Effective Page Refresh Policies for Web Crawlers
and a Semantic Web Document Ranking Model

Roger-Alekos Berkley
IMSE 2012/2014
"Effective Page Refresh Policies for Web Crawlers".
-Junghoo Cho, University of California, Los Angeles.
-Hector Garcia-Molina, Stanford University.

- How to maintain local copies of remote data sources "fresh" when the data source is updated autonomously and independently.
- Special attention is given to the problem of Web crawlers that maintain local copies of remote Web pages for Web search engines.
- This paper proposes various page refresh policies and studies their effectiveness.
"Swoogle: A Search and Metadata Engine for the Semantic Web".
-Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, Joel Sachs, University of Maryland Baltimore County.

What is a web crawler?
A software system for the bulk and automatic downloading of web pages.

What is it used for?
● Search Engines (main component).
● Mirroring/Web Archiving.
● Web Data Mining.
● Web Monitoring.
Web Crawler (2)

Challenges

- Huge Size of the Web
- Inability to Crawl the Web from one machine (Paralellism)
- Limited storage and bandwidth resources
- High Diversity of the Web (formats, quality, page importance)
- Additional load on Web Servers
- High dynamics of the web (Page creations, updates, deletes)
- Huge amount of noise (Spam, duplicates)

Requirements

- Scalability
  Cope with growing load by adding machines & bandwidth
- Quality
  Priority for “important” pages
- Efficiency
  Make efficient use of resources
- Extensibility
  Possibility to add new data formats and protocols
- Politeness
  Respect servers' policies for accessing pages (which, how frequent)
- Freshness
  Make sure that crawled snapshots are current
- Robustness
  Resilience to malicious or unintended crawler traps

Reference: 3,4.
All types of crawlers usually aim to have coverage that is as complete as possible.

Types of Crawlers:

- **Universal Crawler**: supports universal search engines.
- **Focused Crawler**: collects web-pages according to a criteria (e.g. certain domain).
- **Topical Crawler**: collects web-pages about a specific topic (e.g. solar energy).
- **Batch-mode Crawler**: revisits a fixed collection of web-pages periodically.
- **Incremental Crawler**: continuously revisits known pages and increases crawl quality by finding new good pages.
- **Archive/Snapshot crawler**: gets at most a snapshot of a web page.
- **Mirroring/Caching System**: useful for “offline” access to web pages.

Reference: 3,4.
"Effective Page Refresh Policies for Web Crawlers".
- Junghoo Cho, University of California, Los Angeles.
- Hector Garcia-Molina, Stanford University.

1. Freshness and Age.
2. Evolution of real-world elements.
4. Synchronization policies.
5. Weighted Freshness and Age.
1. Freshness and Age

- Data sources do not push changes to local copies.
- Local copies get out of date.
- Local copies have to periodically "poll" data sources to keep the copy "fresh".

**Freshness** (F): the fraction of the local copy that is up-to-date.

\[
F(e_i; t) = \begin{cases} 
1 & \text{if } e_i \text{ is up-to-date at time } t \\
0 & \text{otherwise.}
\end{cases}
\]

**Age** (A): measures "how old" the local copy is.

\[
A(e_i; t) = \begin{cases} 
0 & \text{if } e_i \text{ is up-to-date at time } t \\
 t - t_{m(e_i)} & \text{otherwise.}
\end{cases}
\]

\[
F(S; t) = \frac{1}{N} \sum_{i=1}^{N} F(e_i; t).
\]

\[
A(S; t) = \frac{1}{N} \sum_{i=1}^{N} A(e_i; t).
\]

\( t_{m(e1)} \) is the time of the first modification of element (e1) after its last synchronization.

Reference: 5.
1.a Freshness and Age (Evolution)

Time evolution for Freshness and Age of an element:

Reference: 5.
1.b Freshness and Age (Examples)

Example 1
Database (S) with 100 elements, \( S = \{e_1, e_2, e_3, \ldots, e_{100}\} \).
At time (t), 65 elements are out-of-date and 35 elements fresh.

\[
F(S,t) = \frac{1}{100}(1 \times 35) = 0.35 \\
\text{(1: database fully fresh)} \\
\text{(0: database fully out-of-date)}
\]

Example 2
Database with 3 elements, \( S = \{e_1, e_2, e_3\} \).
At time (t), all 3 elements out-of-date.

\[F(S,t) = 0\]

\[
A(e_1) = t - tm(e_1) = 3 \text{ hours.} \\
A(e_2) = t - tm(e_2) = 7 \text{ hours.} \\
A(e_3) = t - tm(e_3) = 10 \text{ hours.}
\]

\[
A(S,t) = \frac{3+7+10}{3} = 6.66 \text{ hours.}
\]

Reference: 5.
2. Evolution of Real-World Elements

- Freshness is hard to measure exactly in practice since we need to instantaneously compare the real-world data to the local copy.
- But it is possible to estimate freshness given some information about how the real-world data changes.
- A Poisson Process is often used to model events that happen randomly and independently with a fixed rate of change (λ) over time.
- In this paper, it is assumed that the Real-World elements are modified by a Poisson Process.

Reference: 5.
3. Evolution of Real-World Database

We can model the change of the real-world database as a whole by one of the following:

1) Uniform change frequency model
   All real-world database elements change at the same rate ($\lambda$).

2) Non-uniform change frequency model
   Elements change at different rates.

In this paper: each element in the real-world database (web page) is assumed to change independently and at its own rate.

one element may change once a day, and another element may change once a year.

Reference: 5.
4. Synchronization Policies Dimensions

- **Synchronization Frequency**: How frequently should we synchronize the local database.
- **Synchronization Order**: In what order should we synchronize the elements.
- **Synchronization Points**: At which point in time should we synchronize the elements.
- **Resource Allocation**: How many elements should we synchronize per time interval, and how frequently should we sync each individual element.

Reference: 5.
4.a. Synchronization Frequency

Parameters to be decided:

\( \mathcal{N} \): number of elements to be synchronized per time unit.

\( I \): time unit.

By changing "\( I \)" we adjust how often the elements are synchronized.

\[ f = \frac{1}{I} \]

represents the average frequency at which an element in the database is synchronized.

Reference: 5.
4.b. Synchronization Order Policies

**Example:** we maintain a local database of 10,000 Web pages from a certain website. We keep our copy updated by revisiting the pages in the site. In performing the crawl, we may adopt one of the following options:

1. **Fixed synchronization order policy**
   
   We maintain a list of all 10,000 URLs and we visit the URLs repeatedly in the same order.

2. **Random synchronization order policy**
   
   We only maintain the URL of the root page of the site, and whenever we crawl the site, we start from the root page, following links. The link structure and order at a particular crawl determines the page visit order, the synchronization order may change from one crawl to the next.

3. **Purely random synchronization order policy**
   
   At each synchronization point, we select a random element from the database and synchronize it.

Reference: 5.
r = \lambda / f

\lambda = \text{Rate at which a real-world element changes.}

f = \text{Frequency at which a local element is synchronized.}

\textbf{When } r < 1 \text{, we synchronize the elements more often than they change.}

\textbf{When } r > 1 \text{, the elements change more often than we synchronize them.}

Reference: 5.
4.c. Synchronization Points

In some cases, we may need to synchronize the database only in a limited time window.

**Example**: we maintain a local database of 10 pages from a certain website. The site is heavily accessed during daytime. We consider several synchronization policies:

(a) Sync. all 10 pages at the beginning of the day.

(b) Sync. most pages at the beginning of the day, some pages during the rest of the day.

(c) Sync. 10 pages uniformly over a day.

Reference: 5.
Example
Database of 3 elements \{e_1, e_2, e_3\}.

- \(e_1\) changes at the rate \(\lambda_1=4\) (times/day).
- \(e_2\) changes at the rate \(\lambda_2=3\).
- \(e_3\) changes at the rate \(\lambda_3=2\).

We decided to synchronize the database at the total of 9 elements/day.

In deciding how frequently to synchronize each element, we have 2 options:

1. **Synchronize all elements uniformly at the same rate.**
   Meaning: Sync. \(e_1, e_2, e_3\) at the same rate of 3 times a day.

2. **Synchronize an element proportionally more often if it changes more often.**
   Meaning: Sync. \(e_1\) at (frequency) \(f_1=4\), \(e_2\) at \(f_2=3\), \(e_3\) at \(f_3=2\).
Two-Element Database Example

$S = \{ e_1, e_2 \}$

$f(e_1) = 9$ times a day.

$f(e_2) = 1$ time a day.

**Goal**: maximize the freshness of the database.

$e_1$ changes once and only once in each interval, we do not know exactly when it changes within this interval.

$e_2$ changes once and only once a day, we do not know exactly when it changes within this day.

Reference: 5.
Let's assume that we can only synchronize one element per day.

**Question**: which element should we synchronize?

<table>
<thead>
<tr>
<th>Item</th>
<th>e1</th>
<th>e2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency of change</td>
<td>9 times/day</td>
<td>1 time/day</td>
</tr>
<tr>
<td><strong>Point of Change</strong></td>
<td>After/before middle of the period</td>
<td>After/before the middle of the day</td>
</tr>
<tr>
<td>Benefit</td>
<td>1/18 (of a day)</td>
<td>1/2 (of a day)</td>
</tr>
<tr>
<td>Probability of either point</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td><strong>Expected Benefit</strong></td>
<td>1/36 (of a day)</td>
<td>1/4 (of a day)</td>
</tr>
</tbody>
</table>

**Answer**: e2 results in higher expected benefit.

Reference: 5.
Using the same logic, we calculate the expected benefit at different sync. levels:

<table>
<thead>
<tr>
<th>Row</th>
<th>Total Sync. Frequency</th>
<th>Allocation f1 f2</th>
<th>Expected Benefit</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>1 0</td>
<td>$\frac{1}{2} \times \frac{1}{18} = \frac{1}{36}$</td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1</td>
<td>$\frac{1}{2} \times \frac{1}{2} = \frac{9}{36}$</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>2</td>
<td>2 0</td>
<td>$\frac{1}{2} \times \frac{1}{18} + \frac{1}{2} \times \frac{1}{18} = \frac{2}{36}$</td>
<td>0 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1</td>
<td>$\frac{1}{2} \times \frac{1}{18} + \frac{1}{2} \times \frac{1}{2} = \frac{10}{36}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 2</td>
<td>$\frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times \frac{1}{3} = \frac{12}{36}$</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>5</td>
<td>3 2</td>
<td>$\frac{3}{36} + \frac{12}{36} = \frac{30}{72}$</td>
<td>2 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 3</td>
<td>$\frac{2}{36} + \frac{6}{16} = \frac{31}{72}$</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>10</td>
<td>9 1</td>
<td>$\frac{9}{36} + \frac{1}{4} = \frac{36}{72}$</td>
<td>7 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 3</td>
<td>$\frac{7}{36} + \frac{6}{16} = \frac{41}{72}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 5</td>
<td>$\frac{5}{36} + \frac{15}{36} = \frac{40}{72}$</td>
<td></td>
</tr>
</tbody>
</table>

**Proportional Policy**
- e1 changes 9 times, gets synced 9 times.
- e2 changes once, gets synced once.

**Uniform Policy**
- No matter how much the elements change, they get the same amount of sync.

**Rows (a) to (e):** when the synchronization frequency is smaller than the change frequency, it's better to give up syncing the elements that change too fast (and focus on what we can track).

**Rows (h) to (j):** Even if the sync. frequency is larger, the uniform policy is more effective. And, the optimal policy is somewhere in between the proportional and the uniform policies.

Reference: 5.
4.d. Optimal Resource-Allocation Policy

Example: 5-element real-world database.

<table>
<thead>
<tr>
<th></th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
<th>$e_4$</th>
<th>$e_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) change frequency</td>
<td>1.00</td>
<td>2.00</td>
<td>3.00</td>
<td>4.00</td>
<td>5.00</td>
</tr>
<tr>
<td>(b) synchronization frequency (freshness)</td>
<td>1.15</td>
<td>1.36</td>
<td>1.35</td>
<td>1.14</td>
<td>0.00</td>
</tr>
<tr>
<td>(c) synchronization frequency (age)</td>
<td>0.84</td>
<td>0.97</td>
<td>1.03</td>
<td>1.07</td>
<td>1.09</td>
</tr>
</tbody>
</table>

(b) and (c) are the optimal synchronization frequencies for these elements, based on their frequency of change.

Outcome
1. To improve freshness, we should penalize the elements that change too often.
2. To improve age, we should synchronize an element more often when it changes more often.

Optimal Policy
Notice the marginal differences between the synchronization frequencies?

- Optimal resource-allocation policy is rather closer to the uniform policy than the proportional policy.

Reference: 5.
So far, we assumed that the freshness and the age of every element is equally important to the users. But what if the elements have different "importance"?

Element Importance = Traffic

When the users access element $e_1$ **twice more often** than $e_2$, we consider $e_1$ twice as important as $e_2$. By maintaining $e_1$ up-to-date we can make the users see fresh elements twice as often.

$w_i =$ weight (importance) of an element.

\[
F(S; t) = \left( \sum_{i=1}^{N} w_i F(e_i; t) \right) \Bigg/ \left( \sum_{i=1}^{N} w_i \right)
\]

\[
A(S; t) = \left( \sum_{i=1}^{N} w_i A(e_i; t) \right) \Bigg/ \left( \sum_{i=1}^{N} w_i \right)
\]

(1) Weighted Freshness  
(2) Weighted Age

Reference: 5.
### 5. Weighted Freshness and weighted age

**Example:** We maintain 6 elements in the local database, their **weights** and change frequencies are different. Assuming that we synchronize 6 elements per day.

<table>
<thead>
<tr>
<th></th>
<th>$e_{11}$</th>
<th>$e_{12}$</th>
<th>$e_{13}$</th>
<th>$e_{21}$</th>
<th>$e_{22}$</th>
<th>$e_{23}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) weight $(w_i)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b) change frequency (times/day)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>(c) synchronization frequency (freshness)</td>
<td>0.78</td>
<td>0.76</td>
<td>0.00</td>
<td>1.28</td>
<td>1.56</td>
<td>1.62</td>
</tr>
<tr>
<td>(d) synchronization frequency (age)</td>
<td>0.76</td>
<td>0.88</td>
<td>0.94</td>
<td>0.99</td>
<td>1.17</td>
<td>1.26</td>
</tr>
</tbody>
</table>

(c) and (d) are the optimal synchronization frequencies for these elements, based on their weight and frequency of change.

### Outcome

1. We can see that we should visit an element more often when its weight is higher. Such as: $(e_{11}$ and $e_{21})$, $(e_{12}$ and $e_{22})$.

2. But, not necessarily proportionally more often. Such as: $(e_{13}$ and $e_{23})$.

Reference: 5.
6. Experimental Setup

To collect data on how often web pages change (real-life database):

- Crawled 720,000 pages from 270 "popular" websites.
- For a period of 4 months.
- Using Stanford WebBase crawler.
- Indexing speed 60 pages per second.
- Storage 210GB of HTML.

To select sites for the local copy:

- Used a snapshot of the web available from Stanford WebBase crawler.
- Selected 400 most popular websites and contacted them.
- 270 websites agreed to be experimented.
- Starting at the root URL, 3000 pages were crawled daily from each website.
6. Experimental Results (1)

Verification of the comparison of the **synchronization-order** policies.

- **r**: change frequency rate for crawls. (r=2, once a month).
- **Lines**: predictions by theoretical analysis.
- **Points**: experimental results.

Reference: 5.
6. Experimental Results (2)

Study of the **frequency of change** of the experimented web pages. (important for the next step of evaluating the resource-allocation policies).

**Horizontal:** Average Change Interval. (total monitoring period divided by number of detected changes).

**Vertical:** fraction of pages.

![Graph showing frequency of change](image)
6. Experimental Results (3)

Evaluation of the three resource-allocation policies.

<table>
<thead>
<tr>
<th></th>
<th>overall</th>
<th></th>
<th>com</th>
<th></th>
<th>gov</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Freshness</td>
<td>Age</td>
<td>Freshness</td>
<td>Age</td>
<td>Freshness</td>
</tr>
<tr>
<td>Proportional</td>
<td>0.12</td>
<td>400 days</td>
<td>0.07</td>
<td>386 days</td>
<td>0.69</td>
</tr>
<tr>
<td>Uniform</td>
<td>0.57</td>
<td>5.6 days</td>
<td>0.38</td>
<td>8.5 days</td>
<td>0.82</td>
</tr>
<tr>
<td>Optimal</td>
<td>0.62</td>
<td>4.3 days</td>
<td>0.44</td>
<td>7.4 days</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Freshness and Age prediction of a local copy based on real-web data

*Fixed-Order policy adopted (best).*

**Outcome**

1. Optimal policy is significantly better than the two other policies.
2. Uniform policy is significantly better than the Proportional policy.

**Facts**

- In a case, *Freshness* increased 500% by using the *Optimal policy* instead of the *Proportional policy*.
- Age decreases by 23% from the *Uniform policy* to the *Optimal policy*.
- Age of the *Proportional policy* is 93 times worse than the *Optimal policy*.

Reference: 5.
Optimal policy is more useful when pages change more often.

Facts
- For the (.gov) domain, which has fewer frequently changing pages, the freshness difference is 18% between the proportional and optimal policies.
- Difference becomes 84% for the (.com) domain.
- Same kind of improvement for Age, which increases from 1380% improvement for the (.gov) to 5110% for the (.com).
"Swoogle: A Search and Metadata Engine for the Semantic Web".
-Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng,
Pavan Reddivari, Vishal Doshi, Joel Sachs, University of Maryland
Baltimore County.

2. Rational random surfer ranking model for Semantic Web Documents.
Google PageRank

Random Surfing Model

- More incoming links = Higher PageRank.
- More outgoing links = lower outgoing vote.
- Damping factor = maximizes the amount of outgoing votes (Decay).

\[ PR(A) = (1-d) + d \left( \frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)} \right) \]

\( PR(A) \): PageRank of page A.
\( d \) = damping factor, set to 0.85.
\( T1 \): a page that points to page A.
\( Tn \): another page that points to page A.
\( C(T1) \): number of links going out of Page T1.

The nature of hyperlinks leads to a non-uniform probability of following a particular outgoing link. This is not appropriate for the Semantic Web.

**Rational Random Surfing Model:** accounts for the various types of links that can exist between SWDs. Types of links:

1. *Imports (A,B):* A imports all content of B.
2. *Uses-term (A,B):* A uses some of terms defined by B, without importing B.
3. *Extends (A,B):* A extends the definitions of terms defined by B.
4. *Asserts (A,B):* A makes assertions about the individuals defined by B.

These relationships account for the ranking and carry different weights.

For instance, when a surfer imports (A,B) while visiting A, it is natural for it to follow this link because B is semantically part of A.

Similarly, the surfer may follow the extends(A,B) relation because it can understand the defined term completely only when it browses both A and B.

Reference: 7.
Questions?

Thank you for your attention.