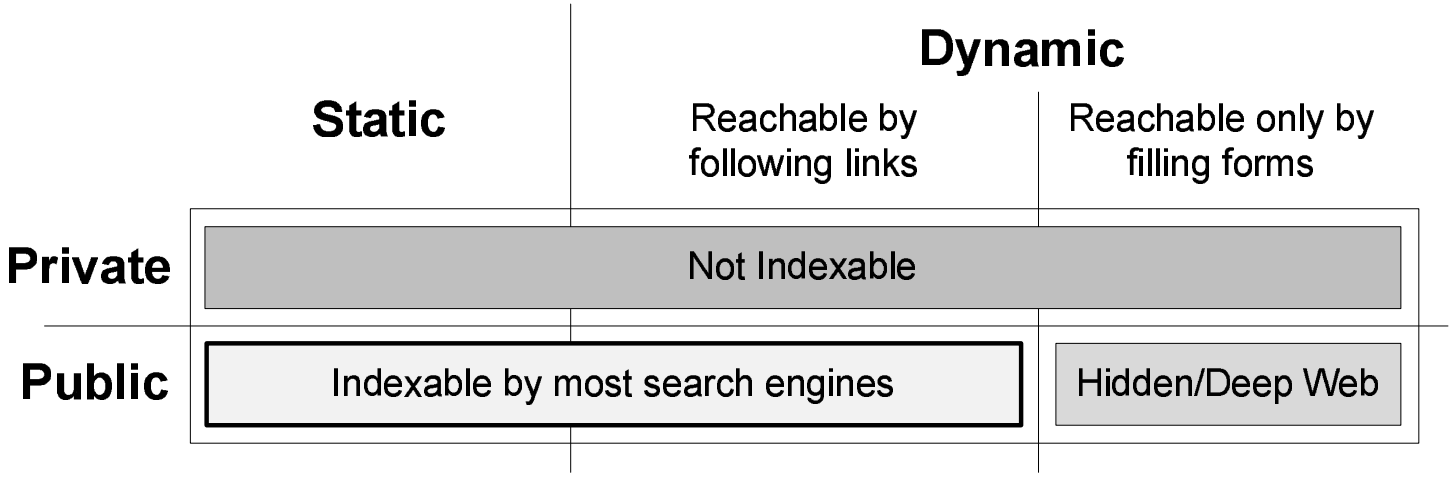


Practical Issues

- The implementation of a crawler involves many **practical issues**
- Most of them is due to the need to interact with many different systems
- Example: How to download maintaining the traffic produced as uniform as possible?
  - The pages are from multiple sources
  - DNS and Web server response times are highly variable
  - Web server up-time cannot be taken for granted
- Other practical issues are related with: types of Web pages, URL canonization, parsing, wrong implementation of HTML standards, and duplicates

Taxonomy of Web pages

- A challenging aspect is related with the taxonomy of Web pages
- Web pages can be classified in **public or private**, and **static or dynamic**
  - There are in practice infinitely many dynamic pages
  - We cannot expect a crawler to download all of them
  - Most crawlers choose a maximum depth of dynamic links to follow



Wrong HTML

- Most Web browsers are very tolerant with HTML wrongly coded
  - This has led to very poor quality in the HTML coding
  - The parser module of the crawler must allow for mistakes in the HTML coding
- In many cases it is difficult to tell if a link is broken or no
  - Some servers use a custom-built error page to show that a page does not exist, without send a response header signaling the error condition
  - *Bar-Yosef et al* refer to these error pages as **soft-404**, and observe that 29% of dead links point to them
  - Some Web crawlers test Web sites by sending a URL that (probably) does not exist

Duplicates

- The prevalence of mirrored content on the web is high
- Types of duplicates
  - **intentional duplicates**: mirroring of other pages
  - **unintentional duplicates**: the result of the way that many Web sites are built
- The worst of unintentional are caused by identifiers embedded in the URLs to track user's behavior
  - e.g.: `/dir/page.html&jsessid=09A89732`
  - a Web crawler must be aware of session-ids and try to keep a consistent session-id across requests

Granularity of Information

- Blogs, Web forums, and mailing list archives: large repositories of information comprised of many small postings
- Useful source of information when the topic is not covered somewhere else
- However, sometimes individual postings are not valuable
- A Web crawler might index only the pages that aggregate information

Parallel and Distributed Crawling

- To achieve better scalability and be more tolerant to failures, Web crawling should be done in parallel and distributed fashion
- In this case, the most important issue is to avoid downloading the same page more than once and/or overloading Web servers
- The coordination among processes is done by exchanging URLs
- The goal of the crawler designer is to minimize the communication overhead
  - Ideally, every page should be downloaded by a single process

Parallel and Distributed Crawling

- A fully distributed crawling system requires a policy for assigning the new URLs discovered
- The decision of which process should download a given URL is done by an **assignment function**
- *Boldi et al* state that an effective assignment function must have three main properties:
  - **Balancing property**: each crawling process should get approximately the same number of hosts
  - **Contra-variance property**: if the number of crawling processes grows, the number of hosts assigned to each process must shrink
  - The assignment must be able to add and remove crawling processes dynamically

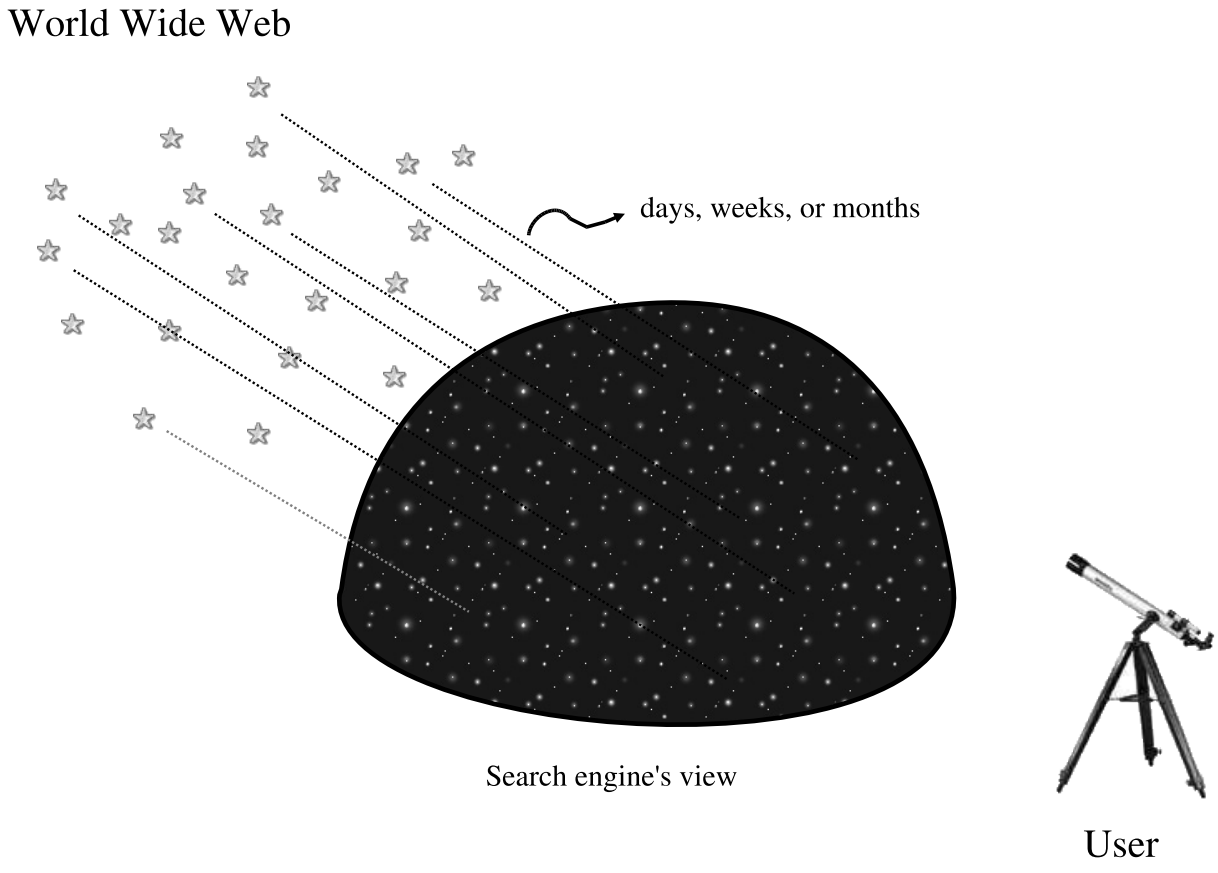
Scheduling Algorithms

Scheduling Algorithms

- A Web crawler needs to balance various objectives that contradict each other
  - It must download new pages and seek fresh copies of downloaded pages
  - It must use network bandwidth efficiently avoiding to download bad pages
  - However, the crawler cannot know which pages are good, without first downloading them
- To further complicate matters there is a huge amount of pages being added, changed and removed every day on the Web

Scheduling Algorithms

- Crawling the Web, in a certain way, resembles watching the sky in a clear night: the star positions that we see reflects the state of the stars at different times



Scheduling Algorithms

- The simplest crawling scheduling is traversing Web sites in a breadth-first fashion
  - This algorithm increases the Web site coverage
  - And is good to the politeness policy by not requesting many pages from a site in a row
- However, can be useful to consider the crawler's behavior as a combination of a series of policies
- To illustrate, a crawling algorithm can be viewed as composed of three distinct policies
  - **selection policy**: to visit the best quality pages, first
  - **re-visit policy**: to update the index when pages change
  - **politeness policy**: to avoid overloading Web sites

Selection Policy

- Even large search engines cover only a portion of the publicly available content
- *Gulli et al* showed in 2005 that the coverage of large search engines is between 58% and 76% of the Web
- It is highly desirable that such downloaded fraction contains the most authoritative pages
- As *Edwards et al* noted

  "Given that the bandwidth for conducting crawls is neither infinite nor free it is becoming essential to crawl the Web in a not only scalable, but efficient way if some reasonable measure of quality or freshness is to be maintained."

Selection Policy

- A crawler must carefully choose, at each step, which pages to visit next
- The selection of which pages to crawl can be divided into two types of restrictions
  - **Off-line limits** that are set beforehand
  - **On-line selection** that is computed as the crawl goes by

Off-line Limits

- Due to storage limitations, in practice, it is frequently necessary to establish beforehand limits for the crawling process
- The off-line limits used more frequently by Web crawlers are the following
  - A maximum number of hosts to be crawled
  - A maximum depth (a maximum number of links to be traversed starting from any home page)
  - A maximum overall number of pages in the collection
  - Maximum number of pages or bytes downloaded from each server
  - A list of accepted mime-types for downloading (e.g.: `text/html` and `text/plain`)

On-line Selection

- A crawler requires a metric of importance for prioritizing Web pages
- The importance of a page may be a function of
  - its intrinsic quality
  - its popularity in terms of links or visits
  - its URL (in the case of vertical search engines restricted to a top-level domain or to a fixed Web site)
- Another difficulty: the crawler must work with partial information, as the complete set of Web pages is not known during crawling

Policies for On-line Selection

- *Cho et al* made the first study on policies for crawling scheduling
- A crawling simulation in stanford.edu domain was done with different strategies
- The ordering metrics tested were breadth-first, backlink-count and partial Pagerank
- Their conclusion: the Partial Pagerank strategy is better, followed by breadth-first and backlink-count

Policies for On-line Selection

- Najork and Wiener crawled pages from different domains, using breadth-first ordering
- They found that a breadth-first crawl captures pages with high Pagerank early in the crawl
- The explanation of the authors: pages with high Pagerank have many links to them, and they are found early

Policies for On-line selection

- Abiteboul proposed an algorithm called OPIC (On-line Page Importance Computation) to guide on-line selection
- In OPIC, each page is given an initial sum of *cash* which is distributed equally among the pages it points to
- This is similar to a Pagerank computation, but it is faster as is done in only one step
- An OPIC-driven crawler downloads first the pages in the crawling frontier with the higher amounts of cash

Policies for On-line Selection

- *Boldi et al* used simulation to testing breadth-first against random ordering
- The winning strategy was breadth-first, although a random ordering also performed surprisingly well
- They also showed that PageRank calculations carried on partial subgraphs of the Web are a poor approximation of the actual PageRank
- Then if those partial calculations are used to guide the crawl a strong bias against good pages appears

Policies for On-line Selection

- *Baeza-Yates et al* also used simulation to testing several crawling strategies
- They showed that both the OPIC strategy and a strategy that uses the length of the per-site queues are both better than breadth-first crawling
- Also, even when the Web changes rapidly, it is very effective to use the information gathered in a previous crawl to guide the current one

Focused Crawling

- A particular case of on-line selection policy is the filtering of pages according to a given topic
- The topic description consists usually of a driving query and sometimes also of a set of example documents
- If the topic is described by example documents, then those documents are used as seed pages
- Relevance to the topic is inferred using
  - the driving query
  - the example pages
  - the pages seen so far during the crawl

Focused Crawling

- Focused crawling exploits **topical locality on the Web**
  - Pages that link to each other are more likely to be on the same topic than pages chosen at random
  - Related pages tend to be co-cited
- As more pages are collected, a feedback effect can be taken in advantage, such that noisy pages can be effectively identified

Focused Crawling

- Main problem of focused crawling: to predict the relevance of a page before downloading the page
- A good source of information are the anchor texts of the links
- *Diligenti et al* proposed an approach based in **context graphs**, in which the content of relevant pages is used to infer the relevance of non-downloaded pages

Re-visit Policy

- The Web has a very dynamic nature
- By the time a Web crawler has finished its crawl, many events might have happened
- We characterize these events as creations, updates and deletions:
  - **Creations**: when a new page is created (and becomes accessible by a link) it can be crawled as determined by the visit politic
  - **Updates**: an update can be either minor (occurs at the paragraph or sentence level), or major (all references to its content are not valid anymore)
  - **Deletions**. Undetected deletions of pages are more damaging for a search engine's reputation than updates

Modeling of Page Events

- From the search engine's point of view, there is a cost associated with not detecting an event
- The cost functions used more frequently are **freshness** and **age**
- Freshness is a binary measure that indicates whether the local copy is up-to-date or not
- The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Modeling of Page Events

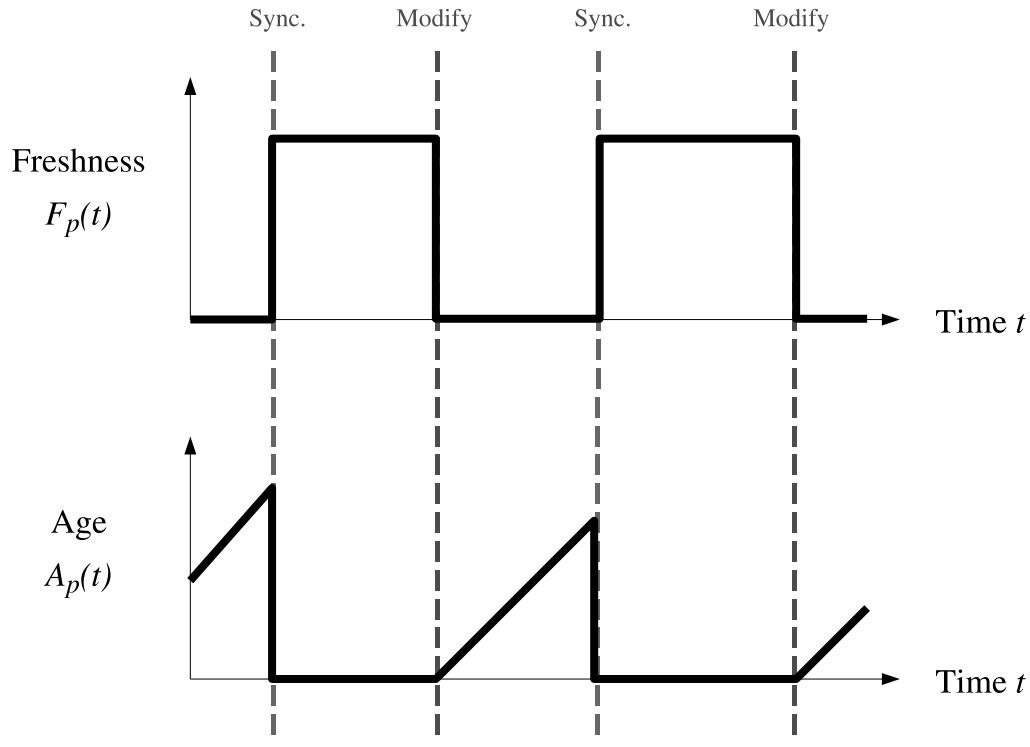
- **Age** is a measure that indicates how outdated the local copy is
- The age of a page p in the repository, at time t is defined as

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ has not been modified at time } t, \\ & \text{since last update} \\ t - lu(p) & \text{otherwise} \end{cases}$$

  - where $lu(p)$ is the last update time for the page p

Modeling of Page Events

- Evolution of freshness and age as time: two types of events can occur
  - **Event modify**, modification of a Web page in the server
  - **Event sync**, downloading of the modified page by the crawler



Strategies

- The objectives of the crawler can be:
  - to keep the average freshness of pages in the collection as high as possible, or
  - to keep the average age of pages in the collection as low as possible
- These are not equivalent objectives:
  - In the first case, the crawler is just concerned with how many pages are out-dated
  - In the second case, the crawler is concerned with how old are the local copies of the pages

Strategies

- Cho and Garcia-Molina studied two re-visiting policies: uniform and proportional
- **Uniform policy**: re-visiting all pages in the collection with the same frequency
- **Proportional policy**: re-visiting more often the pages that change more frequently
- They proved that, in terms of average freshness, the uniform policy is better
- In the proportional policy, the crawler will waste time by trying to re-crawl pages that change too often
- The authors state that to improve freshness, it is necessary to penalize the elements that change too often

Strategies

- Neither the uniform policy nor the proportional policy are optimal
  - The optimal method for keeping average freshness high includes ignoring the pages that change too often
  - The optimal for keeping average age low is to use access frequencies that monotonically increase with the rate of change of each page
- In both cases, the optimal is closer to the uniform policy than to the proportional policy

Strategies

- Explicit formulas for the re-visit policy are not attainable as they depend on the distribution of page changes
- If the distributions are known, an optimal re-visit policy may be obtained numerically
- In general, most large search engines use two or three queues with different turnout times
  - a queue for news sites that is refreshed several times a day
  - a daily or weekly queue for popular or relevant sites
  - and a large queue for the rest of the Web
- Note that the re-visiting policies considered here regard all pages as homogeneous in terms of quality

Estimating Freshness

- For each page p the following information becomes available after every visit:
  - The access time-stamp of page p: visit_p
  - The last-modified time-stamp of page p (provided by most Web servers): modified_p
  - The text of the page, which can be compared to an older copy to detect changes, especially if modified_p is not provided

Estimating Freshness

- The following information can be estimated if the re-visiting period is short enough:
  - The time at which the page first appeared: $\mathrm{created}_p$
  - The time at which the page was no longer reachable: $\mathrm{deleted}_p$
- Note that, in all cases, the results are only an estimation of the actual values

Estimating Freshness

- The probability $u_p(t)$ that a copy of p is up-to-date at time t, decreases with time if the page is not re-visited
- *Brewington and Cybenko* considered that if changes to a given page occur at independent intervals, then this can be modeled as a Poisson process
- In this case, if t units of time have passed since the last visit, then

$$u_p(t) \propto e^{-\lambda_p t}$$

  - where the parameter λ_p corresponds to the average rate of change of the page p

Estimating Freshness

- Let's assume we have visited a page N_p times, and in X_p visits a change has been observed
- Further, let T_p the total time from changes of the page, which is computed as follows
  - If during a visit the page is found not modified, then the time since the last visit is added to this accumulator
  - If during a visit the page is found modified, then the time since its last-modification time is added to this accumulator
- Then λ_p can be estimated as:

$$\lambda_p \approx \frac{(X_p - 1) - \frac{X_p}{N_p \log(1 - X_p/N_p)}}{T_p}$$

Characterization of Page Changes

- While there are different time-related metrics for a Web page, the most used are:
  - Age: $\text{visit}_p - \text{modified}_p$
  - Lifespan: $\text{deleted}_\text{p} - \text{created}_p$
  - Number of changes during the lifespan: changes_p
  - Average change interval: $\text{lifespan}_p / \text{changes}_p$
- Based on these values, useful metrics for the entire sample can be calculated, such as
  - Distribution of change intervals
  - Average lifespan of pages
  - Median lifespan of pages

Politeness Policy

- The use of Web robots, while useful for a number of tasks, comes with a price for the general community
  - Web crawlers require considerable bandwidth
  - They can create server overload, specially if the frequency of access to a given server is high, and/or if the robot is poorly written
- Privacy is also an issue with Web crawlers
  - They may, for instance, access parts of a Web site that were not meant to be public
  - If the crawler keeps a cache of pages, copyright issues which are currently not enforced may arise

Politeness Policy

- A set of guidelines is also important for the continued operation of a Web crawler
- A crawler that is impolite with a Web site may be banned by the hosting provider
- The three basic rules for Web crawler operation are:
  - A Web crawler must identify itself as such, and must not pretend to be a regular Web user
  - A Web crawler must obey the robots exclusion protocol (robots.txt)
  - A Web crawler must keep a low bandwidth usage in a given Web site

Politeness Policy

- To illustrate, the four larger search engines follow these rules, as indicated below
  - Ask search

    `http://about.ask.com/en/docs/about/webmasters.shtml`
  - Google (Googlebot)

    `http://www.google.com/webmasters/bot.html`
  - MSN Search

    `http://www.msnsearch.com/msnbot.htm`
  - Yahoo! Search (Slurp!)

    `http://help.yahoo.com/help/us/ysearch/slurp/`
- Next, we discuss in more detail the issues of robot identification, exclusion protocol, and bandwidth usage

Robot Identification

- Sometimes, the navigational pattern of a Web crawler may be detected by a Web server
- However, this detection is more effective if the Web crawler identifies itself as such in the first place
- The HTTP protocol includes a **user-agent field** that can be used to identify who is issuing a request
- The user-agent field of a Web crawler should include an address to a Web page containing information on the crawler, as well as contact information
- When this information is not present, Web site administrators might send complaints to the listed owner of the entire originating network segment

Robot Exclusion Protocol

- The robot exclusion protocol involves three types of exclusion: server-wide, page-wise exclusions, and cache exclusions
- **Server-wide exclusion** instructs the crawler about directories that should not be crawled
- This is done via a single `robots.txt` file that is located in the root directory of a Web site
- The `robots.txt` file below indicates that all crawlers should not download the directories `/data/private` and `/cgi/bin`:

```
User-agent:  *
Disallow:  /data/private
Disallow:  /cgi-bin
```

Robot Exclusion Protocol

- **Page-wise exclusion** is done by the inclusion of meta-tags in the pages themselves
- Meta-tags are part of the standard HTML syntax and allow a page author to associate pairs of the form `key=value` to Web pages
- To illustrate, the meta-tag below indicates that crawlers should neither index this page nor follow links contained on it

```
<meta name="robots"
content="noindex,nofollow"/>
```

Robot Exclusion Protocol

- **Cache exclusion** is used by publishers that sell access to their information
- While they allow Web crawlers to index the pages, they instruct search engines not to show the user a local cached copy of the page
- This is done in the same HTML tag as the page-wise exclusion by using the nocache keyword, as follows:

```
<meta name="robots" content="nocache"/>
```

- Even with all the precautions, a crawler might access pages that were not meant to be public
- Because of this, it is important to have a fast way of removing a document from the local collection

Controlling Bandwidth Usage

- The bandwidth available for a crawler is usually much higher than the bandwidth of the Web sites it visits
- Using multiple threads, a Web crawler might easily overload a Web server, specially a smaller one
- To avoid this, it is customary
  - to open only one connection to a given Web server at a time
  - to take a delay between two consecutive accesses
- *Cho et al* suggested adopting 10 seconds as the interval between consecutive accesses, the WIRE crawler uses 15 seconds as the default, and other crawlers use 15 or 30 seconds

Controlling Bandwidth Usage

- Recently, several Web crawlers allow Web site operators to decide which is the delay that should be used when indexing their site
- This is done by the robots exclusion protocol, including in the `robots.txt` a line specifying a crawl-delay, such as instance:

  ```
  Crawl-delay: 45
  ```

  which instructs Web crawlers to wait 45 seconds between consecutive accesses

Combining Policies

- The behavior of the crawler can be separated into two parts
  - a **short-term scheduling**, dealing with the politeness policy
  - a **long-term scheduling**, dealing with selection and freshness
- A natural combination of these policies is to consider the profit obtained from downloading a single Web page
- Suppose that a local page has an estimated quality q and that its probability of being up-to-date is p
  - Then we can consider that the value of the page in the index is $q \times p$

Combining Policies

- If we download the page now, its probability of being up-to-date becomes 1, so its value becomes q
  - Then, the expected profit of downloading the page is $q \times (1 - p)$
- A natural policy then is to sort pages by expected profit
- Other types of decay can be used to account for the pages that are not “fresh” in the repository
  - For instance, the value of a page can be $q \times p^{\alpha}$, where α is a exponent that makes freshness more or less important compared to quality
- Notice that to have more pages we need to crawl new pages instead of refreshing pages, but the only way to find new pages is refreshing pages that have a new link in them

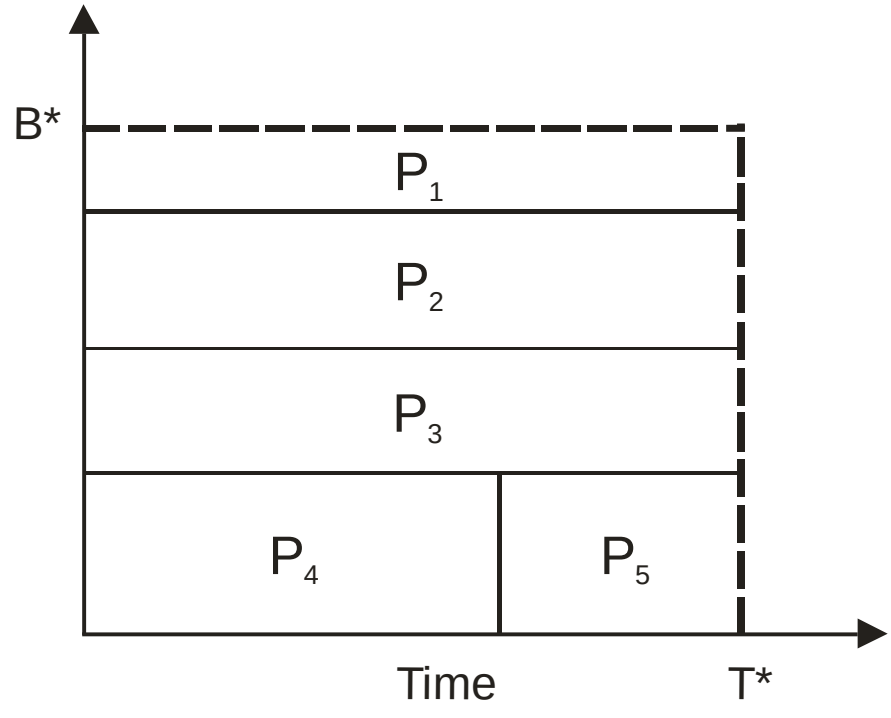
Crawling Evaluation

Evaluating Network Usage

- The diagram below depicts an optimal Web crawling scenario for an hypothetical batch of five pages
- The x-axis is time and the y-axis is speed, so the area of each page is its size (in bytes)
- The downloader has maximum bandwidth B^* so the crawl can be completed in time:

$$T^* = \frac{\sum_i \text{size}(P_i)}{B^*}$$

where $\text{size}(P_i)$ is the size of page P_i

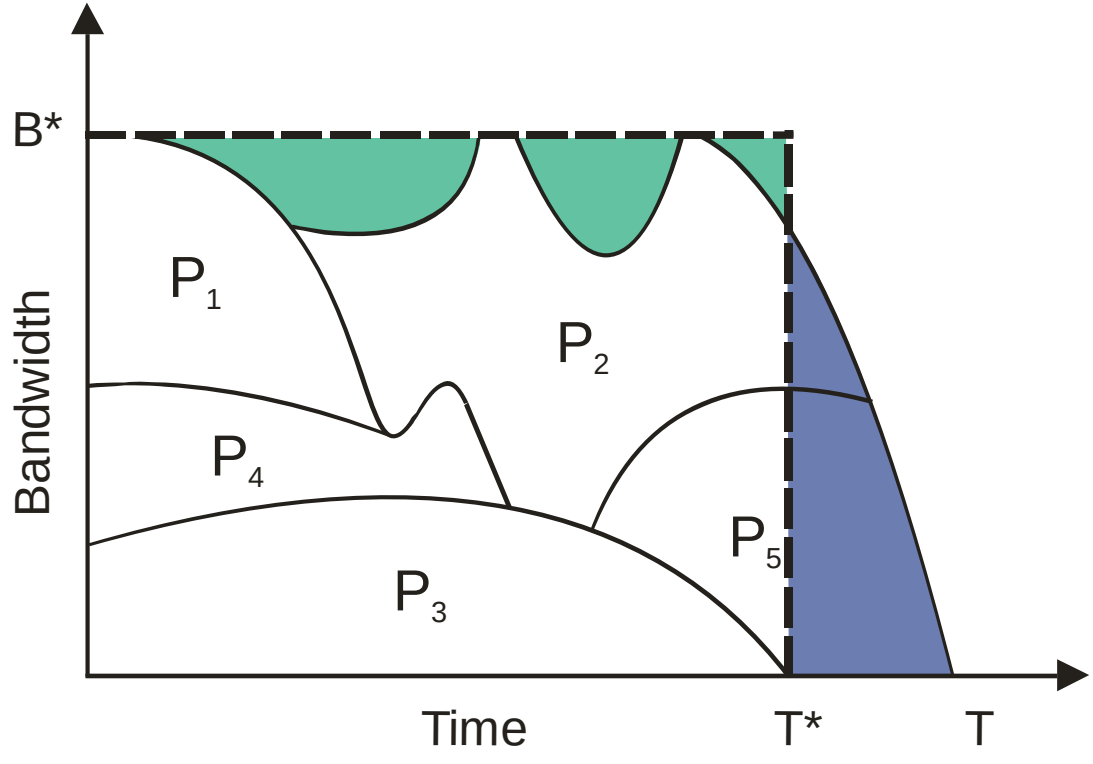


Evaluating Network Usage

- Let us consider now a more realistic setting in which:
  - The speed of download of every page is variable and bounded by the effective bandwidth to a Web site (a fraction of the bandwidth that the crawler would like to use can be lost)
  - Pages from the same site can not be downloaded right away one after the other (politeness policy)

Evaluating Network Usage

- Under these assumptions, a different crawling timeline may occur, such as the one depicted on the figure below
- In the realistic case, the total time T is larger than the optimal case
- The bandwidth lost can be measured and is given by $B^* \times (T - T^*)$
- In the figure, green and blue areas are equal



Evaluating Network Usage

- By the end of the batch of pages, it is very likely that only a few hosts are active
- Once a large fraction of the pages have been downloaded it is reasonable to stop the crawl, if only a few hosts remain at the end
- Particularly, if the number of hosts remaining is very small then the bandwidth cannot be used completely

Evaluating Network Usage

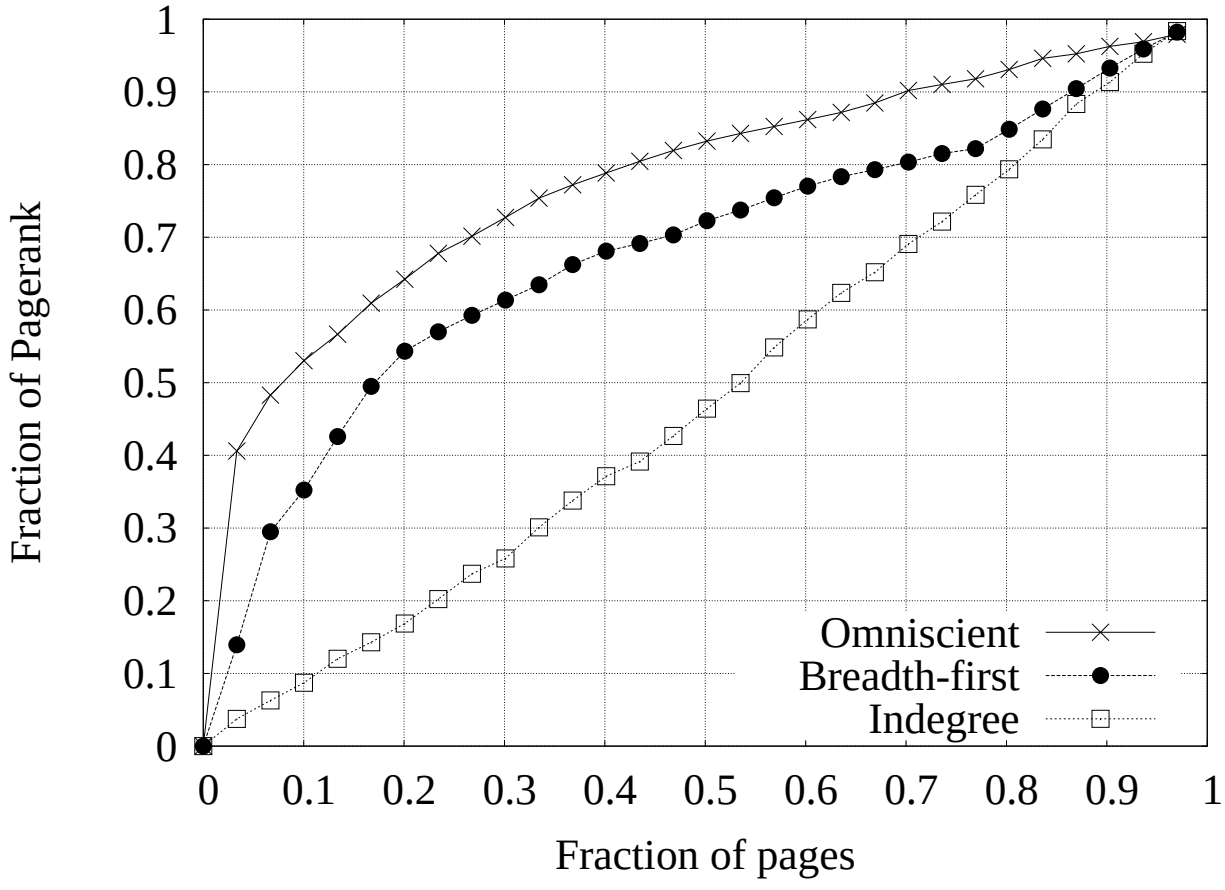
- A short-term scheduler may improve its finishing time by saturating the bandwidth usage, through the use of more threads
- However, if too many threads are used, the cost of switching control among threads becomes prohibitive
- The ordering of the pages can also be optimized by avoiding having a few sites to choose from at any point of the batch

Evaluating Long-term Scheduling

- For comparing strategies for long-term scheduling, many authors use the following idea.
  - First, a metric that estimates the quality of Web pages is calculated for every page in the collection
  - For each strategy the pages are sorted in the order in which they are downloaded
  - Then for each page the cumulative sum of the values of the chosen quality metric is calculated

Evaluating Long-term Scheduling

- The result of such experiment is a graph similar to the one below (from *Baeza-Yates et al*, using PageRank as a quality metric)



Evaluating Long-term Scheduling

- The crawling strategies simulated in the graph are in-degree, breadth-first and omniscient
  - **Breadth-first** implements a FIFO queue to store newly URLs and uses a FIFO discipline to schedule URLs for downloading
  - **In-degree** selects the next page to be downloaded by counting all in-links to it from already downloaded pages and selecting the largest one
  - **Omniscient** is an idealized strategy that queries an *oracle* that knows the complete Web graph and knows the actual Pagerank of each page beforehand
- If Pagerank is used as quality metric, the graph indicates that the omniscient strategy performs better than the other two strategies

Extensions

Crawling the Hidden Web

- The process we have described allows the indexing of all the Web that is reachable by just following links
- *Raghavan and Garcia-Molina* observed that there is an enormous amount of information not reachable by following links, but only by querying or filling forms
- This fraction of the Web is known as the **hidden or deep Web** (*Bright-Planet*)
- It was estimated in the year 2000 as 550 times larger (in terms of pages) than the full Web
- More recent studies suggest that the size of the deep Web might be even larger (*Chang et al*)

Crawling the Hidden Web

- To crawl the hidden Web, first a set of pages found by normal crawling is parsed to search for forms
- These forms are then converted to an internal representation
- Then one or several relevant queries are formulated and issued, and the results are parsed and stored by the crawler

Crawling with the Help of Sites

- So far we have considered that Web servers are passive regarding the crawl, but this is not necessarily so
- The response to the posting describing the first Web crawler back in 1993 already suggested this possibility:

  "Wouldn't it be better if you could just ask each server for it's connectivity? Seems like this would make things run a lot faster. Since each server has local access to all the information it could just find all the HREFs real quick, unique them and report to someone else."

Crawling with the Help of Sites

- The early search engine **Aliweb** actually used this idea
  - Web servers had to advertise their contents in a local file
- Unfortunately, most Web authors were too lazy to implement these systems
- Hence, most of modern Web crawlers consider Web servers as passive entities

Crawling with the Help of Sites

- Nowadays, the only help a Web crawler receives from most Web sites is related to technologies that also help regular users
- This includes *if-modified-since requests* that are used to verify if a Web page has changed
- The most used is the very simple *RSS ping*, which is a scheme used by blogs and news sources to notify search engines and news aggregators of updates
- Actually, what a RSS ping does is to ask the news aggregator to schedule a re-crawl for the RDF feed published by the content provider
- This is expected to change in the future, as RDF feeds are becoming increasingly common

Examples of Web Crawlers

Global-scale Crawlers

- The Internet Archive uses a crawler called **Heritrix**,
  - It was designed with the purpose of archiving periodic snapshots of a large portion of the Web
  - It uses several processes in a distributed fashion, and a fixed number of Web sites are assigned to each process
- The early architecture of **Google** is described by Sergey Brin and Lawrence Page
  - The crawler was integrated with the indexing process
  - During parsing, the URLs found were passed to a URL server that checked if the URL have been previously seen
  - If not, the URL was added to the queue of the URL server

Global-scale Crawlers

- The architecture of the **FAST Search Engine** was described by Risvik and Michelsen
  - It is a distributed architecture in which each machine holds a *document scheduler*
  - Each scheduler maintains a queue of documents to be downloaded by a *document processor*
  - Each crawler communicates with the other crawlers via a *distributor* module

Modular Web Crawlers

- **Mercator** is a modular Web crawler written in Java
  - Its modularity arises from the usage of interchangeable *protocol modules* and *processing modules*
  - Protocols modules are related to how to acquire the Web pages
  - Processing modules are related to how to process Web pages
  - Other processing modules can be used to index the text of the pages, or to gather statistics from the Web

Modular Web Crawlers

- **WebFountain** is a distributed, modular crawler similar to Mercator
  - It features a *controller* machine that coordinates a series of *ant* machines
  - It also includes a module for stating and solving an equation system for maximizing the freshness
- **WebSPHINX** is composed of a Java class library and a development environment for web crawlers
  - It implements multi-threaded Web page collector and HTML parser, and a graphical user interface to set the starting URLs

Open Source Web Crawlers

- **NUTCH** is an open-source crawler written in Java that is part of the Lucene search engine
  - It is sponsored by the Apache Foundation
  - It includes a simple interface for intranet Web crawling as well as a more powerful set of commands for large-scale crawl
- **WIRE** is an open-source web crawler written in C++
  - Includes several policies for scheduling the page downloads
  - Also includes a module for generating reports and statistics on the downloaded pages
  - It has been used for Web characterization
- Other crawlers described in the literature include ht://Dig (in C++), WebBase (in C), CobWeb (in Perl), PolyBot (in C++ and Python), and WebRace (in Java)

Trends and Research Issues

Trends and Research Issues

- There are many research topics related to Web crawling, such as
  - Improving the selection policy, in the sense of developing strategies to discover relevant items early during the crawl
  - Improving memory and network usage
  - Crawling for acquiring facts, including crawling the semantic Web
  - Doing crawling in other environments, such as in peer-to-peer services, etc.