Ανάκληση Πληροφορίας

Διδάσκων –
Δημήτριος Κατσαρός

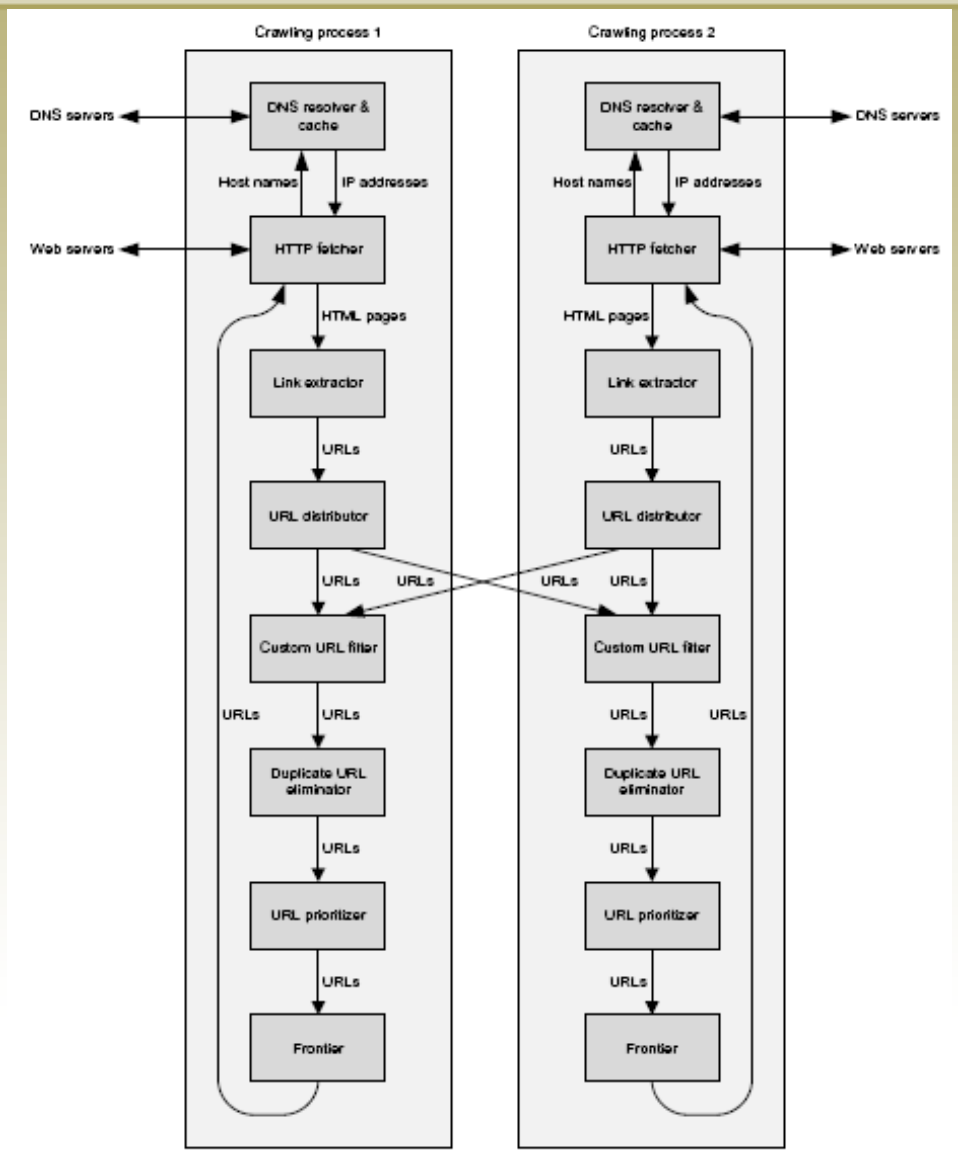
Διάλεξη 19η: 30/05/2014

Web crawlers

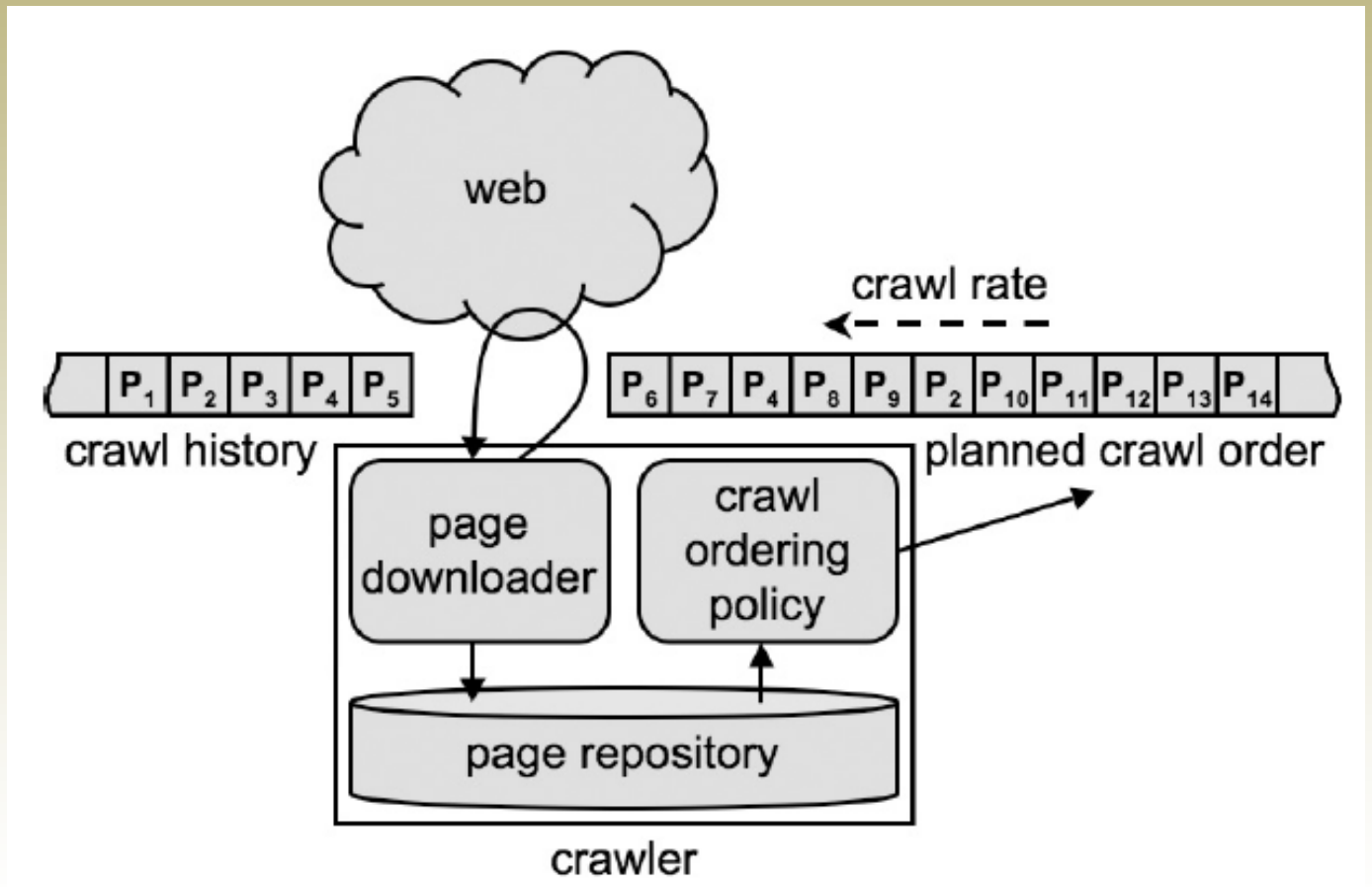
Ερπυστές στον Παγκόσμιο Ιστό

Basic crawler architecture

- Frontier data structure
- HTTP Fetcher
- Link Extractor
- URL Distributor
- Custom URL Filter
- Duplicate URL Eliminator
- URL Prioritizer



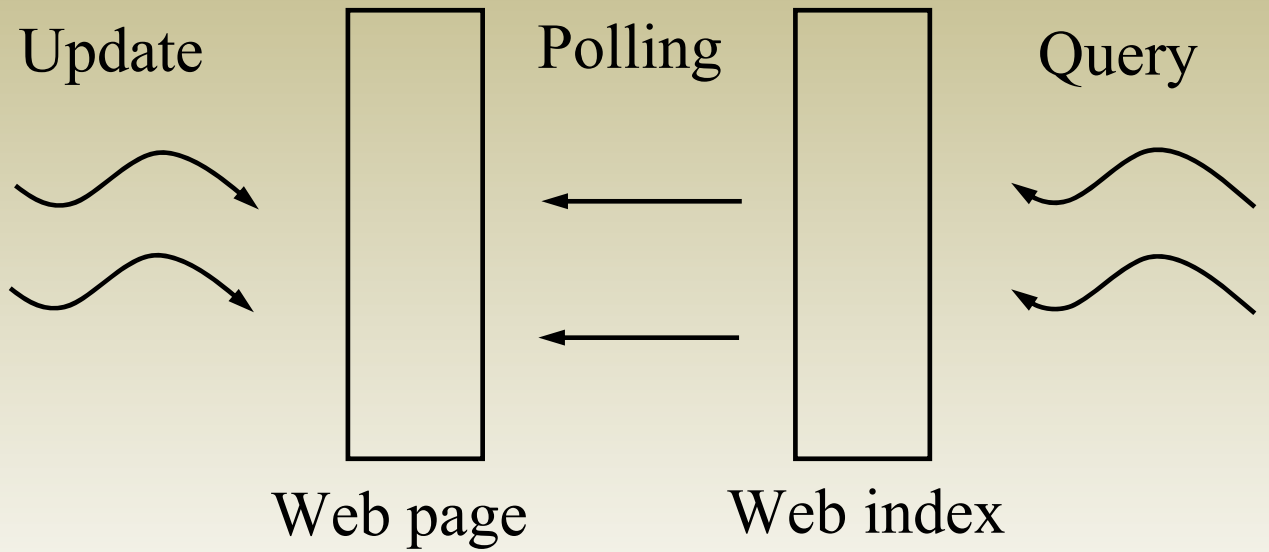
Crawl order model



Taxonomy of crawl ordering techniques

Technique	Objectives		Factors considered		
	Coverage	*Freshness*	*Importance*	*Relevance*	*Dynamicity*
Breadth-first search [43, 95, 108]	√				
Prioritize by indegree [43]	√		√		
Prioritize by PageRank [43, 45]	√		√		
Prioritize by site size [9]	√		√		
Prioritize by spawning rate [48]	√				√
Prioritize by search impact [104]	√		√	√	
Scoped crawling (Section 4.2)	√			√	
Minimize obsolescence [41, 46]		√	√		√
Minimize age [41]		√	√		√
Minimize incorrect content [99]		√	√		√
Minimize embarrassment [115]		√	√	√	√
Maximize search impact [103]		√	√	√	√
Update capture (Section 5.2)		√		√	√
WebFountain [56]	√	√			√
OPIC [1]	√	√	√		

Index page refresh policy: The problem



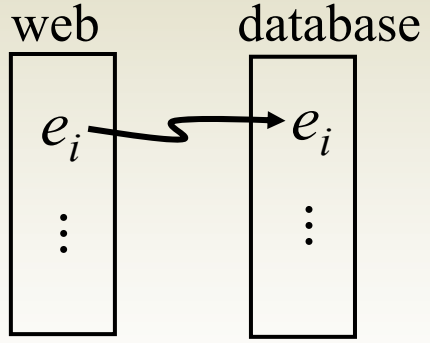
Change metrics

- Freshness
  - Freshness of a local page e_i at time t is

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

Freshness of the database S at time t is

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



(Assume "equal importance" of pages)

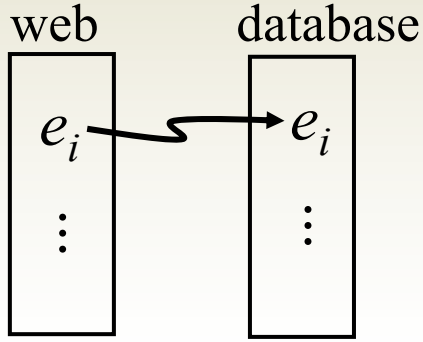
Change Metrics

- Age
  - Age of element e_i at time t is

$$A(e_i ; t) = \begin{cases} 0 & \text{if } e_i \text{ is up-to-date at time } t \\ t\text{-(modification time of } e_i\text{)} & \text{otherwise} \end{cases}$$

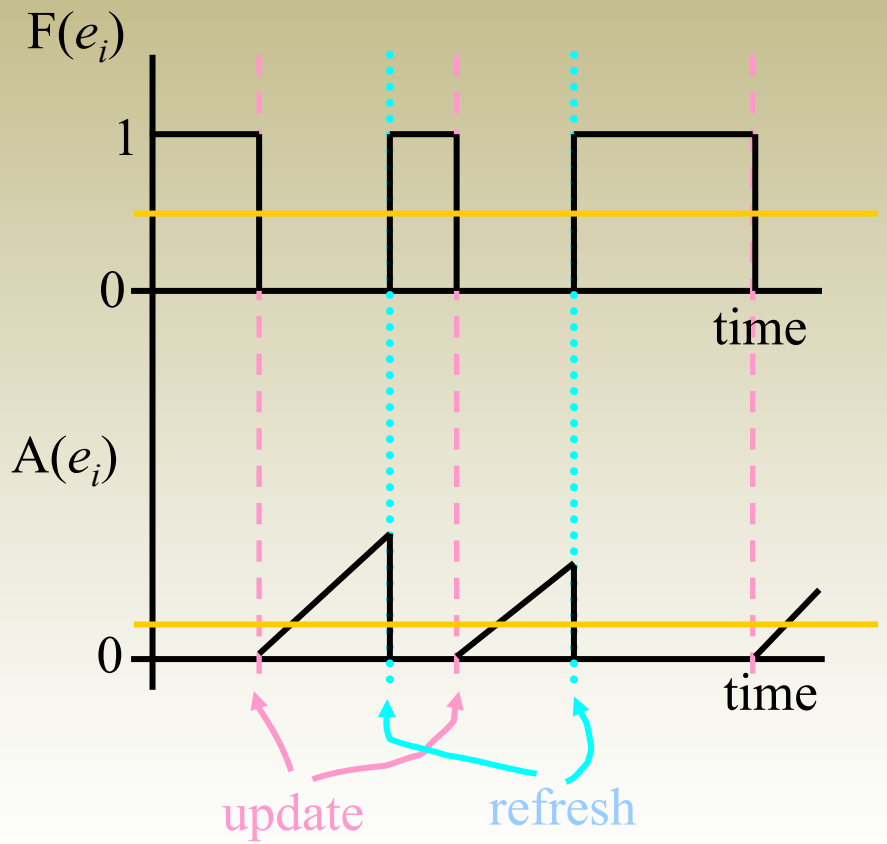
Age of the database S at time t is

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



(Assume "equal importance" of pages)

Change Metrics



Time averages:

$$\bar{F}(e_i) = \lim_{t\to\infty} \frac{1}{t} \int_0^t F(e_i;t)\,dt$$

$$\bar{F}(S) = \lim_{t\to\infty} \frac{1}{t} \int_0^t F(S;t)\,dt$$

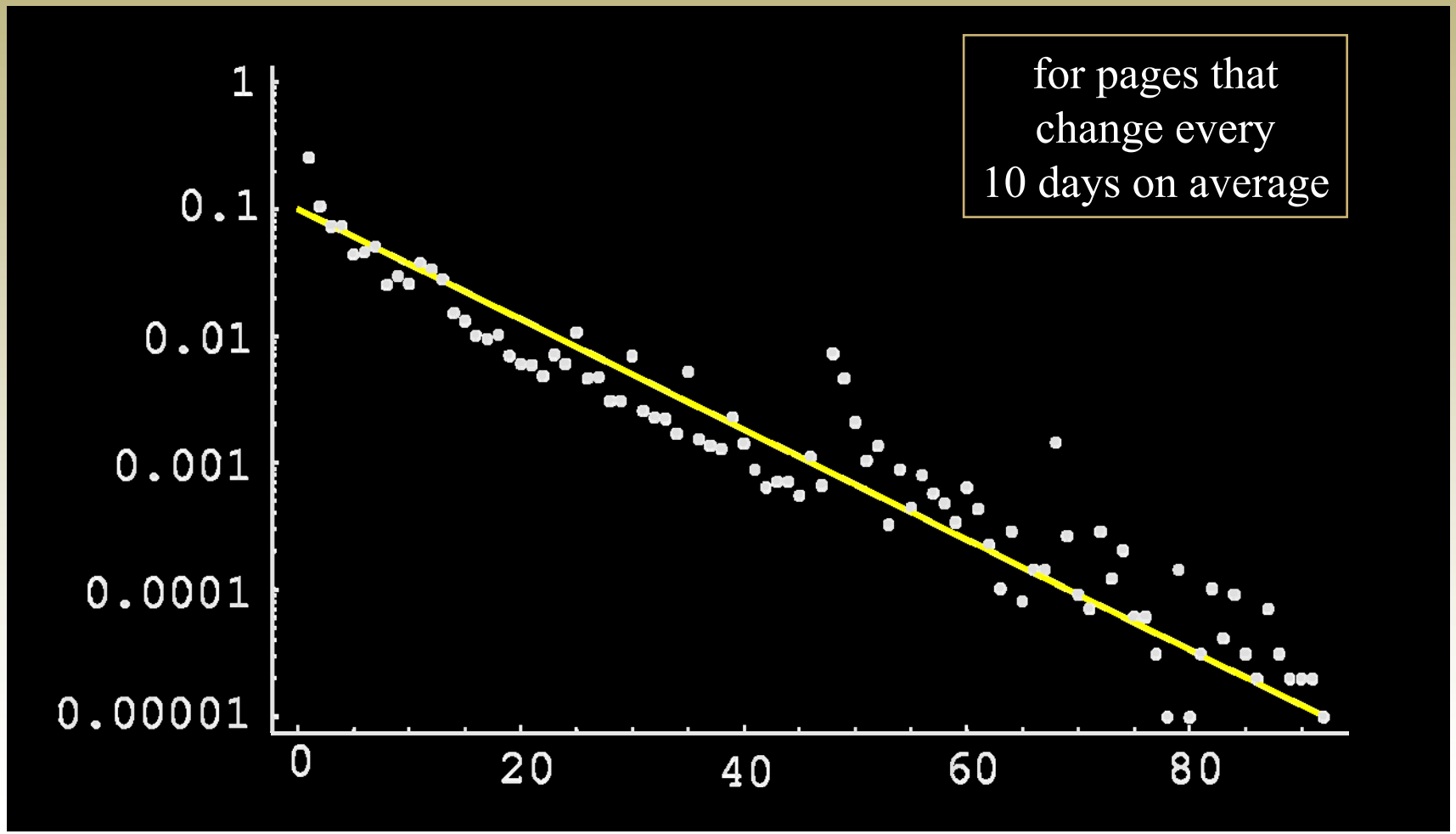
Relation between F(S) and $F(e_i)$

Theorem 5.1 $\bar{F}(S) = \frac{1}{N}\sum_{i=1}^{N}\bar{F}(e_i)$ $\bar{A}(S) = \frac{1}{N}\sum_{i=1}^{N}\bar{A}(e_i)$

Proof

$$\begin{aligned}
\bar{F}(S) &= \lim_{t\to\infty}\frac{1}{t}\int_0^t F(S;t)dt && \text{(definition of } \bar{F}(S)\text{)}\\
&= \lim_{t\to\infty}\frac{1}{t}\int_0^t \left(\frac{1}{N}\sum_{i=1}^{N}F(e_i;t)\right)dt && \text{(definition of } F(S;t)\text{)}\\
&= \frac{1}{N}\sum_{i=1}^{N}\lim_{t\to\infty}\frac{1}{t}\int_0^t F(e_i;t)dt\\
&= \frac{1}{N}\sum_{i=1}^{N}\bar{F}(e_i) && \text{(definition of } \bar{F}(e_i)\text{)}
\end{aligned}$$

Evolution of Web pages



Probabilistic evolution of a Web page

- We assume that each element e_i is modified by a Poisson process with change rate λ_i, That is, each element e_i changes at its own average rate λ_i, and this rate may differ from element to element
- Under the Poisson process model, we can analyze the freshness and the age of the element e_i over time. More precisely, let us compute the expected value of the freshness and the age of e_i at time t
- We assume that we synchronize e_i at t=0 and at t=I
- Since the time to the next event follows an exponential distribution under a Poisson process, we can obtain the probability that e_i changes in the interval (0, t] by the following integration:

$$\Pr\{T \leq t\} = \int_0^t f_T(t)dt = \int_0^t \lambda_i e^{-\lambda_i t} dt = 1 - e^{-\lambda_i t}$$

Probabilistic evolution of a Web page

- **Basic Statistics**

Lemma 4.1 *If T is the time to the occurrence of the next event in a* Poisson process *with rate λ, the probability density function for T is*

$$f_T(t) = \begin{cases} \lambda e^{-\lambda t} & \text{for } t > 0 \\ 0 & \text{for } t \leq 0 \end{cases}$$

- Because e_i is not synchronized in the interval $(0, I)$, the local element e_i may get out-of-date with probability $\Pr\{T \leq t\} = 1 - e^{-\lambda i t}$ at time $t \in (0, I)$
- Hence, the *expected freshness* is

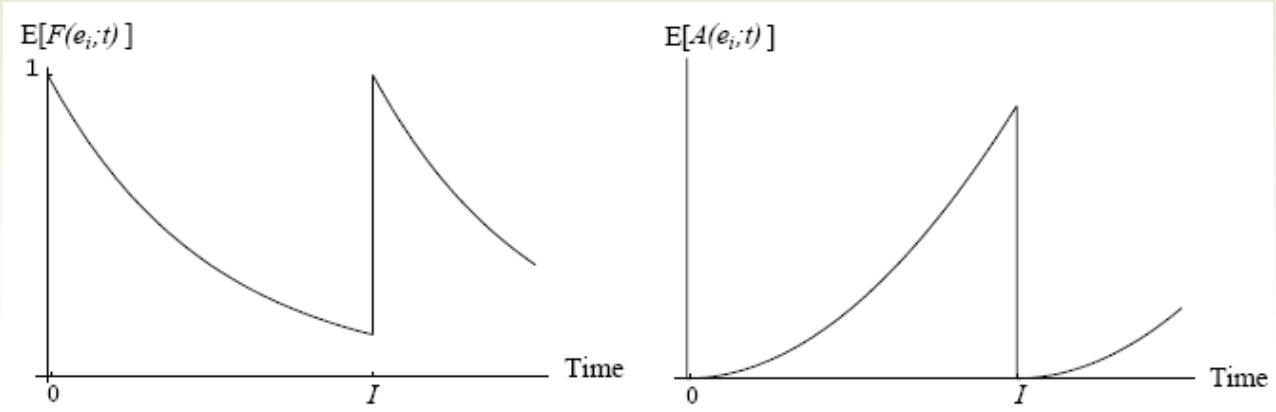
$$\mathrm{E}[F(e_i; t)] = 0 \cdot (1 - e^{-\lambda_i t}) + 1 \cdot e^{-\lambda_i t} = e^{-\lambda_i t} \quad \text{for } t \in (0, I)$$

- Note that the expected freshness is 1 at time t=0 and that the expected freshness approaches 0 as time passes

Probabilistic evolution of a Web page

- We can obtain the expected value of age of e_i similarly
- If e_i is modified at time $s \in (0, I)$, the age of e_i at time $t \in (s,I)$ is $(t - s)$
- e_i changes at time s with probability $\lambda_i e^{-\lambda i s}$, so the expected age at time $t \in (0, I)$ is

$$\mathrm{E}[A(e_i;t)] = \int_0^t (t-s)(\lambda_i e^{-\lambda_i s})ds = t(1 - \frac{1-e^{-\lambda_i t}}{\lambda_i t})$$



(a) $\mathrm{E}[F(e_i;t)]$ graph over time (b) $\mathrm{E}[A(e_i;t)]$ graph over time

Symbols used and their meanings

symbol	meaning
(a) $\bar{F}(S)$, $\bar{F}(e_i)$	Freshness of database S (and element e_i) averaged over time
(b) $\bar{A}(S)$, $\bar{A}(e_i)$	Age of database S (and element e_i) averaged over time
(c) $\bar{F}(\lambda_i, f_i)$, $\bar{A}(\lambda_i, f_i)$	Freshness (and age) of element e_i averaged over time, when the element changes at the rate λ_i and is synchronized at the frequency f_i
(i) λ_i	Change frequency of element e_i
(j) f_i $(= 1/I_i)$	Synchronization frequency of element e_i
(k) λ	Average change frequency of database elements
(l) f $(= 1/I)$	Average synchronization frequency of database elements

Synchronization policies

- We discussed how the real-world database changes over time
- We now study how the local copy can be refreshed
- There are several dimensions to this synchronization process
  1. **Synchronization frequency**: We first need to decide *how frequently* we synchronize the local database. Obviously, as we synchronize the database more often, we can maintain the local database fresher. In our analysis, we assume that we synchronize N elements per I time-units. By varying the value of I, we can adjust how often we synchronize the database
  2. **Resource allocation**: Even after we decide how many elements we synchronize per unit interval, we still need to decide how frequently we synchronize *each individual* element. We illustrate this issue by the following example:

Synchronization policies

- A database consists of three elements, e_1, e_2 and e_3. It is known that the elements change at the rates λ_1=4, λ_2=3, and λ_3=2 (times/day). We have decided to synchronize the database at the total rate of 9 elements/day. In deciding how frequently we synchronize each element, we consider the following options:
  - Synchronize all elements uniformly at the same rate. That is, synchronize e1, e2 and e3 at the same rate of 3 (times/day)
  - Synchronize an element proportionally more often when it changes more often. In other words, synchronize the elements at the rates of f1 = 4, f2 = 3, f3 = 2 (times/day) for e1, e2 and e3, respectively

Synchronization policies

Based on how fixed synchronization resources are allocated to individual elements, we can classify synchronization policies as follows:

- (a) **Uniform-allocation policy**: We synchronize all elements at the same rate, regardless of how often they change. That is, each element ei is synchronized at the fixed frequency f. In the previous example, the first option corresponds to this policy
- (b) **Non-uniform-allocation policy**: We synchronize elements at different rates. In particular, with a *proportional-allocation policy* we synchronize element e_i with a frequency f_i that is proportional to its change frequency λ_i. Thus, the frequency ratio λ_i/f_i, is the same for any i under the proportional-allocation policy. In the previous example, the second option corresponds to this policy

Synchronization policies

3. **Synchronization order**: Now we need to decide in what order we synchronize the elements in the database

Example. We maintain a local database of 10,000Web pages from site A. In order to maintain the local copy up-to-date, we continuously update our local database by revisiting the pages in the site. In performing the update, we may adopt one of the following options:

- We maintain an explicit list of all URLs in the site, and we visit the URLs repeatedly in the same order. Notice that if we update our local database at a fixed rate, say 10,000 pages/day, then we synchronize a page, say p1, at the fixed interval of one day

Synchronization policies

- 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. Since the link structure (and the order) at a particular crawl determines the page visit order, the synchronization order may change from one crawl to the next. Notice that under this policy, we synchronize a page, say p_1, at variable intervals. For instance, if we visit p1 at the end of one crawl and at the beginning of the next crawl, the interval is close to zero, while in the opposite case it is close to two days
- Instead of actively synchronizing pages, we synchronize pages on demand, as they are *requested* by a user. Since we do not know which page the user will request next, the synchronization order may appear random. Under this policy, the synchronization interval of p_1 is not bound by any value. It may range from zero to infinity

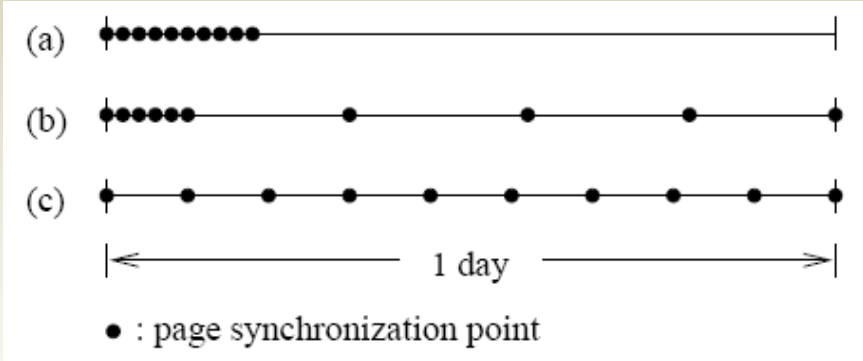
Synchronization policies

We can summarize the above options as follows:

- (a) **Fixed order**: We synchronize all elements in the database in the same order repeatedly. Therefore, a particular element is synchronized at a fixed interval under this policy. This policy corresponds to the first option of the above example
- (b) **Random order**: We synchronize all elements repeatedly, but the synchronization order may be different in each iteration. This policy corresponds to the second option in the example
- (c) **Purely random**: At each synchronization point, we select a random element from the database and synchronize it. Therefore, an element is synchronized at intervals of arbitrary length. This policy corresponds to the last option in the example

Synchronization policies

4. **Synchronization points**: In some cases, we may need to synchronize the database only in a limited time window. For instance, if a Web site is heavily accessed during daytime, it might be desirable to crawl the site only in the night, when it is less frequently visited. We illustrate several options for dealing with this constraint by an example



We assume that we synchronize a database uniformly over time. Since crawlers cannot guess the best time to visit each site, they typically visit sites at a uniform rate that is convenient to the crawler.

Synchronization-order policies

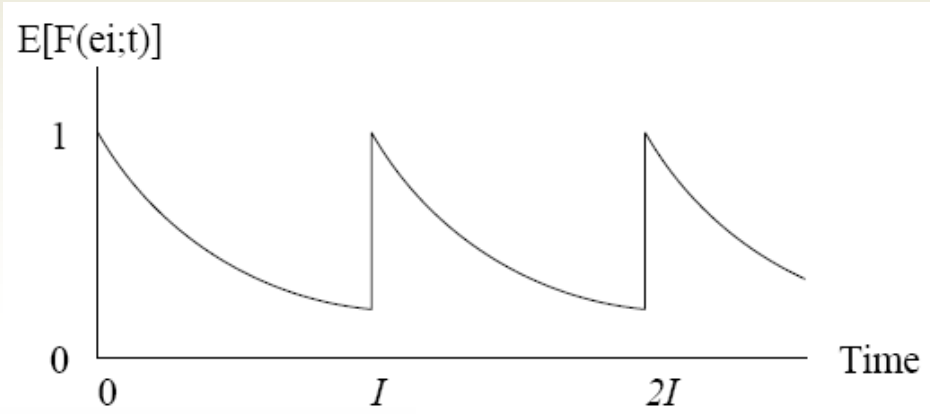
- Clearly, we can increase the database freshness by synchronizing more often
- But exactly how often should we synchronize, for the freshness to be, say, 0.8?
- Conversely, how much freshness do we get if we synchronize 100 elements per second?
- We will now address these questions by analyzing synchronization-order policies
- Through the analysis, we will also learn which synchronization-order policy is the best in terms of freshness and age

Synchronization-order policies

- We assume that all real-world elements are modified at the same average rate λ
- That is, we adopt the uniform change-frequency model
- When the elements change at the same rate, it does not make sense to synchronize the elements at different rates, so we also assume uniform-allocation policy
- These assumptions significantly simplify our analysis, while giving us solid understanding on the issues that we address
- Based on these assumptions, we analyze different synchronization-order policies in the subsequent subsections

Fixed-order policy

- Under the fixed-order policy, we synchronize the local elements in the *same order* repeatedly
- Now we compute the freshness of the database S. Since we can compute the freshness of S from freshness of its elements, we first study how a random element e_i evolves over time
- Assuming that it takes I seconds to synchronize all elements in S, the expected freshness of e_i will evolve as in Figure. In the graph, we assumed that we synchronize e_i initially at t=0
- Note that E[F(ei; t)] recovers to 1 every I seconds, when we synchronize it. Intuitively, e_i goes through exactly the same process every I seconds, so we can expect that we can learn anything about e_i by studying how e_i evolves in the interval (0, I)



Time evolution of E[$F(e_i;t)$] for fixed-order policy

Fixed-order policy

Theorem 5.2 *When the element e_i is synchronized at the fixed interval of I seconds, the time average of the freshness of e_i is the same as the time average of $\mathrm{E}[F(e_i;t)]$ over the interval $(0, I)$.*

$$\bar{F}(e_i) = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt$$

□

Proof

$$\begin{aligned}
\bar{F}(e_i) &= \lim_{t\to\infty} \frac{\int_0^t F(e_i;t)dt}{t} \\
&= \lim_{n\to\infty} \frac{\sum_{j=0}^{n-1}\int_{jI}^{(j+1)I} F(e_i;t)dt}{\sum_{j=0}^{n-1} I} \\
&= \lim_{n\to\infty} \frac{\sum_{j=0}^{n-1}\int_0^I F(e_i;t+jI)dt}{nI} \\
&= \frac{1}{I}\int_0^I \left[\lim_{n\to\infty}\frac{1}{n}\sum_{j=0}^{n-1} F(e_i;t+jI)\right] dt \qquad (5.1)
\end{aligned}$$

Because we synchronize e_i every I seconds from $t = 0$, $F(e_i;t+jI)$ is the freshness of e_i

Fixed-order policy

at t *seconds after each synchronization.* Therefore, $\frac{1}{n}\sum_{j=0}^{n-1} F(e_i; t+jI)$, the average of freshness at t seconds after synchronization, will converge to its expected value, $\mathrm{E}[F(e_i;t)]$, as $n \to \infty$. That is,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{j=0}^{n-1} F(e_i; t+jI) = \mathrm{E}[F(e_i;t)].$$

Then,

$$\frac{1}{I}\int_0^I \left[\lim_{n\to\infty} \frac{1}{n}\sum_{j=0}^{n-1} F(e_i; t+jI)\right] dt = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt. \qquad (5.2)$$

From Equation 5.1 and 5.2, $F(e_i) = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt.$ ■

Based on Theorem 5.2, we can compute the freshness of e_i.

$$\bar{F}(e_i) = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt = \frac{1}{I}\int_0^I e^{-\lambda t}dt = \frac{1-e^{-\lambda I}}{\lambda I} = \frac{1-e^{-\lambda/f}}{\lambda/f}$$

Fixed-order policy

- We assumed that all elements change at the same frequency λ and that they are synchronized at the same interval I, so the above equation holds for any element e_i

- Therefore, the freshness of database S is

$$\bar{F}(S) = \frac{1}{N}\sum_{i=1}^{N}\bar{F}(e_i) = \frac{1-e^{-\lambda/f}}{\lambda/f}$$

- We can analyze the age of S similarly, and we get

$$\bar{A}(S) = I\left(\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)$$

Random-order policy

- Under the random-order policy, the synchronization order of elements might be different from one crawl to the next
- Obviously, the random-order policy is more complex to analyze than the fixed-order policy. Since we may synchronize e_i at any point during interval I, the synchronization interval of e_i is not fixed any more. In one extreme case, it may be almost 2I, when e_i is synchronized at the beginning of the first iteration and at the end of the second iteration.
- In the opposite case, it may be close to 0, when e_i is synchronized at the end of the first iteration and at the beginning of second iteration Therefore, the synchronization interval of e_i, W, is not a fixed number any more, but follows a certain distribution $f_W(t)$. Therefore the equation of Theorem 5.2 should be modified accordingly:

Random-order policy

$$\bar{F}(e_i) = \lim_{t\to\infty} \frac{1}{t} \int_0^t F(e_i;t)dt = \frac{\int_0^{2I} f_W(s) \left(\int_0^s \mathrm{E}[F(e_i;t)]dt\right) ds}{\int_0^{2I} f_W(s)\ s\ ds} \tag{5.3}$$

In Theorem 5.2, we simply divided $\int_0^I \mathrm{E}[F(e_i;t)]dt$ by I, because the synchronization interval was fixed to I. But now we take the average of $\int_0^s \mathrm{E}[F(e_i;t)]dt$ and s weighted by the relative frequencies of the interval s ($f_W(s)$). To perform the above integration, we need to derive the closed form of $f_W(t)$.

Lemma 5.1 *Let T_1 (T_2) be the time when element e_i is synchronized in the first (second) iteration under the random-order policy. Then the p.d.f. of $W = T_1 - T_2$, the synchronization interval of e_i, is*

$$f_W(t) = \begin{cases} \frac{t}{I^2} & 0 \le t \le I \\ \frac{2I-t}{I^2} & I \le t \le 2I \\ 0 & \text{otherwise.} \end{cases}$$

□

Random-order policy

Proof The p.d.f.'s of T_1 and T_2 are

$$f_{T_1}(t) = \begin{cases} \frac{1}{I} & 0 \le t \le I \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad f_{T_2}(t) = \begin{cases} \frac{1}{I} & I \le t \le 2I \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\begin{aligned} f_W(t) &= f(T_2 - T_1 = t) \\ &= \int_0^I f(T_1 = s) f(T_2 - T_1 = t | T_1 = s)\, ds \\ &= \int_0^I f(T_1 = s) f(T_2 = s + t)\, ds \\ &= \frac{1}{I} \int_0^I f(T_2 = s + t)\, ds. \end{aligned}$$

■

Random-order policy

When $t < 0$ or $t > 2I$, $f(T_2 = s + t) = 0$ for any $s \in (0, I)$. Therefore,

$$f_W(t) = \frac{1}{I} \int_0^I f(T_2 = s + t)\, ds = 0.$$

When $0 \le t \le I$, $f(T_2 = s + t) = \frac{1}{I}$ for $s \in (I - t, I)$. Then,

$$f_W(t) = \frac{1}{I} \int_{I-t}^{I} \frac{1}{I}\, ds = \frac{t}{I^2}.$$

When $I \le t \le 2I$, $f(T_2 = s + t) = \frac{1}{I}$ for $s \in (0, 2I - t)$, and therefore

$$f_W(t) = \frac{1}{I} \int_0^{2I-t} \frac{1}{I}\, ds = \frac{2I - t}{I^2}.$$

Random-order policy

- Based on the Lemma 5.1 and Equation 5.3, we can compute the freshness of the random-order policy, and the result is

$$\bar{F}(e_i) = \frac{1}{\lambda/f}\left[1 - \left(\frac{1-e^{-\lambda/f}}{\lambda/f}\right)^2\right]$$

- Since the above analysis is valid for any element e_i, the freshness of S becomes

$$\bar{F}(S) = \frac{1}{\lambda/f}\left[1 - \left(\frac{1-e^{-\lambda/f}}{\lambda/f}\right)^2\right]$$

- We can compute $\bar{A}(S)$ similarly

$$\bar{A}(S) = I\left[\frac{1}{3} + \left(\frac{1}{2} - \frac{1}{\lambda/f}\right)^2 - \left(\frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)^2\right]$$

Purely-random policy

- The analysis of purely-random policy is similar to that of random-order policy. Here again, the time between synchronizations of e_i, W, is a random variable with a probability density function $f_W(t)$, and the freshness of e_i becomes:

$$\bar{F}(e_i) = \lim_{t\to\infty} \frac{1}{t} \int_0^t F(e_i;t)dt = \frac{\int_0^\infty f_W(s) \left(\int_0^s \mathrm{E}[F(e_i;t)]dt\right) ds}{\int_0^\infty f_W(s)\, s\, ds}$$

- We can prove (from the law of rare events) that $f_W(t)$ is:

$$f_W(t) = \begin{cases} fe^{-ft} & t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and we get: $\bar{F}(S) = \dfrac{1}{1+\lambda/f}$ $\qquad \bar{A}(S) = I\left(\dfrac{\lambda/f}{1+\lambda/f}\right)$

Synchronization-order policies: Summary

policy	*Freshness* $\bar{F}(S)$	*Age* $\bar{A}(S)$
Fixed-order	$\frac{1-e^{-r}}{r}$	$I(\frac{1}{2} - \frac{1}{r} + \frac{1-e^{-r}}{r^2})$
Random-order	$\frac{1}{r}(1 - (\frac{1-e^{-r}}{r})^2)$	$I(\frac{1}{3} + (\frac{1}{2} - \frac{1}{r})^2 - (\frac{1-e^{-r}}{r^2})^2)$
Purely-random	$\frac{1}{1+r}$	$I(\frac{r}{1+r})$

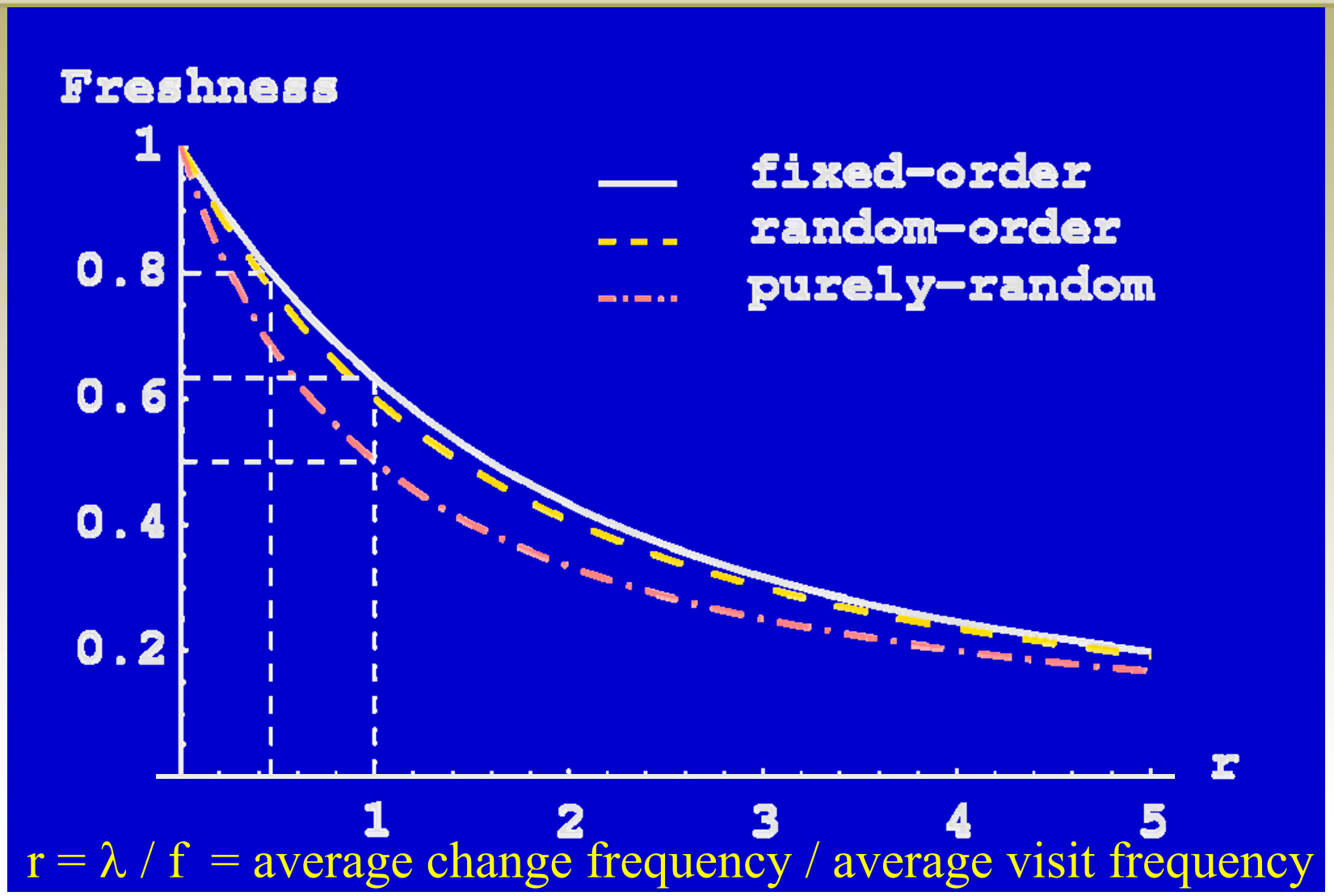
Freshness and age formula for various synchronization-order policies

Algorithm 5.4.1 Fixed-order synchronization
Input: `ElemList` $= \{e_1, e_2, \ldots, e_N\}$
Procedure
[1] While (TRUE)
[2] `SyncQueue` := `ElemList`
[3] While (not Empty(`SyncQueue`))
[4] `e` := Dequeue(`SyncQueue`)
[5] Synchronize(`e`)

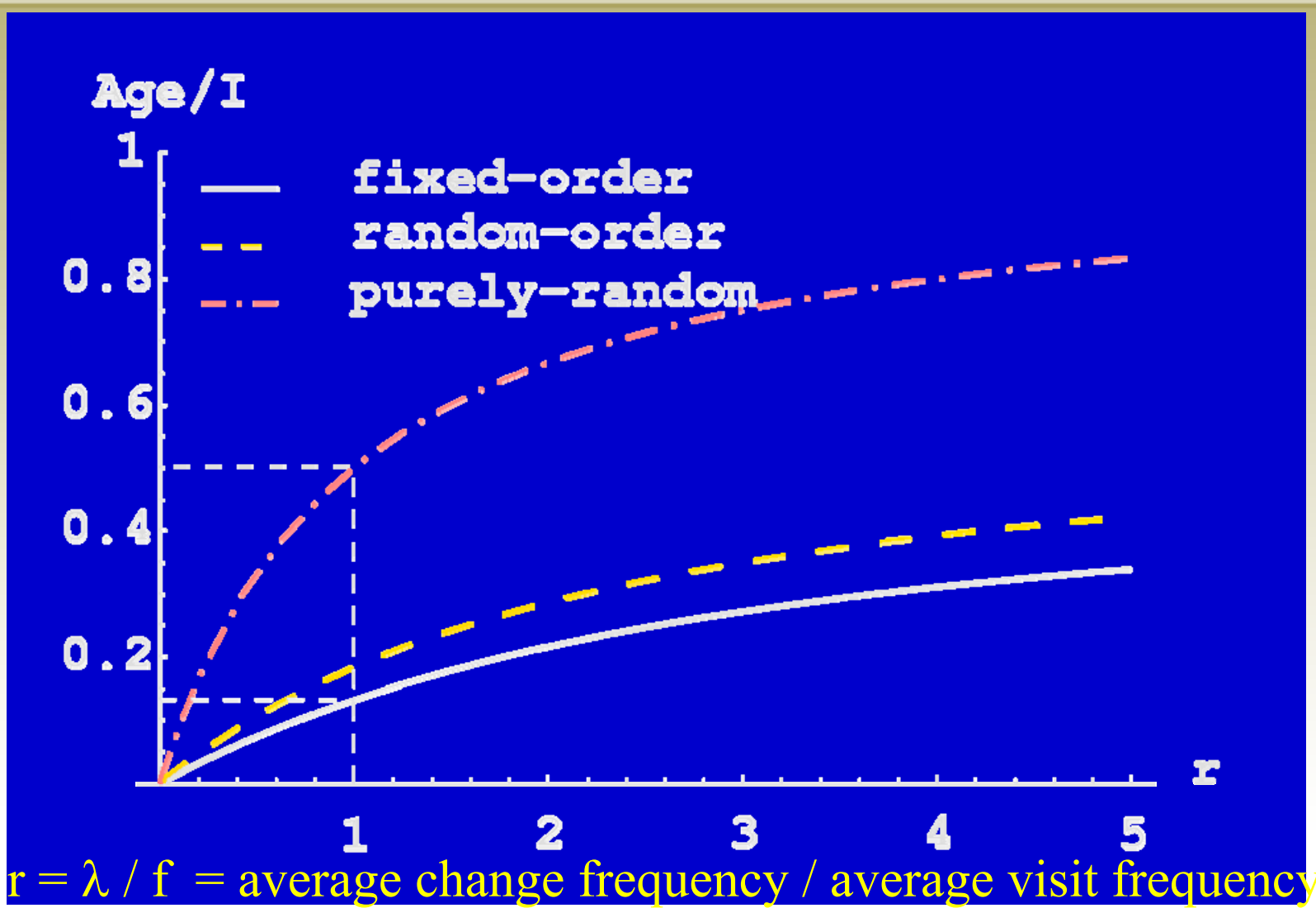
Algorithm 5.4.2 Random-order synchronization
Input: `ElemList` $= \{e_1, e_2, \ldots, e_N\}$
Procedure
[1] While (TRUE)
[2] `SyncQueue` := RandomPermutation(`ElemList`)
[3] While (not Empty(`SyncQueue`))
[4] `e` := Dequeue(`SyncQueue`)
[5] Synchronize(`e`)

Algorithm 5.4.3 Purely-random synchronization
Input: `ElemList` $= \{e_1, e_2, \ldots, e_N\}$
Procedure
[1] While (TRUE)
[2] `SyncQueue` := RandomPermutation(`ElemList`)
[3] `e` := Dequeue(`SyncQueue`)
[4] Synchronize(`e`)

Synchronization-order policies: Summary



Synchronization-order policies: Summary



r = λ / f = average change frequency / average visit frequency

Resource allocation policies

- Previously, we addressed various questions assuming that all elements in the database change at the same rate
- But what can we do if the elements change at *different* rates and we know how often each element changes?
- Is it better to synchronize an element more often when it changes more often?
- We address this question by analyzing different resource-allocation policies

Resource allocation policies

- For the analysis, we model the real-world database by the *non-uniform* change-frequency model, and we assume the *fixed-order* policy for the synchronization-order policy, because the fixed-order policy is the best synchronization-order policy
- In other words, we assume that the element ei changes at the frequency λ_i (λ_i's may be different from element to element), and we synchronize e_i at the *fixed interval* $I_i(=1/f_i$, where f_i is synchronization frequency of $e_i)$
- Remember that we synchronize N elements in I(= 1/f) time units. Therefore, the average synchronization frequency $(1/N * \Sigma_{i=1}^{N} f_i)$ should be equal to f

Resource allocation policies

- We first assume that the change frequencies of real-world elements follow the *gamma distribution*, and compare how effective the *proportional* and the *uniform* policies are
- The gamma distribution is often used to model a random variable whose domain is non-negative numbers
- Also, the distribution is known to cover a wide array of distributions
  - For instance, the exponential and the chi-square distributions are special instances of the gamma distribution, and the gamma distribution is close to the normal distribution when the variance is small. This mathematical property and versatility makes the gamma distribution a desirable one for describing the distribution of change frequencies

Resource allocation policies

To compute the freshness, we assume that we synchronize element e_i at the *fixed* frequency f_i (Fixed-order policy, Item 3a of Section 5.3). In Section 5.4.1, we showed that the freshness of e_i in this case is

$$\bar{F}(\lambda_i, f_i) = \frac{1 - e^{-\lambda_i/f_i}}{\lambda_i/f_i} \tag{5.4}$$

and the age of e_i is

$$\bar{A}(\lambda_i, f_i) = \frac{1}{f_i}\left(\frac{1}{2} - \frac{1}{\lambda_i/f_i} + \frac{1 - e^{-\lambda_i/f_i}}{(\lambda_i/f_i)^2}\right). \tag{5.5}$$

The gamma distribution $g(x)$ with parameters $\alpha > 0$ and $\mu > 0$ is

$$g(x) = \frac{\mu}{\Gamma(\alpha)}(\mu x)^{\alpha-1} e^{-\mu x} \qquad \text{for} \quad x > 0 \tag{5.6}$$

and the mean and the variance of the distribution are

$$\mathrm{E}[X] = \frac{\alpha}{\mu} \quad \text{and} \quad \mathrm{Var}[X] = \frac{\alpha}{\mu^2}. \tag{5.7}$$

Resource allocation policies: Uniform

Now let us compute the freshness of S for the uniform allocation policy. By the definition of the uniform allocation policy, $f_i = f$ for any i. Then from Theorem 5.1,

$$\bar{F}(S)_u = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(e_i) = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f)$$

where subscript u stands for the uniform allocation policy. When N is large, we can approximate the above average by the weighted integral

$$\bar{F}(S)_u = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f) = \int_0^{\infty} g(\lambda)\bar{F}(\lambda, f)d\lambda \tag{5.8}$$

where $g(\lambda)$ is the gamma distribution. By substituting $g(\lambda)$ and $\bar{F}(\lambda, f)$ using Equation 5.6 and 5.4, we get

$$\bar{F}(S)_u = \frac{\mu f}{1-\alpha}\left((1+\frac{1}{\mu f})^{1-\alpha} - 1\right).$$

It is more intuitive to deal with the *mean* and the *variance* of the distribution than α and μ, so we reformulate the above formula with

$$\lambda = (\text{mean}) = \frac{\alpha}{\mu} \quad \text{and} \quad \delta^2 = \frac{(\text{variance})}{(\text{mean})^2} = \frac{\alpha/\mu^2}{(\alpha/\mu)^2} \qquad \text{(From Equation 5.7).}$$

Resource allocation policies: Uniform

Then $\bar{F}(S)_u$ becomes

$$\bar{F}(S)_u = \frac{1-(1+r\delta^2)^{1-\frac{1}{\delta^2}}}{r(1-\delta^2)} \quad (r=\lambda/f)$$

.

We can compute the age of the database, $\bar{A}(S)_u$, similarly.

$$\bar{A}(S)_u = \frac{1}{N}\sum_{i=1}^{N}\bar{A}(\lambda_i, f) = \int_0^\infty g(\lambda)\bar{A}(\lambda, f)d\lambda$$

$$= \frac{1}{f(1-\delta^2)}\left[\frac{1-\delta^2}{2} - \frac{1}{r} + \frac{1-(1+r\delta^2)^{2-\frac{1}{\delta^2}}}{r^2(1-2\delta^2)}\right]$$

Resource allocation policies: Proportional

Now let us compute the freshness of the proportional allocation policy. By the definition of the proportional policy, $\lambda_i/f_i = \lambda/f$ for any i. Then from Equation 5.4 and 5.5, we can derive

$$\bar{F}(\lambda_i, f_i) = \frac{1-e^{-\lambda/f}}{\lambda/f} \qquad \bar{A}(\lambda_i, f_i) = \frac{1}{\lambda_i}\left[\frac{\lambda}{f}\left(\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)\right].$$

Therefore, $\bar{F}(S)_p$ and $\bar{A}(S)_p$ becomes

$$\bar{F}(S)_p = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f_i) = \frac{1-e^{-\lambda/f}}{\lambda/f}$$

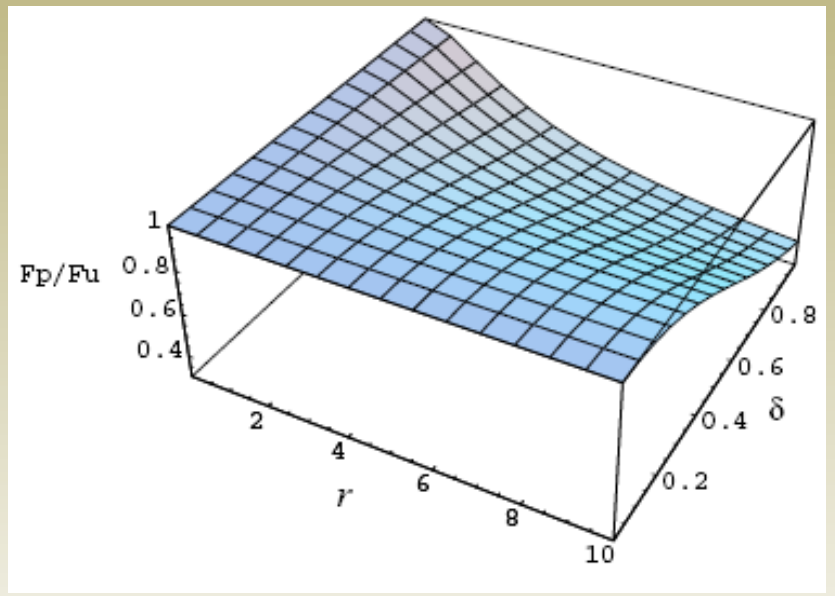
$$\begin{aligned}
\bar{A}(S)_p &= \frac{1}{N}\sum_{i=1}^{N} \bar{A}(\lambda_i, f_i) \\
&= \left[\frac{\lambda}{f}\left(\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)\right]\left(\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\lambda_i}\right) \\
&= \left[\frac{\lambda}{f}\left(\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)\right]\int_0^{\infty} g(\lambda)\,\frac{1}{\lambda}\,d\lambda \\
&= \frac{1}{f(1-\delta^2)}\left[\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right].
\end{aligned}$$

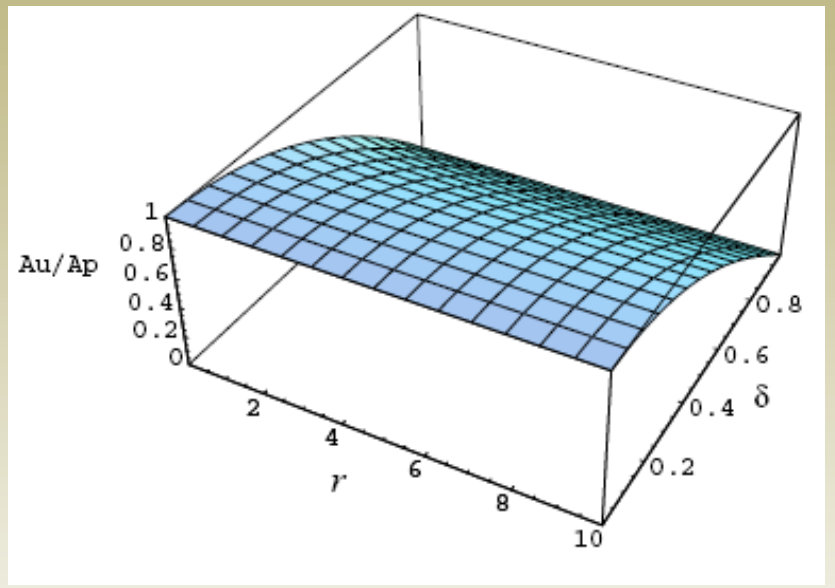
Resource allocation policies: Summary

policy	*Freshness* $\bar{F}(S)$	*Age* $\bar{A}(S)$
Uniform allocation	$\frac{1-(1+r\delta^2)^{1-\frac{1}{\delta^2}}}{r(1-\delta^2)}$	$\frac{I}{(1-\delta^2)}[\frac{1-\delta^2}{2} - \frac{1}{r} + \frac{1-(1+r\delta^2)^{2-\frac{1}{\delta^2}}}{r^2(1-2\delta^2)}]$
Proportional allocation	$\frac{1-e^{-r}}{r}$	$\frac{I}{(1-\delta^2)}[\frac{1}{2} - \frac{1}{r} + \frac{1-e^{-r}}{r^2}]$

- In the table, r represents the frequency ratio λ/f, where λ is the average rate at which elements change (the mean of the gamma distribution), and f is the average rate at which we synchronize them (1/I)
- Also, δ represents the standard deviation of change frequencies (more precisely, δ^2 =(variance)/(mean)^2 of the gamma distribution)
- Note that the uniform policy is better than the proportional one if $\bar{F}(S)_p < \bar{F}(S)_u$ and $\bar{A}(S)_u < \bar{A}(S)_p$

Resource allocation policies: Summary





- Surprisingly, we can clearly see that the ratios are below 1 for any r and δ values: **The uniform policy is always better than the proportional policy!**
- In fact, the uniform policy gets more effective as the elements change at more different frequencies

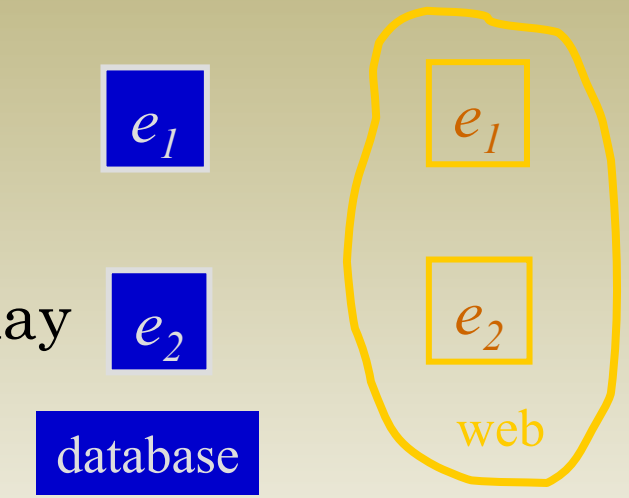
Uniform vs. Proportional

- When the variance of change frequencies is zero ($\delta=0$), all elements change at the same frequency, the two policies give the same result
- We can observe that the age ratio does not change much as r increases, while the freshness ratio heavily depends on the r value

- While we showed that the uniform policy is better than the proportional one only for the gamma-distribution assumption, it is in fact a very general conclusion: we can prove that the uniform policy is always better than the proportional policy under *any* distribution

Uniform vs. Proportional: A two page Web

- Two page database
- *e1* changes 9 times/day
- *e2* changes once/day
- Can synchronize only one page per day
- We do not know exactly when the element changes in one interval
- How should we visit pages?
  - *e1 e2 e1 e2 e1 e2 e1 e2*... [uniform]
  - *e1 e1 e1 e1 e1 e1 e1 e2 e1 e1* ... [proportional]
  - *e1 e1 e1 e1 e1 e1* ...
  - *e2 e2 e2 e2 e2 e2* ...
  - ?



Uniform vs. Proportional: A two page Web

- To answer this question, we need to compare how the freshness changes if we pick one element over the other
- If the element e2 changes in the middle of the day and if we synchronize e2 right after it changed, it will remain up-to-date for the remaining half of the day
- Therefore, by synchronizing element e2 we get 1/2 day "benefit"(or freshness increase)
- However, the probability that e2 changes before the middle of the day is 1/2, so the "expected benefit" of synchronizing e2 is 1/2 × 1/2 day = 1/4 day

Uniform vs. Proportional: A two page Web

- By the same reasoning, if we synchronize e1 in the middle of an interval, e1 will remain up-to-date for the remaining half of the interval (1/18 of the day) with probability ½
- Therefore, the expected benefit is 1/2×1/18 day = 1/36 day

- From this crude estimation, we can see that it is more effective to select e2 for synchronization!

Uniform vs. Proportional: A two page Web

- The table shows the expected benefits for several other scenarios
- The second column shows the total synchronization frequencies (f1+f2) and the third column shows how much of the synchronization is allocated to f1 and f2
- In the fourth column we estimate the expected benefit, and in the last column we show the f1 and f2 values that give the highest expected benefit
- Note that since $\lambda 1 = 9$ and $\lambda 2 = 1$, row (h) corresponds to the proportional policy (f1 = 9, f2 = 1), and row (j) corresponds to the uniform policy (f1=f2=5)

row	$f_1 + f_2$	f_1	f_2	benefit	best	
(a)	1	1	0	$\frac{1}{2} \times \frac{1}{18} = \frac{1}{36}$	0	1
(b)		0	1	$\frac{1}{2} \times \frac{1}{2} = \frac{9}{36}$		
(c)	2	2	0	$\frac{1}{2} \times \frac{1}{18} + \frac{1}{2} \times \frac{1}{18} = \frac{2}{36}$	0	2
(d)		1	1	$\frac{1}{2} \times \frac{1}{18} + \frac{1}{2} \times \frac{1}{2} = \frac{10}{36}$		
(e)		0	2	$\frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times \frac{1}{3} = \frac{12}{36}$		
(f)	5	3	2	$\frac{3}{36} + \frac{12}{36} = \frac{30}{72}$	2	3
(g)		2	3	$\frac{2}{36} + \frac{6}{16} = \frac{31}{72}$		
(h)	10	9	1	$\frac{9}{36} + \frac{1}{4} = \frac{36}{72}$	7	3
(i)		7	3	$\frac{7}{36} + \frac{6}{16} = \frac{41}{72}$		
(j)		5	5	$\frac{5}{36} + \frac{15}{36} = \frac{40}{72}$		

Uniform vs. Proportional: A two page Web

- From the table, we can observe the following interesting trends:
  1. **Rows (a)-(e)**: When the synchronization frequency (f1 + f2) is much smaller than the change frequency ($\lambda 1+\lambda 2$), it is better to give up synchronizing the elements that change too fast. In other words, when it is not possible to keep up with everything, it is better to focus on what we can track.
  2. **Rows (h)-(j)**: Even if the synchronization frequency is relatively large (f1 + f2 = 10), the uniform allocation policy (row (j)) is more effective than the proportional allocation policy (row (h)). The optimal point (row (i)) is located somewhere between the proportional policy and the uniform policy

Auxiliary slides on crawling

Web Crawling

Christopher Olston and Marc Najork

Slides created by *Aécio Solano Rodrigues Santos* and *Nivio Ziviani*
based on the survey *Web Crawling* from *Foundations and Trends in Information Retrieval* (2010).

Summary

Summary

Introduction

- A *web crawler* (also known as a *robot* or a *spider*) is a system for the bulk downloading of web pages.
- They are used for a variety of purposes:
  - Web Search Engines
  - Web Archiving
  - Web Data Mining
  - Web Monitoring

Introduction

- The web is not a centrally managed repository of information, but rather consists of hundreds of millions of independent web content providers.
- Content aggregators (such as search engines or web data miners) have two choices:
  - Adopt a **pull** model where they will proactively scour the web for new or updated information
  - Try to establish a convention and a set of protocols enabling content providers to **push** content of interest to the aggregators.
- One of the earliest search services, adopted the push model. However, this approach did not succeed, and virtually all content aggregators adopted the pull aproach.

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 broad coverage and good freshness must achieve extremely high throughput, which poses many difficult engineering problems.
- **Content selection tradeoffs**
  - Crawlers do not purport to crawl the whole web, or keep up with all the changes.
  - The goals are to acquire high-value content quickly, ensure eventual coverage of all reasonable content, and bypass low-quality, irrelevant, redundant, and malicious content.
  - The crawler must balance competing objectives such as 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 the right safety mechanisms a high-throughput crawler can inadvertently carry out a *denial-of-service attack*.
- **Adversaries**
  - Some content providers seek to inject useless or misleading content into the corpus assembled by the crawler.
  - Such behavior is often motivated by financial incentives, for example (mis)directing traffic to commercial web sites.

Summary

Chonology

- In the spring of 1993, Matthew Gray implemented the *World Wide Web Wanderer*
- The pages crawled by the Wanderer were compiled into an index (the “Wandex”), thus giving rise to the first search engine on the Web.
- It was used until 1996 to collect statistics about the evolution of the Web.
- In December 1993, three more crawler-based Internet Search engines became available:
  - Jump-Station
  - World Wide Web Worm [90]
  - RBSE spider [57]

Chonology

- *WebCrawler* [108] joined the field in April 1994
- *MOMspider* [61] was drescribed in the same year
- This first generation of crawlers identified some of the defining issues in web crawler design.
- *MOMspider* considered politeness policies:
  - it limited the rate of requests to each site
  - it allowed web sites to exclude themselves from purview through the nascent robots exclusion protocol [83]
  - it provided a “black-list” mechanism that allowed the crawl operator to exclude sites.

Chonology

- However, the design of these early crawlers did not focus on scalability.
- Several of them (RBSE spider and WebCrawler) used general-purpose database management systems to store the state of the crawl.
- The following few years saw the arrival of several commercial search engines:
  - Lycos, Infoseek, Excite, AltaVista, and HotBot
- All of which used crawlers to index tens of millions of pages; however, the design of these crawlers remains undocumented.

Chonology

- Brin and Page's 1998 paper outlining the architecture of the first generation *Google* [25] system contains a short description of their crawler.
- With the *Mercator* web crawler, Heydon and Najork presented a "blueprint design" for web crawlers [75, 94].
  - The second Mercator paper gave statistics of a 17-day, four-machine crawl that covered 891 million pages.
- Shkapenyuk and Suel's *Polybot* web crawler [111] represents another "blueprint design".
  - Polybot is a distributed system, consisting of a crawl manager process, multiple downloader processes, and a DNS resolver process.
  - Polybot was able to download 120 million pages over 18 days using four machines.

Chonology

- The *IBM WebFountain* crawler [56] represented another industrial-strength design.
  - The WebFountain crawler was fully distributed.
  - The three major components were:
    - Multi-threaded crawling processes (“Ants”).
    - Processes responsible for identifying downloaded pages with near-duplicate content.
    - A central controller.
- *UbiCrawler* [21] is another scalable distributed web crawler.
  - It uses consistent hashing to partition URLs according to their host component across crawling machines, leading to graceful performance degradation in the event of the failure of a crawling machine.
  - UbiCrawler was able to download about 10 million pages per day using five crawling machines.

Chonology

- Recently, Yan et al. described IRLbot [84]:
  - A single-process web crawler that is able to scale to extremely large web collections without performance degradation.
  - The paper describes a crawl that ran over two months and downloaded about 6.4 billion web pages.
- Finally, there are a number of open-source crawlers, two of which deserve special mention:
  - Heritrix [78, 93] is the crawler used by the Internet Archive. It is written in Java and highly componentized, and its design is quite similar to that of Mercator.
  - The Nutch crawler [62, 81] is written in Java as well. It supports distributed operation and should therefore be suitablefor very large crawls.

Architecture Overview

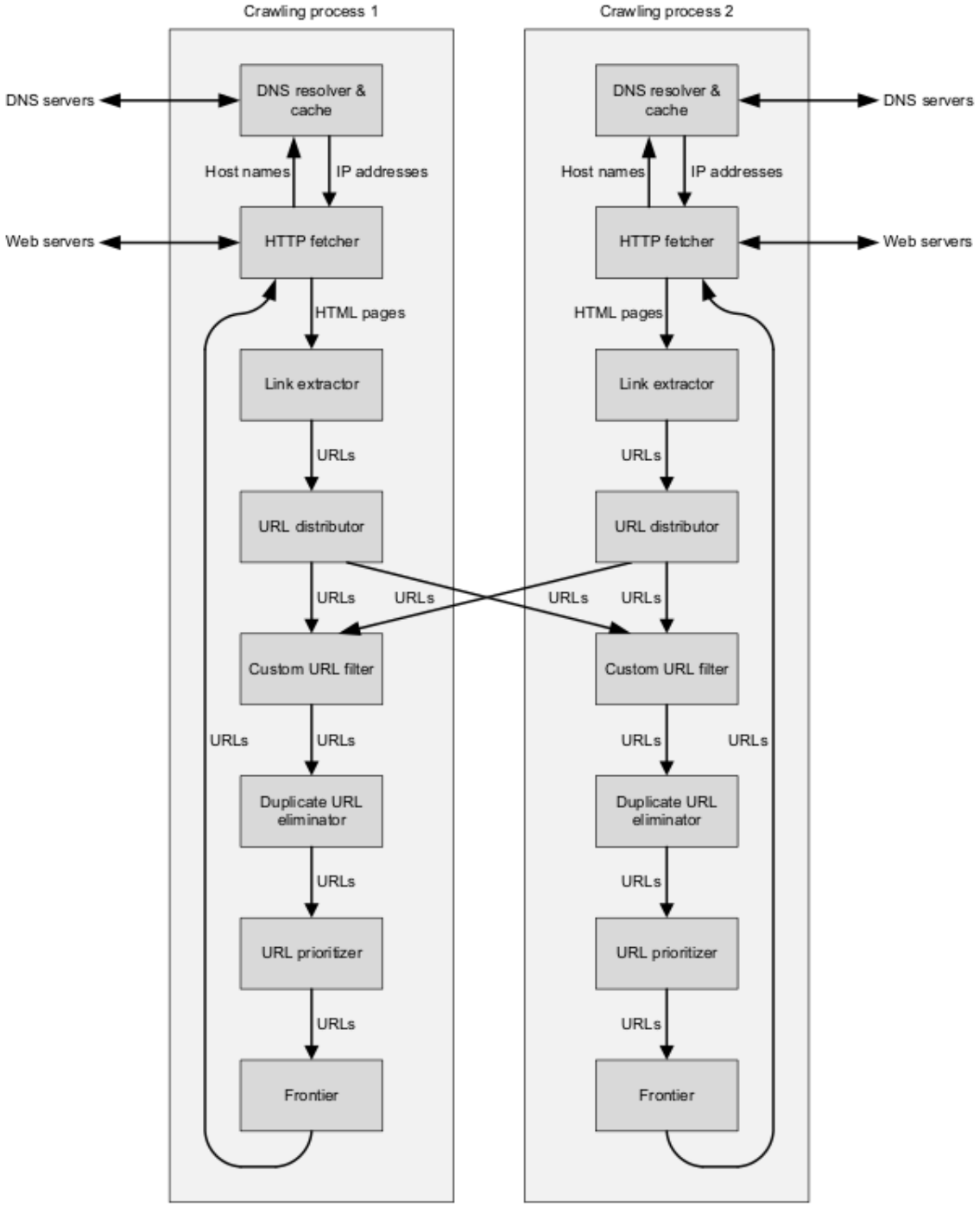
- The crawler consists of multiple processes running on different machines connected by a high-speed network.
- Each crawling process consists of multiple worker threads.
- Each worker thread performs repeated work cycles.
- At the beginning of each work cycle, a worker obtains a URL from the **Frontier** data structure, which dispenses URLs according to their priority and to politeness policies.
- The worker thread then invokes the **HTTP fetcher**.
  - The fetcher first calls a DNS sub-module to resolve the host component of the URL into the IP address of the corresponding web server (using cached results of prior resolutions if possible)
  - Then connects to the web server and checks for any robots exclusion rules (which typically are cached as well) and attempts to download the web page.

Architecture Overview

- If the download succeeds, the web page may or may not be stored in a repository of harvested web pages (not shown).
- The page is passed to the **Link extractor**, which parses the page's HTML content and extracts hyperlinks contained therein.
- The corresponding URLs are then 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 (the latter requires additional DNS resolutions).
- Since most hyperlinks refer to pages on the same web site, assignment to the local crawling process is the common case.

Architecture Overview

- Next, the URL passes through the two following modules:
  - **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**, which maintains the set of all URLs discovered so far and passes on only never-before-seen URLs.
- Finally, the **URL prioritizer** selects a position for the URL in the Frontier, based on factors such as estimated page importance or rate of change.


Crawling process 1
Crawling process 2
DNS servers
DNS resolver & cache
DNS servers
Host names
IP addresses
Host names
IP addresses
Web servers
HTTP fetcher
HTTP fetcher
Web servers
HTML pages
HTML pages
Link extractor
Link extractor
URLs
URLs
URL distributor
URL distributor
URLs
URLs
URLs
URLs
Custom URL filter
Custom URL filter
URLs
URLs
URLs
URLs
Duplicate URL eliminator
Duplicate URL eliminator
URLs
URLs
URL prioritizer
URL prioritizer
URLs
URLs
Frontier
Frontier

Key Design Points

- Any web crawler needs to keep track both of the URLs that are to be downloaded, as well as those that have already been downloaded.
- The required state is a set of URLs, each associated with a flag indicating whether the page has been downloaded.
- The operations that must be supported are:
  - Adding a new URL
  - Retrieving a URL
  - Marking a URL as downloaded
  - Testing whether the set contains a URL
- However, in-memory data structures that support these operations (e.g., trees or sorted lists) does not scale to web corpus sizes that exceed the amount of memory available on a single machine.

Key Design Points

- To scale beyond this limitation, one could either maintain the data structure (e.g., the tree or sorted list) on disk, or use an off-the-shelf database management system.
- However, the cost of accessing items in the set (particularly for the purpose of set membership test) typically involves a disk seek, making it a fairly expensive operation.
- To achieve high performance, a more specialized approach is needed.
- Virtually every modern web crawler splits the crawl state into two major data structures:
  - **URL-seen test** or **Duplicate URL eliminator**, which maintains the set of URLs that have been discovered (whether downloaded or not)
  - **Frontier**, which maintains the set of URLs that have yet to be downloaded.

Frontier Data Structure and Politeness

- A straightforward implementation of the frontier data structure is a First-in-First-out (FIFO) queue.
- Such an implementation results in a breadth-first traversal of the web graph.
- However, this simple approach has drawbacks:
  - Most hyperlinks on the web refer to another page on the same web server, resulting in the crawler issuing many consecutive HTTP requests to that server.
  - A barrage of requests in short order is considered “impolite”, and may be construed as a denial-of-service attack.
  - On the other hand, it would be wasteful for the web crawler to space out requests to the same server without doing other useful work in the meantime.

Frontier Data Structure and Politeness

- This problem is compounded in a multithreaded or distributed crawler that issues many HTTP requests in parallel.
- Most web crawlers obey a policy of not issuing multiple overlapping requests to the same server.
- An easy way to realize this is to maintain a mapping from web servers to crawling threads, e.g., by hashing the host component of each URL.
- In this design, each crawling thread has a separate FIFO queue, and downloads only URLs obtained from that queue.

Frontier Data Structure and Politeness

- A more conservative politeness policy is to 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.
- In other words, this crawling policy is biased toward well-provisioned web sites.

Frontier Data Structure and Politeness

- Such a policy is well-suited to the objectives of search engines, since large and popular web sites tend also to be well-provisioned.
- In addition, web crawlers may also want to prioritize the URLs in the frontier.
- For example, it may be desirable to prioritize pages according to their estimated usefulness, based for example on:
  - PageRank
  - Traffic they receive
  - Reputation of the web site
  - Rate at which the page has been updated in the past

URL Seen Test

- The second major data structure in any modern crawler is sometimes called **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, in order to cope with URLs that no longer point to a valid page.

URL Seen Test

- There are multiple straightforward in-memory implementations of a UST
  - e.g., a hash table or Bloom filter.
- In-memory implementations do not scale to arbitrarily large web corpora.
- However, they scale much further than in-memory implementations of the frontier, since each URL can be compressed to a much smaller token.
  - e.g., a 10-byte hash value.
- Commercial search engines employ distributed crawlers, and a hash table realizing the UST can be partitioned across the machines in the crawling cluster, further increasing the limit of how far such an in-memory implementation can be scaled out.

URL Seen Test

- In a disk-based hash table, each lookup requires a disk seek, severely limiting the throughput.
- Caching popular URLs can increase the throughput by about an order of magnitude [27] to a few thousand lookups per second.
- But given that the average web page contains on the order of a hundred links and that each link needs to be tested for novelty, the crawling rate would still be limited to tens of pages per second under such an implementation.
- While latency of disk seeks is poor (a few hundred seeks per second), the *bandwidth* of disk reads and writes is quite high.
- So, implementations performing random file accesses perform poorly, but those that perform streaming sequential reads or writes can achieve reasonable throughput.

URL Seen Test

- The Mercator crawler leveraged this observation by:
  - Aggregating many set lookup and insertion operations into a single large batch.
  - Processing this batch by sequentially reading a set of sorted URL hashes from disk and writing them (plus the hashes of previously undiscovered URLs) out to a new file [94].
- This approach implies that:
  - The set membership is delayed: we only know whether a URL is new after the batch containing the URL has been merged with the disk file.
  - Adding URLs to the frontier in a delayed fashion also means that there is a lower bound on how soon they can be crawled; however, given that the frontier is usually far larger than a DUE batch, this delay is imperceptible except for the most high-priority URLs.

URL Seen Test

- The IRLbot crawler [84] uses a refinement of the Mercator scheme.
- Batch of URLs arriving at the DUE is also 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.
- Because IRLbot stores the batch on disk, the size of a single batch can be much larger than Mercator's in-memory batches, so the cost of the merge-sort with the main URL hash file is amortized over a much larger set of URLs.

Auxiliary Data Structures

- Web crawlers maintain various auxiliary data structures. we discuss two: The robots exclusion rule cache and the DNS cache.
- The *Robots Exclusion Protocol* [83], is a convention that allows a web site administrator to bar web crawlers from crawling their site, or some pages within the site.
- This is done by providing a file at URL `/robots.txt` containing rules that specify which pages the crawler is allowed to download.
- Before attempting to crawl a site, 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 typically cache the results of previous requests of that file.

Auxiliary Data Structures

- 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.
- As with the robots exclusion rule cache, entries are expired according to both a standard eviction policy (such as least-recently used), and to expiration directives.

Distributed Crawling

- Web crawlers can be distributed over multiple machines to increase their throughput.
- This is done by partitioning the URL space, such that each crawler machine or node is responsible for a subset of the URLs on the web.
- Partitioning the URL space across site boundaries makes it easy to obey politeness policies, since each crawling process can schedule downloads without having to communicate with other crawler nodes.
- Moreover, all the major data structures can easily be partitioned across site boundaries.

Distributed Crawling

- Thanks to the prevalence of relative links on the web, they will be themselves responsible for the large majority of extracted URLs.
- When a process extracts a URL u that falls under the responsibility of another crawler node, it forwards u to that node.
- The amount of communication with other crawler nodes can be reduced by maintaining a cache of popular URLs, used to avoid repeat forwardings [27].

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*, where the resources of the crawler are divided between downloading newly discovered pages and re-downloading 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 frontier and downloaded as in batch crawling, but they are subsequently reentered into the frontier.

Incremental Web Crawling

- The priority of a previously downloaded URL should be dependent on a model of the page's temporal behavior based on past observations.
- In addition to content evolution, other factors such as page quality are also often taken into account.
- Indeed there are many fast-changing "spam" web pages.

Summary

Introduction

Crawler Architecture

Crawl Ordering Problem

Batch Crawl Ordering

Incremental Crawl Ordering

Avoiding Problematic and Undesirable Content

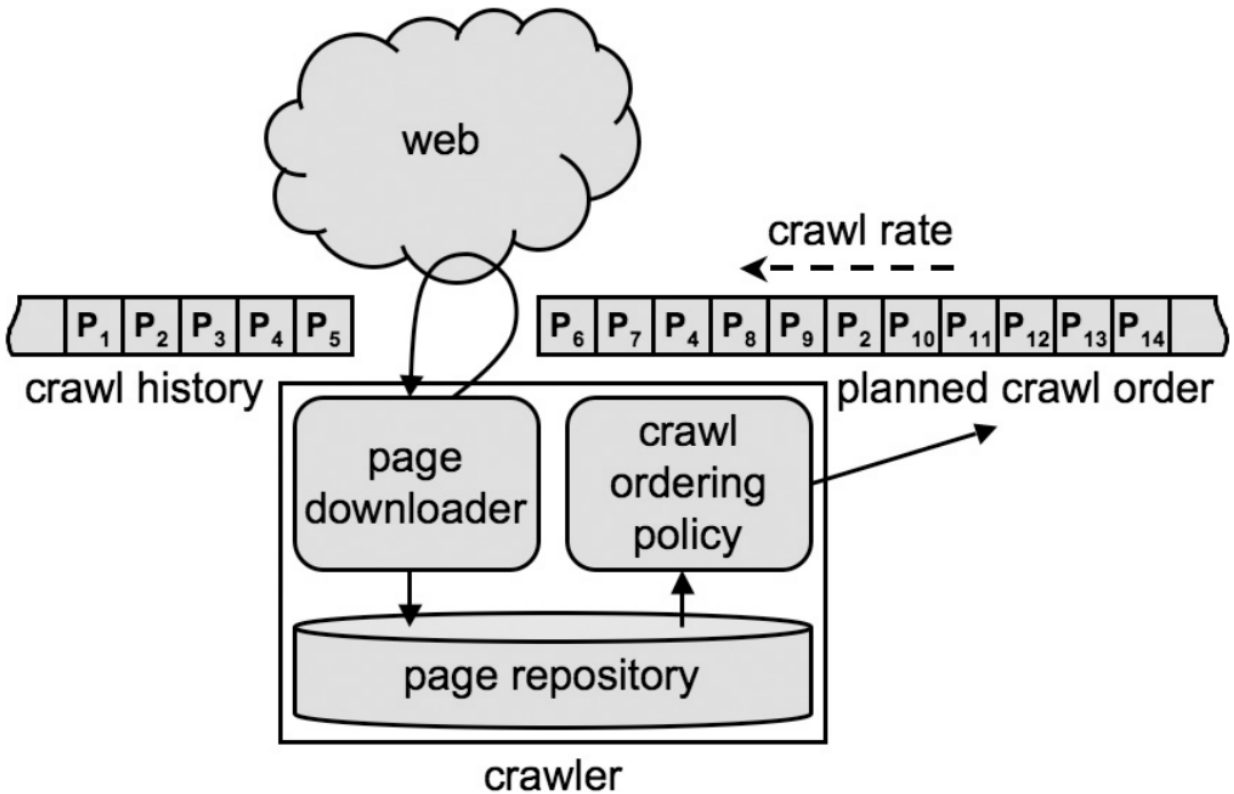
Deep Web Crawling

Crawl Ordering Problem

- Aside from the intra-site politeness considerations, a crawler is free to visit URLs in any order.
- The crawl order is extremely significant, because for the purpose of crawling the web can be considered infinite
  - due to the growth rate of new content
  - due to dynamically generated content [8]
- Next sections survey work on selecting a good crawler order, with a focus on two basic considerations:
  - **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.

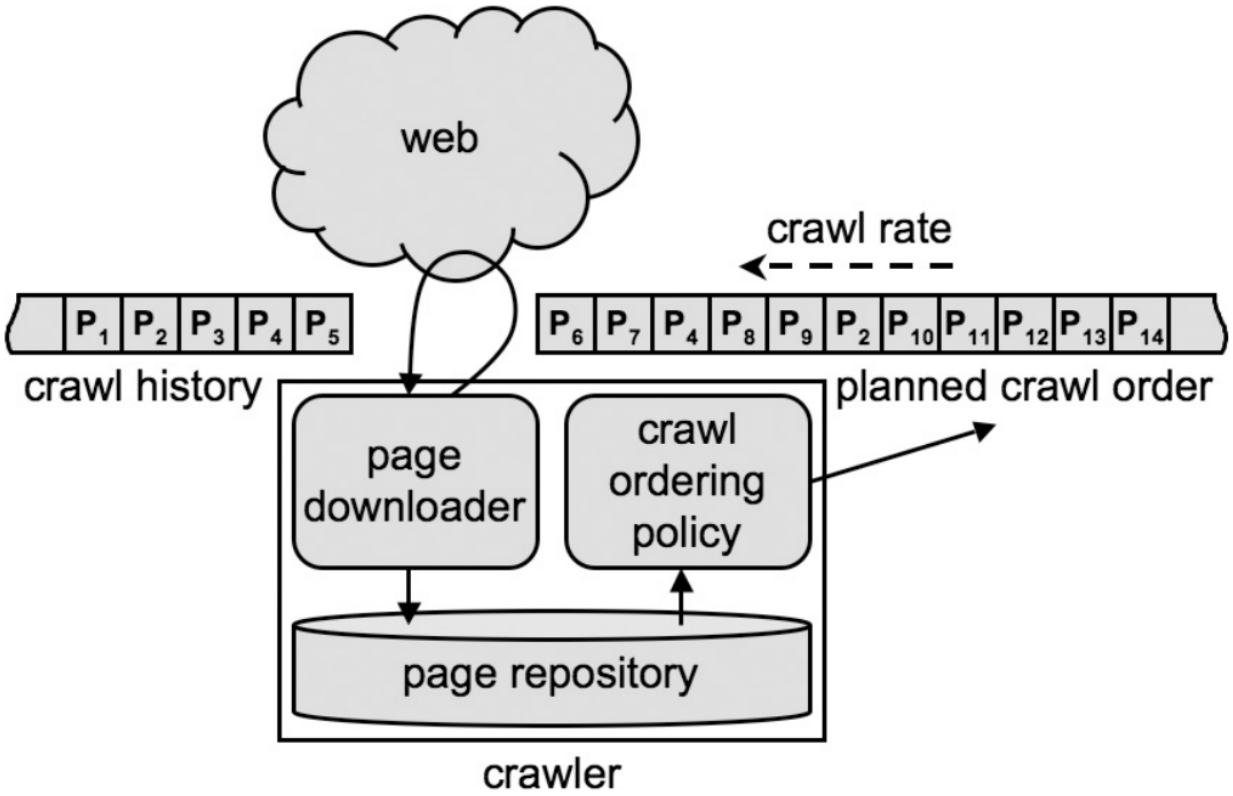
Model

- Most work on crawl ordering abstracts away the architectural details of a crawler, and assumes that URLs in the frontier data structure can be reordered freely.
- The resulting simplified crawl ordering model is depicted in figure.



Model

- 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.
- For example, *breadth-first search*, extracts hyperlinks from pages entering the repository, identifies linked-to pages that are not already part of the (historical or planned) crawl order, and adds them to the end of the planned crawl order.
- 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 exist for managing repeated downloads.
- **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.
- It is believed that most modern commercial crawlers perform incremental crawling, which is more powerful because it allows re-visitation of pages at different rates.

Limitations

- This model has led to a good deal of research with practical implications.
- However, as with all models, it simplifies reality.
- Some real-world considerations that fall outside the model.
- A large-scale crawler maintains its frontier data structure on disk, which limits opportunities for reordering.
- Some pages (or even versions of a page) take longer to download than others, due to differences in size and network latency.
- Crawlers take special care to space out downloads of pages from the same server, to obey politeness constraints.

Limitations

- Modern commercial crawlers utilize many simultaneous page downloader threads, running on many independent machines. Hence rather than a single totally ordered list of pages to download, it is more accurate to think of a set of parallel lists, encoding a partial order.
- Special care must be taken to avoid crawling redundant and malicious content.
- If the page repository runs out of space, and expanding it is not considered worthwhile, is becomes necessary to retire some of the pages stored there (although it may make sense to retain some metadata about the page, to avoid recrawling it).

Web Characteristics

- Before proceeding, we describe some structural and evolutionary properties of the web that are relevant to the crawl ordering question.
- Several studies of the structure of the web graph have been conducted.
- 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”.

Web Characteristics

- Hence there exist many ordered pairs of pages (P_1, P_2) such that there is no way to reach P_2 by starting at P_1 and repeatedly following hyperlinks.
- The implications for crawling are:
  1. One cannot simply crawl to depth N , for a reasonable value of N like $N = 20$, and be assured of covering the entire web graph;
  2. crawling “seeds” (the pages at which a crawler commences) should be selected carefully, and multiple seeds may be necessary to ensure good coverage.

Web Characteristics

- 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.

Web Characteristics

- Chang et al. [35] studied the “deep web”, i.e., web sites whose content is not reachable via hyperlinks and instead can only be retrieved by submitting HTML forms.
- The findings include:
  1. there are over one million deep web sites;
  2. more deep web sites have structured (multi-field) query interfaces than unstructured (single-field) ones;
  3. most query interfaces are located within a few links of the root of a web site, and are thus easy to find by shallow crawling from the root page.

Temporal Characteristics

- It is important to understand the temporal characteristics of the web, both in terms of:
  - Site-level evolution: the appearance and disappearance of pages on a site.
  - Page-level evolution: changing content within a page.
- Dasgupta et al. [48] and Ntoulas et al. [96] studied creation and retirement of pages and links inside a number of web sites, and found the following characteristics.

Temporal Characteristics

- **Site-Level Evolution**
  - New pages are created at a rate of 8% per week.
  - Pages are retired at a rapid pace, such that during the course of one year 80% of pages disappear.
  - New links are created at the rate of 25% per week, which is significantly faster than the rate of new page creation.
  - Links are retired at about the same pace as pages, with 80% disappearing in the span of a year.
  - It is possible to discover 90% of new pages by monitoring links spawned from a small, well-chosen set of old pages
    - for most sites, five or fewer pages suffice.
  - However, discovering the remaining 10% requires substantially more effort.
    - for some sites hundreds of pages must be monitored.

Temporal Characteristics

- **Page-Level Evolution**
  - Page change events are governed by a Poisson process, which means that changes occur randomly and independently, at least in the case of pages that change less frequently than once a day [39].
  - Page change frequencies span multiple orders of magnitude
    - sub-hourly, hourly, daily, weekly, monthly, annually
  - Each order of magnitude includes a substantial fraction of pages on the web [2, 39]. This finding motivates the study of non-uniform page revisitation schedules.
  - Change frequency is correlated with visitation frequency, URL depth, domain and topic [2], as well as page length [60].
  - A page's change frequency tends to remain stationary over time, such that past change frequency is a fairly good predictor of future change frequency [60].

Temporal Characteristics

- Pages with high change frequency tend to exhibit less cumulative change than pages with moderate change frequency.
- The amount of content that changed on a page in the past is a fairly good predictor of the amount of content that will change in the future.
- The degree of predictability varies from web site to web site [60, 96].
- Many changes are confined to a small, contiguous region of a web page [60, 85], and/or only affect transient words that do not characterize the core, time-invariant theme of the page [2].

Temporal Characteristics

- The temporal behavior of (regions of) web pages can be divided into three categories:
  - *Static*: no changes
  - *Churn*: new content supplants old content.
    - e.g., quote of the day.
  - *Scroll*: new content is appended to old content.
    - e.g., blog entries.
- Most web pages include at least some static content, resulting in an upper bound on the divergence between an old snapshot of a page and the live copy.
- One simple way to characterize a page is by:
  1. the divergence upper bound (i.e., the amount of non-static content), under some divergence measure;
  2. the amount of time it takes to reach the upper bound (i.e., the time taken for all non-static content to change) [2].

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

Technique	Objectives		Factors considered		
	Coverage	*Freshness*	*Importance*	*Relevance*	*Dynamicity*
Breadth-first search [43, 95, 108]	√				
Prioritize by indegree [43]	√		√		
Prioritize by PageRank [43, 45]	√		√		
Prioritize by site size [9]	√		√		
Prioritize by spawning rate [48]	√				√
Prioritize by search impact [104]	√		√	√	
Scoped crawling (Section 4.2)	√			√	
Minimize obsolescence [41, 46]		√	√		√
Minimize age [41]		√	√		√
Minimize incorrect content [99]		√	√		√
Minimize embarrassment [115]		√	√	√	√
Maximize search impact [103]		√	√	√	√
Update capture (Section 5.2)		√		√	√
WebFountain [56]	√	√			√
OPIC [1]	√	√	√		

Summary

Batch Crawl Ordering

- A batch crawler traverses links outward from an initial seed set of URLs.
- The seed set may be selected algorithmically, or by hand, based on criteria such as importance, outdegree, or other structural features of the web graph [120].
- A common, simple approach is to use the root page of a web directory site such as OpenDirectory, which links to many important sites across a broad range of topics.
- After the seed set has been visited and links have been extracted from the seed set pages, the crawl ordering policy takes over.

Batch Crawl Ordering

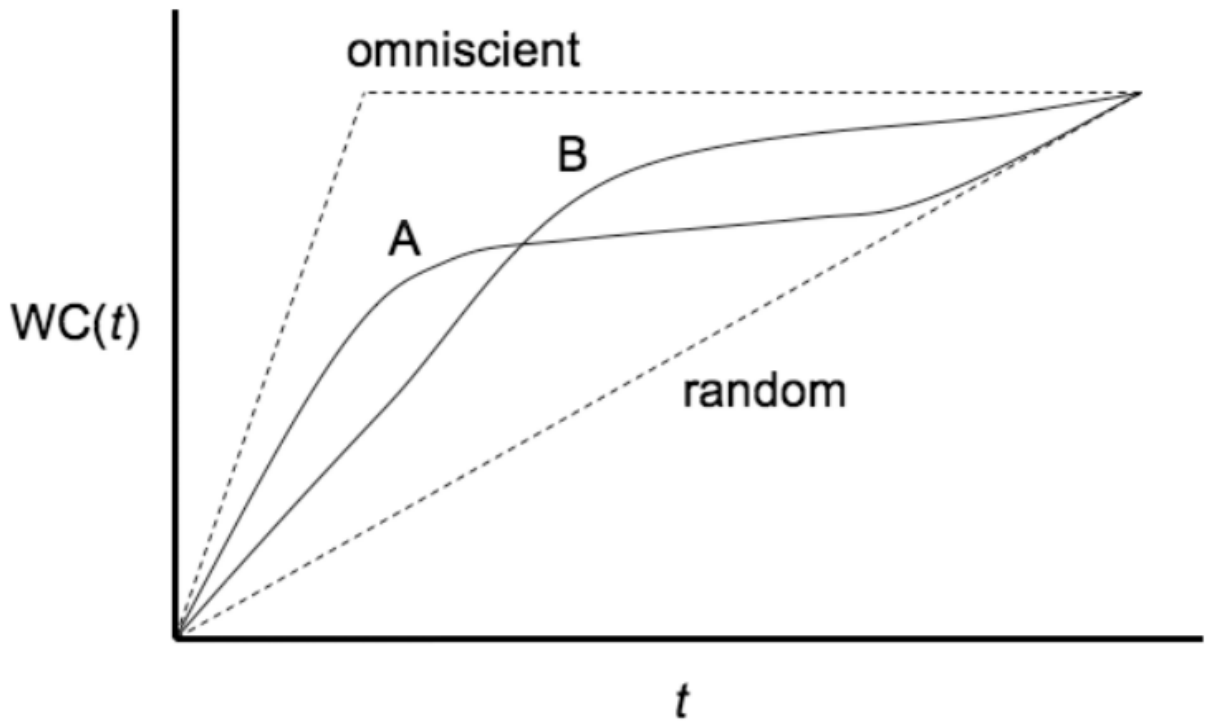
- The goal of the crawl ordering policy is to maximize the weighted coverage (WC) achieved over time:

$$WC(t) = \sum_{p \in C(t)} w(p)$$

  where,
  - t denotes the time elapsed since the crawl began.
  - $C(t)$ denotes the set of pages crawled up to time t (under the fixed crawl rate assumption, $|C(t)| \propto t$).
  - $w(p)$ denotes a numeric weight associated with page p.
- The weight function $w(p)$ is chosen to reflect the purpose of the crawl.

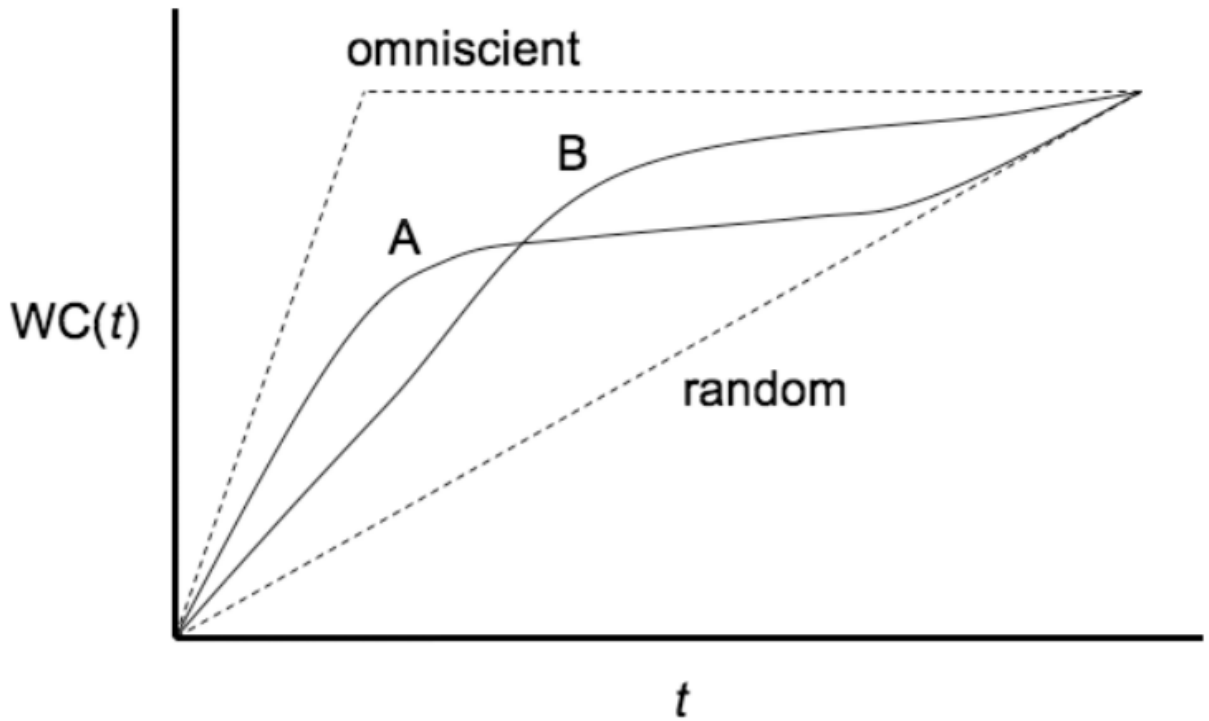
Batch Crawl Ordering

- Figure shows some hypothetical WC curves. Typically, $w(p) \geq 0$, and hence $WC(t)$ is monotonic in t.



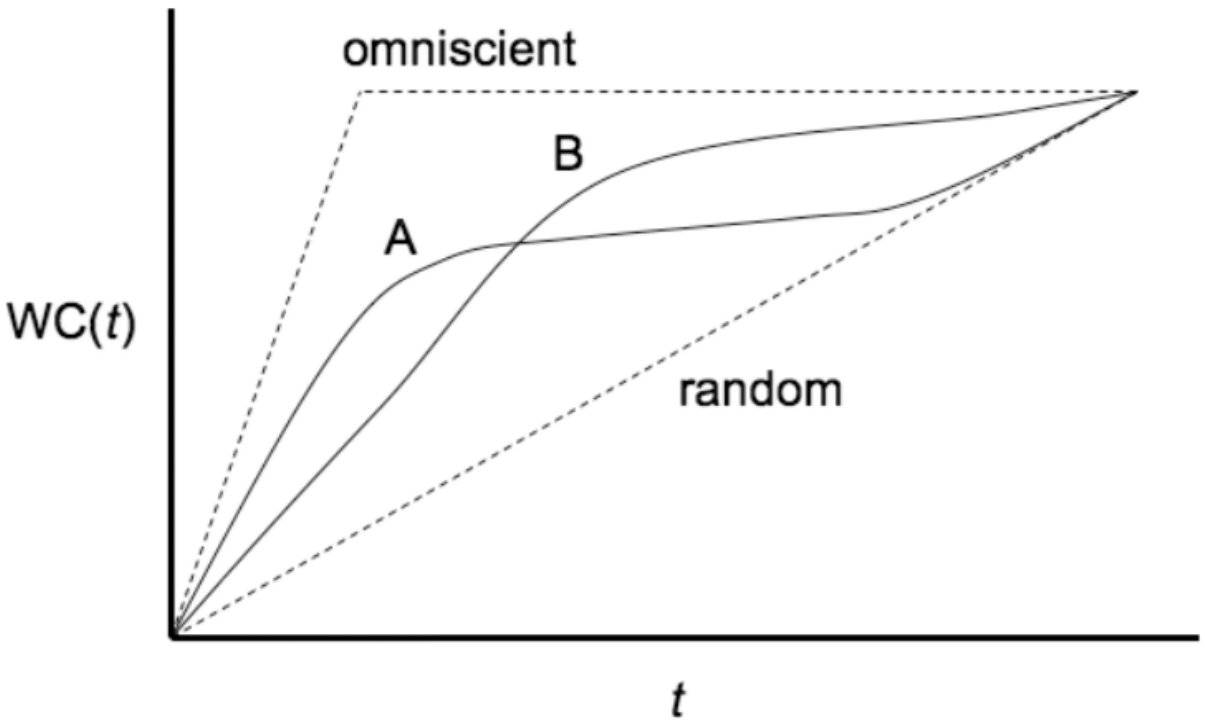
Batch Crawl Ordering

- Under a random crawl ordering policy, $WC(t)$ is roughly linear in t;
- This line serves as a baseline upon which other policies strive to improve.



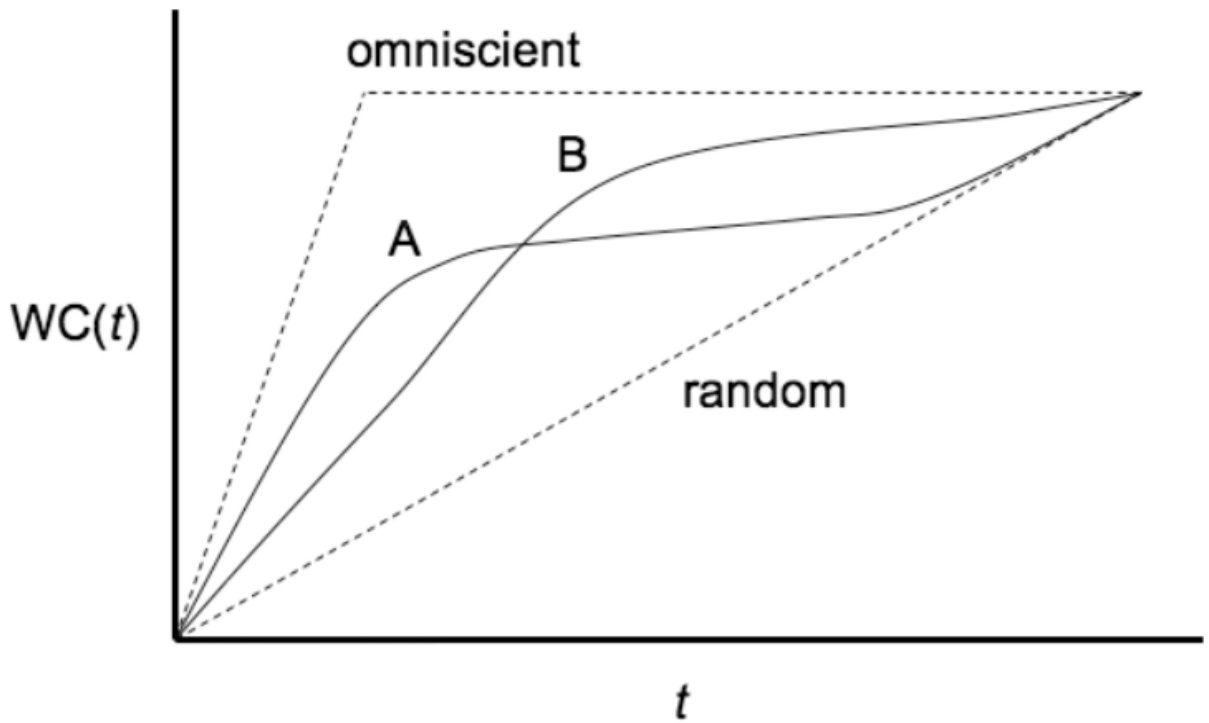
Batch Crawl Ordering

- An omniscient policy, which downloads pages in descending order of $w(p)$ yields a theoretical upper-bound curve.



Batch Crawl Ordering

- Policies A and B fall in-between the random and omniscient cases, with A performing better in the early stages of the crawl, but B performing better toward the end.
- The choice between A and B depends on how long the crawl is allowed to run before being stopped.



Batch Crawl Ordering

- The above framework can be applied to:
  - **comprehensive** batch crawling, in which the goal is to achieve broad coverage of all types of content.
  - **scoped** batch crawling, where the crawler restricts its attention to a relatively narrow slice of the webscoped.

Comprehensive Crawling

- When the goal is to cover high-quality content of all varieties, a popular choice of weight function is

$$w(p) = PR(p)$$

  where $PR(p)$ is p's importance score as measured by PageRank [101].
- In view of maximizing coverage weighted by PageRank or some variant, three main types of crawl ordering policies have been examined in the literature.

Comprehensive Crawling

- **Breadth-first search**
  - Pages are downloaded in the order in which they are first discovered.
- **Prioritize by indegree**
  - The page with the highest number of incoming hyperlinks from previously downloaded pages, is downloaded next.
- **Prioritize by PageRank (variant/estimate)**
  - Pages are downloaded in descending order of PageRank (or some variant), as estimated based on the pages and links acquired so far by the crawler.

Comprehensive Crawling

- Straightforward application of Prioritize by PageRank involves:
  - Recomputing PageRank scores after each download, or updating the PageRank scores incrementally [38].
  - To recompute PageRank scores only periodically, and rely on an approximation scheme between recomputations.
  - Lastly, Abiteboul et al. [1] gave an efficient online method of estimating a variant of PageRank that does not include random jumps, designed for use in conjunction with a crawler.

Comprehensive Crawling

- Three published empirical studies evaluated the above policies over real web data.
- Starting from high-PageRank seeds, breadth-first crawling performs well early in the crawl (low t), but not as well as the other policies later in the crawl (medium to high t).
- The shortcut of only recomputing PageRank periodically leads to poor performance, but the online approximation scheme by Abiteboul et al. [1] performs well.
- Furthermore, in the context of repeated batch crawls, it is beneficial to use PageRank values from previous iterations to drive the current iteration.

Comprehensive Crawling

- There is no consensus on prioritization by indegree
  - One study (Cho et al. [43]) found that it worked fairly well (almost as well as prioritization by PageRank).
  - Another study (Baeza-Yates et al. [9]) found that it performed very poorly.
- In addition, Baeza-Yates et al. [9] proposed a crawl policy that gives priority to sites containing a large number of discovered but uncrawled URLs.
  - Their empirical study imposed per-site politeness constraints
  - Toward the end of the crawl (high t) the proposed policy outperforms policies based on breadth-first search, indegree, and PageRank.
  - The reason is that it avoids the problem of being left with a few very large sites at the end, which can cause a politeness bottleneck.

Search Relevance as the Crawling Objective

- Fetterly et al. [58] and Pandey and Olston [104] argued that when the purpose of crawling is to supply content to a search engine, PageRank weighted coverage may not be the most appropriate objective.
- It instead makes sense to crawl pages that would be viewed or clicked by search engine users, if present in the search index.
- Fetterly et al. [58] evaluated four crawl ordering policies under two relevance metrics:
  - **MaxNDCG**: The total Normalized Distributed Cumulative Gain (NDCG) [79] score of a set of queries evaluated over the crawled pages, assuming optimal ranking.
  - **Click count**: The total number of clicks the crawled pages attracted via a commercial search engine in some time period.

Search Relevance as the Crawling Objective

- The main findings were:
  - prioritization by PageRank is the most reliable and effective method on these metrics;
  - Imposing per-domain page limits boosts effectiveness.
- Pandey and Olston [104] proposed a technique for explicitly ordering pages by expected relevance impact, under the objective of maximizing coverage weighted by the number of times a page appears among the top N results of a user query.
- The relatively high computational overhead of the technique is mitigated by concentrating on queries whose results are likely to be improved by crawling additional pages (*deemed needy queries*).
- Relevance of frontier pages to needy queries is estimated from cues found in URLs and referring *anchortext*.

Scoped Crawling

- A scoped crawler strives to limit crawling activities to pages that fall within a particular category or scope,
- It acquires in-scope content much faster and more cheaply than via a comprehensive crawl.
- Scope may be defined according to:
  - topic (e.g., pages about aviation)
  - geography (e.g., pages about locations in and around Oldenburg, Germany [6])
  - format (e.g., images and multimedia)
  - genre (e.g., course syllabi [51])
  - language (e.g., pages in Portuguese [65])
  - other aspects.

Scoped Crawling

- The mathematical objective typically associated with scoped crawling is maximization of weighted coverage

$$WC(t) = \sum_{p \in C(t)} w(p)$$

- In scoped crawling, the weight function $w(p)$ reflects the degree to which page p falls within the intended scope.
- In the simplest case, $w(p) \in \{0, 1\}$, where
  - 0 denotes that p is outside the scope.
  - 1 denotes that p is in-scope.
- Measures the fraction of crawled pages that are in-scope.
- Analogous to the *precision* metric used in IR.

Scoped Crawling

- Typically the in-scope pages form a finite set, hence it makes sense to measure recall in addition to precision.
- Two recall-oriented evaluation techniques have been proposed:
  1. designate a few representative in-scope pages by hand, and measure what fraction of them are discovered by the crawler [92].
  2. measure the overlap among independent crawls initiated from different seeds, to see whether they converge on the same set of pages [34].
- *Topical crawling* (also known as "focused crawling"), in which in-scope pages are ones that are relevant to a particular topic or set of topics, is by far the most extensively studied form of scoped crawling.

Topical Crawling

- The basic observation exploited by topical crawlers is that relevant pages tend to link to other relevant pages, either directly or via short chains of links.
- The first crawl ordering technique to exploit this observation was *fish search* [53]:
  - It categorized each crawled page p as either relevant or irrelevant.
  - Explored the neighborhood of each relevant page up to depth d looking for additional relevant pages.

Topical Crawling

- A second generation of topical crawlers [43, 74, 108] explored the neighborhoods of relevant pages in a non-uniform fashion.
- The most promising links are traversed first.
- The link traversal order was governed by individual relevance estimates assigned to each linked-to page.
- If a crawled page p links to an uncrawled page q, the relevance estimate for q is computed via analysis of
  - The text surrounding p's link to q
    - i.e., the anchortext and text near the anchortext
  - The URL of page q

Topical Crawling

- A third-generation approach based on machine learning and link structure analysis was introduced by Chakrabarti et al. [33, 34].
- The approach leverages pre-existing topic taxonomies such as the Open Directory and Yahoo!'s web directory, which supply examples of web pages matching each topic.
- These example pages are used to train a classifier to map newly encountered pages into the topic taxonomy.
- The user selects a subset of taxonomy nodes (topics) of interest to crawl.
- The crawler preferentially follows links from pages that the classifier deems most relevant to the topics of interest.

Topical Crawling

- In addition, an attempt is made to identify pages with a collection of links to topical pages (*hub pages*) using the HITS link analysis algorithm [82].
- Links from hub pages are followed with higher priority than other links.
- The empirical findings of Chakrabarti et al. [34] established topical crawling as a viable and effective paradigm:
  - A general web crawler seeded with topical pages quickly becomes mired in irrelevant regions of the web, yielding very poor weighted coverage. In contrast, a topical crawler successfully stays within scope, and explores a steadily growing population of topical pages over time.
  - Two topical crawler instances, started from disparate seeds, converge on substantially overlapping sets of pages.

Greediness

- Paths between pairs of relevant pages sometimes pass through one or more irrelevant pages.
- A topical crawler that is too greedy will stop when it reaches an irrelevant page, and never discovers subsequent relevant page(s).
- The question of how greedily to crawl is an instance of the *explore versus exploit* tradeoff observed in many contexts.
- In this context, the question is:
  - How should the crawler balance exploitation of direct links to (apparently) relevant pages, with exploration of other links that may, eventually, lead to relevant pages?

Adaptivity

- In most topical crawling approaches the page ordering strategy is fixed for the duration of the crawl.
- Some have studied ways for a crawler to adapt its strategy over time, in response to observations made while the crawl is in flight.
- Agarwal et al. [5] proposed a method to learn on the fly how best to combine relevance signals found into a single relevance estimate on which to base the crawl order.
- Evolutionary algorithms (e.g., genetic algorithms) have been explored as a means to adapt crawl behavior over time [37, 80, 91].

Summary

Incremental Crawl Ordering

- In contrast to a batch crawler, a *continuous* or *incremental crawler* never "starts over".
- An incremental crawler interleaves revisitation of previously crawled pages with first-time visitation of new pages.
- The aim is to achieve good freshness and coverage simultaneously.
- Coverage is measured according to the same *weighted coverage*.
- An analogous *weighted freshness* metric is as follows:

$$WF(t) = \sum_{p \in C(t)} w(p) \cdot f(p,t)$$

where $f(p,t)$ is page p's freshness level at time t.

Incremental Crawl Ordering

- One is typically interested in the steady-state average of WT:

$$\overline{WT} = \lim_{t \to \infty} \frac{1}{t} \int_0^1 WF(t)dt$$

- At each step, an incremental crawler faces a choice between two basic actions:
  1. **Download a new page**.
     - May improve coverage.
     - May supply new links, which can lead to discovery of new pages.
  2. **Re-download an old page**.
     - May improve freshness.
     - May supply new links or reveal the removal of links.

Incremental Crawl Ordering

- In the presence of dynamic pages and finite crawling resources, there is a tradeoff between coverage and freshness.
- There is no consensus about the best way to balance the two.
- Some contend that balancing coverage and freshness should be left as a business decision
  - Do we prefer broad coverage of content that may be somewhat out-of-date?
  - Or do we prefer narrower coverage with fresher content?
- Others have proposed specific schemes for combining the two objectives into a single framework.
- Most published work on crawling focuses either uniquely on coverage or uniquely on freshness.

Maximizing Freshness

- To simplify the study of this problem, it is standard practice to assume that:
  - The set of crawled pages is fixed (i.e., $C(t)$ is static, so we drop the dependence on t)
  - Each page $p \in C$ exhibits a stationary stochastic pattern of content changes over time.
- Freshness maximization divides into three relatively distinct sub-problems:
  - **Model estimation**. Construct a model for the temporal behavior of each page $p \in C$.
  - **Resource allocation**. Given a maximum crawl rate r, assign to each page $p \in C$ a revisitation frequency $r(p)$ such that $\sum_{p \in C} r(p) = r$.
  - **Scheduling**. Produce a crawl order that adheres to the target revisitation frequencies as closely as possible.

Maximizing Freshness

- Resource allocation is generally viewed as the central aspect of freshness maximization.
- We divide work on resource allocation into two categories, according to the freshness model adopted:
  - Binary Freshness Model
  - Continuous Freshness Models

Binary Freshness Model

- Binary freshness model is also known as *obsolescence*.
- In the binary freshness model, $f(p,t) \in \{0,1\}$.
- If the cached copy of p is identical to the live copy then:
  - $f(p,t) = 1$
  - p is said to be “fresh”.
- Otherwise,
  - $f(p,t) = 0$
  - p is termed “stale”.

Binary Freshness Model

- The first to study the freshness maximization problem were Coffman et al. [46] who postulated a Poisson model of web page change.
- A page p undergoes discrete change events, which cause the copy cached by the crawler to become stale.
- The occurrence of change events is governed by a Poisson process with rate parameter $\lambda(p)$
- This means that changes occur randomly and independently, with an average rate of $\lambda(p)$ changes per time unit.
- In the case of uniform page weights (i.e., all $w(p)$ values are equal), revisitation frequencies in proportion to page change rates, i.e., $r(p) \propto \lambda(p)$ (called proportional resource allocation), can be suboptimal.

Binary Freshness Model

- Cho and García-Molina [41] continued the work of Coffman et al. [46], and derived a famously counterintuitive result:
  - In the uniform weights case, a uniform resource allocation policy, in which $r(p) = r/|C|$ for all p, achieves higher average binary freshness than proportional allocation.
  - The superiority of the uniform policy to the proportional one holds under any distribution of change rates ($\lambda(p)$ values).
- The optimal resource allocation policy for binary freshness, also given by Cho and García-Molina [41], exhibits the following intriguing property:
  - Pages with a very fast rate of change (i.e., $\lambda(p)$ very high relative to $r/|C|$) ought never to be revised by the crawler, i.e., $r(p) = 0$.

Binary Freshness Model

- A page p_1 that changes once per second, and is revisited once per second by the crawler, is on average only half synchronized ($f(p_1) = 0.5$).
- A page p_2 that changes once per day, and is revisited once per hour by the crawler, has much better average freshness ($f(p_2) = 24/25$ under randomized scheduling, according to the formula given by Cho and García-Molina [41]).
- The crawling resources required to keep p_1 weakly synchronized can be put to better use keeping several slow-changing pages like p_2 tightly synchronized, assuming equal page weights.
- In terms of average binary freshness, it is best for the crawler to "give up on" fast-changing pages, and put its energy into synchronizing moderate and slow-changing ones.

Binary Freshness Model

- Under a Poison model, the crawler cannot time its visits to coincide with page change events. The following approaches relax the independence assumption.
- Wolf et al. [115] studied incremental crawling under a quasi-deterministic page change model in which:
  - page change events are non-uniform in time;
  - the distribution of likely change times is known a priori.
- This work also introduced a search-centric page weighting scheme (embarrassment level) in which $w(p) \propto c(p)$
  - $c(p)$ denotes the probability that a user will click on p after issuing a search query, as estimated from historical search engine usage logs.
- The aim is to revisit frequently clicked pages preferentially, thereby minimizing "embarrassing" incidents in which a search result contains a stale page.

Continuous Freshness Models

- In a real crawling scenario, some pages may be “fresher” than others.
- There is no consensus about the best way to measure freshness.
- Cho and García-Molina [41] introduced a temporal freshness metric, in which $f(p,t) \propto -age(p,t)$:

$$age(p,t) = \begin{cases} 0, & \text{if the cached copy of } p \text{ is identical to the live copy} \\ a, & \text{otherwise} \end{cases}$$

  where a denotes the amount of time the copies have differed.
- The longer a cached page remains unsynchronized, the more their content tends to drift apart.

Continuous Freshness Models

- The optimal resource allocation policy under this age-based freshness metric, assuming a Poisson model of page change, is given by Cho and García-Molina [41].
- Unlike in the binary freshness case, the revisitation frequency $r(p)$ increases monotonically with the page change rate $\lambda(p)$.
- Since age increases without bound, the crawler cannot afford to "give up on" any page.

Continuous Freshness Models

- Olston and Pandey [99] introduced an approach in which, the idea is to measure changes in page content directly.
- A page is divided into a set of content fragments $f_1, f_2, ..., f_n$.
- A corresponding weight $w(f_i)$ captures the fragment's importance and/or relevance.
- Freshness is measured as the (weighted) fraction of fragments in common between the cached and live page snapshots, using the Jaccard set similarity measure.

Continuous Freshness Models

- They also characterize the *longevity* of newly updated content.
- Long-lived content (e.g., today's blog entry, which will remain in the blog indefinitely) is more valuable to crawl than ephemeral content (e.g., today's "quote of the day," which will be overwritten tomorrow).
- In separate work, Pandey and Olston [103] proposed a search-centric method of assigning weights to changes.
- Weights are assigned based on the degree to which a change is expected to impact search ranking.
- Even if a page undergoes periodic changes, if the search engine's treatment of the page is unaffected by these changes, there is no need for the crawler to revisit it.

Summary

Redundant Content

- There is a prevalence of **duplicate** and **near-duplicate** content on the web.
- Some duplication stems from the fact that:
  - many web sites allow multiple URLs to refer to the same content;
  - content that is identical modulo ever-changing elements such as rotating banner ads, evolving comments by readers, and timestamps.

Redundant Content

- Schonfeld et al. proposed the “duplicate URL with similar text” (DUST) algorithm [12] to detect this form of aliasing, and to infer rules for normalizing URLs into a canonical form.
- Rules inferred using these algorithms can be used by a web crawler to normalize URLs after extracting them from downloaded pages and before passing them through the duplicate URL eliminator and into the frontier.

Redundant Content

- Another source of duplication is **mirroring** [18, 19, 52]:
  - Providing all or parts of the same web site on different hosts.
- Mirrored web sites can be divided into two groups:
  - Sites that are mirrored by the same organization
    - having one web server serving multiple domains with the same content;
    - having multiple web servers provide synchronized content.
  - Content that is mirrored by multiple organizations
    - schools providing Unix man pages on the web;
    - web sites republishing Wikipedia content, often somewhat reformatted.

Redundant Content

- Detecting mirrored content differs from detecting DUST in two ways:
  - The duplication occurs across multiple sites, so mirror detection algorithms have to consider the entire corpus.
  - Entire trees of URLs are mirrored, so detection algorithms can use URL trees as a feature to detect mirror candidates, and then compare the content of candidate subtrees.

Crawler Traps

- Another phenomenon that inflates the corpus without adding utility is **crawler traps**:
  - Web sites that populate a large, possibly infinite URL space on that site with mechanically generated content.
- Some crawler traps are nom-malicious.
  - e.g., calendaring tools with a hyperlink from each month to the next (and previous) month, thereby forming an unbounded chain of dynamically generated pages.
- Other crawler traps are malicious.
- Often set up by “spammers” to inject large amounts of their content into a search engine, in the hope of:
  - having their content show up high in search result pages;
  - providing many hyperlinks to their “landing page,” thus biasing link-based ranking algorithms such as PageRank.

Web Spam

- Web spam may be defined as: “web pages that are crafted for the sole purpose of increasing the ranking of these or some affiliated pages, without improving the utility to the viewer” [97].
- Web spam is motivated by the monetary value of achieving a prominent position in search-engine result pages.
- There is a multi-billion dollar industry devoted to *search engine optimization* (SEO), most of it being legitimate but some of it misleading.

Web Spam

- Web spam can be broadly classified into three categories [69]:
  - **Keyword stuffing**, populating pages with highly searched or highly monetizable terms;
  - **Link spam**, creating cliques of tightly inter-linked web pages with the goal of biasing link-based ranking algorithms such as PageRank [101];
  - **Cloaking**, serving substantially different content to web crawlers than to human visitors (to get search referrals for queries on a topic not covered by the page).

Web Spam

- The problem of identifying web spam can be framed as a classification problem.
- The main challenge is to identify features that are predictive of web spam and can thus be used as inputs to the classifier.
- Many such features have been proposed, including:
  - hyperlink features [16, 17, 50, 116];
  - term and phrase frequency [97];
  - DNS lookup statistics [59];
  - HTML markup structure [114].
- Web spam detection is a constant arms race, with both spammers and search engines evolving their techniques in response to each other's actions.

Cloaked Content

- *Cloaking* refers to the practice of serving different content to web crawlers than to human viewers of a site [73].
- Not all cloaking is malicious:
  - Many web sites with interactive content rely heavily on JavaScript;
  - But most web crawlers do not execute JavaScript;
  - It is reasonable for such a site to deliver alternative, script-free versions of its pages to a search engine's crawler to enable the engine to index and expose the content.
- Web sites distinguish mechanical crawlers from human visitors either based on:
  - `User-Agent` field (an HTTP header that is used to distinguish different web browsers, and by convention is used by crawlers to identify themselves);
  - Crawler's IP address.

Cloaked Content

- A variant of cloaking is called *redirection spam*.
- A web server utilizing redirection spam serves the same content both to crawlers and to human-facing browser software.
- However, the content will cause a browser to immediately load a new page presenting different content.
- Redirection spam is facilitated either through:
  - HTML `META REFRESH` tag (whose presence is easy to detect)
  - JavaScript, which most browsers execute but most crawlers do not.
- Chellapilla and Maykov argued for the use of lightweight JavaScript parsers and execution engines in the crawling/indexing pipeline to evaluate scripts to determine whether redirection occurs.

Summary

Deep Web Crawling

- Some content is accessible only by filling in HTML forms.
- This kind of content cannot be reached via conventional crawlers that just follow hyperlinks.
- Crawlers that automatically fill in forms to reach the content behind them are called *hidden web* or *deep web* crawlers.
- Types of Deep Web Sites:
  - Content is either:
    - unstructured (e.g., free-form text);
    - or structured (e.g., data records with typed fields).
  - The form interface used to query the content is either:
    - unstructured (i.e., a single query box that accepts a free-form query string)
    - or structured (i.e., multiple query boxes that pertain to different aspects of the content).

Types of Deep Web Sites

	Unstructured Content	Structured Content
Unstructured query interface	News archive (simple search)	Product review site
Structured query interface	News archive (advanced search)	Online bookstore

- For simplicity most work focuses on either the upper-left quadrant (which we henceforth call the unstructured case), or the lower-right quadrant (structured case).

Deep Web Crawling

- Deep web crawling has three steps:
  1. **Locate deep web content sources.** A human or crawler must identify web sites containing form interfaces that lead to deep web content.
  2. **Select relevant sources.** For a scoped deep web crawling task (e.g., crawling medical articles), one must select a relevant subset of the available content sources. In the unstructured case this problem is known as database or resource selection [32, 66].
  3. **Extract underlying content.** Finally, a crawler must extract the content lying behind the form interfaces of the selected content sources.
- Step 3 (content extraction) is the core problem in deep web crawling.

Content Extraction

- The main approach to extracting content from a deep web site proceeds in four steps (the first two steps apply only to the structured case):
  1. Select a subset of form elements to populate, or perhaps multiple such subsets.
  2. If possible, decipher the role of each of the targeted form elements (e.g., book author versus publication date), or at least understand their domains (proper nouns versus dates).
  3. Create an initial database of valid data values (e.g., “Ernest Hemingway” and 1940 in the structured case; English words in the unstructured case).
  4. Use the database to issue queries to the deep web site (e.g., publisher = “Scribner”), parse the result and extract new data values to insert into the database (e.g., author = “Ernest Hemingway”), and repeat.

References

[1] S. Abiteboul, M. Preda, and G. Cobena, “Adaptive on-line page importance computation,” in Proceedings of the 12th International World Wide Web Conference, 2003.

[2] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas, “The web changes everything: Understanding the dynamics of web content,” in Proceedings of the 2nd International Conference on Web Search and Data Mining, 2009.

[3] Advanced Triage (medical term), http://en.wikipedia.org/wiki/Triage# Advanced triage.

[4] A. Agarwal, H. S. Koppula, K. P. Leela, K. P. Chitrapura, S. Garg, P. K. GM, C. Haty, A. Roy, and A. Sasturkar, “URL normalization for de-duplication of web pages,” in Proceedings of the 18th Conference on Information and Knowledge Management, 2009.

[5] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu, “Intelligent crawling on the world wide web with arbitrary predicates,” in Proceedings of the 10th International World Wide Web Conference, 2001.

[6] D. Ahlers and S. Boll, “Adaptive geospatially focused crawling,” in Proceedings of the 18th Conference on Information and Knowledge Management, 2009.

[7] Attributor. http://www.attributor.com.

[8] R. Baeza-Yates and C. Castillo, “Crawling the infinite web,” Journal of Web Engineering, vol. 6, no. 1, pp. 49–72, 2007.

References

[9] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez, “Crawling a country: Better strategies than breadth-first for web page ordering,” in Proceedings of the 14th International World Wide Web Conference, 2005.

[10] B. Bamba, L. Liu, J. Caverlee, V. Padliya, M. Srivatsa, T. Bansal, M. Palekar, J. Patrao, S. Li, and A. Singh, “DSphere: A source-centric approach to crawling, indexing and searching the world wide web,” in Proceedings of the 23rd International Conference on Data Engineering, 2007.

[11] Z. Bar-Yossef and M. Gurevich, “Random sampling from a search engine’s index,” in Proceedings of the 15th International World Wide Web Conference, 2006.

[12] Z. Bar-Yossef, I. Keidar, and U. Schonfeld, “Do not crawl in the DUST: Different URLs with similar text,” in Proceedings of the 16th International World Wide Web Conference, 2007.

[13] L. Barbosa and J. Freire, “Siphoning hidden-web data through keyword-based interfaces,” in Proceedings of the 19th Brazilian Symposium on Databases SBBD, 2004.

[14] L. Barbosa and J. Freire, “An adaptive crawler for locating hidden-web entry points,” in Proceedings of the 16th International World Wide Web Conference, 2007.

[15] L. Barbosa, A. C. Salgado, F. de Carvalho, J. Robin, and J. Freire, “Looking at both the present and the past to efficiently update replicas of web content,” in Proceedings of the ACM International Workshop on Web Information and Data Management, 2005.

References

[16] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates, “Link- based characterization and detection of web spam,” in Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web, 2006.

[17] A. Benczúr, K. Csalogány, T. Sarlós, and M. Uher, “Spamrank — fully automatic link spam detection,” in Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web, 2005.

[18] K. Bharat and A. Broder, “Mirror, mirror on the web: A study of host pairs with replicated content,” in Proceedings of the 8th International World Wide Web Conference, 1999.

[19] K. Bharat, A. Broder, J. Dean, and M. Henzinger, “A comparison of techniques to find mirrored hosts on the WWW,” Journal of the American Society for Information Science, vol. 51, no. 12, pp. 1114–1122, 2000.

[20] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.

[21] P. Boldi, B. Codenotti, , M. Santini, and S. Vigna, “UbiCrawler: A scalable fully distributed web crawler,” Software — Practice & Experience, vol. 34, no. 8, pp. 711–726, 2004.

[22] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, “Structural properties of the African web,” in Poster Proceedings of the 11th International World Wide Web Conference, 2002.

[23] P. Boldi, M. Santini, and S. Vigna, “Paradoxical effects in pagerank incremental computations,” Internet Mathematics, vol. 2, no. 3, pp. 387–404, 2005.

References

[24] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, “The Harvest information discovery and access system,” in Proceedings of the 2nd International World Wide Web Conference, 1994.

[25] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” in Proceedings of the 7th International World Wide Web Conference, 1998.

[26] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, “Graph structure in the web,” in Proceedings of the 9th International World Wide Web Conference, 2000.

[27] A. Broder, M. Najork, and J. Wiener, “Efficient URL caching for World Wide Web crawling,” in Proceedings of the 12th International World Wide Web Conference, 2003.

[28] A. Z. Broder, S. C. Glassman, and M. S. Manasse, “Syntactic clustering of the web,” in Proceedings of the 6th International World Wide Web Conference, 1997.

[29] M. Burner, “Crawling towards eternity: Building an archive of the world wide web,” Web Techniques Magazine, vol. 2, no. 5, pp. 37–40, 1997.

[30] J. Callan, “Distributed information retrieval,” in Advances in Information Retrieval, (W. B. Croft, ed.), pp. 127–150, Kluwer Academic Publishers, 2000.

[31] J. Callan and M. Connell, “Query-based sampling of text databases,” ACM Transactions on Information Systems, vol. 19, no. 2, pp. 97–130, 2001.

References

[32] J. P. Callan, Z. Lu, and W. B. Croft, “Searching distributed collections with inference networks,” in Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1995.

[33] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg, “Automatic resource compilation by analyzing hyperlink structure and associated text,” in Proceedings of the 7th International World Wide Web Conference, 1998.

[34] S. Chakrabarti, M. van den Berg, and B. Dom, “Focused crawling: A new approach to topic-specific web resource discovery,” in Proceedings of the 8th International World Wide Web Conference, 1999.

[35] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, “Structured databases on the web: Observations and implications,” ACM SIGMOD Record, vol. 33, no. 3, pp. 61–70, 2004.

[36] K. Chellapilla and A. Maykov, “A taxonomy of JavaScript redirection spam,” in Proceedings of the 16th International World Wide Web Conference, 2007.

[37] H. Chen, M. Ramsey, and C. Yang, “A smart itsy bitsy spider for the web,” Journal of the American Society for Information Science, vol. 49, no. 7, pp. 604–618, 1998.

[38] S. Chien, C. Dwork, R. Kumar, D. R. Simon, and D. Sivakumar, “Link evolution: Analysis and algorithms,” Internet Mathematics, vol. 1, no. 3, pp. 277–304, 2003.

[39] J. Cho and H. García-Molina, “The evolution of the web and implications for an incremental crawler,” in Proceedings of the 26th International Conference on Very Large Data Bases, 2000.

References

[40] J. Cho and H. García-Molina, “Parallel crawlers,” in Proceedings of the 11th International World Wide Web Conference, 2002.

[41] J. Cho and H. García-Molina, “Effective page refresh policies for web crawlers,” ACM Transactions on Database Systems, vol. 28, no. 4, pp. 390–426, 2003.

[42] J. Cho and H. García-Molina, “Estimating frequency of change,” ACM Transactions on Internet Technology, vol. 3, no. 3, pp. 256–290, 2003.

[43] J. Cho, J. Garcíıa-Molina, and L. Page, “Efficient crawling through URL ordering,” in Proceedings of the 7th International World Wide Web Conference, 1998.

[44] J. Cho and A. Ntoulas, “Effective change detection using sampling,” in Proceedings of the 28th International Conference on Very Large Data Bases, 2002.

[45] J. Cho and U. Schonfeld, “RankMass crawler: A crawler with high personalized PageRank coverage guarantee,” in Proceedings of the 33rd International Conference on Very Large Data Bases, 2007.

[46] E. G. Coffman, Z. Liu, and R. R. Weber, “Optimal robot scheduling for web search engines,” Journal of Scheduling, vol. 1, no. 1, 1998.

[47] CrawlTrack, “List of spiders and crawlers,” http://www.crawltrack.net/ crawlerlist.php.

[48] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins, “The discoverability of the web,” in Proceedings of the 16th International World Wide Web Conference, 2007.

References

[49] A. Dasgupta, R. Kumar, and A. Sasturkar, “De-duping URLs via rewrite rules,” in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008.

[50] B. Davison, “Recognizing nepotistic links on the web,” in Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search, 2000.

[51] G. T. de Assis, A. H. F. Laender, M. A. Gonçalves, and A. S. da Silva, “A genre-aware approach to focused crawling,” World Wide Web, vol. 12, no. 3, pp. 285–319, 2009.

[52] J. Dean and M. Henzinger, “Finding related pages in the world wide web,” in Proceedings of the 8th International World Wide Web Conference, 1999.

[53] P. DeBra and R. Post, “Information retrieval in the world wide web: Making client-based searching feasible,” in Proceedings of the 1st International World Wide Web Conference, 1994.

[54] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, “Focused crawling using context graphs,” in Proceedings of the 26th International Conference on Very Large Data Bases, 2000.

[55] C. Duda, G. Frey, D. Kossmann, and C. Zhou, “AJAXSearch: Crawling, indexing and searching web 2.0 applications,” in Proceedings of the 34th International Conference on Very Large Data Bases, 2008.

[56] J. Edwards, K. S. McCurley, and J. A. Tomlin, “An adaptive model for optimizing performance of an incremental web crawler,” in Proceedings of the 10th International World Wide Web Conference, 2001.

References

[57] D. Eichmann, “The RBSE spider — Balancing effective search against web load,” in Proceedings of the 1st International World Wide Web Conference, 1994.

[58] D. Fetterly, N. Craswell, and V. Vinay, “The impact of crawl policy on web search effectiveness,” in Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2009.

[59] D. Fetterly, M. Manasse, and M. Najork, “Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages,” in Proceedings of the 7th International Workshop on the Web and Databases, 2004.

[60] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener, “A large-scale study of the evolution of web pages,” in Proceedings of the 12th International World Wide Web Conference, 2003.

[61] R. Fielding, “Maintaining distributed hypertext infostructures: Welcome to MOMspider’s web,” in Proceedings of the 1st International World Wide Web Conference, 1994.

[62] A. S. Foundation, “Welcome to Nutch!,” http://lucene.apache.org/nutch/.

[63] W. Gao, H. C. Lee, and Y. Miao, “Geographically focused collaborative crawling,” in Proceedings of the 15th International World Wide Web Conference, 2006.

[64] GigaAlert, http://www.gigaalert.com.

[65] D. Gomes and M. J. Silva, “Characterizing a national community web,” ACM Transactions on Internet Technology, vol. 5, no. 3, pp. 508–531, 2005.

References

[66] L. Gravano, H. García-Molina, and A. Tomasic, “The effectiveness of GlOSS for the text database discovery problem,” in Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, 1994.

[67] M. Gray, “Internet growth and statistics: Credits and background,” http://www.mit.edu/people/mkgray/net/background.html.

[68] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien, “How to build a WebFountain: An architecture for very large-scale text analytics,” IBM Systems Journal, vol. 43, no. 1, pp. 64–77, 2004.

[69] Z. Gyöngyi and H. García-Molina, “Web Spam Taxonomy,” in Proceedings of the 1st International Workshop on Adversarial Information Retrieval, 2005.

[70] Y. Hafri and C. Djeraba, “High performance crawling system,” in Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval, 2004.

[71] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork, “Measuring index quality using random walks on the web,” in Proceedings of the 8th International World Wide Web Conference, 1999.

[72] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork, “On near- uniform URL sampling,” in Proceedings of the 9th International World Wide Web Conference, 2000.

[73] M. R. Henzinger, R. Motwani, and C. Silverstein, “Challenges in web search engines,” SIGIR Forum, vol. 36, no. 2, pp. 11–22, 2002.

References

[74] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur, “The shark-search algorithm — An application: Tailored web site mapping,” in Proceedings of the 7th International World Wide Web Conference, 1998.

[75] A. Heydon and M. Najork, “Mercator: A scalable, extensible web crawler,” World Wide Web, vol. 2, no. 4, pp. 219–229, 1999.

[76] International Workshop Series on Adversarial Information Retrieval on the Web, 2005.

[77] Internet Archive, http://archive.org/.

[78] Internet Archive, “Heritrix home page,” http://crawler.archive.org/.

[79] K. Jarvelin and J. Kekalainen, “Cumulated gain-based evaluation of IR techniques,” ACM Transactions on Information Systems, vol. 20, no. 4, pp. 422–446, 2002.

[80] J. Johnson, K. Tsioutsiouliklis, and C. L. Giles, “Evolving strategies for focused web crawling,” in Proceedings of the 20th International Conference on Machine Learning, 2003.

[81] R. Khare, D. Cutting, K. Sitakar, and A. Rifkin, “Nutch: A flexible and scalable open-source web search engine,” Technical Report, CommerceNet Labs, 2004.

[82] J. Kleinberg, “Authoritative sources in a hyperlinked environment,” Journal of the ACM, vol. 46, no. 5, pp. 604–632, 1999.

[83] M. Koster, “A standard for robot exclusion,” http://www.robotstxt.org/ orig.html, 1994.

References

[84] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov, “IRLbot: Scaling to 6 billion pages and beyond,” in Proceedings of the 17th International World Wide Web Conference, 2008.

[85] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. C. Agarwal, “Characterizing web document change,” in Proceedings of the International Conference on Advances in Web-Age Information Management, 2001.

[86] L. Liu, C. Pu, W. Tang, and W. Han, “CONQUER: A continual query system for update monitoring in the WWW,” International Journal of Computer Systems, Science and Engineering, vol. 14, no. 2, 1999.

[87] B. T. Loo, O. Cooper, and S. Krishnamurthy, “Distributed web crawling over DHTs,” UC Berkeley Technical Report CSD-04-1305, 2004.

[88] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy, “Google’s deep-web crawl,” in Proceedings of the 34th International Conference on Very Large Data Bases, 2008.

[89] M. Mauldin, “Lycos: Design choices in an internet search service,” IEEE Expert, vol. 12, no. 1, pp. 8–11, 1997.

[90] O. A. McBryan, “GENVL and WWWW: Tools for taming the web,” in Proceedings of the 1st International World Wide Web Conference, 1994.

[91] F. Menczer and R. K. Belew, “Adaptive retrieval agents: Internalizing local context and scaling up to the web,” Machine Learning, vol. 39, pp. 203–242, 2000.

References

[92] F. Menczer, G. Pant, and P. Srinivasan, “Topical web crawlers: Evaluating adaptive algorithms,” ACM Transactions on Internet Technology, vol. 4, no. 4, pp. 378–419, 2004.

[93] G. Mohr, M. Stack, I. Ranitovic, D. Avery, and M. Kimpton, “An introduction to Heritrix, an open source archival quality web crawler,” in Proceedings of the 4th International Web Archiving Workshop, 2004.

[94] M. Najork and A. Heydon, “High-performance web crawling,” Technical report, Compaq SRC Research Report 173, 2001.

[95] M. Najork and J. L. Wiener, “Breadth-first search crawling yields high-quality pages,” in Proceedings of the 10th International World Wide Web Conference, 2001.

[96] A. Ntoulas, J. Cho, and C. Olston, “What’s new on the web? The evolution of the web from a search engine perspective,” in Proceedings of the 13th International World Wide Web Conference, 2004.

[97] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly, “Detecting spam web pages through content analysis,” in Proceedings of the 15th International World Wide Web Conference, 2006.

[98] A. Ntoulas, P. Zerfos, and J. Cho, “Downloading textual hidden web content through keyword queries,” in Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, 2005.

[99] C. Olston and S. Pandey, “Recrawl scheduling based on information longevity,” in Proceedings of the 17th International World Wide Web Conference, 2008.

References

[100] V. J. Padliya and L. Liu, “Peercrawl: A decentralized peer-to-peer architecture for crawling the world wide web,” Georgia Institute of Technology Technical Report, 2006.

[101] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Technical Report, Stanford University, 1998.

[102] S. Pandey, K. Dhamdhere, and C. Olston, “WIC: A general-purpose algorithm for monitoring web information sources,” in Proceedings of the 30th International Conference on Very Large Data Bases, 2004.

[103] S. Pandey and C. Olston, “User-centric web crawling,” in Proceedings of the 14th International World Wide Web Conference, 2005.

[104] S. Pandey and C. Olston, “Crawl ordering by search impact,” in Proceedings of the 1st International Conference on Web Search and Data Mining, 2008.

[105] S. Pandey, K. Ramamritham, and S. Chakrabarti, “Monitoring the dynamic web to respond to continuous queries,” in Proceedings of the 12th International World Wide Web Conference, 2003.

[106] G. Pant and P. Srinivasan, “Learning to crawl: Comparing classification schemes,” ACM Transactions on Information Systems, vol. 23, no. 4, pp. 430–462, 2005.

[107] G. Pant and P. Srinivasan, “Link contexts in classifier-guided topical crawlers,” IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 1, pp. 107–122, 2006.

[108] B. Pinkerton, “Finding what people want: Experiences with the WebCrawler,” in Proceedings of the 2nd International World Wide Web Conference, 1994.

References

[109] S. Raghavan and H. García-Molina, “Crawling the hidden web,” in Proceedings of the 27th International Conference on Very Large Data Bases, 2001.

[110] U. Schonfeld and N. Shivakumar, “Sitemaps: Above and beyond the crawl of duty,” in Proceedings of the 18th International World Wide Web Conference, 2009.

[111] V. Shkapenyuk and T. Suel, “Design and implementation of a high- performance distributed web crawler,” in Proceedings of the 18th International Conference on Data Engineering, 2002.

[112] A. Singh, M. Srivatsa, L. Liu, and T. Miller, “Apoidea: A decentralized peer- to-peer architecture for crawling the world wide web,” in SIGIR Workshop on Distributed Information Retrieval, 2003.

[113] Q. Tan, Z. Zhuang, P. Mitra, and C. L. Giles, “A clustering-based sampling approach for refreshing search engine’s database,” in Proceedings of the 10th International Workshop on the Web and Databases, 2007.

[114] T. Urvoy, T. Lavergne, and P. Filoche, “Tracking web spam with hidden style similarity,” in Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web, 2006.

[115] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen, “Optimal crawling strategies for web search engines,” in Proceedings of the 11th International World Wide Web Conference, 2002.

[116] B. Wu and B. Davison, “Identifying link farm spam pages,” in Proceedings of the 14th International World Wide Web Conference, 2005.

References

[117] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma, “Query selection techniques for efficient crawling of structured web sources,” in Proceedings of the 22nd International Conference on Data Engineering, 2006.

[118] Yahoo! Research Barcelona, “Datasets for web spam detection,” http://www.yr-bcn.es/webspam/datasets.

[119] J.-M. Yang, R. Cai, C. Wang, H. Huang, L. Zhang, and W.-Y. Ma, “Incorporating site-level knowledge for incremental crawling of web forums: A list-wise strategy,” in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009.

[120] S. Zheng, P. Dmitriev, and C. L. Giles, “Graph-based seed selection for web- scale crawlers,” in Proceedings of the 18th Conference on Information and Knowledge Management, 2009.

[121] K. Zhu, Z. Xu, X. Wang, and Y. Zhao, “A full distributed web crawler based on structured network,” in Asia Information Retrieval Symposium, 2008.