WIC: A General-Purpose Algorithm for Monitoring Web Information Sources

Sandeep Pandey, Kedar Dhamdhere*, Christopher Olston

Carnegie Mellon University
Pittsburgh, PA 15213
{spandey, kedar, olston}@cs.cmu.edu

Abstract

The Web is becoming a universal information dissemination medium, due to a number of factors including its support for content dynamicity. A growing number of Web information providers post near real-time updates in domains such as auctions, stock markets, bulletin boards, news, weather, roadway conditions, sports scores, etc. External parties often wish to capture this information for a wide variety of purposes ranging from online data mining to automated synthesis of information from multiple sources. There has been a great deal of work on the design of systems that can process streams of data from Web sources, but little attention has been paid to how to produce these data streams, given that Web pages generally require "pull-based" access.

In this paper we introduce a new general-purpose algorithm for monitoring Web information sources, effectively converting pull-based sources into push-based ones. Our algorithm can be used in conjunction with continuous query systems that assume information is fed into the query engine in a push-based fashion. Ideally, a Web monitoring algorithm for this purpose should achieve two objectives: (1) timeliness and (2) completeness of information captured. However, we demonstrate both analytically and empirically using real-world data that these objectives are fundamentally at odds. When resources available for Web monitoring are limited, and the number of sources to monitor is large, it may be necessary to sacrifice some timeliness to achieve better completeness, or vice versa. To take this fact into account, our algorithm is highly parameterized and targets an application-specified balance between timeliness and completeness. In this paper we formalize the problem of optimizing for a flexible combination of timeliness and completeness, and prove that our parameterized algorithm is a 2-approximation in all cases, and in certain cases is optimal.

* Supported by NSF ITR grants CCR-0085982 and CCR-0122581.

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

1 Introduction

The Web is becoming a universal medium for disseminating information of all kinds, including highly dynamic information. A significant amount of valuable dynamic information is being posted to the Web, and people want to access it. In many situations, direct manual viewing of dynamic Web pages is not an adequate mode of access for one or both of the following two reasons. First, most information posted on the Web is not made available forever, and may disappear or be replaced by new information at any time [5]. This aspect presents a challenge, especially for applications in which historical information is of interest. Second, many applications require automated synthesis of information from multiple dynamic Web sources [10].

As a result, there is significant interest in systems that monitor and process updates to frequently updated Web pages automatically. These systems perform a variety of information management functions including synthesis, archiving, and continuous query processing. Web-based continuous query (CQ) processing systems proposed in the literature include CONQUER [9], Niagara [10], OpenCQ [7] and WebCQ [8].

The main focus of this work has been on language design and efficient query processing, and the crucial issue of how to capture information from dynamically changing Web pages has largely been ignored. Most work on continuous query processing assumes that data is "pushed" into the query engine in the form of *data streams*. However, generally speaking, Web data must be "pulled," *i.e.*, continuous query systems must explicitly download Web pages, check for changes, and submit any resulting new data to the query processor.

So far, only heuristics with no formal guarantees on effectiveness have been proposed for converting pull-

oriented Web sources into push-oriented data streams. The designers of Niagara [10] and other CQ systems for the Web have suggested that simple periodic polling be used for this purpose. However, periodic polling breaks down in the presence of a large number of frequently-updated Web sources, when resources become inadequate for polling all Web pages at a fast rate. The work in [13] was the first to study ways to improve upon simple periodic polling, but the algorithm proposed in [13], called CAM, has a number of serious drawbacks: (1) CAM is only suitable for a narrow range of applications in which timeliness of information captured is of utmost importance, whereas many real-life applications must balance timeliness with completeness, a serious issue we discuss shortly, (2) even for the specialized set of applications handled by CAM, there is no formal guarantee that CAM performs well at capturing information, and (3) CAM relies on a computationally intensive, offline algorithm to schedule monitoring.

In this paper we introduce a new general-purpose Web monitoring algorithm called the *Web Information Collector* (WIC), which is suitable for use in conjunction with a variety of CQ systems. WIC has the following desirable properties: (1) it handles a wide range of application scenarios, (2) it provably performs within a factor of two of the optimal offline Web monitoring algorithm in all cases, and (3) it is highly efficient and executes in an online fashion, making it practical for real-world use.

1.1 Web Monitoring Objectives

In continuous Web monitoring applications, it is usually desirable to capture as much information as possible, with as little delay as possible. Dynamic Web pages undergo updates over time, and each updated version of the page potentially contains new information of value to the application. Therefore, an ideal Web monitoring system would capture every change to each page of interest, immediately following the update that causes the change.

Unfortunately, this ideal situation is often difficult to achieve in practice. Due to the nature of Web protocols, obtaining updates to Web pages generally requires polling those pages. For applications that monitor a large number of Web pages, high-frequency polling and processing of all pages of interest can be prohibitively expensive. Typically it is not feasible or desirable to provision systems with adequate communication bandwidth and processing power to support exhaustive and rapid polling of a large number of Web pages.[1]

As a result, polling must be performed selectively, and some criteria for deciding when to poll each page must be established. Although it may not be possible to capture all changes to all pages of interest in a timely fashion due to resource limitations, it is generally desirable to come as close as possible to that ideal. Hence, scheduling polling of pages for Web monitoring can be viewed as a constrained optimization problem with two objectives:

1. **Completeness:** Maximize the number of changes captured.
2. **Timeliness:** Minimize the delay in capturing changes.

In certain cases these two objectives are at odds with each other. We illustrate this property through a very simple example. Suppose we wish to capture changes to a single Web page that is updated in such a way that new information is always appended (thus, information is never removed from the page). Given a single opportunity to download a snapshot of this page, we are clearly faced with a tradeoff between timeliness and completeness. If we download the page early, we will only capture a few changes, but with little delay. Conversely, if we wait a long time before downloading the page, we may capture more changes, but there will be a high delay, on average, between the time at which a change occurs and the time at which it is captured.

When facing resource-constrained situations, the appropriate balance to strike between timeliness and completeness depends on the application. Applications that need to react rapidly to new information, such as stock market day-trading programs, may value timeliness over completeness when both cannot be had. On the other hand, applications whose purpose is to compile historical archives for offline querying may opt for completeness.

To accommodate a diverse variety of applications with differing requirements, we introduce a flexible method of specifying the relative importance of timeliness and completeness: application designers supply a function $urgency : \mathcal{Z}^+ \rightarrow [0, 1]$ specifying the *utility* of a captured change as a function of the delay between the occurrence of the change and the time of capture. Example urgency functions are illustrated in Figure 1. For applications in which timeliness is critical, an urgency function with steep downward slope, such as function *a* in Figure 1, should be used. Conversely, for applications that value completeness over timeliness, a more gradually decreasing function such as function *b* is more appropriate.

In this paper we formalize our notions of timeliness, completeness, and utility, and provide both analytical and empirical evidence of the existence of a tradeoff between

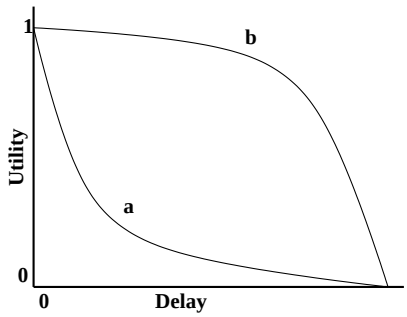

Figure 1: Urgency functions.

[1]While processing of unchanged pages may be avoided using fast checksum comparisons or by outfitting HTTP requests with an "if-modified-since" qualifier, such techniques are usually ineffective in the presence of frequently changing superficial content such as advertisements, counters, etc.

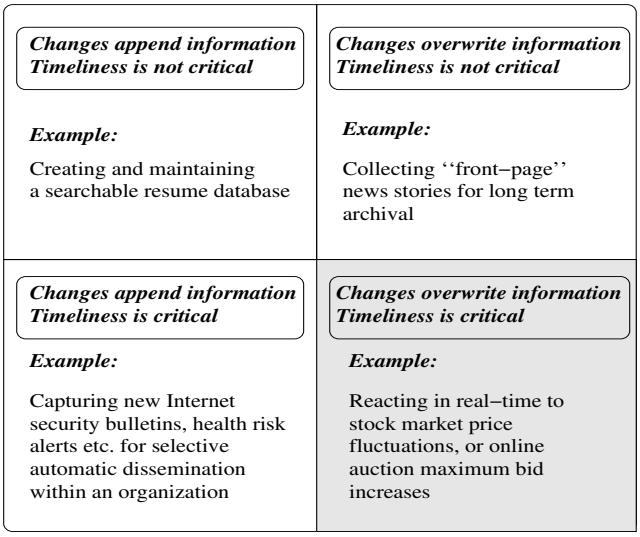

Figure 2: Extreme scenarios.

timeliness and completeness. We formulate the problem of scheduling polling of remote Web pages as an optimization problem whose objective is to maximize utility under the constraint of limited resources available for polling. Our formulation is parameterized by an application-dictated urgency function, allowing the solution to be customized to the needs of specific applications. Of course, the appropriate polling schedule also depends heavily on the way in which the Web pages being monitored change over time, discussed next.

1.2 Modeling Changes to Web Pages

We define a *change* to a Web page as an update that causes information of value to the application to be added to the page. The information added during a change to a page may not remain on the page forever. For example, typical financial reporting sites only display the most recent news reports. Similarly, some online auction sites only show the most recent bids. To model this fact, we assume a certain *lifetime* of information, which may vary among pages and by application. Lifetime indicates the probability that information made available by a change at time t is removed at any future time $t + x$.

It is instructive to consider the two extreme possibilities for lifetime. First, on long-lived, append-only pages, the lifetime of information is essentially infinite. In the opposite extreme, some pages are updated such that each change overwrites the information presented by the previous change completely. In the case of complete overwrites, the lifetime of information made available due to a given change extends only to the time of the subsequent change.

Figure 2 provides examples of application scenarios that can roughly be categorized into each of these two extremes in terms of information lifetime. The applications are also categorized based on the orthogonal dimension of whether timeliness is critical (see Section 1.1), resulting in four extreme categories overall. The flexibility of our formulation makes it suitable for a large variety of Web monitoring applications. The only previous approach we are aware of, CAM [13], only handles applications that fit into the shaded region of Figure 2.

It is important to note that our classification of applications into quadrants in Figure 2 is very rough, and variants of these applications may fit more or less well into their assigned category (which represent extremes in terms of information lifetime and criticality of delay). Our approach accommodates this fact, and can handle applications falling in between these extreme categories since our urgency and lifetime specifications are highly adjustable, as we shall see later in the paper.

1.3 Contributions

The specific contributions of this paper are as follows:

- We formalize the scheduling problem in Web monitoring as a parameterized optimization problem.
- We demonstrate that there exists a fundamental tradeoff between timeliness and completeness, which makes our parameterized formulation necessary.
- We present an efficient, online Web monitoring algorithm that meets the needs of all applications encompassed by Figure 2.
- We prove that our algorithm is a 2-approximation for all cases, and is optimal for the shaded region of Figure 2.

1.4 Outline

The remainder of this paper is organized as follows. In Section 2 we formalize the scheduling problem in Web monitoring as a parameterized optimization problem. We present an efficient, online algorithm and prove that it is a 2-approximation for our Web monitoring scheduling problem in Section 3. Then, in Section 4 we show analytically that, when resources are limited, a fundamental tradeoff exists between timeliness and completeness. In Section 5 we report the results of extensive experiments on real-world data. We confirm that a tradeoff does exist between timeliness and completeness, and that our urgency parameter enables application designers to control that tradeoff.

2 Monitoring the Web: Models and Assumptions

Our models for the Web monitoring scheduling problem and the way in which Web pages change extend the framework introduced in [13, 14]. Let $\mathcal{P}$ be the set of Web pages under consideration for monitoring. Each page $P_i \in \mathcal{P}$ has an associated *importance weight* $W_i \in [0, 1]$, denoting the relative importance of capturing changes to P_i. Time is divided into discrete time instants, and monitoring is performed in epochs of N consecutive time instants. $\mathcal{T}$ denotes the sequence of time instants $T_1, T_2, \ldots, T_N$ in an epoch.

We focus on the problem of scheduling monitoring of the pages in $\mathcal{P}$ during a single epoch. Monitoring a page includes the duties of fetching the page from its remote source, determining whether it has undergone one or more changes of interest and, if so, processing the change(s) and propagating them to the target application. We assume the cost of monitoring a page to be uniform across all pages and across time. This simplification is based on the assumption that the fixed overhead for the operations required (*i.e.*, polling, downloading, and processing a page) is the dominant factor, which is consistent with the assumption made in most work on Web crawling, *e.g.*, [2, 14].

Let C denote the maximum number of pages that can be monitored in a single time instant. The value of C depends on the availability of resources for monitoring, including CPU cycles, communication bandwidth, etc. If C equals or exceeds the number of pages, $|\mathcal{P}|$, then the scheduling problem is trivial: simply monitor each page at every instant. In practice, however, we expect that C may be much less than $|\mathcal{P}|$, making careful scheduling a requirement. A legal *monitoring schedule* for an epoch is one that performs at most C monitorings of pages during each time instant $T_1, T_2, \ldots, T_N$. A monitoring schedule $S = \{s_{1,1}, s_{1,2} \ldots s_{1,N}, s_{2,1}, s_{2,2} \ldots s_{|\mathcal{P}|,N}\}$ consists of a set of Boolean variables $s_{i,j} \in \{0, 1\}$, where $s_{i,j} = 1$ iff page $P_i \in \mathcal{P}$ is scheduled to be monitored at time instant T_j, and $s_{i,j} = 0$ otherwise.

For convenience, a summary of the symbols used in this paper is provided in Table 1. Some of these symbols are not introduced until later in the paper, and should be ignored for now.

2.1 Nature of Changes

A *change* to a page is defined to be an update that causes information of value to the application to be added to the page. We assume that the information presented with each change to a particular page carries equal value, or importance.[2] However, we do not assume that all pages are equally valuable to monitor. For example, in financial monitoring applications, pages providing periodic earnings reports may be of significantly higher importance for certain purposes than those displaying stock prices, even though stock prices are typically updated much more frequently. To model this fact we allow custom importance weights to be associated with each page, as stated above.

As discussed in Section 1.2, we model information posted to Web pages as having an associated lifetime. Let $life_i(j, k)$ denote the probability that information made available by a change at time T_j to page P_i remains available at time T_k. (It is assumed that $1 \leq j \leq k \leq N$.) We assume that life is a monotonically nonincreasing function of $k - j$. Our life function can be used to model a variety of common Web page update patterns ranging from ones in which changes strictly append information to ones in which all changes overwrite information supplied by previous changes, and covering situations in between. We supply some examples later in Section 2.3.

[2]While our model can be extended to enable differentiation among changes in terms of importance, we believe that even with this restriction it is adequate to capture the basic properties of most applications.

Notation	*Definition*
$\mathcal{P}$	Set of pages that are considered for monitoring. Variable i is used for iterating over this set.
$\mathcal{T}$	Sequence of time instants $\{T_1, T_2, \ldots, T_N\}$ in an epoch. Variables j,k,q and z are used for iterating over this set.
C	Maximum number of monitorings allowed in each time instant.
$urgency_i$	A function to model the value of information of page P_i as a function of timeliness. $urgency(0)$ is always assumed to equal 1.
$life_i$	A function to model the decay of existence of information on page P_i with time.
$\pi_{i,j}$	The estimated probability that the page P_i is updated at time instant T_j.
$s^x_{i,j}$	A decision variable, set to 1 if page P_i is monitored at time instant T_j by algorithm x (o for OPTIMAL algorithm and g for WIC algorithm), and 0 otherwise. For notational convenience, we define $s^x_{i,0} = 0$.
S^x	A monitoring schedule that consists of a set of Boolean variables $s^x_{i,j}$, where $s^x_{i,j} = 1$ iff page $P_i \in \mathcal{P}$ is scheduled to be monitored at time T_j by algorithm $x \in \{o, g\}$, and $s^x_{i,j} = 0$ otherwise.
$prev^x_{i,j}$	The most recent time instant before T_j at which page P_i was monitored by algorithm $x \in \{o, g\}$. $prev^x_{i,j}$ is 0 if the page is never monitored before time T_j by algorithm x. Mathematically, $prev^x_{i,j} = max\{j' : 1 \leq j' < j \wedge s^x_{i,j'} = 1\}$.
$seq^x_i(j, k)$	The sequence of time instants in the open interval (j, k) at which page P_i is monitored by algorithm x. Mathematically, $seq^x_i(j, k) = \{j' : j < j' < k \wedge s^x_{i,j'} = 1\}$.
seq^x_i	For convenience, we refer to $seq^x_i(0, N + 1)$ as seq^x_i. We use $seq^x_i(k)$ to refer to the k^{th} element of the sequence. $seq^x_i(0)$ is defined to be 0. Also, $seq^x = \cup_{i \in \mathcal{P}}(seq^x_i)$.

Table 1: Summary of symbols and their meanings.

We assume that each page $P_i \in \mathcal{P}$ has an associated probability of change $\pi_{i,j} \in [0, 1]$ at each time instant $T_j \in \mathcal{T}$ that has been estimated in advance. This so-called *quasi-deterministic* model of change probability has been shown to be appropriate for modeling frequently updated Web pages [11, 14]. The problem of assigning probabilities of change to Web pages is beyond the scope of this paper. We demonstrate in Section 5.2 that our approach is tolerant of a moderate degree of inaccuracy in the estimated change probabilities.

2.2 Web Monitoring Objective

Given a life parameter $life_i()$ and change probabilities $\pi_{i,*}$ for each page $P_i \in \mathcal{P}$, we can compute the expected number of changes captured by monitoring P_i at a particular time instant T_j. For any prior time instant T_k, $1 \leq k \leq j$, the probability that a change occurred at time T_k and can still be captured at time T_j is $\pi_{i,k} \cdot life_i(k,j)$. Suppose for the moment that the first monitoring of P_i during the epoch occurs at time T_j. Summing over all time instants in the epoch up to T_j we obtain:

$$\sum_{k=1}^{j} \left(\pi_{i,k} \cdot life_i(k,j) \right)$$

Now, consider a monitoring schedule S, which consists of a set of Boolean variables $s_{i,j}$ giving monitoring times for each of the pages in $\mathcal{P}$ during the epoch. The expected total number of changes captured by S from all pages during the epoch is given by:

$$\sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} \left(s_{i,j} \cdot \sum_{k=prev_{i,j}+1}^{j} \pi_{i,k} \cdot life_i(k,j) \right)$$

where $prev_{i,j}$ denotes the index of the most recent time instant prior to T_j at which page P_i was monitored.

Not all captured changes may be of equal value to an application. Instead, each page P_i has an associated importance weight $W_i \in [0,1]$. Furthermore, recall from Section 1.1 that an application-specific *urgency* function $urgency_i : \mathcal{Z}^+ \rightarrow [0,1]$ may be associated with each page P_i. Together, W_i and P_i specify the *utility* of a captured change in P_i as a function of the delay between the occurrence of the change and the time of capture. In particular, if a change in P_i occurs at time T_k, and is captured later at time T_j, $j > k$, then the utility of capturing that change is $W_i \cdot urgency_i(j-k)$. Note that $W_i \cdot urgency_i(j-k) \leq W_i \leq 1$. If a change is captured during the same instant in which it occurred, *i.e.*, at time T_k, then the utility of capturing the change is $W_i \cdot urgency_i(0)$. We require that $urgency_i(0) = 1$, so the utility of capturing a change to P_i immediately is W_i.

The expected total utility U accrued by executing monitoring schedule S in the epoch is given by:

$$U = \sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} \left(s_{i,j} \sum_{k=prev_{i,j}+1}^{j} W_i \cdot urgency_i(j-k) \cdot \pi_{i,k} \cdot life_i(k,j) \right)$$

The objective when selecting a monitoring schedule is to maximize U.

2.3 Life and Urgency Parameters

Note that life and urgency are tuning parameters in the above objective function. Life is used to model different Web page change behaviors, while urgency can be tuned according to application requirements in terms of timeliness and completeness. Below are some examples of how life and urgency can be set in order to model various data and application scenarios that arise in practice.

- $life(k,j)$, for $k \leq j$:
  1. **Unbounded-Append:** All changes are of an append-only nature and information is never deleted:
  $$life_i(k,j) = 1.$$
  2. **Time-window–Append(W):** Changes append new information, and old information is removed after W time instants:
  $$life_i(k,j) = \begin{cases} 1 & \text{if } j-k \leq W \\ 0 & \text{otherwise} \end{cases}$$
  3. **Change-window–Append(Z):** Changes append new information, and old information is removed after Z subsequent changes occur. In general, it is difficult to write a concise formula for this scenario. For the special case in which the change probability for page P_i has the same value π_i at all time instants, it can be written as:
  $$life_i(k,j) = \sum_{q=0}^{Z-1} \binom{j-k}{q} \pi_i^q (1-\pi_i)^{N-1-q}$$
  4. **Overwrite:** Each change completely obliterates all information made available by previous changes:
  $$life_i(k,j) = \prod_{q=k+1}^{j} (1-\pi_{i,q})$$
- $urgency(t)$, for $t \geq 0$:
  1. **Uniform:** Utility is independent of delay:
  $$urgency(t) = 1$$
  2. **Exponential Decay(r):** For a decay parameter $r \in [0,1]$:
  $$urgency(t) = r^t$$
  3. **Sliding Window(W):** For a window size parameter $W \geq 0$:
  $$urgency(t) = \begin{cases} 1 & \text{if } t \leq W \\ 0 & \text{otherwise} \end{cases}$$

  Other decay functions may also be used to specify urgency, such as polynomial decay, polyexponential decay, and chordal decay (see [3]).

Figure 3 shows how our life and urgency parameters should be set in order to model the extreme data and application scenarios represented in Figure 2.

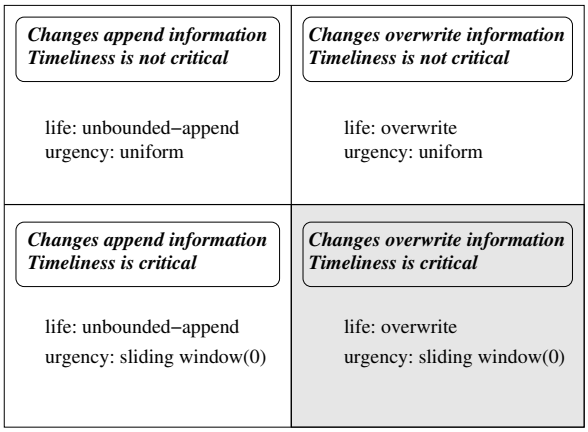

Figure 3: Life and urgency under various scenarios.

Relaxed versions of these extreme life and urgency settings, such as the windowed and exponentially decaying functions outlined above, can be used to accommodate applications falling in between these extreme scenarios. For example, some online auction sites display a sliding window of recent bids for each item, which can be modeled using Time-window–Append or Change-window–Append for $life$. Auction monitoring applications needing access to bid histories up to the last hour may specify urgency as Sliding-window(1 hour).

We note that the RIR objective presented in [13] assumes that any nonzero delay in capturing changes is unacceptable and that changes in the target pages are fully overwritten, which corresponds to the shaded region of Figure 3. Hence, by setting life and urgency to Overwrite and Sliding-window(0), respectively, our utility objective is equivalent to RIR as a special case.

3 General-Purpose Web Monitoring Algorithm

In selecting a monitoring schedule S we are faced with the optimization problem of choosing values for the boolean variables $s_{i,j}$, $P_i \in \mathcal{P}$, $T_j \in \mathcal{T}$ so that the total utility U is maximized, given a constraint C on the number of monitorings allowed at each time instant. To simplify exposition in this and subsequent sections, we define a function $g_i : \mathcal{Z}^+ \times \mathcal{Z}^+ \rightarrow \mathbb{R}$ for page P_i as:

$$g_i(j_1, j_2) = W_i \sum_{k=j_1+1}^{j_2} \left(urgency_i(j_2 - k) \cdot \pi_{i,k} \cdot life_i(k, j_2) \right)$$

This quantity represents the utility accrued by monitoring page P_i at time T_{j_2}, assuming that the most recent monitoring of P_i occurred at T_{j_1}, $j_1 < j_2$.

We can express our optimization problem in terms of g_i as follows:

$$\text{maximize} \quad \sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s_{i,j} \cdot g_i(prev_{i,j}, j)$$

subject to the resource constraint:

$$\forall j, \quad \sum_{P_i \in \mathcal{P}} s_{i,j} \leq C$$

where $s_{i,j} \in \{0, 1\}$.

3.1 Optimal Offline Algorithm

This problem can be formulated as a nonserial constrained optimization problem, and solved using nonserial dynamic programming [1]. The running time complexity of nonserial dynamic programs depends on the interaction among decision variables [1, 6]. It turns out that in our problem the decision variables (*i.e.*, $s_{i,j}$ variables) are highly intertwined, so the optimal dynamic programming algorithm is likely to be too expensive for large-scale applications.

We address this issue by proposing a greedy algorithm that serves as a 2-approximation and runs in time linear in the number of decision variables, *i.e.*, $O(|\mathcal{P}| \cdot |\mathcal{T}|)$.

3.2 Efficient Online Algorithm

We present the following greedy algorithm for scheduling monitoring of dynamic Web pages, which we call *WIC* for "Web Information Collector":

Algorithm 1 *(WIC):*

1. *For all pages $P_i \in \mathcal{P}$ and time instants $T_j \in \mathcal{T}$:*

   Initialize $s_{i,j} \leftarrow 0$.

2. *For $j = 1$ to N:*

   For each $P_i \in \mathcal{P}$ let $u_i = g_i(prev_{i,j}, j)$.

   Let $\mathcal{L}$ contain the pages P_i with the top C values of u_i.

   For each $P_i \in \mathcal{L}$ set $s_{i,j} = 1$.

   For each P_i,

   if $P_i \in \mathcal{L}$,

   set $prev_{i,j+1} = j$

   else,

   set $prev_{i,j+1} = prev_{i,j}$

Recall that $g_i(prev_{i,j}, j)$ denotes the utility accrued by monitoring page P_i on instant T_j. Since at each instant the above algorithm monitors those pages which offer maximum current utility, it operates in a greedy manner. WIC maximizes utility locally, at each time instant, but does not necessarily maximize overall utility accrued during the entire epoch.

We now examine the running-time complexity of WIC, which depends on the nature of the life and urgency parameters. For all example settings of life and urgency outlined in Section 2.3 except for life = Change-window–Append, the value of $g_i(prev_{i,j}, j)$ can be computed in constant time from its value in the previous time instant,

$g_i(prev_{i,j-1}, j-1)$, and $\pi_{i,j}$. In those cases the running time of WIC is linear in the number of decision variables, *i.e.*, $O(|\mathcal{P}| \cdot |\mathcal{T}|)$.

WIC can be executed in an *online* fashion, meaning that the values of decision variables $s_{*,j}$ are assigned immediately prior to time T_j. Therefore, it is compatible with algorithms for estimating change probabilities at the "last minute," *i.e.*, ones that assign change probability estimates $\pi_{*,j}$ as late as time T_{j-1}. When executed in an online fashion, WIC requires only $O(|\mathcal{P}|)$ computations per time instant.

3.3 WIC is a 2-Approximation

We show that for monotonic urgency functions WIC is a 2-approximation algorithm for the optimization problem formulated above. Let S^g denote the schedule selected by WIC and S^o denote an optimal schedule. Our claim is that the expected total utility accrued by S^g is not less than half of that accrued by S^o. Mathematically, our claim is:

$$\sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^o_{i,j} \cdot g_i(prev^o_{i,j}, j) \quad \leq \quad (1+a) \cdot \sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^g_{i,j} \cdot g_i(prev^g_{i,j}, j)$$

where the superscripts "g" and "o" denote aspects of S^g and S^o, respectively, and a depends on urgency as follows:

$$a = \max_i \max_t \left(\frac{urgency_i(t+1)}{urgency_i(t)} \right)$$

If *urgency* is a monotonically nonincreasing function, $0 \leq a \leq 1$ and this inequality implies that WIC is a 2-approximation.

Our complete formal proof is rather involved, and is given in the extended technical report version of this paper [12]. Here we present the main idea behind our proof, focusing on the special case of $C = 1$ for simplicity.

We begin by stating a simple property of WIC that follows from its construction:

$$s^g_{i,j} = 1 \implies \forall P_{i'} \in \mathcal{P},\ g_i(prev^g_{i,j}, j) \geq g_{i'}(prev^g_{i',j}, j) \quad (1)$$

Suppose that at a certain time instant T_j, in the schedule selected by WIC , S^g, page $P_{i'}$ is monitored, and in S^o page $P_{i''}$ is monitored, where $P_{i'}$ and $P_{i''}$ may be the same or different. Consider two cases:

- *$P_{i''}$ is not monitored in S^g at any time T_k, $1 \leq k \leq j$*: In this case the WIC and optimal schedules for $P_{i''}$ are as follows (* denotes either 0 or 1):

Time Instant	T_1	T_2	$\cdot$	$\cdot$	$\cdot$	T_{j-1}	T_j
$s^o_{i'',j}$	*	*	*	*	*	*	1
$s^g_{i'',j}$	0	0	0	0	0	0	*

  $prev^g_{i'',j} = 0$, so it must be the case that $prev^g_{i'',j} \leq prev^o_{i'',j}$. By Lemma 1 in [12], this fact implies $g_{i''}(prev^o_{i'',j}, j) \leq g_{i''}(prev^g_{i'',j}, j)$. Combining this result with Equation 1, which states that $g_{i''}(prev^g_{i'',j}, j) \leq g_{i'}(prev^g_{i',j}, j)$, we obtain:

$$g_{i''}(prev^o_{i'',j}, j) \leq g_{i'}(prev^g_{i',j}, j)$$

  which means that the utility accrued by S^o at time T_j is not greater than the utility accrued by S^g.

- *$P_{i''}$ is monitored in S^g at some time T_k, $1 \leq k \leq j$*: In this case the WIC and optimal schedules for $P_{i''}$ may look as follows, for example:

Time Instant	T_1	T_2	$\cdot$	$\cdot$	$\cdot$	T_{j-1}	T_j
$s^o_{i'',j}$	*	*	*	*	*	*	1
$s^g_{i'',j}$	0	1	0	1	0	0	*

  Here the inequality $g_{i''}(prev^o_{i'',j}, j) \leq g_{i'}(prev^g_{i',j}, j)$ does not necessarily hold. However, we prove in [12] that the difference $g_{i''}(prev^o_{i'',j}, j) - g_{i'}(prev^g_{i',j}, j)$ is bounded by a times the utility accrued in S^g for page $P_{i''}$ in the time interval $[prev^g_{i'',j}, j]$, *i.e.*:

$$g_{i''}(prev^o_{i'',j}, j) \quad \leq \quad g_{i'}(prev^g_{i',j}, j) + a \cdot \sum_{q \in seq^g_{i''}(prev^g_{i'',j}, j)} g_{i''}(prev^g_{i'',q}, q)$$

  where $seq^g_{i''}(prev^g_{i'',j}, j)$ denotes the set of time instants T_q with $prev^g_{i'',j} \leq q \leq j$ and $s^g_{i'',q} = 1$.

Combining both cases and making two simple transformations, we find that for all pages $P_{i''} \in \mathcal{P}$ and all time instants $T_j \in \mathcal{T}$:

$$s^o_{i'',j} \cdot g_{i''}(prev^o_{i'',j}, j) \quad \leq \quad \sum_{P_{i'} \in \mathcal{P}} s^g_{i',j} \cdot g_{i'}(prev^g_{i',j}, j) + a \cdot \sum_{q \in seq^g_{i''}(prev^g_{i'',j}, j)} g_{i''}(prev^g_{i'',q}, q)$$

By summing over all $P_{i''} \in \mathcal{P}$ and all $T_j \in \mathcal{T}$ and transforming the resulting expression (see [12]) we obtain our desired result:

$$\sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^o_{i,j} \cdot g_i(prev^o_{i,j}, j) \leq (1+a) \cdot \sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^g_{i,j} \cdot g_i(prev^g_{i,j}, j)$$

Corollaries:

(i) For monotonic urgency functions, $a \leq 1$, so

$$\sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^o_{i,j} \cdot g_i(prev^o_{i,j}, j) \leq 2 \sum_{P_i \in \mathcal{P}} \sum_{j=1}^{N} s^g_{i,j} \cdot g_i(prev^g_{i,j}, j)$$

and WIC is a 2-approximation.

(ii) For the Sliding Window(0) setting of urgency (see Section 2.3), $a = 0$ and WIC is guaranteed to produce an optimal schedule.

4 Timeliness-Completeness Tradeoff

In this section we study the tradeoff between timeliness and completeness analytically, and show that this tradeoff can be controlled by adjusting the urgency parameter. (In Section 5 we measure this effect empirically.)

For our analysis we focus on the following simple example scenario for which optimal schedules are easy to find. (Comprehensive analytical study of the nature of the timeliness-completeness tradeoff in a wider context is left as future work.) Suppose that all changes are append-only in nature, *i.e.*, for all pages $P_i \in \mathcal{P}$, $life_i(k, j) = 1$, independent of k and j. Further suppose that for all pages in $\mathcal{P}$ except a special page $P_{i'}$, the probability of change $\pi_{i,j}$ is uniform and equal to some constant π for all time instants T_j, $1 \leq j \leq N$. For page $P_{i'}$, let $\pi_{i',j} = \pi'$ at each time instant T_j, where $\pi' > \pi$. Page $P_{i'}$ is more likely to change than any other page at each time instant.

Furthermore, let the number of time instants in an epoch be much larger than the number of pages under consideration for monitoring, *i.e.*, $N \geq |\mathcal{P}| \geq 2$, and let $W_i = 1$ for all $P_i \in \mathcal{P}$. Finally, assume that at most one page can be monitored at each time instant, *i.e.*, $C = 1$.

Now consider two extreme scenarios for urgency:

- **Timeliness-Only**: *No delay in capturing information is acceptable.* Information not captured immediately is of no value to the application. This is the Sliding Window(0) scenario for urgency described in Section 2.3, in which $urgency(0) = 1$ and $urgency(t) = 0$ for all $t > 0$. In this scenario,

$$g_i(prev_{i,j}, j) = \begin{cases} \pi & \text{if } i \neq i' \\ \pi' & \text{otherwise} \end{cases}$$

  Here, the optimization problem reduces to that of maximizing the number of changes captured with zero delay. The unique optimal schedule in this case is as follows: Monitor page $P_{i'}$ at each time instant, and do not monitor any other pages. In this scenario prior changes have no bearing and overall utility is maximized by always monitoring the page with the highest probability of change in the current time instant.

- **Completeness-Only**: *Any delay in capturing the changes is acceptable.* This is the uniform scenario for urgency described in Section 2.3, in which urgency is set to $urgency(t) = 1$, independent of t. In this scenario,

$$g_i(prev_{i,j}, j) = \begin{cases} \sum_{k=prev_{i,j}+1}^{j} \pi & \text{if } i \neq i' \\ \sum_{k=prev_{i,j}+1}^{j} \pi' & \text{otherwise} \end{cases}$$

  Here, the optimization problem reduces to that of maximizing the total number of changes captured, regardless of delay. One optimal schedule for this scenario is as follows: During time instants T_1, T_2, ..., $T_{N-|\mathcal{P}|}$ monitor page $P_{i'}$ repeatedly. Then, during time instants $T_{N-|\mathcal{P}|+1}$,..., T_{N-1} monitor each page $P_i \in \mathcal{P}$, $i \neq i'$ exactly once in some order. Finally, at the last time instant T_N monitor $P_{i'}$ again.

  We now argue informally that this schedule is optimal in terms of expected total utility accrued. A formal proof is omitted for brevity. Since changes only append information and timeliness has no bearing, only the last monitoring of each page during the epoch is important. Furthermore, the last monitoring of a given page captures the most information if it is as late in the schedule as possible. Therefore, an optimal schedule for this scenario is one that monitors a different page in each of the final $|\mathcal{P}|$ time instants, in ascending order of probability of change. The monitorings scheduled for time instants between T_1 and $T_{N-|\mathcal{P}|}$ are irrelevant in terms of utility.

Below we tabulate the number of changes captured as well as the number captured with zero delay between occurrence and capture:

Urgency setting	*Number of changes captured*	*Number captured with zero delay*
Timeliness-Only	$N \cdot \pi'$	$N \cdot \pi'$
Completeness-Only	$N \cdot \pi' + [(\lvert\mathcal{P}\rvert - 1) \cdot N - \frac{\lvert\mathcal{P}\rvert \cdot (\lvert\mathcal{P}\rvert - 1)}{2}] \cdot \pi$	$\leq N \cdot \pi' - (\lvert\mathcal{P}\rvert - 1) \cdot (\pi' - \pi)$

In the Timeliness-Only case, all changes to $P_{i'}$ are captured, and all are captured in the same time instant in which they occur. The expected number of such changes is $N \cdot \pi'$, which represents the maximum number of changes that can be captured with zero delay under any schedule, on expectation. In the Completeness-Only case, not only are all $N \cdot \pi'$ changes to $P_{i'}$ captured, but most of the changes to the other pages are captured as well. The expected number of such changes is $[(|\mathcal{P}| - 1) \cdot N - |\mathcal{P}| \cdot (|\mathcal{P}| - 1)/2] \cdot \pi$. Overall, $N \cdot \pi' + [(|\mathcal{P}| - 1) \cdot N - |\mathcal{P}| \cdot (|\mathcal{P}| - 1)/2] \cdot \pi$ changes are captured, which represents the maximum number of changes that can be captured under any schedule, on expectation. Since we assume $N > |\mathcal{P}|/2$, the expected total number of changes captured in the Completeness-Only case is greater than in the Timeliness-Only case.

Although more changes are captured in the Completeness-Only case, the delay between the time of occurrence and time of capture of those changes tends to be longer than in the Timeliness-Only case. We quantify the difference by comparing the expected number of changes captured immediately, with zero delay, in the two cases. First observe that the particular choice of optimal schedule given above for the Completeness-Only scenario represents the best case for expected number of changes captured with zero delay. In that best case, the expected total number of zero-delay changes captured is $(N - |\mathcal{P}|) \cdot \pi' + (|\mathcal{P}| - 1) \cdot \pi + \pi' = N \cdot \pi' - (|\mathcal{P}| - 1) \cdot (\pi' - \pi)$. This quantity is less than the number of zero-delay changes captured in the Timeliness-Only case, $N \cdot \pi'$ (recall that $\pi' > \pi$). Hence, more changes are captured immediately

after they occur in the Timeliness-Only case than in the Completeness-Only case, even though the total number of changes captured is fewer.

In this example scenario, if we maximize the expected total number of changes captured, the expected number captured with zero delay is less than maximal. Conversely, if we maximize the expected number captured with zero delay, the total number of changes we expect to capture is less than maximal. Based on our analysis of this example scenario we conclude that the following two facts appear to be true:

1. A fundamental tradeoff exists between timeliness and completeness of information captured during monitoring.

2. Our urgency parameter serves as a knob to control this tradeoff.

Extending our analysis to encompass a broader range of scenarios appears nontrivial and is left as a topic of future work. In the rest of this paper we prefer to focus on empirical measurements.

5 Experiments

To evaluate our WIC algorithm empirically, we used real-world online auction data from a major auction site. Auction bids are of significant interest to monitor for purposes of offline trend analysis as well as real-time counter-bidding.

We obtained 7550 Web pages from the site, each of which contains bidding histories for one item up for auction. Since bids have timestamps, we were able to reconstruct the past temporal behavior of these pages. Each page is updated whenever a new bid is made for the corresponding item, at which time information about the new bid (including bidder, price, time) is appended to the bidding history. However, for the sake of testing the flexibility of our approach, some of our experiments assume that each page displays only the most recent, or maximum, bid for the item, and prior bids are erased. Some auction sites only display the maximum bid.

For our experiments we treated one day as an epoch, with time instants corresponding to one-minute intervals. Hence, $N = 60 \cdot 24 = 1440$ time instants. The number of pages $|\mathcal{P}| = 7550$. We set $W_i = 1$ for all pages $P_i \in \mathcal{P}$. Change probabilities ($\pi_{*,*}$) are determined as follows: we begin with the "exact probabilities," in which each $\pi_{i,j} \in \{0, 1\}$, depending on whether page P_i undergoes a change at time instant T_j. Then, we add noise to simulate inaccuracies introduced by a change probability estimation algorithm in the following two ways:

- **False positives and false negatives**: Given an error factor $FPN \in [0, 1]$, we remove each change with probability FPN, *i.e.*, set $\pi_{i,j} = 0$ when originally $\pi_{i,j} = 1$. Each time a change to page P_i at time instant T_j is removed, we insert a spurious change to P_i at a randomly-selected time instant $T_{j'}$, *i.e.*, set $\pi_{i,j'} = 1$. The smaller FPN is, the more accurate the change probability estimates are.

- **Spread**: For each change to page P_i at time instant T_j, we spread the probability of change according to a Gaussian distribution parameterized by standard deviation $\sigma \geq 0$ (in units of time instants). As with FPN, the smaller σ is, the more accurate the change probability estimates are.

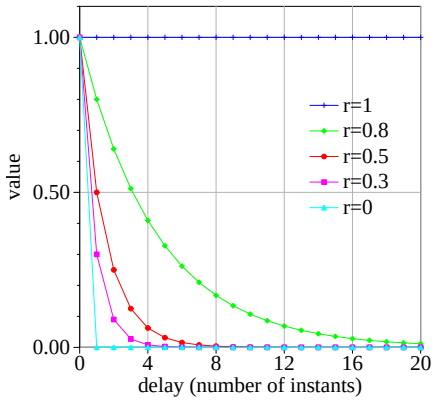

Figure 4: Urgency function for different values of r.

5.1 Metric and Parameters

We evaluate the performance of monitoring scheduling algorithms in terms of total utility accrued. To normalize our measurements in the range $[0, 1]$ we divide total utility by the total number of changes undergone by all pages at all time instants in the epoch. If a scheduling algorithm captures all changes with zero delay, the normalized utility is 1.

Recall that utility is parameterized by an urgency specification. For our experiments we used exponential decay with parameter $r \in [0, 1]$ for urgency. Figure 4 shows urgency functions for different values of r. By tuning r, the desired balance between timeliness and completeness can be specified. Using a small value for r, timeliness is preferred over completeness. In the extreme, setting $r = 0$ signals that changes not captured immediately, *i.e.*, during the same time instant in which they occur, are of no value and do not increase utility. On the other hand, using a large value for r, completeness is preferred over timeliness. In the extreme, setting $r = 1$ signals that timeliness has no bearing and utility depends only on completeness.

5.2 Effect of Inaccuracies in Change Probability Estimation

In our first experiment we investigate how inaccuracies in change probability estimation effect the performance of our WIC algorithm (Section 3.2). First we vary the estimation accuracy by changing the value of σ (standard deviation of spread), while fixing $FPN = 0$. Figure 5 shows the result. In each graph, the x-axis plots the resource constraint

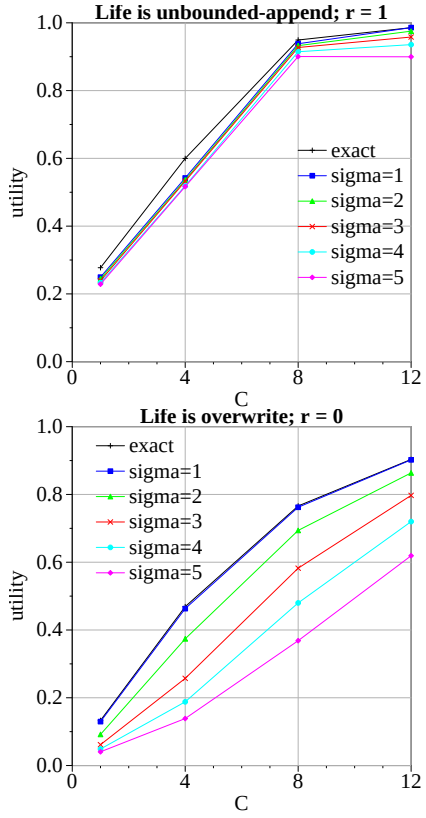

Figure 5: Effect of change probability estimation inaccuracy in terms of spread (σ) on performance.

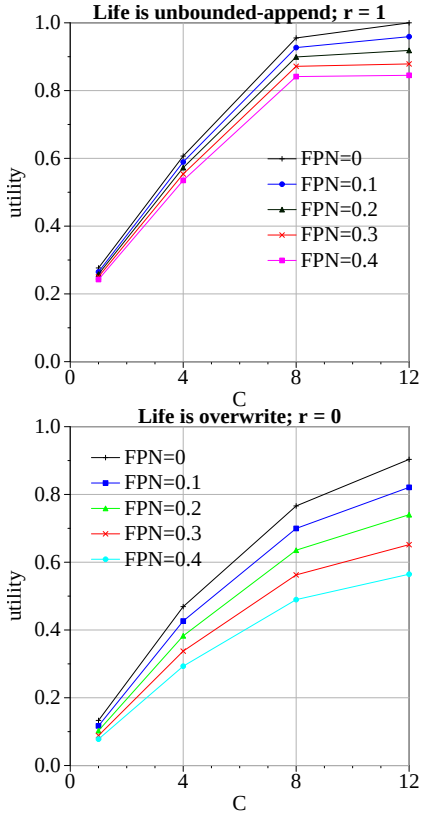

Figure 6: Effect of change probability estimation inaccuracy in terms of false positives and negatives (FPN) on performance.

C (maximum number of pages that can be monitored per time instant), and the y-axis plots utility captured. In both graphs the utility obtained decreases with increasing σ, as would be expected. The two graphs shown correspond to the best and worst cases in terms of loss in utility due to spread in change probability estimation. The graphs for other combinations of life and urgency fall in between these two extremes, so we omit them.

As we can see from Figure 5, our algorithm is fairly tolerant of a modest degree of spread on this data. When $\sigma = 4$, the height of the central peak of the distribution of change probability estimates falls at around 0.1 (it is well below 0.1 for $\sigma = 5$). In other words, with $\sigma = 4$ the estimate only indicates a 10% probability of change at the time instants in which a change does occur, yet our approach still performs reasonably well.

Next we measure the effect of introducing false positives and false negatives by varying FPN, fixing $\sigma = 0$ (no spread). Figure 6 shows the result. As before, we show the two graphs corresponding to the settings of life and urgency yielding the best and worst cases in terms of utility lost due to inaccuracy in change probability estimation. Again, our algorithm appears to be fairly intolerant of a modest degree of inaccuracy in estimation of change probability due to false positives and negatives. Furthermore, both in terms of spread and false positives and negatives, the degree to which estimation inaccuracy undermines the ability to achieve high utility is highly dependent on the life and urgency parameters. We fix $FPN = 0.1$ and $\sigma = 2$ for the rest of our experiments.

5.3 Timeliness-Completeness Tradeoff

In our next experiment we demonstrate the control provided by our urgency parameter in trading off timeliness against completeness. We fix $FPN = 0.1$ and $\sigma = 2$, and life is set to Unbounded-Append for all pages in $\mathcal{P}$ (similar results were obtained with life set to Overwrite).

In Figure 7 we show the number of changes captured by our WIC algorithm, as a fraction of the total number of changes that occurred, under different urgency functions. The resource constraint C is plotted on the $x - axis$. As we expect, in each case as C increases, more changes are captured. The number of changes captured also increases as r (the urgency parameter) increases. (Recall that increasing r increases the relative importance of completeness compared with that of timeliness.) When availability of resources is relatively low ($C \leq 12$ in this case), our algorithm captures between around 20% and 80% more changes when $r = 1$ than when $r = 0$.

We now turn to timeliness. Figure 8 plots the distribution of delay between the times of occurrence and capture of the changes captured, with $C = 8$ and for different

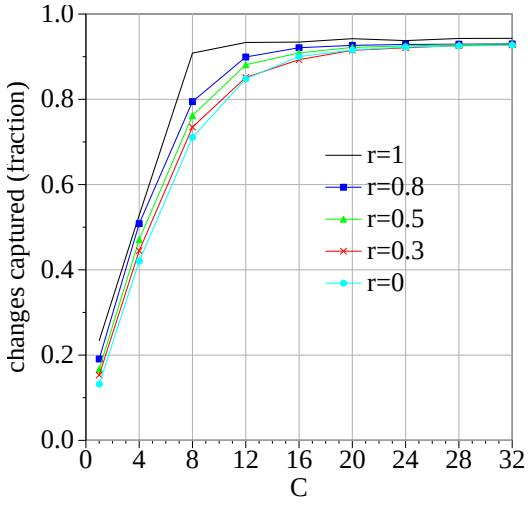

Figure 7: Number of changes captured under various exponential urgency functions (r). Life is Unbounded-Append.

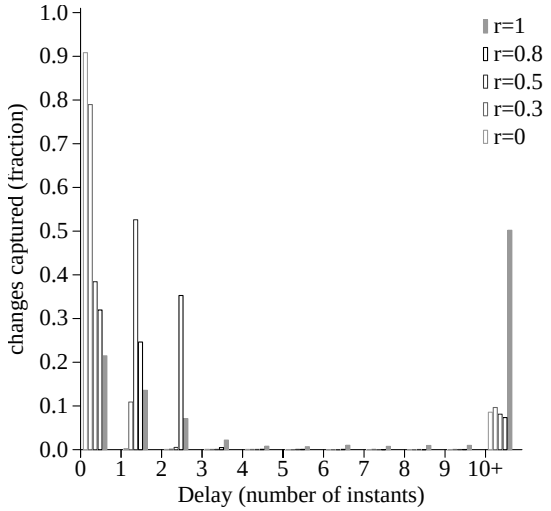

Figure 8: Distribution of delay in changes captured under various exponential urgency functions (r). Life is Unbounded-Append.

settings of r (urgency). Delay that exceeds 10 minutes is shown in the rightmost bar, labeled "10+." For $r = 0$, over 90% of the changes captured are captured with zero delay. However, this figure drops below 25% for the case of $r = 1$, in which roughly 50% of the changes captured are captured with a delay of 10 minutes or more.

These results indicate a tradeoff does indeed exist between timeliness and completeness, and it can be controlled by adjusting our urgency parameter, as our analytical results of Section 4 indicated. The tradeoff is perhaps best visualized as plotted in Figure 9. This graph shows the total number of changes captured (as a fraction of the total number of changes that occurred), as well as the total number captured with zero delay (again as a fraction of the total number that occurred), when $C = 8$, for different settings of urgency (r). It can clearly be seen that by adjusting the urgency parameter r, timeliness can be traded off against completeness.

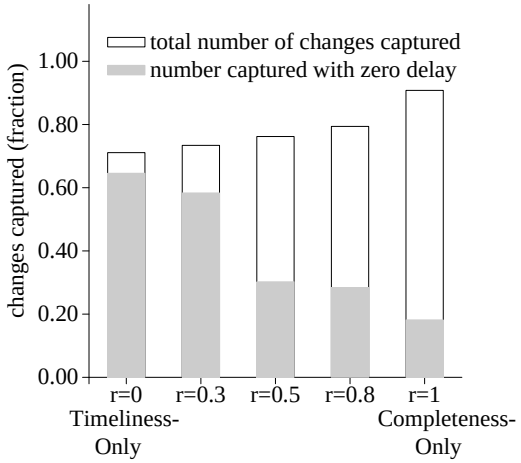

Figure 9: Tradeoff between timeliness and completeness.

5.4 Comparison Against Prior Approach

For our final experiment we compare our approach against the only prior work we are aware of on scheduling monitoring of dynamic Web pages, CAM [13]. Since CAM was designed to optimize for the "returned information ratio" (RIR) objective, we set our life parameter to Overwrite and $r = 0$ (equivalently, Sliding Window(0) urgency) to make utility equivalent to RIR. Note that the RIR objective strongly favors timeliness over completeness and assumes that all changes overwrite information due to previous changes. In this way our approach generalizes that of [13]. However, our approach is less general in the sense that we assume unit time to monitor a page, whereas CAM handles cases in which each pages takes a different amount of time to download. Therefore, neither approach subsumes the other.

We compared the two approaches on the scenario in which they overlap, namely: scheduling unit-time monitorings of pages that undergo changes that completely overwrite information, while optimizing for timeliness. Note that our WIC algorithm is guaranteed to find the optimal monitoring schedule in this case (Section 3.3). We measured total utility accrued for different values of C (available resources), under the two algorithms, with $FPN = 0.1$ and $\sigma = 2$. Our WIC algorithm outperformed CAM by as much as a factor of two.

6 Related Work

Web monitoring has been addressed in the context of systems that evaluate continuous queries over the Web [7–10]. The main focus of this work has been on language design and scalability of the query engine, rather than on how best to capture information from sources requiring pull-based

access, like Web pages. Our work addresses this largely ignored yet important research topic.

The only prior work we are aware of that addresses this topic in a nontrivial way is [13], which introduced the CAM Web monitoring algorithm. CAM requires access to predicted change probabilities in advance (*i.e.*, it is not an online algorithm), and is geared toward maximizing an objective called "returned information ratio" (RIR). RIR strongly favors timeliness over completeness, and assumes that all changes overwrite information due to previous changes. RIR is a special case of our much broader formulation, in which the tradeoff between timeliness and completeness of information captured, as well as the way in which information is posted to Web pages, are exposed as parameters. When our algorithm parameters are set to match the RIR objective, our online algorithm (WIC) is guaranteed to find the optimal solution. In all other cases WIC is a 2-approximation. No formal guarantees about the effectiveness of CAM were provided in [13], although the CAM heuristic does handle cases in which the cost of monitoring is nonuniform across pages (WIC does not).

Work on scheduling Web crawlers, *e.g.*, [2,4,14] focuses on maximizing the current "freshness" of a local repository containing copies of Web pages. In contrast, in our work the focus is on capturing the history of changes to pages.

7 Summary

In this paper we studied the problem of scheduling polling of remote Web pages for the purpose of monitoring the dynamic Web. The goal is to use limited resources most effectively in order to maximize the overall utility of information captured. Utility is a highly application-dependent notion, and our approach is parameterized by custom specifications of (1) the relative importance of information available from individual pages under consideration and (2) the sensitivity of the application to delay in captured information. Our highly parameterized formulation makes it suitable for a wide variety of Web monitoring applications.

We formalized the scheduling problem as a parameterized optimization problem. We then presented an efficient online algorithm that we showed always achieves total utility within a factor of two of the optimal offline solution in all cases. Both analysis and experiments on real-world online auction data confirmed that a fundamental tradeoff exists between timeliness and completeness of information captured during monitoring; our urgency parameter serves as a knob to control this tradeoff.

References

[1] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, New York, 1972.

[2] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, May 2000.

[3] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART*, June 2003.

[4] J. Edwards, K. S. McCurley, and J. A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the Tenth International World Wide Web Conference*, May 2001.

[5] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International World Wide Web Conference*, May 2003.

[6] T. Ibaraki and N. Katoh. Resource allocation problems: Algorithmic approaches. *MIT Press, Cambridge, MA*, 1988.

[7] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *Knowledge and Data Engineering*, 11(4):610–628, 1999.

[8] L. Liu, C. Pu, and W. Tang. WebCQ: Detecting and delivering information changes on the web. In *Proceedings of International Conference on Information and Knowledge Management*, November 2000.

[9] L. Liu, C. Pu, W. Tang, and W. Han. CONQUER: A continual query system for update monitoring in the WWW. *International Journal of Computer Systems, Science and Engineering*, 1999.

[10] J. Naughton, D. DeWitt, D. Maier, A. Aboulnaga, J. Chen, L. Galanis, J. Kang, R. Krishnamurthy, Q. Luo, N. Prakash, R. Ramamurthy, J. Shanmugasundaram, F. Tian, K. Tufte, E. Viglas, Y. Wang, C. Zhang, B. Jackson, A. Gupta, and R. Chen. The Niagara internet query system. *IEEE Data Engineering Bulletin*, 24(2):27–33, 2001.

[11] V. N. Padmanabhan and L. Qui. The content and access dynamics of a busy web site: findings and implications. In *Proceedings of ACM SIGCOMM*, August, 2000.

[12] S. Pandey, K. Dhamdhere, and C. Olston. WIC: A General-Purpose Algorithm for Monitoring Web Information Sources. Technical report, June 2004. Available at: `http://www.cs.cmu.edu/~olston/publications/wic.html`.

[13] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *Proceedings of the Twelfth International World Wide Web Conference*, May 2003.

[14] J. Wolf, M. Squillante, P. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the Eleventh International World Wide Web Conference*, May 2002.