Effective Page Refresh Policies For Web Crawlers

JUNGHOO CHO
University of California, Los Angeles
and
HECTOR GARCIA-MOLINA
Stanford University

In this paper we study how we can maintain local copies of remote data sources "fresh," when the source data is updated autonomously and independently. In particular, we study the problem of *Web crawlers* that maintain local copies of remote Web pages for Web search engines. In this context, remote data sources (Web sites) do not notify the copies (Web crawlers) of new changes, so we need to periodically *poll* the sources to maintain the copies up-to-date. Since polling the sources takes significant time and resources, it is very difficult to keep the copies completely up-to-date.

This paper proposes various refresh policies and studies their effectiveness. We first formalize the notion of "freshness" of copied data by defining two freshness metrics, and we propose a Poisson process as the change model of data sources. Based on this framework, we examine the effectiveness of the proposed refresh policies analytically and experimentally. We show that a Poisson process is a good model to describe the changes of Web pages and we also show that our proposed refresh policies improve the "freshness" of data very significantly. In certain cases, we got orders of magnitude improvement from existing policies.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Distributed databases*; H.3.3 [**Information Search and Retrieval**]: Search process; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Web crawlers, World-Wide Web, Web search engines, Page refresh

1. INTRODUCTION

Frequently, one needs to maintain local copies of remote data sources for better performance or availability. For example, a Web search engine copies a significant subset of the Web and maintain copies and/or indexes of the pages to help users

Authors' email addresses: Junghoo Cho (cho@cs.ucla.edu), Hector Garcia-Molina (hector@cs.stanford.edu)

access relevant information. Also, in a data warehouse environment, one may locally copy remote customer and transaction records for later analysis. Typically, the remote sources are managed autonomously and updated independently of the local copies. Because the sources do not often push updates to the copies, we must periodically poll the sources to detect changes and refresh the copies.

In this scenario, a part of the local copy may get out-of-date because changes at the sources are not immediately propagated to the local copy. Therefore, it becomes important to design a good refresh policy that maximizes the "freshness" of the local copy. In this paper we address some important questions in this context. For instance, how often should we refresh the copy to maintain, say, 80% of the copy up-to-date? How much fresher does the copy get if we refresh it twice as often? Should we refresh a data item more often when its "importance" is twice as high as others? Should we poll frequently-changing objects more frequently?

Even though this synchronization problem arises in various contexts, our work is mainly motivated by our need to manage Web data. In our WebBase project, we try to store a significant portion of the Web (currently 100 million pages), in order to provide the pages to researchers for their own experiments on Web searching and data mining [Page and Brin 1998; Cho et al. 1998]. Web search engines (i.e., Google and AltaVista) also maintain copies and/or indexes of Web pages, and they need to periodically revisit the pages to maintain them up-to-date. This task is typically done by a program called a *Web crawler*.

As the Web grows larger, it becomes more important to refresh the data more effectively. Recent studies show that it often takes more than 6 months for a new page to be indexed by Web search engines [Lawrence and Giles 1998; 1999]. Also, many search engines return obsolete links, frustrating users. For example, reference [Lawrence and Giles 1999] estimates that up to 14% of the links in search engines are broken.

To improve the "freshness" of a local copy, we need to address many important challenges. First, it is not easy to even *measure* the freshness of the copy. Intuitively, the copy is considered fresh when the data in the local copy is the same as the remote sources. However, how can we instantaneously know the *current status* of the remote data when it is spread across thousands of Web sites? Second, even if a data item changes at a certain average rate, we do not know exactly when the item will change. For instance, the pages in the New York Times Web site are updated about once a day, but the update of a particular page depends on how the news related to that page develops over time. Therefore, visiting the page once a day does not guarantee its freshness.

In this paper, we will study how to synchronize data to maximize its freshness. The main contributions of this paper are:

—We present a formal framework to study the synchronization problem.

—We present several synchronization policies that are currently employed, and we compare how effective they are. Our study will show that some policies that may be intuitively appealing might actually perform *worse* than a naive policy.

—We also propose a new synchronization policy which can improve freshness by orders of magnitude in certain cases. Our policy takes into account how often a page changes and how important the pages are, and makes an appropriate

synchronization decision.

—We validate our analysis using experimental data collected from 270 Web sites over 4 month period.

The rest of this paper is organized as follows. In Section 2, we present a framework for the synchronization problem. Then in Section 3, we explain what options exist for synchronizing a local copy, and we compare these options in Sections 4 and 5. In Section 7, we verify our analysis using data collected from the World Wide Web. Finally, Section 8 discusses related work.

2. FRAMEWORK

To study the synchronization problem, we first need to understand the meaning of "freshness," and we need to know how data changes over time. In this section we present our framework to address these issues. In our discussion, we refer to the Web sites (or the data sources) that we monitor as the *real-world database* and their local copies as the *local database*, when we need to distinguish them. Similarly, we refer to individual Web pages (or individual data items) as the *real-world elements* and as the *local elements*.

In Section 2.1, we start our discussion with the definition of two freshness metrics, *freshness* and *age*. Then in Section 2.2, we discuss how we model the evolution of individual real-world elements. Finally in Section 2.3 we discuss how we model the real-world database as a whole.

2.1 Freshness and age

Intuitively, we consider a database "fresher" when the database has more up-to-date elements. For instance, when database A has 10 up-to-date elements out of 20 elements, and when database B has 15 up-to-date elements, we consider B to be fresher than A. Also, we have a notion of "age:" Even if all elements are obsolete, we consider database A "more current" than B, if A was synchronized 1 day ago, and B was synchronized 1 year ago. Based on this intuitive notion, we define *freshness* and *age* as follows:

(1) *Freshness:* Let $S = \{e_1, \ldots, e_N\}$ be the local database with N elements. Ideally, all N elements will be maintained up-to-date, but in practice, only $M(< N)$ elements will be up-to-date at a specific time. (By up-to-date we mean that their values equal those of their real-world counterparts.) We define the *freshness* of S at time t as $F(S;t) = M/N$. Clearly, the *freshness* is the fraction of the local database that is up-to-date. For instance, $F(S;t)$ will be one if all local elements are up-to-date, and $F(S;t)$ will be zero if all local elements are out-of-date. For mathematical convenience, we reformulate the above definition as follows:

Definition 2.1. The *freshness* of a local element e_i at time t is

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

Then, the *freshness* of the local database S at time t is

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

Note that freshness is hard to measure exactly in practice since we need to "instantaneously" compare the real-world data to the local copy. But as we will see, it is possible to estimate freshness given some information about how the real-world data changes.

Also note that under the above definition, the freshness of every element is considered "equal." Every element contributes the same freshness $1/N$ to the freshness of the database S. Later in Section 6, we will consider how we can extend the above definition to a general case when the freshness of one element might be more "important" than another.

(2) *Age:* To capture "how old" a database is, we define the metric *age* as follows:

Definition 2.2. We use $t_m(e_i)$ to represent the time of the first modification of e_i after the most recent synchronization. Then, the *age* of the local element e_i at time t is

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

Then the *age* of the local database S is

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

The *age* of S tells us the average "age" of the local database. For instance, if all real-world elements changed one day ago and we have not synchronized them since, $A(S;t)$ is one day.

Again, the above definition considers that the age of every element is equal. In Section 6, we consider an extension where the age of an element is more important than another.

In Figure 1, we show the evolution of $F(e_i;t)$ and $A(e_i;t)$ of an element e_i. In this graph, the horizontal axis represents time, and the vertical axis shows the value of $F(e_i;t)$ and $A(e_i;t)$. We assume that the real-world element changes at the dotted lines and the local element is synchronized at the dashed lines. The *freshness* drops to zero when the real-world element changes, and the *age* increases linearly from that point on. When the local element is synchronized to the real-world element, its *freshness* recovers to one, and its *age* drops to zero. Note that it is possible that an element changes multiple times between synchronization. Once an element changes after a synchronization, however, the following changes do not affect the freshness or age values as we show in Figure 1.

Obviously, the freshness (and age) of the local database may change over time. For instance, the freshness might be 0.3 at one point of time, and it might be 0.6 at another point of time. To compare different synchronization methods, it is important to have a metric that fairly considers freshness over a period of time, not

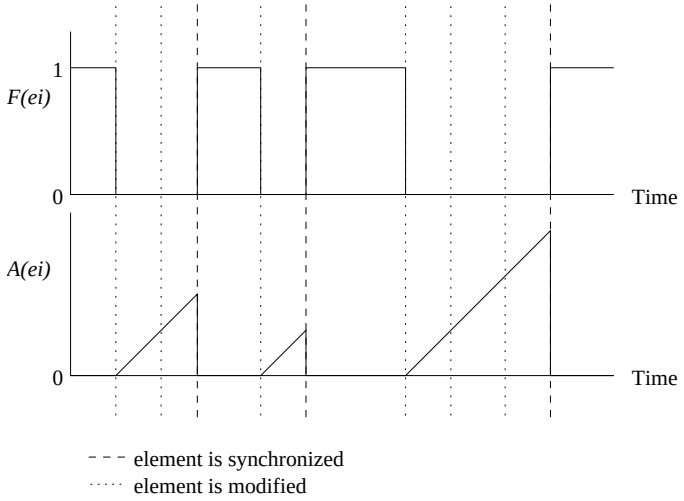


Fig. 1. An example of the time evolution of $F(e_i;t)$ and $A(e_i;t)$

just at one instant. In this paper we use the freshness *averaged over time* as this metric.

Definition 2.3. We define the time average of freshness of element e_i, $\bar{F}(e_i)$, and the time average of freshness of database S, $\bar{F}(S)$, as

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

The time average of age can be defined similarly.

From the definition, we can prove that $\bar{F}(S)$ is the average of $\bar{F}(e_i)$.

THEOREM 2.4. $$\bar{F}(S) = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(e_i) \qquad \bar{A}(S) = \frac{1}{N}\sum_{i=1}^{N} \bar{A}(e_i)$$

PROOF.

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

The proof for age is similar. □

2.2 Poisson process and probabilistic evolution of an element

To study how effective different synchronization methods are, we need to know how the real-world element changes. In this paper, we assume that the elements are modified by a *Poisson process*. The Poisson process is often used to model a sequence of events that happen *randomly* and *independently* with a *fixed rate* over time. For instance, the occurrences of fatal auto accidents, or the arrivals of customers at a service center, are usually modeled by *Poisson processes.* Under the Poisson process, it is well-known that the time to the next event is exponentially distributed [Taylor and Karlin 1998]. That is, if T is the time that the next event occurs in a Poisson process with change rate λ, the probability density function for T is

$$f_T(t) = \begin{cases} \lambda e^{-\lambda t} & \text{for } t > 0 \\ 0 & \text{for } t \leq 0. \end{cases} \quad (1)$$

Also, it is known that the probability that e_i changes at least once in the time interval $(0, t]$ is

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

In this paper, we assume that each element e_i is modified by the Poisson process with change rate λ_i. That is, each element changes at its own rate λ_i, and this rate may differ from element to element. For example, one element may change once a day, and another element may change once a year. Existing literature strongly indicates that a Poisson process is a good approximate model to describe real Web page changes [Brewington and Cybenko 2000a; 2000b]. Later in Section 7, we also experimentally verify the Poisson process model using real Web data.

Under the Poisson process model, we can analyze the freshness and age of the element e_i over time. More precisely, let us compute the *expected freshness* of e_i at time t. For the analysis, we assume that we synchronize e_i at $t = 0$ and at $t = I$. Since e_i is not synchronized in the interval $(0, I)$, the local element e_i may get out-of-date with probability $\Pr\{T \leq t\} = 1 - e^{-\lambda t}$ at time $t \in (0, I)$. Hence, the *expected freshness* is

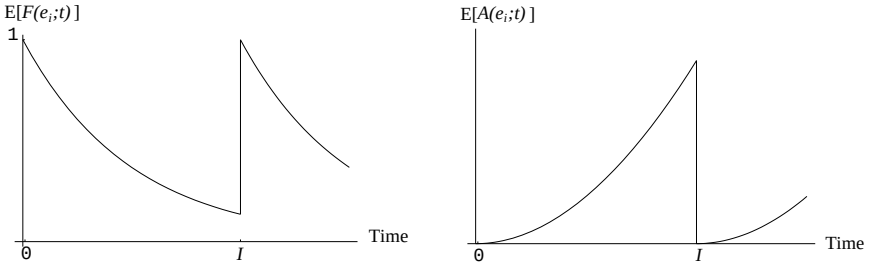
$$\mathrm{E}[F(e_i; t)] = 0 \cdot (1 - e^{-\lambda t}) + 1 \cdot e^{-\lambda t} = e^{-\lambda t} \quad \text{for } t \in (0, I).$$

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

We can obtain the *expected age* of e_i similarly. If the first time e_i is modified is at time $s \in (0, I)$, the age of e_i at time $t \in (s, I)$ is $(t - s)$. From Equation 1, e_i changes at time s with probability $\lambda e^{-\lambda s}$, so the expected age at time $t \in (0, I)$ is

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

Note that $\mathrm{E}[A(e_i; t)] \to 0$ as $t \to 0$ and that $\mathrm{E}[A(e_i; t)] \approx t$ as $t \to \infty$; the expected age is 0 at time 0 and the expected age is approximately the same as the elapsed time when t is large. In Figure 2, we show the graphs of $\mathrm{E}[F(e_i; t)]$ and $\mathrm{E}[A(e_i; t)]$. Note that when we resynchronize e_i at $t = I$, $\mathrm{E}[F(e_i; t)]$ recovers to one and $\mathrm{E}[A(e_i; t)]$ goes to zero.



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

Fig. 2. Time evolution of E[$F(e_i;t)$] and E[$A(e_i;t)$]

2.3 Evolution model of database

In the previous subsection we modeled the evolution of an element. Now we discuss how we model the database as a whole. Depending on how its elements change over time, we can model the real-world database by one of the following:

—*Uniform change-frequency model*: In this model, we assume that all real-world elements change at the *same* frequency λ. This is a simple model that could be useful in the following cases:
  —We do not know how often the *individual* element changes over time. We only know how often the entire database changes *on average*, so we may assume that all elements change at the same *average* rate λ.
  —The elements change at *slightly* different frequencies. In this case, this model will work as a good approximation.

—*Non-uniform change-frequency model*: In this model, we assume that the elements change at *different* rates. We use λ_i to refer to the change frequency of the element e_i. When the λ_i's vary, we can plot the histogram of λ_i's as we show in Figure 3. In the figure, the horizontal axis shows the range of change frequencies (e.g., $9.5 < \lambda_i \leq 10.5$) and the vertical axis shows the fraction of elements that change at the given frequency range. We can approximate the discrete histogram by a continuous distribution function $g(\lambda)$, when the database consists of many elements. We will adopt the continuous distribution model whenever convenient.

For the reader's convenience, we summarize our notation in Table I. As we continue our discussion, we will explain some of the symbols that have not been introduced yet.

3. SYNCHRONIZATION POLICIES

So far we discussed how the real-world database changes over time. In this section we 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,

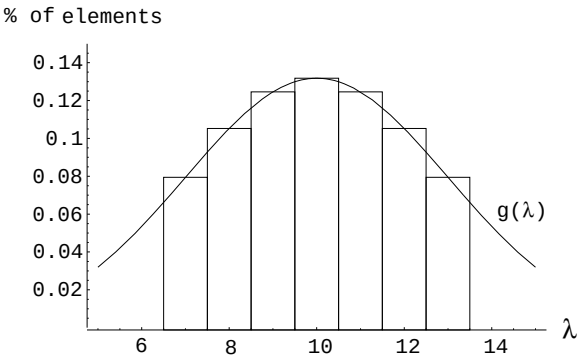


Fig. 3. Histogram of the change frequencies

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
(m) N	Number of elements in the database

Table I. The symbols that are used throughout this chapter and their meanings

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. We use the symbol f to represent $1/I$. f represents the average frequency at which an element in the database is synchronized.

(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 an example.

Example 3.1. A database consists of three elements, e_1, e_2 and e_3. It is known that the elements change at the rates $\lambda_1 = 4$, $\lambda_2 = 3$, and $\lambda_3 = 2$ (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 e_1, e_2 and e_3 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 $f_1 = 4$, $f_2 = 3$, $f_3 = 2$ (times/day) for e_1, e_2 and e_3, respectively.

Based on how fixed synchronization resources are allocated to individual elements, we can classify synchronization policies as follows. We study these policies later in Section 5.

(a) *Uniform-allocation policy*: We synchronize all elements at the same rate, regardless of how often they change. That is, all elements are synchronized at the same frequency f. In Example 3.1, 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 Example 3.1, the second option corresponds to this policy.

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

Example 3.2. We maintain a local database of 10,000 Web 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 p_1, at a regular interval of one day.

—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 p_1 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.

We can generalize the above options as follows. In Section 4 we will compare how effective these synchronization order policies are.

(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. More precisely we take a random permutation of elements in each iteration and synchronize elements in the permuted order. This policy roughly 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. While this policy is rather hypo-

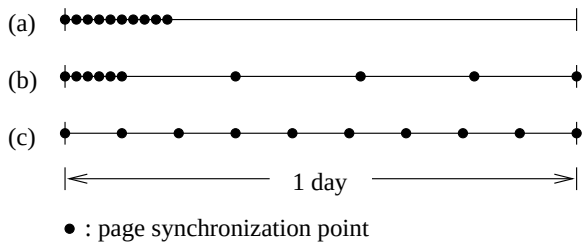


Fig. 4. Several options for the synchronization points

thetical, we believe it is a good comparison point because it has the most variability in the visit intervals.

(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.

Example 3.3. We maintain a local database of 10 pages from site A. The site is heavily accessed during daytime. We consider several synchronization policies, including the following:

—Figure 4(a): We synchronize all 10 pages in the beginning of a day, say midnight.

—Figure 4(b): We synchronize most pages in the beginning of a day, but we still synchronize some pages during the rest of the day.

—Figure 4(c): We synchronize 10 pages uniformly over a day.

In this paper we assume that we synchronize a database uniformly over time. We believe this assumption is valid especially for the Web environment. Because the Web sites are located in many different time zones, it is not easy to identify which time zone a particular Web site resides in. Also, the access pattern to a Web site varies widely. For example, some Web sites are heavily accessed during daytime, while others are accessed mostly in the evening, when the users are at home. 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.

4. SYNCHRONIZATION-ORDER POLICIES

In this section, we study the effectiveness of synchronization-order policies and identify which synchronization-order policy is the best in terms of freshness and age.

In this section we assume that all real-world elements are modified at the same average rate λ. That is, we adopt the *uniform change-frequency model* (Section 2.3). 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* (Item 2a in Section 3). These assumptions significantly simplify our analysis, while giving us solid understanding on the issues that we address. Based on these assumptions,

ALGORITHM 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`)

Fig. 5. Algorithm of fixed-order synchronization policy

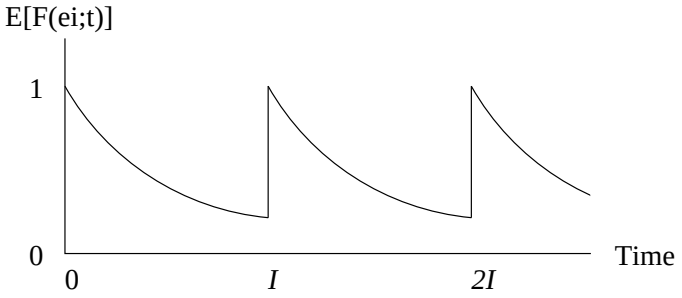


Fig. 6. Time evolution of $\mathrm{E}[F(e_i;t)]$ for fixed-order policy

we analyze different synchronization-order policies in the subsequent subsections. A reader who is not interested in mathematical details may skip to Section 4.4.

4.1 Fixed-order policy

Under the fixed-order policy, we synchronize the local elements in the *same order* repeatedly. We describe the fixed-order policy more formally in Figure 5. Here, `ElemList` records *ordered* list of *all* local elements, and `SyncQueue` records the elements *to be synchronized* in each iteration. In steps [3] through [5], we synchronize all elements once, and we repeat this loop forever. Note that we synchronize elements in the same order in every iteration, because the order in `SyncQueue` is always the same.

Now we compute the freshness of the database S. (Where convenient, we will refer to the time-average of freshness simply as freshness, if it does not cause any confusion.) Since we can compute the freshness of S from freshness of its elements (Theorem 2.4), we first study how the freshness of a random element e_i evolves over time.

Since we assume that it takes I time units to synchronize all N elements in S, the expected freshness of e_i will evolve as in Figure 6. In the graph, we assumed that we synchronize e_i initially at $t = 0$ without losing generality. Note that $\mathrm{E}[F(e_i;t)]$ recovers to 1 every I time units, when we synchronize it. Intuitively, e_i goes through exactly the same process every I time units, so we can expect that we can learn everything about e_i by studying how e_i evolves in the interval $(0, I]$. In particular, we suspect that the freshness of e_i averaged over time, $\lim_{t\to\infty} \frac{1}{t}\int_0^t F(e_i;t)dt$, should be equal to the expected freshness of e_i averaged over the interval $(0, I)$,

$\frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt$. The following theorem proves it more formally.

THEOREM 4.1. *When the element e_i is synchronized at the fixed interval of I time units, 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.

$$\begin{aligned}
\bar{F}(e_i) &= \lim_{t\to\infty} \frac{\int_0^t F(e_i;t)dt}{t} \\
&= \lim_{n\to\infty} \frac{\sum_{j=0}^{n-1}\int_{jI}^{(j+1)I} F(e_i;t)dt}{\sum_{j=0}^{n-1} I} \\
&= \lim_{n\to\infty} \frac{\sum_{j=0}^{n-1}\int_0^I F(e_i;t+jI)dt}{nI} \\
&= \frac{1}{I}\int_0^I \left[\lim_{n\to\infty}\frac{1}{n}\sum_{j=0}^{n-1} F(e_i;t+jI)\right] dt \qquad (2)
\end{aligned}$$

Because we synchronize e_i every I time units from $t=0$, $F(e_i;t+jI)$ is the freshness of e_i at t *time units after each synchronization.* Therefore, $\frac{1}{n}\sum_{j=0}^{n-1} F(e_i;t+jI)$, the average of freshness at t time units 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,

$$\frac{1}{I}\int_0^I \left[\lim_{n\to\infty}\frac{1}{n}\sum_{j=0}^{n-1} F(e_i;t+jI)\right] dt = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt. \qquad (3)$$

From Equations 2 and 3, $F(e_i) = \frac{1}{I}\int_0^I \mathrm{E}[F(e_i;t)]dt$. □

Based on Theorem 4.1, we can compute the freshness of e_i.

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

Here, f is $1/I$, the (average) synchronization rate of an element. Throughout this section, we assume 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

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

ALGORITHM 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`)

Fig. 7. Algorithm of random-order synchronization policy

We can analyze the age of S similarly.

$$\bar{A}(S) = \bar{A}(e_i) = \frac{1}{I}\int_0^I \mathrm{E}[A(e_i;t)]dt = \frac{1}{I}\int_0^I t(1-\frac{1-e^{-\lambda t}}{\lambda t})dt = \frac{1}{f}(\frac{1}{2} - \frac{1}{\lambda/f} + \frac{1-e^{-\lambda/f}}{(\lambda/f)^2})$$

4.2 Random-order policy

Under the random-order policy, the synchronization order of elements might be different from one crawl to the next. Figure 7 describes the random-order policy more formally. Note that we randomize the order of elements before every iteration by applying random permutation (step [2]).

The random-order policy is slightly 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 the second iteration. Therefore, the synchronization interval of e_i, W, is not a fixed number any more, but follows a certain distribution $f_W(w)$. Therefore the equation of Theorem 4.1 should be modified accordingly. In Theorem 4.1, we simply divided $\int_0^I \mathrm{E}[F(e_i;t)]dt$ by the fixed synchronization interval I. But now because the synchronization interval w spans on $[0, 2I]$ following the distribution $f_W(w)$, we need to take the average of $\int_0^w \mathrm{E}[F(e_i;t)]dt$ weighted by the frequency of synchronization interval $w \in [0, 2I]$, and divide it by the average synchronization interval $\int_0^{2I} f_W(w)\, w\, dw$:

$$\bar{F}(e_i) = \lim_{t\to\infty}\frac{1}{t}\int_0^t F(e_i;t)dt = \frac{\int_0^{2I} f_W(w)\left(\int_0^w \mathrm{E}[F(e_i;t)]dt\right)dw}{\int_0^{2I} f_W(w)\, w\, dw} \qquad (4)$$

To perform the above integration, we need to derive the closed form of $f_W(w)$.

LEMMA 4.2. *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*

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

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

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

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

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

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

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

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

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

□

Using Lemma 4.2, we can compute each part of Equation 4:

$$\begin{gathered} \textstyle\int_0^{2I} f_W(w)\, w\, dw = \int_0^I \frac{w}{I^2}\, w\, dw + \int_I^{2I} \frac{2I-w}{I^2}\, w\, dw = I \\ \textstyle\int_0^w \mathrm{E}[F(e_i; t)] dt = \int_0^w e^{-\lambda t} dt = \frac{1}{\lambda}(1 - e^{-\lambda w}) \\ \textstyle\int_0^{2I} f_W(w) \left(\int_0^w \mathrm{E}[F(e_i; t)] dt \right) dw = \int_0^I \left(\frac{w}{I^2}\right) \frac{1}{\lambda}(1 - e^{-\lambda w}) dw + \int_I^{2I} \left(\frac{2I-w}{I^2}\right) \frac{1}{\lambda}(1 - e^{-\lambda w}) dw \\ = \frac{1}{\lambda} \left[1 - \left(\frac{1 - e^{-\lambda I}}{\lambda I} \right)^2 \right] \end{gathered}$$

Thus,

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

Since this analysis is valid for any element e_i, the freshness of S is the same as above:

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

ALGORITHM 4.3. **Purely-random synchronization**
Input: `ElemList` $= \{e_1, e_2, \ldots, e_N\}$
Procedure
[1] While (TRUE)
[2] `e` := PickRandom(`ElemList`)
[3] Synchronize(`e`)

Fig. 8. Algorithm of purely-random synchronization policy

We can compute $\bar{A}(S)$ through the same steps and get the following result.

$$\bar{A}(S) = \frac{\int_0^{2I} f_W(w) \left(\int_0^w \mathrm{E}[A(e_i;t)]dt\right) dw}{\int_0^{2I} f_W(w)\ w\ dw} = \frac{1}{f}\left[\frac{1}{3} + \left(\frac{1}{2} - \frac{1}{\lambda/f}\right)^2 - \left(\frac{1-e^{-\lambda/f}}{(\lambda/f)^2}\right)^2\right]$$

4.3 Purely-random policy

Whenever we synchronize an element, we pick an arbitrarily random element under the purely-random policy. Figure 8 describes the policy more formally.

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

$$\bar{F}(e_i) = \lim_{t\to\infty} \frac{1}{t}\int_0^t F(e_i;t)dt = \frac{\int_0^{\infty} f_W(w) \left(\int_0^w \mathrm{E}[F(e_i;t)]dt\right) dw}{\int_0^{\infty} f_W(w)\ w\ dw}. \qquad (5)$$

Note that the outer integral is over $(0,\infty)$, since the synchronization interval of e_i may get arbitrarily large. The law of rare events [Taylor and Karlin 1998] shows that $f_W(w)$ is

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

Using $f_W(w)$, we can compute each part of Equation 5:

$$\int_0^{\infty} f_W(w)\ w\ dw = \int_0^{2I} fe^{-fw}\ w\ dw = 1/f$$
$$\int_0^w \mathrm{E}[F(e_i;t)]dt = \int_0^w e^{-\lambda t}dt = \tfrac{1}{\lambda}(1-e^{-\lambda w})$$
$$\int_0^{\infty} f_W(w) \left(\int_0^w \mathrm{E}[F(e_i;t)]dt\right) dw = \int_0^{\infty} (fe^{-fw})\tfrac{1}{\lambda}(1-e^{-\lambda w})\ dw = \tfrac{1}{f+\lambda}$$

Therefore,

$$\bar{F}(S) = \bar{F}(e_i) = \frac{1}{1+\lambda/f}$$

We can compute $\bar{A}(S)$ through the same steps and get the following result:

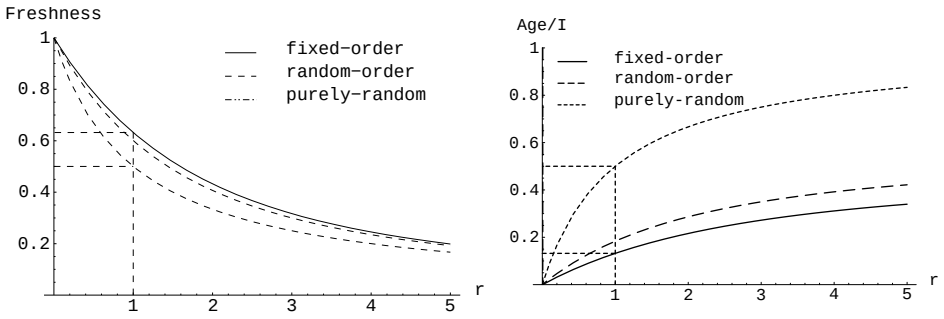
$$\bar{A}(S) = \frac{\int_0^{\infty} f_W(w) \left(\int_0^w \mathrm{E}[A(e_i;t)]dt\right) dw}{\int_0^{\infty} f_W(w)\ w\ dw} = \frac{1}{f}\left(\frac{\lambda/f}{1+\lambda/f}\right).$$

4.4 Comparison of synchronization-order policies

In Table II, we summarize the results in the preceding subsections. In the table, we use r to represent the frequency ratio λ/f, where λ is the frequency at which a real-world element changes and $f(=1/I)$ is the frequency at which a local element

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

Table II. Freshness and age formula for various synchronization-order policies



(a) Freshness graph over $r = \lambda/f$

(b) Age graph over $r = \lambda/f$

Fig. 9. Comparison of freshness and age of various synchronization policies

is synchronized. When $r < 1$, we synchronize the elements more often than they change, and when $r > 1$, the elements change more often than we synchronize them.

To help readers interpret the formulas, we show the freshness and the age graphs in Figure 9. In the figure, the horizontal axis is the frequency ratio r, and the vertical axis shows the freshness and the age of the local database. From the graph, it is clear that the fixed-order policy performs best by both metrics. For instance, when we synchronize the elements as often as they change ($r = 1$), the freshness of the fixed-order policy is $(e-1)/e \approx 0.63$, which is 30% higher than that of the purely-random policy. The difference is more dramatic for age. When $r = 1$, the age of the fixed-order policy is only one fourth of the random-order policy. In general, as the variability in the time between visits increases, the policy gets less effective. This result is expected given that the fixed-order policy guarantees a "bound" on freshness and age values while the purely-random policy may lead to unlimited freshness and age values.

Notice that as we synchronize the elements more often than they change ($\lambda \ll f$, thus $r = \lambda/f \to 0$), the freshness approaches 1 and the age approaches 0. Also, when the elements change more frequently than we synchronize them ($r = \lambda/f \to \infty$), the freshness becomes 0, and the age increases. Finally, notice that the freshness is not equal to 1, even if we synchronize the elements as often as they change ($r = 1$). This fact has two reasons. First, an element changes at random points of time, even if it changes at a fixed *average* rate. Therefore, the element may not change between some synchronizations, and it may change more than once between other synchronizations. For this reason, it cannot be always

up-to-date. Second, some delay may exist between the change of an element and its synchronization, so some elements may be "temporarily obsolete," decreasing the freshness of the database.

5. RESOURCE-ALLOCATION POLICIES

In the previous section, we compared synchronization-order policies 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? In this section we address this question by studying different resource-allocation policies (Item 2 in Section 3). For the study, we model the real-world database by the *non-uniform* change-frequency model (Section 2.3), and we assume the *fixed-order* policy for the synchronization-order policy (Item 3 in Section 3), because the fixed-order policy is the best synchronization-order policy. In other words, we assume that the element e_i 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 ($\frac{1}{N}\sum_{i=1}^{N} f_i$) should be equal to f.

In Section 5.1, we start our discussion by comparing the two most straightforward resource allocation policies: the uniform-allocation and the proportional-allocation policy. While we may intuitively expect that the proportional policy performs better than the uniform policy, our result in Section 5.1 will show that the uniform policy is *always* more effective than the proportional policy under any scenario. Then in Section 5.2 we try to understand why this happens by studying a simple example. Finally in Section 5.3 we derive the optimal resource-allocation policy that either maximizes freshness or minimizes age. A reader who only wants to learn the optimal resource-allocation policy may skip to Section 5.3.

5.1 Superiority of the uniform policy to the proportional policy

In this subsection, we prove that the uniform policy is better than the proportional policy under any distribution of λ values. To help our proof, we use $\bar{F}(\lambda_i, f_i)$ to refer to the time average of freshness of e_i when it changes at λ_i and is synchronized at f_i. Our proof is based on the convexity of the freshness function $\bar{F}(\lambda_i, f_i)$ over λ_i. When $f(x)$ is a convex function, it is well known [Thomas, Jr. 1969] that

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \geq f(\frac{1}{n}\sum_{i=1}^{n} x_i) \quad \text{for any } x_i\text{'s } (i = 1, 2, \ldots, n). \tag{6}$$

Similarly, when $f(x)$ is concave,

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \leq f(\frac{1}{n}\sum_{i=1}^{n} x_i) \quad \text{for any } x_i\text{'s } (i = 1, 2, \ldots, n). \tag{7}$$

Our goal is to prove that the freshness of the uniform policy (represented as $\bar{F}(S)_u$) is better than the freshness of the proportional policy (represented as $\bar{F}(S)_p$). We first note that the function $\bar{F}(\lambda_i, f_i)$ derived in Section 4 is convex

over λ_i for any synchronization-order policy.[1] Therefore, $\bar{F}(\lambda_i, f_i)$ satisfies Equation 6 for variable λ_i.

THEOREM 5.1. *It is always true that* $\bar{F}(S)_u \geq \bar{F}(S)_p$.

PROOF. By definition, the uniform policy is $f_i = f$ for any i. Then

$$\bar{F}(S)_u = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f_i) = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f). \tag{8}$$

For the proportional policy, $\lambda_i/f_i = \lambda/f = r$ for any i, and because the analytical form of $\bar{F}(\lambda_i, f_i)$ only depends on the ratio $r = \lambda_i/f_i$, $\bar{F}(e_i) = \bar{F}(\lambda_i, f_i) = \bar{F}(\lambda, f)$. Therefore,

$$\bar{F}(S)_p = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(e_i) = \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda, f) = \bar{F}(\lambda, f). \tag{9}$$

Then

$$\begin{aligned}
\bar{F}(S)_u &= \frac{1}{N}\sum_{i=1}^{N} \bar{F}(\lambda_i, f) && \text{(Equation 8)} \\
&\geq \bar{F}\left(\frac{1}{N}\sum_{i=1}^{N} \lambda_i, f\right) && \text{(convexity of } \bar{F} \text{ over } \lambda_i) \\
&= \bar{F}(\lambda, f) && \text{(definition of } \lambda) \\
&= \bar{F}(S)_p. && \text{(Equation 9)}
\end{aligned}$$

□

Similarly, we can prove that the age of the uniform policy, $\bar{A}(S)_u$, is always less than the age of the proportional policy, $\bar{A}(S)_p$, based on the concavity of $\bar{A}(\lambda_i, f_i)$ over λ_i.

THEOREM 5.2. *It is always true that* $\bar{A}(S)_u \leq \bar{A}(S)_p$.

PROOF. From the definition of the uniform and the proportional policies,

$$\bar{A}(S)_u = \frac{1}{N}\sum_{i=1}^{N} \bar{A}(\lambda_i, f) \tag{10}$$

$$\bar{A}(S)_p = \frac{1}{N}\sum_{i=1}^{N} \bar{A}(\lambda_i, f_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{\lambda}{\lambda_i}\bar{A}(\lambda, f). \tag{11}$$

[1] We can verify its convexity by computing $\partial^2 \bar{F}(\lambda_i, f_i)/\partial \lambda_i^2$ of the derived function.

Then

$$\begin{aligned}
\bar{A}(S)_u &= \frac{1}{N}\sum_{i=1}^{N}\bar{A}(\lambda_i, f) && \text{(Equation 10)}\\
&\le \bar{A}(\frac{1}{N}\sum_{i=1}^{N}\lambda_i, f) && \text{(concavity of } \bar{A} \text{ over } \lambda_i)\\
&= \bar{A}(\lambda, f) && \text{(definition of } \lambda)\\
\bar{A}(S)_p &= \lambda\bar{A}(\lambda, f)\left(\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\lambda_i}\right) && \text{(Equation 11)}\\
&\ge \lambda\bar{A}(\lambda, f)\frac{1}{\frac{1}{N}\sum_{i=1}^{N}\lambda_i} && \text{(convexity of function } \frac{1}{x})\\
&= \lambda\bar{A}(\lambda, f)\frac{1}{\lambda} && \text{(definition of } \lambda)\\
&= \bar{A}(\lambda, f).
\end{aligned}$$

Therefore, $\bar{A}(S)_u \le \bar{A}(\lambda, f) \le \bar{A}(S)_p$. □

5.2 Two-element database

Intuitively, we expected that the proportional policy would be better than the uniform policy, because we allocate more resources to the elements that change more often, which may need more of our attention. But why is it the other way around? In this subsection, we try to understand why we get the counterintuitive result, by studying a very simple example: a database consisting of two elements. The analysis of this simple example will let us understand the result more concretely, and it will reveal some intuitive trends. We will confirm the trends more precisely when we study the optimal synchronization policy later in Section 5.3.

Now we analyze a database consisting of two elements: e_1 and e_2. For the analysis, we assume that e_1 changes at 9 times/day and e_2 changes at once/day. We also assume that our goal is to maximize the freshness of the database averaged over time. In Figure 10, we visually illustrate our simple model. For element e_1, one day is split into 9 intervals, and e_1 changes *once and only once* in each interval. However, we do not know exactly when the element changes in one interval. For element e_2, it changes *once and only once* per day, and we do not know when it changes. While this model is not exactly a Poisson process model, we adopt this model due to its simplicity and concreteness.

Now let us assume that we decided to synchronize only *one* element per day. Then what element should we synchronize? Should we synchronize e_1 or should we synchronize e_2? To answer this question, we need to compare how freshness changes if we pick one element over the other. If the element e_2 changes in the middle of the day and if we synchronize e_2 right after it changed, it will remain up-to-date for the remaining half of the day. Therefore, by synchronizing element e_2 we get 1/2 day "benefit"(or freshness increase). However, the probability that e_2 changes before the middle of the day is 1/2, so the "expected benefit" of synchronizing

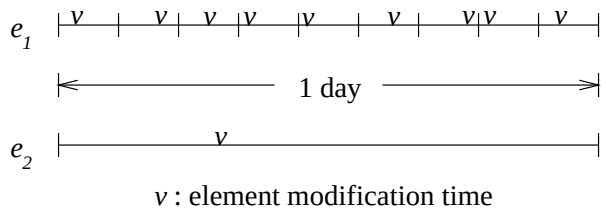


Fig. 10. A database with two elements with different change frequency

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	$\frac{5}{36} + \frac{15}{36} = \frac{40}{72}$		

Table III. Estimation of benefits for different choices

e_2 is $1/2 \times 1/2$ day $= 1/4$ day. By the same reasoning, if we synchronize e_1 in the middle of an interval, e_1 will remain up-to-date for the remaining half of the interval (1/18 of the day) with probability 1/2. Therefore, the expected benefit is $1/2 \times 1/18$ day $= 1/36$ day. From this crude estimation, we can see that it is more effective to select e_2 for synchronization!

Table III shows the expected benefits for several other scenarios. The second column shows the total synchronization frequencies ($f_1 + f_2$) and the third column shows how much of the synchronization is allocated to f_1 and f_2. In the fourth column we estimate the expected benefit, and in the last column we show the f_1 and f_2 values that give the *highest* expected benefit. To reduce clutter, when $f_1 + f_2 = 5$ and 10, we show only some interesting (f_1, f_2) pairs. Note that since $\lambda_1 = 9$ and $\lambda_2 = 1$, row (h) corresponds to the proportional policy ($f_1 = 9, f_2 = 1$), and row (j) corresponds to the uniform policy ($f_1 = f_2 = 5$). From the table, we can observe the following interesting trends:

(1) Rows (a)-(e): When the synchronization frequency ($f_1 + f_2$) 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 ($f_1 + f_2 = 10$), 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.

We note that it may be surprising that the uniform policy performs better than the proportional policy. Is this result due to an artifact of our discrete freshness

metric (either 0 or 1)? Is the uniform policy better for *any* freshness metric? While we do not have a formal proof, we believe that this result is not due to the discreteness of our definition for the following reasons: First, in [Cho 2001], we generalize the definition of freshness to a continuous value (between zero and one) that gradually "decays" over time. Even under this continuous definition, we still get the same result (the uniform policy is better than the proportional policy). Second, the uniform policy also performs better for the age metric, which is based on continuous age values.

In general, it will be an interesting research topic to study exactly what "codition" guarantees the superiority of the uniform policy over the proportional policy.

5.3 The optimal resource-allocation policy

In this section, we formally verify the trend we have just observed by deriving the optimal policy. Mathematically, we can formulate our goal as follows:

PROBLEM 5.3. *Given λ_i's ($i = 1, 2, \ldots, N$), find the values of f_i's ($i = 1, 2, \ldots, N$) that maximize*

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

when f_i's satisfy the constraints

$$\frac{1}{N}\sum_{i=1}^{N} f_i = f \quad \text{and} \quad f_i \geq 0 \quad (i = 1, 2, \ldots, N)$$

Because we can derive the closed form of $\bar{F}(\lambda_i, f_i)$,[2] we can solve the above problem.

SOLUTION Using the *method of Lagrange multipliers* [Thomas, Jr. 1969], we can derive that the freshness of database S, $\bar{F}(S)$, takes its maximum when all f_i's satisfy the following equations[3]

$$\frac{\partial \bar{F}(\lambda_i, f_i)}{\partial f_i} = \mu \quad \text{and} \quad \frac{1}{N}\sum_{i=1}^{N} f_i = f.$$

Notice that we introduced another variable μ in the solution,[4] and the solution consists of $(N+1)$ equations (N equations of $\partial \bar{F}(\lambda_i, f_i)/\partial f_i = \mu$ and one equation of $\frac{1}{N}\sum_{i=1}^{N} f_i = f$) with $(N+1)$ unknown variables $(f_1, \ldots, f_N, \mu)$. We can solve these $(N+1)$ equations for f_i's, since we know the closed form of $\bar{F}(\lambda_i, f_i)$.

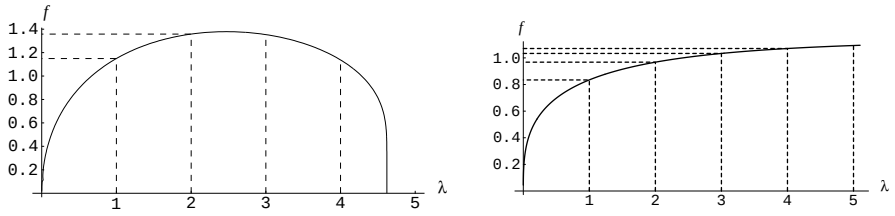
From the solution, we can see that all optimal f_i's satisfy the equation $\partial \bar{F}(\lambda_i, f_i)/\partial f_i = \mu$. That is, all optimal (λ_i, f_i) pairs are on the graph of $\partial \bar{F}(\lambda, f)/\partial f = \mu$. To illustrate the property of the solution, we use the following example.

Example 5.4. The real-world database consists of five elements, which change at the frequencies of 1, 2, ..., 5 (times/day). We list the change frequencies in

[2]For instance, $\bar{F}(\lambda_i, f_i) = (1 - e^{-\lambda_i/f_i})/(\lambda_i/f_i)$ for the fixed-order policy.
[3]When $\partial \bar{F}(\lambda_i, f_i)/\partial f_i = \mu$ does not have a solution with $f_i \geq 0$, f_i should be equal to zero.
[4]This is a typical artifact of the method of Lagrange multipliers.

	e_1	e_2	e_3	e_4	e_5
(a) change frequency	1	2	3	4	5
(b) synchronization frequency (freshness)	1.15	1.36	1.35	1.14	0.00
(c) synchronization frequency (age)	0.84	0.97	1.03	1.07	1.09

Table IV. The optimal synchronization frequencies of Example 5.4



(a) synchronization frequency as a function of change frequency for freshness optimization

(b) synchronization frequency as a function of change frequency for age optimization

Fig. 11. Solution of the freshness and age optimization problem of Example 5.4

row (a) of Table IV. (We explain the meaning of rows (b) and (c) later, as we continue our discussion.) We decided to synchronize the local database at the rate of 5 elements/day total, but we still need to find out how often we should synchronize each element.

For this example, we can solve Problem 5.3 analytically, and we show the graph of its solution in Figure 11(a). The horizontal axis of the graph corresponds to the change frequency of an element, and the vertical axis shows the optimal synchronization frequency of the element with that given change frequency. For instance, the optimal synchronization frequency of e_1 is 1.15 ($f = 1.15$), because the change frequency of element e_1 is 1 ($\lambda = 1$). Similarly from the graph, we can find the optimal synchronization frequencies of other elements, and we list them in row (b) of Table IV.

Notice that while e_4 changes twice as often as e_2, we need to synchronize e_4 less frequently than e_2. Furthermore, the synchronization frequency of e_5 is zero, while it changes at the highest rate. This result comes from the shape of Figure 11(a). In the graph, when $\lambda > 2.5$, f decreases as λ increases. Therefore, the synchronization frequencies of the elements e_3, e_4 and e_5 gets smaller and smaller.

While we obtained Figure 11(a) by solving Example 5.4, we prove in the next subsection that the shape of the graph is the same for *any* distributions of λ_i's. That is, the optimal graph for *any* database S is *exactly the same* as Figure 11(a), except that the graph of S is scaled by a constant factor from Figure 11(a). Since the shape of the graph is always the same, the following statement is true in any scenario: *To improve freshness, we should penalize the elements that change too often.*

Similarly, we can compute the optimal *age* solution for Example 5.4, and we show the result in Figure 11(b). The axes in this graph are the same as before. Also,

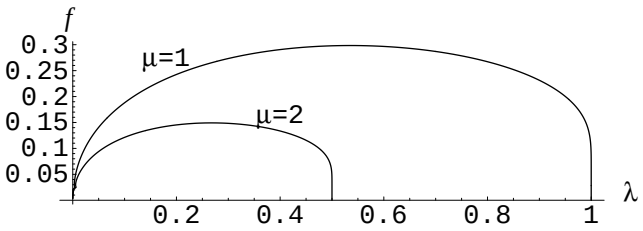


Fig. 12. $\partial \bar{F}(\lambda, f)/\partial f = \mu$ graph when $\mu = 1$ and $\mu = 2$

we list the optimal synchronization frequencies in row (c) of Table IV. Contrary to the freshness, we can observe that we should synchronize an element more often when it changes more often ($f_1 < \cdots < f_5$). However, notice that the difference between the synchronization frequencies is marginal: All f_i's are approximately close to one. In other words, the optimal solution is rather closer to the uniform policy than to the proportional policy. Similarly for age, we can prove that the shape of the optimal age graph is always the same as Figure 11(b). Therefore, the trend we observed here is very general and holds for *any* database.

5.4 The graph of the equation $\partial \bar{F}(\lambda, f)/\partial f = \mu$

We now study the property of the solutions for Problem 5.3. Since the solutions are always on the graph $\partial \bar{F}(\lambda, f)/\partial f = \mu$, and since only μ changes depending on scenarios, we can study the property of the solutions simply by studying how the graph changes for various μ values.

In particular, we prove that the graph $\partial \bar{F}(\lambda, f)/\partial f = \mu$ is scaled *only by a constant factor* for different μ values. For example, Figure 12 shows the graphs of the equation for $\mu = 1$ and 2. The graph clearly shows that the $\mu = 2$ graph is simply a scaled version of the $\mu = 1$ graph by a factor of 2; the $\mu = 1$ graph converges to zero when $\lambda \to 1$, while the $\mu = 2$ graph converges to zero when $\lambda \to 1/2$. Also, the $\mu = 1$ graph takes its maximum at $(0.53, 0.3)$, while the $\mu = 2$ takes its maximum at $(0.53/2, 0.3/2)$. The following theorem generalizes this scaling property.

THEOREM 5.5. *We assume* (λ, f) *satisfy the equation* $\frac{\partial \bar{F}}{\partial f}(\lambda, f) = \mu$. *Then* $(\mu\lambda, \mu f)$, *the point scaled by the factor* μ *from* (λ, f), *satisfies the equation* $\frac{\partial \bar{F}}{\partial f}(\mu\lambda, \mu f) = 1$.

PROOF. For the fixed-order policy,[5] $\frac{\partial \bar{F}}{\partial f}(\lambda, f) = -\frac{e^{-\lambda/f}}{f} + \frac{1-e^{-\lambda/f}}{\lambda}$. Then,

$$\frac{\partial \bar{F}}{\partial f}(\mu\lambda, \mu f) = -\frac{e^{-\mu\lambda/\mu f}}{\mu f} + \frac{1-e^{-\mu\lambda/\mu f}}{\mu\lambda} = \frac{1}{\mu}\left(-\frac{e^{-\lambda/f}}{f} + \frac{1-e^{-\lambda/f}}{\lambda}\right) = \frac{1}{\mu}\,\frac{\partial \bar{F}}{\partial f}(\lambda, f) = \frac{1}{\mu}\,\mu = 1. \tag{12}$$

Therefore, the points $(\mu\lambda, \mu f)$ satisfies the equation $\frac{\partial \bar{F}}{\partial f}(\mu\lambda, \mu f) = 1$. □

From Theorem 5.5, we can see that the graph of $\partial \bar{F}(\lambda, f)/\partial f = \mu$ for any μ value is essentially a *scaled version* of the graph $\partial \bar{F}(\lambda, f)/\partial f = 1$. Therefore, its shape is *always* the same for *any* value of μ. Since only the μ value changes as the

[5]We can similarly prove the theorem for other synchronization-order policies.

distribution of λ_i's change, this "scaling property" for different μ values shows that the shape of the graph is *always* the same under any distribution.

Similarly for age, we can prove that the graphs are scaled only by a constant factor for any distributions of λ_i's.

6. WEIGHTED FRESHNESS

So far, we assumed that the freshness and the age of every element is equally important to the users. But what if the elements have different "importance"? For example, if the database S has two elements, e_1 and e_2, and if the users access e_1 twice as often as e_2, how should we synchronize the elements to maximize the freshness of the database *perceived by the users*? Should we refresh e_1 more often than e_2? In this section, we study how we can extend our model to address this scenario.

6.1 Weighted freshness metrics

When the users access element e_1 twice more often than e_2, we may consider that e_1 is twice as important as e_2, because we can make the users see fresh elements twice as often by maintaining e_1 up-to-date. To capture this concept of "importance", we may give different "weights" w_i's to element e_i's and define the freshness of a database as follows:

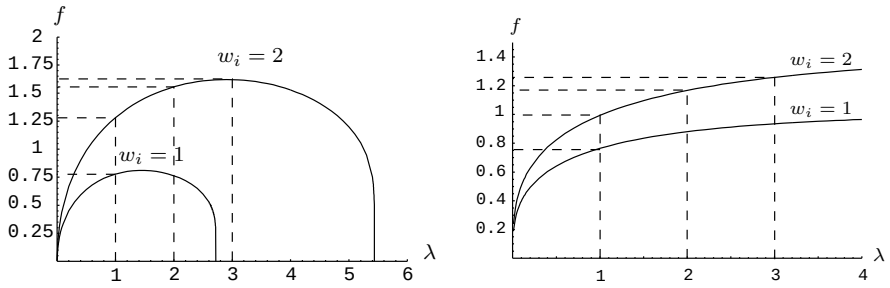
Definition 6.1. Let w_i be the weight (or importance) of element e_i. The freshness and the age of a database S is defined as

$$F(S;t) = \left(\sum_{i=1}^{N} w_i\, F(e_i;t)\right) \Bigg/ \left(\sum_{i=1}^{N} w_i\right) \qquad A(S;t) = \left(\sum_{i=1}^{N} w_i\, A(e_i;t)\right) \Bigg/ \left(\sum_{i=1}^{N} w_i\right)$$

Note that when all w_i's are equal to 1, the above definition reduces to our previous definition. Under this weighted definition, we can analyze the Poisson-process model using a similar technique described in Section 5.3. We first illustrate the result with the following example.

Example 6.2. We maintain 6 elements in our local database, whose weights and change frequencies are listed in the rows (a) and (b) of Table V. In this example, there exist two classes of elements, the elements with weight 1 (e_{1*}) and the elements with weight 2 (e_{2*}). Each class contains 3 elements (e_{*1}, e_{*2}, e_{*3}) with different change frequencies. We assume that we synchronize a total of 6 elements every day. Under these parameters, we can analyze the freshness maximization problem, whose result is shown in Figure 13(a).

The horizontal axis of the graph represents the change frequency of an element, and the vertical axis shows the optimal synchronization frequency for the given change frequency. The graph consists of two curves because the elements in different classes follow different curves. The $w_i = 2$ elements (e_{2*}) follow the outer curve, and the $w_i = 1$ elements (e_{1*}) follow the inner curve. For example, the optimal synchronization frequency of e_{11} is 0.78, because its weight is 1 and its change frequency is once a day. We list the optimal synchronization frequencies for all elements in row (c) of Table V. From this result, we can clearly see that we should visit an element more often when its weight is higher. For instance, the



(a) synchronization frequency as a function of change frequency for freshness optimization

(b) synchronization frequency as a function of change frequency for age optimization

Fig. 13. Solution of the freshness and age optimization problem of Example 6.2

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

Table V. The optimal synchronization frequencies of Example 6.2

synchronization frequency of e_{21} is 1.28, which is higher than 0.78 of e_{11}. (Note that both e_{21} and e_{11} change at the same frequency.)

However, note that the optimal synchronization frequency is *not* proportional to the weight of an element. For example, the optimal synchronization frequency of e_{23} is infinitely larger than that of e_{13}! In fact, in the following theorem, we prove that the weight of an element determines the *scale of the optimal curve* that the element follows. That is, the optimal curve that the $w_i = 2$ elements follow is *exactly twice* as large as the curve that the $w_i = 1$ elements follow.

Similarly, we can analyze the weighted age metric for the example, and we show the result in Figure 13(b). Again, we can see that we should visit an element more often when its weight is higher, but not necessarily proportionally more often. In general, we can prove the following property for age: *When the weight of e_1 is k times as large as that of e_2, the optimal age curve that e_1 follows is $\sqrt{k}$ times larger than the curve that e_2 follows.*

THEOREM 6.3. *When the weight of e_1 is k times as large as that of e_2, the optimal freshness curve that e_1 follows is k times larger than the curve that e_2 follows.*

PROOF. Similarly to Problem 5.3, we can solve the freshness optimization problem under the weighted freshness definition and show that the optimal f_i values satisfy the equation

$$\partial \bar{F}(\lambda_i, f_i)/\partial f_i = \mu/w_i.$$

To prove the theorem, we need to prove that if the (λ_i, f_i) satisfy the above equa-

tion for $w_i = 1$, then the pair $(k\lambda_i,\ kf_i)$ satisfy the equation for $w_i = k$. It is straightforward to prove this fact. For example, for the fixed-order policy,[6]

$$\frac{\partial \bar{F}}{\partial f}(\lambda, f_i) = -\frac{e^{-\lambda/f_i}}{f_i} + \frac{1 - e^{-\lambda/f_i}}{\lambda}.$$

Then

$$\frac{\partial \bar{F}}{\partial f}(k\lambda_i, kf_i) = -\frac{e^{-k\lambda_i/kf_i}}{kf_i} + \frac{1-e^{-k\lambda_i/kf_i}}{k\lambda_i} = \frac{1}{k}\left(-\frac{e^{-\lambda_i/f_i}}{f_i} + \frac{1-e^{-\lambda_i/f_i}}{\lambda_i}\right) = \frac{1}{k}\ \frac{\partial \bar{F}}{\partial f}(\lambda_i, f_i) = \mu/k$$

□

THEOREM 6.4. *When the weight of e_1 is k times as large as that of e_2, the optimal age curve that e_1 follows is $\sqrt{k}$ times larger than the curve that e_2 follows.*

PROOF. The proof is similar to that of Theorem 6.3. □

7. EXPERIMENTS

Throughout this paper we modeled database changes as a Poisson process. In this section, we first verify the Poisson process model using experimental data collected from 270 sites. Then, using the observed change frequencies on the Web, we compare the effectiveness of our various synchronization policies. The experimental results will show that our optimal policy performs significantly better than the current policies used by crawlers.

7.1 Experimental setup

To collect the data on how often Web pages change, we crawled around 720,000 pages from 270 "popular" sites every day, from February 17th through June 24th, 1999. This was done with the Stanford WebBase crawler, a system designed to create and maintain large Web repositories. The system is capable of high indexing speeds (about 60 pages per second), and can handle relatively large data repositories (currently 210GB of HTML is stored). In this section we briefly discuss how the particular sites were selected for our experiments.

To select the sites for our experiment, we used the snapshot of the Web in our WebBase repository. At that time, WebBase maintained the snapshot of 25 million Web pages, and based on this snapshot we identified the top 400 "popular" sites as the candidate sites. To measure the popularity of sites, we essentially counted how many pages in our repository had a link to each site, and we used the count as the popularity measure of a site.[7]

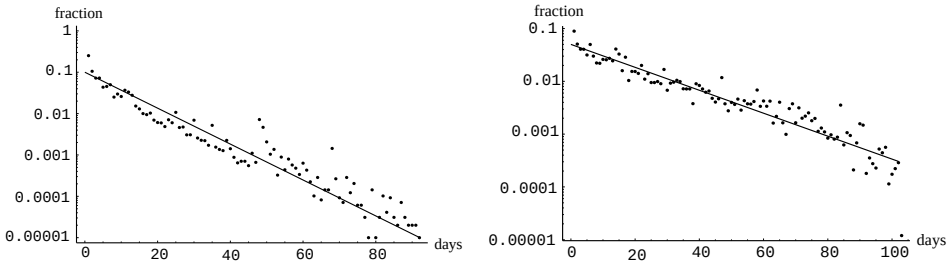
Then, we contacted the Webmasters of all candidate sites asking their permission for our experiment. After this step, 270 sites remained, including sites such as Yahoo (`http://yahoo.com`), Microsoft (`http://microsoft.com`), and Stanford (`http://www.stanford.edu`). Focusing on the "popular" sites biases our results to a certain degree, but we believe this bias is toward what most people are interested in. In Table VI, we show how many sites in our list are from which domain. In our site list, 132 sites belong to `com` and 78 sites to `edu`. The sites ending with ".net"

[6] We can similarly prove the theorem for other synchronization-order policies.

[7] More precisely, we used PageRank as the popularity measure, which is similar to the link count. To learn more about PageRank, please refer to [Page and Brin 1998; Cho et al. 1998].

domain	number of sites
com	132
edu	78
netorg	30 (org: 19, net: 11)
gov	30 (gov: 28, mil: 2)

Table VI. Number of sites within a domain



(a) For the pages that change every 10 days on average

(b) For the pages that change every 20 days on average

Fig. 14. Change intervals of pages

and ".org" are classified as `netorg` and the sites ending with ".gov" and ".mil" as `gov`.

From each of the selected sites, we crawled 3,000 pages every day. That is, starting from the root pages of the selected sites we followed links in a breadth-first search, up to 3,000 pages per site. This "3,000 page window" was decided for practical reasons. In order to minimize the load on a site, we ran the crawler only at night (9PM through 6AM PST), waiting at least 10 seconds between requests to a single site. Within these constraints, we could crawl at most 3,000 pages from a site every day.

7.2 Verification of Poisson process

In this subsection, we verify whether a Poisson process adequately models Web page changes. Similar experimental results have been presented in [Cho and Garcia-Molina 2000; Brewington and Cybenko 2000a].

From Equation 1, we know how long it takes for a page to change under a Poisson process. According to the equation, the time between changes follow an exponential distribution $\lambda e^{-\lambda t}$ if the change frequency of the page is λ. We can use this result to verify our assumption. That is, if we plot the time between changes of a page p, the time should be distributed as $\lambda e^{-\lambda t}$, if changes of p follow a Poisson process.

To compare our experimental data against the Poisson model, we first select only the pages whose *average* change intervals are, say, 10 days, and plot the distribution of their change intervals. In Figure 14, we show some of the graphs plotted this way. Figure 14(a) is the graph for the pages with 10 day change interval, and Figure 14(b) is for the pages with 20 day change interval. The horizontal axis

represents the interval between successive changes, and the vertical axis shows the fraction of changes with that interval. The vertical axis in the graph is logarithmic to emphasize that the distribution is exponential. The lines in the graphs are the predictions by a Poisson process. While there exist small variations, we can clearly see that a Poisson process predicts the observed data very well. We also plotted the same graph for the pages with other change intervals and got similar results when we had sufficient data.

Although our results indicate that a Poisson process describes the Web page changes very well, we note that our results are also limited due to the constraint of our experiment. We crawled Web pages on a daily basis, so our result does not verify the Poisson model for the pages that change very often. Also, the pages that change very slowly were not verified either, because we conducted our experiment for four months and did not detect any changes to those pages. However, given the typical crawling rate of commercial search engines, we believe that the exact change model of these pages are of relatively low importance in practice. For example, Google [Google] tries to refresh their index about once a month on average, so if a page changes more than once a day, exactly when in a day the page changes does not make significant difference in their index freshness. For this reason, we believe the results that we obtain from our dataset is still useful for practical applications despite its limitation.

7.3 Synchronization-order policy

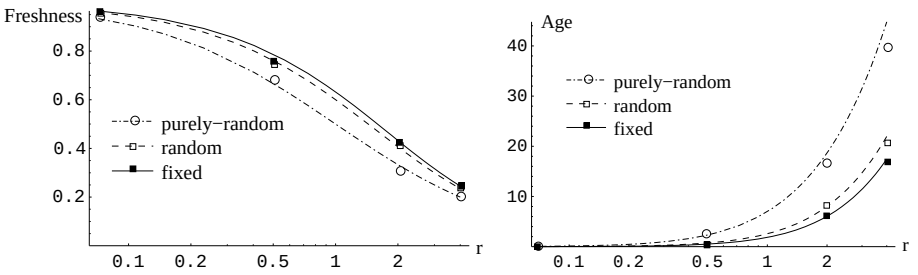
In this subsection, we report our experimental results on synchronization-order policies (Section 4). For the comparison of the synchronization-order policies, we conducted the following experiments: We first selected the set of pages whose average change frequency were once every two weeks.[8]

Then on the set of selected pages, we ran multiple simulated crawls where we downloaded the pages at the average rate of 1) once a day, 2) once every week, 3) once every month and 4) once every two months. Since we knew exactly when each page changed (at the granularity of a day), we could measure the freshness and the age value of every page in each simulated crawl and compute their time averages. In measuring the freshness and age, we assumed that a page changed in the middle of a day when the page changed.

Figure 15 shows the results from these experiments. The horizontal axis represents the frequency ratio $r = \lambda/f$ for a particular simulated crawl. For example, $r = 2$ corresponds to the case when we downloaded pages at the average rate of once every month (once two weeks/once a month). In order to cover a wide range of r values, the horizontal axis is logarithmic. Figure 15(a) shows freshness and Figure 15(b) shows age. The dots in the graph correspond to our experimental results. The lines are the predictions from our theoretical analysis as we showed in Figure 9.[9] From the graph it is clear that the experimental results confirm our theoretical analysis. Most of the dots sit closely to our theoretical graph. Given

[8]We performed the same experiments for the pages whose average change frequencies were other than once every two weeks and observed similar results.

[9]Note that the vertical axis in Figure 15(b) is age, not age/I. In Figure 9(b) it is age/I, so the shapes of the graphs are different.



(a) Freshness metric (b) Age metric

Fig. 15. Experimental results for synchronization-order policies

that the page change interval matches the Poisson model well, this result is not surprising.

Note that the results of Figure 15 have many practical implications. For instance, we can answer all of the following questions using this graph.

—*How can we measure how fresh a local database is?* By measuring how frequently a Web page changes (using a similar monitoring technique that we employed for our experiments), we can estimate how fresh a local database is. For instance, when a Web page changes once a week, and when we download the page also once a week ($r = 1$), the freshness of the local page is 0.63, under the fixed-order policy.

—*How can we guarantee certain freshness of a local database?* From the graph, we can find how frequently we should download a page in order to achieve certain freshness. For instance, if we want at least 0.8 freshness, the frequency ratio r should be less than 0.46 (fixed-order policy). That is, we should re-download the page at least $1/0.46 \approx 2$ times as frequently as it changes.
Therefore, a Web search engine that wants to guarantee a certain level of freshness for their index may want to perform a small-scale change monitoring experiments to estimate how often the indexed pages change. Using this estimate, they can determine how often they will have to download the pages. This technique can be very useful during a planning stage of a Web search engine, when it decides how much network bandwidth it wants to allocate for its crawling activity.

Note we obtained the results in this subsection only for a set of pages with similar change frequency. In Section 7.5, we estimate the expected freshness and age values for the overall Web.

7.4 Frequency of change

Based on the collected data we now study the change frequencies of Web pages. Estimating the change frequencies of pages is essential to evaluate resource-allocation policies.

We can estimate the *average change interval* of a page by dividing our monitoring period by the number of detected changes in a page. For example, if a page changed

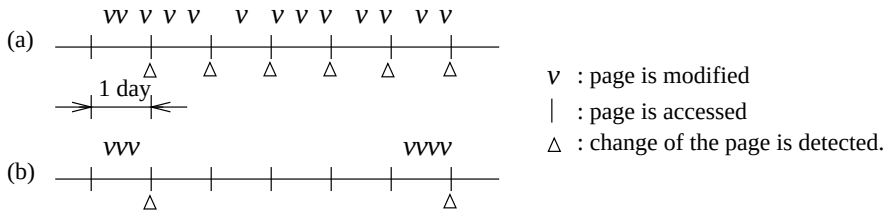
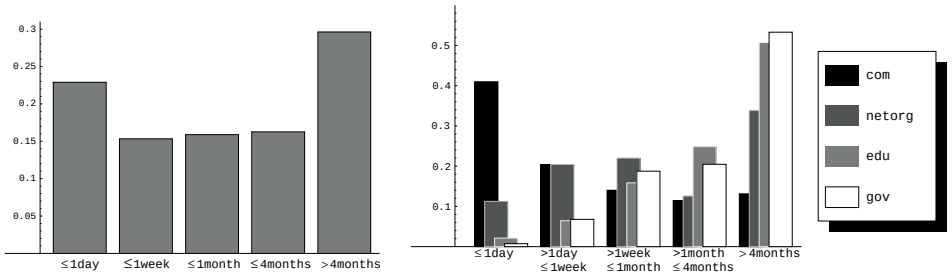


Fig. 16. The cases when the estimated change interval is lower than the real value



(a) Over all domains (b) For each domain

Fig. 17. Fraction of pages with given average interval of change

4 times within the 4-month period, we can estimate the average change interval of the page to be 4 months/4 = 1 month.

In Figure 17 we summarize the result of this estimation. In the figure, the horizontal axis represents the average change interval of pages, and the vertical axis shows the fraction of pages changed at the given average interval. Figure 17(a) shows the statistics collected over all domains, and Figure 17(b) shows the statistics broken down into each domain. For instance, from the second bar of Figure 17(a) we can see that 15% of the pages have a change interval longer than a day and shorter than a week.

From the first bar of Figure 17(a), we can observe that a surprisingly large number of pages change at very high frequencies: More than 20% of pages had changed whenever we visited them. As we can see from Figure 17(b), these frequently updated pages are mainly from the `com` domain. More than 40% of pages in the `com` domain changed every day, while less than 10% of the pages in other domains changed at that frequency (Figure 17(b) first bars). In particular, the pages in `edu` and `gov` domain are very static. More than 50% of pages in those domains did not change at all for 4 months (Figure 17(b) fifth bars). Clearly, pages at commercial sites, maintained by professionals, are updated frequently to provide timely information and attract more users.

While our estimation method gives us a reasonable average change interval for most pages, we also note that our estimation may not be accurate for certain pages. For example, the granularity of the estimated change interval is one day, because we can detect at most one change per day even if the page changes more

	overall		com		gov	
	Freshness	Age	Freshness	Age	Freshness	Age
Proportional	0.12	400 days	0.07	386 days	0.69	19.3 days
Uniform	0.57	5.6 days	0.38	8.5 days	0.82	2.0 days
Optimal	0.62	4.3 days	0.44	7.4 days	0.85	1.3 days

Table VII. Freshness and age prediction based on the real Web data

often (Figure 16(a)). Also, if a page changes several times a day and then remains unchanged, say, for a week (Figure 16(b)), the estimated interval might be much longer than the true value.

7.5 Resource-allocation policy

From our dataset, we now estimate how much freshness and age values we may expect for the overall Web if we use three resource-allocation policies. That is, we assume the change-frequency distribution that we observed in the previous subsection and estimate how well each resource-allocation policy performs under the real distribution. For this evaluation, we assume that we maintain a billion pages locally and that we synchronize all pages every month.[10] Also based on Figure 17(a), we assume that 23% of pages change every day, 15% of pages change every week, etc. For the pages that did not change in 4 months during our experiments, we assume that they change every year. While it is a crude approximation, we believe we can get some idea on how effective different policies are.

In the second and the third columns of Table VII, we show the predicted freshness and age for various resource-allocation policies. To obtain the numbers, we assumed the fixed-order policy (Item 3a in Section 3) as the synchronization-order policy. We can clearly see that the optimal policy is significantly better than any other policies. For instance, freshness increases from 0.12 to 0.62 (500% increase!), if we use the optimal policy instead of the proportional policy. Also, age decreases by 23% from the uniform policy to the optimal policy. From these numbers, we can also learn that we need to be very careful when we optimize the policy based on the frequency of change. For instance, the proportional policy, which people may intuitively prefer, is significantly worse than any other policies: The age of the proportional policy is 93 times worse than that of the optimal policy!

In the remaining columns, we also show the estimated freshness and age if a crawler focuses only on a particular domain of the Web (`com` and `gov`). Again, we assumed the distribution from our experiments (Figure 17(b)) and that the crawler downloaded every page once a month. For example, we assumed that 40% of the pages in the `com` domain changed once a day.

The result shows that our optimal policy is more useful when pages change more often (conversely, when we have less download resources). For example, for the `gov` domain (that has fewer frequently changing pages), the freshness difference is 18% between the proportional and the optimal policy, while the difference becomes 84% for the `com` domain. Similarly, the age difference between the proportional and the optimal policy increases from 1380% to 5110% from the `gov` domain to the `com`

[10]Many popular search engines report numbers similar to these.

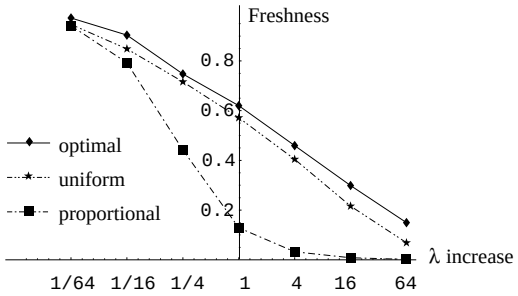


Fig. 18. Estimated freshness for different average change frequencies

domain.

In order to verify whether this trend is generally true, we conducted another set of experiments. In these experiments, we wanted to see how the average freshness value would change if the Web pages changed more (or less) often than the current Web. For this purpose, we artificially increased the average change frequency of all the pages in our dataset by a certain factor and reevaluated the freshness of the pages. We assumed that we still downloaded every page once a month on average. Figure 18 shows the results from these experiments. In the figure, the vertical axis represents the expected freshness values of the three resource-allocation policies. The horizontal axis represents how many times we increased the average change frequency of each page. For example, when $x = 4$, the average change frequency of all pages increased by a factor of 4. Thus, the freshness at $x = 1$ is identical to the second column of Table VII. When $x < 1$, the pages changed less often than our dataset. We performed similar experiments for the age metric and got comparable graphs.

The result shows that the proportional policy performs very poorly when pages change very often. For instance, the proportional policy resulted in almost 0 freshness when the average change frequency increased by a factor of 4 ($x = 4$), while the uniform and the optimal policy gave higher than 0.4 freshness. Also, note that the optimal policy becomes relatively more effective than the uniform policy as λ increases. For example, when λ increased by a factor of 64 ($x = 64$), the optimal policy performed twice as well as the uniform policy. From this result, we believe that our optimal policy will be more useful for the applications that need to monitor frequently changing information, such as daily news update. Also, as the size of the Web grows and as it becomes more difficult to download all pages at the current crawling rate, we believe our new crawling policies can help search engines avoid sudden drops in their index freshness.

8. RELATED WORK

In general, we may consider the problem of this paper as a data replication problem. A lot of work has been done to maintain the consistency of replicated-data [Bernstein and Goodman 1984; Alonso et al. 1990; Bernstein et al. 1980; Barbara and Garcia-Molina 1995; de Carvalho and Roucairol 1982; Colby et al. 1997; Krishnakumar and Bernstein 1991; 1994; Ladin et al. 1992; Golding and Long 1993; Pu and

Leff 1991; Olston and Widom 2000]. This body of work studies the tradeoff between consistency and read/write performance [Krishnakumar and Bernstein 1994; Ladin et al. 1992; Alonso et al. 1990; Olston and Widom 2000] and tries to guarantee a certain type of consistency [Alonso et al. 1990; Barbara and Garcia-Molina 1995; Bernstein et al. 1980]. For example, reference [Yu and Vahdat 2000] tries to limit the number of pending writes that have not been propagated to all replicas. It also proposes a new distributed protocol that can guarantee this property. Reference [Olston and Widom 2000] guarantees a "divergence bound" on the difference between the values of the replicated data and the source data through the cooperation of sources. In most of the existing work, however, researchers have assumed a *push* model, where the sources *notify* the replicated data sites of the updates. In the Web context, this push model is not very appropriate, because most of the Web site managers do not inform others of the changes they made. We need to assume a *pull* model where updates are made *independently* and *autonomously* at the sources.

Reference [Coffman, Jr. et al. 1998] studies how to schedule a Web crawler to improve freshness. The model used for Web pages is similar to the one used in this paper; however, the model for the Web crawler and freshness are very different. In particular, the reference assumes that the "importance" or the "weight" of a page is proportional to the change frequency of the page. While this assumption makes analysis simple, it also makes it hard to discover the fundamental trends that we identified in this paper. We believe the results of this paper is more general, because we study the impact of the change frequency and the importance of a page separately. We also proposed an *age* metric, which was not studied in the reference. Reference [Edwards et al. 2001] proposes another refresh algorithm based on linear programming. The algorithm shows some promising results, but because algorithm becomes more complex over time, the authors report that the algorithm has to periodically "reset" and "start from scratch;" The algorithm takes (practically) infinite amount of time to finish after a certain number of refreshes. A large volume of literature studies various issues related to Web crawlers [Pinkerton 1994; Menczer et al. 2001; Heydon and Najork 1999; Page and Brin 1998], but most of these studies focus on different issues, such as the crawler architecture [Heydon and Najork 1999; Page and Brin 1998], page selection [Menczer et al. 2001; Cho et al. 1998; Chakrabarti et al. 1999; Diligenti et al. 2000] crawler parallelization [Shkapenyuk and Suel 2002; Cho and Garcia-Molina 2002], etc.

There exist a body of literature that studies the evolution of the Web [Wills and Mikhailov 1999; Wolman et al. 1999; Douglis et al. 1999; Pitkow and Pirolli 1997; Cho and Garcia-Molina 2000] and estimates how often Web pages change. For example, reference [Pitkow and Pirolli 1997] studies the relationship between the "desirability" of a page and its lifespan. Reference [Wills and Mikhailov 1999] presents statistics on Web page changes and the responses from Web servers. In reference [Cho and Garcia-Molina 2003], we propose new ways to estimate the change frequencies of Web pages when we do not know the complete change histories of the pages. References [Brewington and Cybenko 2000a; 2000b] estimate the distribution of change frequencies based on experimental data. The work in this category will be very helpful to design a good refresh policy, because we need to

know how often pages change in order to implement the policies proposed in this paper.

9. CONCLUSION

In this paper we studied how to refresh a local database to improve its "freshness." We formalized the notion of freshness by defining *freshness* and *age*, and we developed a model to describe Web page changes. We then analytically compared various refresh policies under the Poisson model and proposed optimal policies that can maximize freshness or minimize age. Our optimal policies consider how often a page changes and how important the pages are, and make an appropriate refresh decision.

Through our study, we also identified that the proportional-synchronization policy, which might be intuitively appealing, does not work well in practice. Our study showed that we need to be very careful in adjusting the synchronization frequency of a page based on its change frequency. Finally, we investigated the changes of real web pages and validated our analysis based on this experimental data. We also showed that our optimal policies can improve freshness and age very significantly using real Web data.

As more and more digital information becomes available, it will be increasingly important to collect it effectively. A crawler or a data warehouse simply cannot refresh all its data constantly, so it must be very careful in deciding what data to poll and check for freshness. The policies we have studied in this paper can make a significant difference in the "temporal quality" of the data that is collected.

REFERENCES

Alonso, R., Barbara, D., and Garcia-Molina, H. 1990. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems 15,* 3 (September), 359–384.

Barbara, D. and Garcia-Molina, H. 1995. The demarcation protocol: A technique for maintaining linear arithmetic constraints in distributed database systems. In *Proceedings of the Internationl Conference on Extending Database Technology*. Vienna, Austria, 373–388.

Bernstein, P., Blaustein, B., and Clarke, E. 1980. Fast maintenance of semantic integrity assertions using redundant aggregate data. In *Proceedings of the Sixth International Conference on Very Large Databases*. Montreal, Canada, 126–136.

Bernstein, P. and Goodman, N. 1984. The failure and recovery problem for replicated distributed databases. *ACM Transactions on Database Systems 9,* 4 (December), 596–615.

Brewington, B. E. and Cybenko, G. 2000a. How dynamic is the web. In *Proceedings of the Ninth International World-Wide Web Conference*.

Brewington, B. E. and Cybenko, G. 2000b. Keeping up with the changing web. *IEEE Computer 33,* 5 (May), 52–58.

Chakrabarti, S., van den Berg, M., and Dom, B. 1999. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the Eighth International World-Wide Web Conference*. Toronto, Canada.

Cho, J. 2001. Crawling the web: Discovery and maintenance of a large-scale web data. Ph.D. thesis, Stanford University.

Cho, J. and Garcia-Molina, H. 2000. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*. Cairo, Egypt.

Cho, J. and Garcia-Molina, H. 2002. Parallel crawlers. In *Proceedings of the Eleventh International World-Wide Web Conference*. Honolulu, Hawaii.

Cho, J. and Garcia-Molina, H. 2003. Estimating frequency of change. *ACM Transactions on Internet Technology 3,* 3 (August).

Cho, J., Garcia-Molina, H., and Page, L. 1998. Efficient crawling through URL ordering. In *Proceedings of the Seventh International World-Wide Web Conference*. Brisbane, Australia.

Coffman, Jr., E. G., Liu, Z., and Weber, R. R. 1998. Optimal robot scheduling for web search engines. *Journal of Scheduling 1,* 1 (June), 15–29.

Colby, L. S., Kawaguchi, A., Lieuwen, D. F., and l Singh Mumick, I. 1997. Supporting multiple view maintenance policies. In *Proceedings of the International Conference on Management of Data*. Tuscon, Arizona, 405–416.

de Carvalho, O. S. F. and Roucairol, G. 1982. On the distribution of an assertion. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. Ottawa, Canada, 121–131.

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. 2000. Focused crawling using context graphs. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*. Cairo, Egypt, 527–534.

Douglis, F., Feldmann, A., and Krishnamurthy, B. 1999. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of the Second USENIX Symposium on Internetworking Technologies and Systems*. Boulder, Colorado.

Edwards, J., McCurley, K., and Tomlin, J. 2001. An adaptive model for optimizing performance of an incremental web c rawler. In *Proceedings of the Tenth International World-Wide Web Conference*.

Golding, R. A. and Long, D. D. 1993. Modeling replica divergence in a weak-consistency protocol for global-sc ale distributed data bases. Technical report UCSC-CRL-93-09, Computer and Information Sciences Board, University of California, Santa Cruz.

Google. Google Inc. `http://www.google.com`.

Heydon, A. and Najork, M. 1999. Mercator: A scalable, extensible web crawler. In *Proceedings of the Eighth International World-Wide Web Conference*. Toronto, Canada, 219–229.

Krishnakumar, N. and Bernstein, A. 1991. Bounded ignorance in replicated systems. In *Proceedings of the 10th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Denver, Colorado, 63–74.

Krishnakumar, N. and Bernstein, A. 1994. Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM Transactions on Database Systems 19,* 4 (December), 586–625.

Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. 1992. Providing high availability using lazy replication. *ACM Transactions on Computer Systems 10,* 4 (November), 360–391.

Lawrence, S. and Giles, C. L. 1998. Searching the World Wide Web. *Science 280,* 5360 (April), 98–100.

Lawrence, S. and Giles, C. L. 1999. Accessibility of information on the web. *Nature 400,* 6740 (July), 107–109.

Menczer, F., Pant, G., and Ruiz, M. E. 2001. Evaluating topic-driven web crawlers. In *Proceedings of the Twenty-Fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New Orleans, LA.

Olston, C. and Widom, J. 2000. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*. Cairo, Egypt.

Page, L. and Brin, S. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World-Wide Web Conference*. Brisbane, Australia.

Pinkerton, B. 1994. Finding what people want: Experiences with the web crawler. In *Proceedings of the Second World-Wide Web Conference*. Chicago, Illinois.

Pitkow, J. and Pirolli, P. 1997. Life, death, and lawfulness on the electronic frontier. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'97*. Atlanta, Georgia, 383–390.

Pu, C. and Leff, A. 1991. Replica control in distributed systems: An asynchronous approach. In *Proceedings of the International Conference on Management of Data*. Denver, Colorado, 377–386.

Shkapenyuk, V. and Suel, T. 2002. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Engineering*. San Jose, California.

Taylor, H. M. and Karlin, S. 1998. *An Introduction To Stochastic Modeling*, 3rd ed. Academic Press.

Thomas, Jr., G. B. 1969. *Calculus and analytic geometry*, 4th ed. Addison-Wesley.

Wills, C. E. and Mikhailov, M. 1999. Towards a better understanding of web resources and server responses for improved caching. In *Proceedings of the Eighth International World-Wide Web Conference*. Toronto, Canada.

Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. M. 1999. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*. 16–31.

Yu, H. and Vahdat, A. 2000. Efficient numerical error bounding for replicated network services. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*. Cairo, Egypt, 123–133.