Einführung in Webspider

Dr. Christian Herta

June 8, 2009

Outline

Web-Crawler

- Aufgaben eines Web-Crawler
  - Den Inhalt von vielen Internet-Dokumenten herunterzuladen, um sie für weitere Anwendungen zur Verfügung zu stellen, z.B. Indizieren für eine Web-Suche.
  - Finden von neuen *URLs*
- Synonyme: (Web-)Spider, Roboter

Muss-Anforderungen an Crawler

- *Robustness* bezüglich *Spider traps, broken links, timeouts*
- *Politeness*: Verhindern der Überlastung von Webserver
- **Legal**: Seiten, die von Web-Mastern als nicht-spidern gekennzeichnet sind, dürfen nicht gespidert werden

Soll-Anforderungen an Crawler (1)

- **Quality and Coverage**: Die heruntergelagenen Seiten müssen nützlich bezüglich der Anwendung sein und eine gute Abdeckung erreichen
- **Aktualität**: Wenn der Crawler in einem kontinuierlichem Modus läuft, soll er möglichst bald geänderte Seiten neu herunterladen
- **Extensibility - Heterogeneous nature**: Crawler sollen modular aufgebaut sein, so dass sie leicht erweiterbar sind, z.B. auf neue (Netzwerk-)Protokolle und Datenformate.

Soll-Anforderungen an Crawler (2)

- **Skalierbarkeit** (*Scalability*) in der Anzahl der Web-Seiten N, die der Crawler verarbeiten kann.
- **Verteilbarkeit**: Dies erlaubt ein *Scale up* durch Hinzufügen von mehr Maschinen und erhöhter Netz-Bandbreite.
- **Performanz** (*Performance*): Die Geschwindigkeit S (Anzahl der pro Zeiteinheit verarbeiteten Seiten). S ist in der Regel eine Funktion der bisher verarbeiteten Seiten.
- Nutzen der Resourcen (**Resource Usage**): CPU und RAM Resourcen Σ die nötig sind um die Anzahl n an Seiten herunterzuladen.

Outline

Spider Traps

- *Spider Traps* können unterschiedliche Ursachen haben, z.B.:
  - *Link Farmen*: dynamisch generierte Links- und Webseiten (Link-Spam)
  - schlecht programmierte Webseiten - Zyklen durch symbolische Links im Dateisystem
- So ist es möglich, dass der Crawler in "Schleifen" gerät und auf Domains verbleibt.
  - Spider Traps sind an der größen Zahl von heruntergeladenen Seiten einer Domain erkennbar
  - Aber es gibt auch Domains, die eine wiklich große Anzahl von verschiedenen Seiten haben (große Portale und Verlagsseiten)

Anforderung

Spider soll robust bezüglich *Spider Traps* sein

URL Aliases und Duplikate

- *URL Aliases* sind URLs die auf den gleichen *Content* zeigen
- Verschiedene Ursachen:
  - *Host name aliases*: unterschiedliche Hostnamen für gleiche IP
  - Alternative Pfade auf dem gleichen Host
  - Replikationen zwischen unterschiedlichen Hosts
  - Angegebene bzw. ausgelassene Portnummern

Anforderung

Es soll falls möglich- vermieden werden, gleiche Seiten mehrfach herunterzuladen.

Anforderung

Duplikate sollen erkannt werden.

Politeness Policies

- Würde man viele Seiten parallel von einem Webserver herunterladen, würden viele Server unter der Last zusammenbrechen $\Rightarrow$ *Politeness Policies*
  - Nicht mehr als eine Seite parallel von einem Host herunterladen
  - *politeness window*: Wartezeit zwischen zwei Anfragen (im Sekundenbereich)
- logische Aufteilung der *request queue*:*single queue* pro Host/Web-Server

/robots.txt

- in der Resource `/robots.txt` können *Web-Master* Angaben zum gewünschten Crawling-Verhalten machen:
  - Welche Bereiche von welchen Spidern nicht gecrawled werden dürfen.
  - Zeitliche Verzögerungen
- wichtig: Caching der `/robots.txt` Inhalte
- Es können auch direkt auf Websiten über *robot Meta-Tags* Angaben zum Verhalten gemacht werden, z.B. "noindex"

Beispiel für robot.txt

```
User-agent: *
Dissallow: /archiv/
Dissallow: /forum/
Allow: /public

User-agent: WebReaper
Dissallow: /

User-agent: Slurp
Crawl-delay: 18

Sitemap: http://mysite.de/sitemap.xml.gz
```

Quality and Coverage

- Ausgehend von einer Menge von Start-URLs Ω_0 (*Seeds*) folgt der Crawler den Hyper-Links in Web-Dokumenten. So erhält er neue URLs, aus denen wiederum die URLs der Hyper-Links extrahiert werden. So kann sich der Crawler im Internet bewegen.
- Die Menge aller Seiten, die ausgehend von Ω_0 erhalten werden kann, ist Ω_∞.
- In der Regel erhält man nur eine Teilmenge von Ω_∞. $\Omega_U \subseteq \Omega_\infty$ sei die Menge der für die Anwendung nützlichen Seiten.
- Ein möglichst gute Abdeckung (*Coverage*) von Ω_U soll von Crawler geleistet werden.

Deep Web

- Nicht alle Teile des Internets sind von Crawlwern zu erreichen $\longrightarrow$ *Deep Web*
- Die meisten Siten des *Deep Webs* gehören folgenden Kategorien an:
  - *Private sites*: Durch Passworte geschützt oder keine eingehenden Links
  - *Form Results*: Seiten die nur erreicht werden, wenn Daten in ein Formular (*Form*) eingetragen werden, z.B. Flugdaten. Kann durch Erraten durch den Crawler teilweise zugänglich gemacht werden.
  - *Scripted Pages*: Falls Links zu Seiten nur innerhalb *Client-seitige* Sprachen, wie JavaScript, Flash ausgedrückt werden. Technisch möglich, aber langsam.
- *statische* vs. *dynamische* Webseiten

Aktualität

- Wenn eine heruntergeladene Webseite (Kopie) der aktuellen Webseite im Netz entspricht, sagt man sie ist *fresh* oder *up-to-date*
- Andernfalls ist sie *stale*
- per *HTTP Head Request* kann der *response header* einer *resource* ausgelesen werden
- Im *Header* gibt es Angaben die zur Aktualitätsprufung genutzt werden können:
  - `Last-Modified` (optional)
  - `ETag`
  - `Content-MD5` (optional)
- Angaben sind nicht immer vorhanden oder zuverlässig

Metriken für die Aktualität

- Fragestellung:
  - Wie soll ein Crawler optimiert werden, damit die heruntergeladenen Seiten möglichst aktuell sind?
  - Wie kann die Aktualität eingeschätzt werden?
  - Welche Seiten sollen überprüft bzw. neu gelagen werden?

Metriken für die Aktualität

- Fragestellung:
  - Wie soll ein Crawler optimiert werden, damit die heruntergeladenen Seiten möglichst aktuell sind?
  - Wie kann die Aktualität eingeschätzt werden?
  - Welche Seiten sollen überprüft bzw. neu gelagen werden?
- Hierzu braucht man Metriken für die Aktualität der heruntergeladenen Webseiten; nach Cho et. all. [1]:
  - *fressness*
  - *age*

Metrik *Freshness*

Freshness einer heruntergeladenen Seite e_i zur Zeit t

$$F_i(t) = \begin{cases} 1, & \text{falls } e_i \textit{ up-to-date} \text{ zur Zeit } t \\ 0, & \text{sonst} \end{cases} \quad (1)$$

Freshness der lokalen Kopien zur Zeit t

$$F_S(t) = \frac{1}{N} \sum_{i=0}^{N} F_i(t) \quad (2)$$

Problem mit der Metrik *Freshness*

- Schafft man es nicht gewisse Seiten *up-to-date* zu halten, so wäre bei einer Optimierung bezüglich der *Freshness* es am besten, diese Seiten gar nicht mehr zu *crawlen*.
- Mittels der erwarteten *Freshness* kann man auch zeigen, dass diese Metrik nicht geeignet ist

Metrik *Age*

Age einer heruntergeladenen Seite e_i zur Zeit t

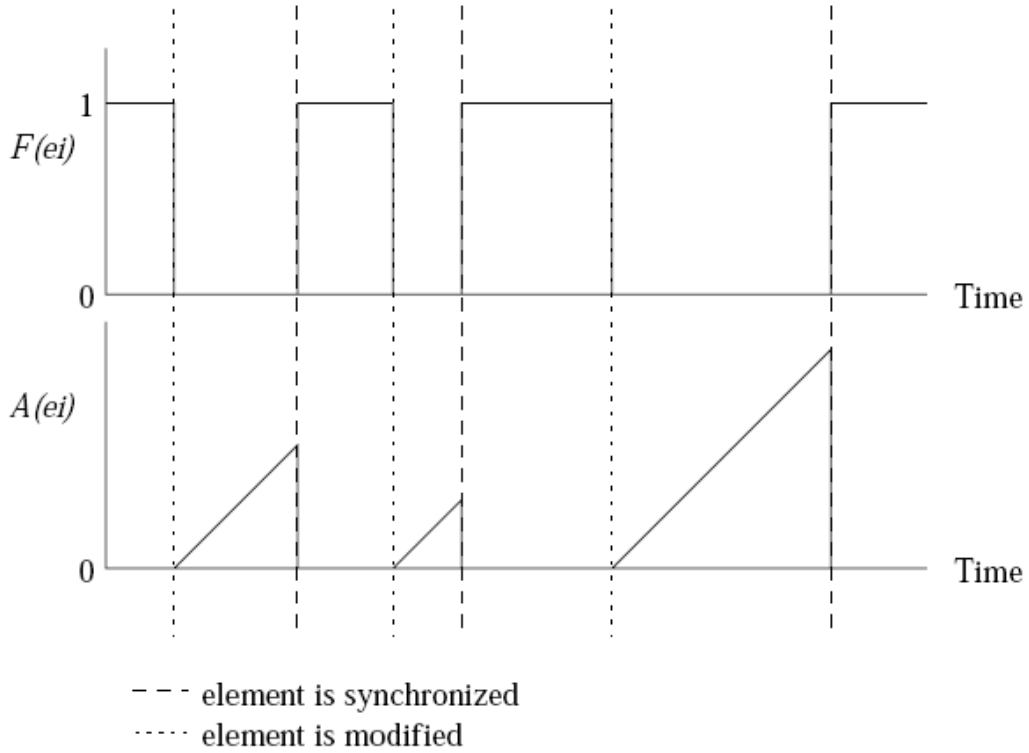
$$A_i(t) = \begin{cases} 0, & \text{falls } e_i \textit{ up-to-date} \text{ zur Zeit } t \\ t - t_{i,mod}, & \text{sonst} \end{cases} \quad (3)$$

mit $t_{i,mod}$: Zeit der Änderung von e_i

Age der lokalen Kopien zur Zeit t

$$A_S(t) = \frac{1}{N} \sum_{i=0}^{N} A_(t) \quad (4)$$

Beispielverlauf von *freshness* und *age*



An example of the time evolution of $F(e_i;t)$ and $A(e_i;t)$

Erwartete *freshness* einer Seite

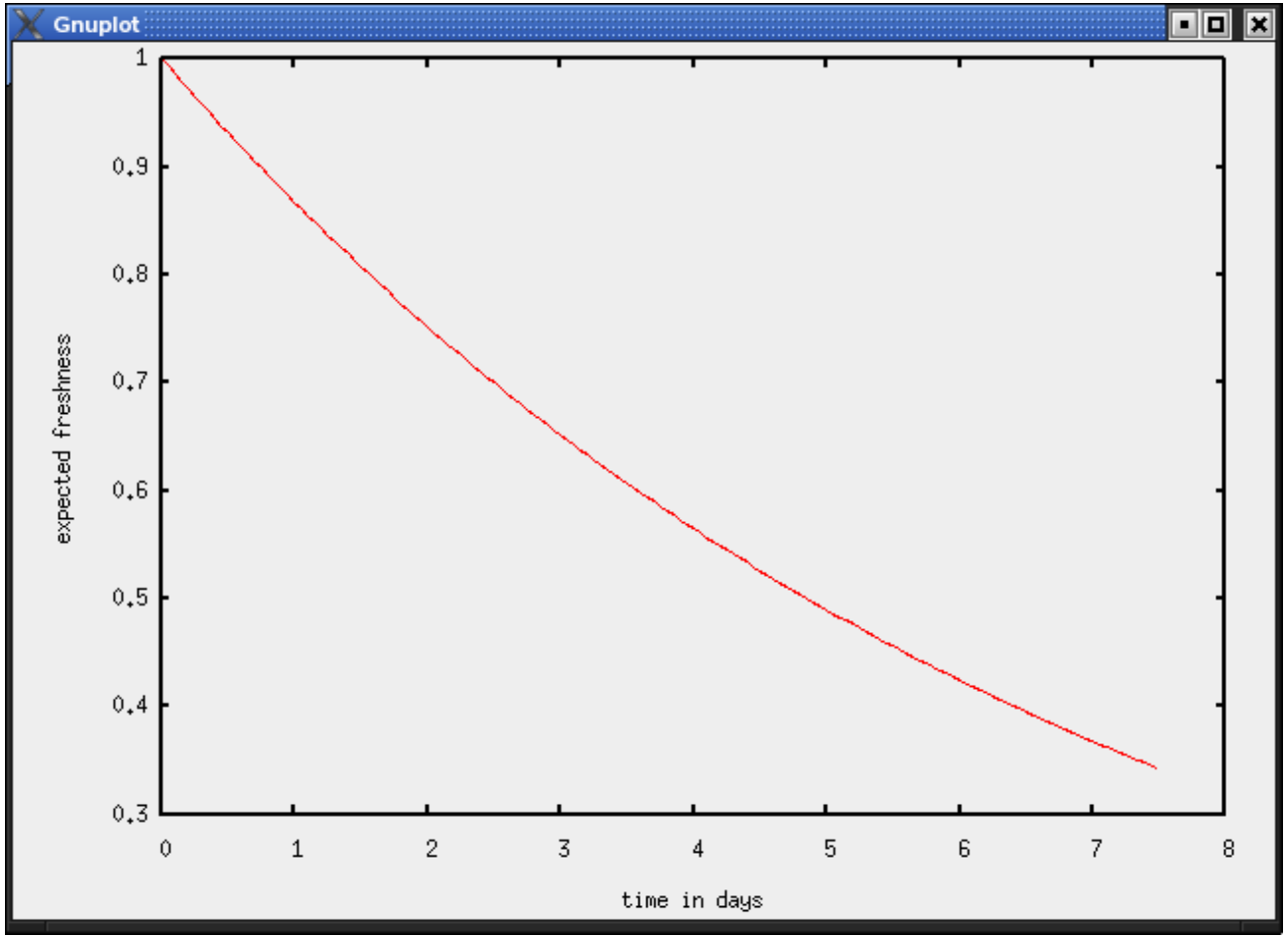
- Annahme: Poisson-Prozess - Modellierung als zufälliger und unabhängige Sequenz von Ereignissen - Details siehe [1]

$$\overline{F_i}(\lambda, t) = e^{-\lambda t} \tag{5}$$

mit

- λ: Änderungsrate (Einheit: $1/t$)

Expected freshness as function of time; $\lambda = 1/7 days$



Erwartetes *age* einer Seite

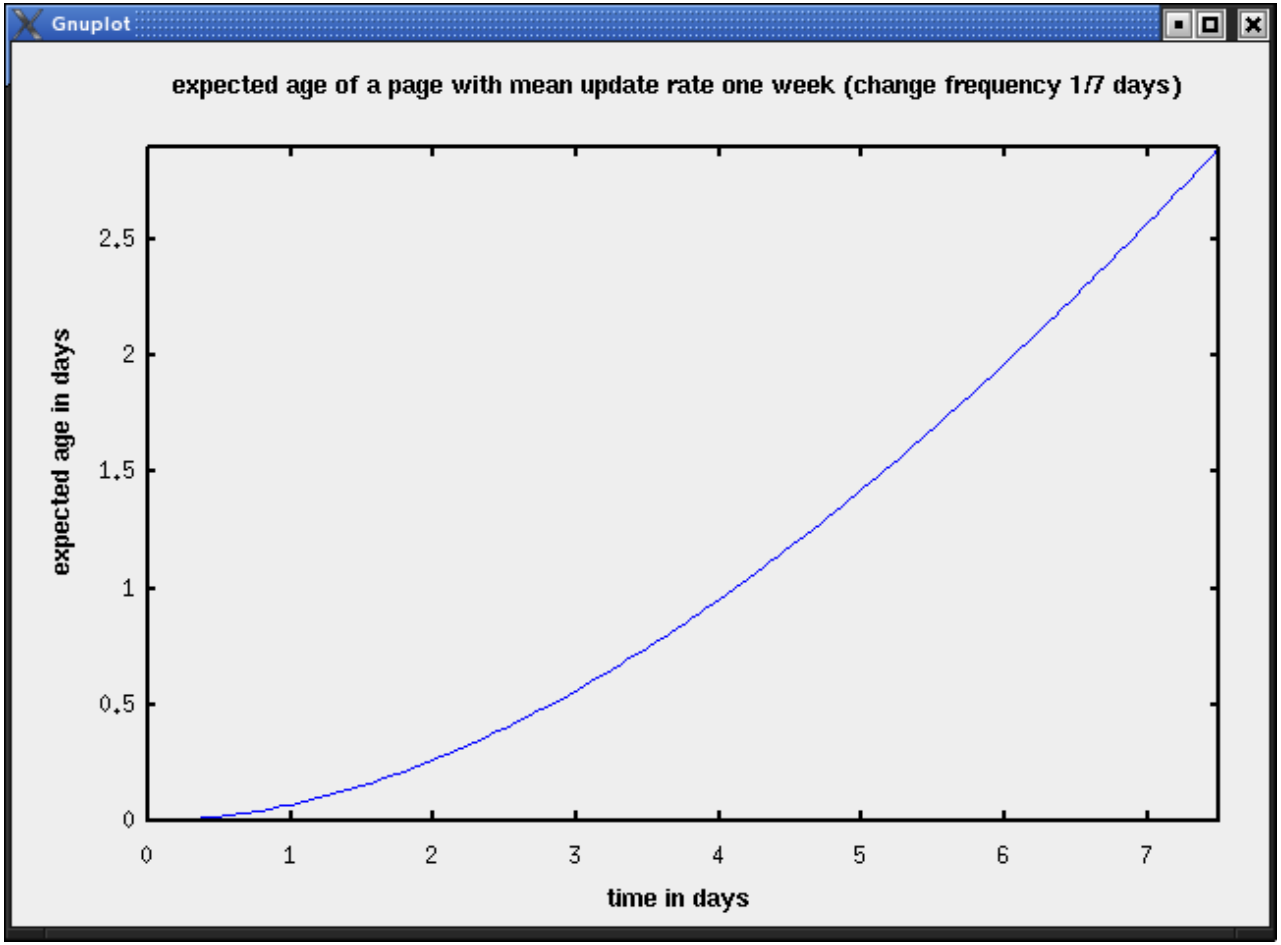
$$expectedAge(\lambda, t) = \int_0^t P_c(t')age(t')dt' = \int_0^t P_c(t')(t - t')dt' \tag{6}$$

mit

- $P_c(t')$: Wahrscheinlichkeit, dass sich die Seite zum Zeitpunkt t' ändert
- Annahme: Poisson-Prozess - Modellierung als zufälliger und unabhängige Sequenz von Ereignissen - Details siehe [1]

$$expectedAge(\lambda, t) = \int_0^t \lambda e^{-\lambda t'}(t - t')dt' \tag{7}$$

Expected age as function of time; $\lambda = 1/7 days$



Age versus Fressness

- Aus dem Vergleich der beiden Kurven sieht man:
  - (Zeitliche) Änderung der *freshness* wird mit der Zeit immer kleiner
  - Zweite Ableitung der *expected Age* ist immer positiv; d.h. je älter die Seite wird, desto teurer wird es, wenn die Seite nicht *gecrawled* wird

Schlussfolgerung

Optimierung bezüglich der *age* führt nicht zu der Schlussfolgerung, dass es sich nicht mehr rentiert, die Seite noch zu *crawlen*. Im Gegensatz zur Optimierung bezüglich der *freshness*.

Sitemaps

- Probleme
  - Aktualitätsproblem
  - Seiten nicht per Links zugänglich (z.B. nur über Formular erreichbar - *deep web*)
- Webmaster erstellen Sitemaps mit Angaben zu
  - Seiten bzw. URLs mit
  - *priority, modification time, modification frequency*

Architektur von Webspidern

- Die Anforderungen Erweiterbarkeit, Verteilbarkeit, Skalierbarkeit und Performanz müssen durch die Architektur gewährleistet werden
- Hierzu gibt es eine eigene Vorlesung

Multithreaded - Asynchroner IO

- Single-threaded Server bei synchronem IO wäre nicht performant: $\longrightarrow$ kein Ausnutzen der Netzbandbreite und *idle* des Prozessors wegen
  - Warten auf DNS-Server Response
  - Warten auf Web-Server Response
- daher sind Crawler *multithreaded* und holen parallel Web-Seiten

Outline

Crawl Typen

- *Broad Crawling*: breites Crawlen, bei der die Zahl der geholten Seiten und die Vollständigkeit im Vordergrund steht: sehr großes Datenvolumen
- *Focused Crawling*: mittleres bis kleines Datenvolumen
  - möglichst alle Seiten von ausgewählten Hosts
  - möglichst alle Seiten zu einem Thema, wie z.B. Gesundheitswesen

Crawl Typen: inkrementell vs. batch

- *Batch Crawl*: Innerhalb eines Crawls wird jede Seite nur einmal geholt; eventuell gesamten Crawl wiederholen - wegen Aktualität
- *Continous or incremental Crawling*: kontinuierliches Crawlen der schon vorhandenen Seiten, um diese aktuell zu halten.


J. Cho and H. Garcia-Molina.
Synchronizing a database to improve freshness.
Technical Report 1999-40, Stanford University, 1999.