Efficient Monitoring Algorithm for Fast News Alerts

Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho

***Abstract*— Recently, there has been a dramatic increase in the use of XML data to deliver information over the Web. Personal weblogs, news Web sites, and discussion forums are now publishing RSS feeds for their subscribers to retrieve new postings. As the popularity of personal weblogs and the RSS feeds grow rapidly, RSS aggregation services and blog search engines have appeared, which try to provide a central access point for simpler access and discovery of new content from a large number of diverse RSS sources. In this paper, we study how the RSS aggregation services should monitor the data sources to retrieve new content quickly using minimal resources and to provide its subscribers with fast news alerts. We believe that the change characteristics of RSS sources and the general user access behavior pose distinct requirements that make this task significantly different from the traditional index refresh problem for Web-search engines. Our studies on a collection of 10K RSS feeds reveal some general characteristics of the RSS feeds, and show that with proper resource allocation and scheduling the RSS aggregator provides news alerts significantly faster than the best existing approach.**

***Index Terms*— Information Search and Retrieval, Online Information Services, Performance evaluation, User profiles and alert services**

I. Introduction

Recently, there has been a dramatic increase in the use of XML data to deliver information over the Web. In particular, personal weblogs, news Web sites, and discussion forums are now delivering up-to-date postings to their subscribers using the RSS protocol [32]. To help users access new content in this RSS domain, a number of RSS aggregation services and blog search engines have appeared recently and are gaining popularity [2], [3], [33], [38]. Using these services, a user can either (1) specify the set of RSS sources that she interested in, so that the user is notified whenever new content appears at the sources (either through email or when the user logs in the service) or (2) conduct a keyword-based search to retrieve all content containing the keyword. Clearly, having a central access point makes it significantly simpler to discover and access new content from a large number of diverse RSS sources.

A. Challenges and contributions

In this paper, we investigate one of the important challenges in building an effective RSS aggregator: how can we minimize the delay between the publication of new content at a source and its appearance at the aggregator? Note that the aggregation can be done either at a desktop (e.g., RSS feed readers) or at a central server (e.g., Personalized Yahoo/Google homepage). While some of our developed techniques can be applied to the desktop-based aggregation, in this paper we primarily focus on the server-based aggregation scenario. This problem is similar to the index refresh problem for Web-search engines [7], [9], [11], [13], [15], [30], [31], [40], but two important properties of the information in the RSS domain make this problem unique and interesting:

- The information in the RSS domain is often *time sensitive*. Most new RSS content is related to current world events, so its value and significance deteriorates rapidly as time passes. An effective RSS aggregator, therefore, has to retrieve new content quickly and make it available to its users close to real time. This requirement is in contrast to general Web search engines where the temporal requirement is not as strict. For example, it is often acceptable to index a new Web page within, say, a month of its creation for the majority of Web pages.
- For general search engines, users mainly focus on the quality of the *returned pages* and largely ignore (or not care about) what is not returned [22], [24]. Based on this observation, researchers have argued and mainly focused on improving the *precision* of the top-k result [30], and the page-refresh policies have also been designed to improve the freshness of the indexed pages. For RSS feeds, however, many users often have a set of their favorite sources and are particularly interested in reading the new content from these sources. Therefore, users do notice (and complain) if the new content from their favorite sources is missing from the aggregator.

As we will see later, the time-sensitivity of the RSS domain fundamentally changes how we should model the generation of new content in this domain and makes it necessary to design a new content-monitoring policy. In the rest of this paper, we investigate the problem of how we can effectively monitor and retrieve time sensitive new content from the RSS domain as follows:

- In Section II, we describe a formal framework for this problem. In particular, we propose a *periodic inhomogeneous Poisson process* to model the generation of postings at the RSS feeds. We also propose to use the *delay metric* to evaluate the monitoring policies for RSS feeds.
- In Section III, we develop the optimal ways to retrieve new content from RSS feeds through a careful analysis of the proposed model and metric.
- In Section IV, we examine the general characteristics of the RSS feeds based on real RSS-feed data. We also evaluate the effectiveness of our retrieval policies

Ka Cheung Sia and Junghoo Cho are with the Department of Computer Sicence, University of California, Los Angeles, CA 90095, USA. E-mail:{kcsia,cho}@cs.ucla.edu

Hyun-Kyu Cho is with the Fundamental Intelligent Robot Research Team of Intelligent Robot Research Division, ETRI, 161 Gajeong-dong, Yuseong-gu, Daejeon, 305-700, Korea. E-mail: hkcho@etri.re.kr

Manuscript received January 27, 2006; revised October 3, 2007; accepted January 11, 2007.

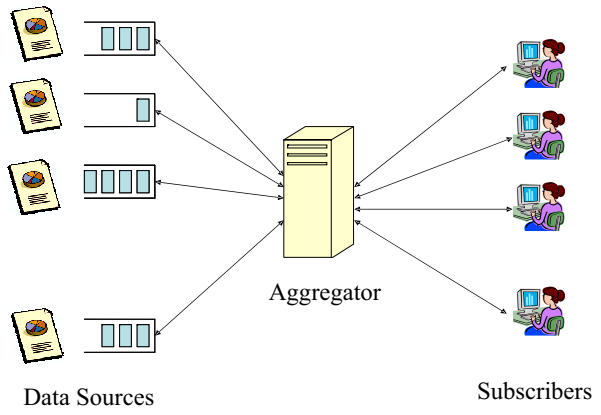


Fig. 1. Framework of an information aggregator.

using the data. Our experiments show that our policy significantly reduces the retrieval delay compared to the best existing policies.

Note that while our work is primarily motivated by our desire to aggregate the content from the RSS domain, our approach is general and independent of the particular type of the data source (e.g., whether we monitor the content from a general Web page or from an RSS feed) as we will see later. As long as the content is time sensitive and it is important to redownload the content frequently (say, more than once a day), the traditional homogeneous Poisson model for changes often does not hold anymore, which makes our new approach important.

II. Framework

In Figure 1 we show the high-level architecture of an RSS aggregator. We consider a distributed information system that consists of n *data sources*[1], a single *aggregator* and a number of *subscribers*. The data sources constantly generate new pieces of information referred to as new *postings*. We assume a *pull*-based architecture, where the aggregator periodically collects the most recent k postings from each source.[2] A subscriber, in turn, retrieves new postings from the aggregator.

Resource constraints: We assume that both the aggregator and the sources have limited computational and network resources for the retrieval of new postings. For example, the aggregator may have a dedicated T1 line that allows the aggregator to contact the sources up to one million times per day, or due to the limit of its networking stack, the aggregator may issue up to 500,000 HTTP requests per hour. In this paper, we model the resource constraint by assuming that the aggregator can contact the sources a total of M times in each period. (The notion of "period" will become clear when we discuss the posting generation model.)

[1]In here, one data source typically corresponds to a single RSS feed, but if multiple RSS feeds can be downloaded through one single HTTP connection to a Web server, they may be grouped together and be considered as one data source.

[2]k is typically in the range of 10–15

Retrieval delay: An effective RSS aggregator has to retrieve new postings from the sources quickly and make them available to its users with minimal delay. We formalize this notion of delay as follows:

Definition 1 Consider a data source O that generates postings at times $t_1, \ldots, t_k$. We also use t_i to represent the posting itself generated at time t_i unless it causes confusion. The aggregator retrieves new postings from O at times $\tau_1, \ldots, \tau_m$. The delay associated with the posting t_i is defined as

$$D(t_i) = \tau_j - t_i$$

where τ_j is the minimum value with $t_i \leq \tau_j$. The total delay of the postings from source O is defined as

$$D(O) = \sum_{i=1}^{k} D(t_i) = \sum_{i=1}^{k} (\tau_j - t_i) \text{ with } t_i \in [\tau_{j-1}, \tau_j]. \quad \square$$

It is also possible to use the metrics that have been widely used in the general search-engine research [7], [10], [13], such as *freshness* and *age*, by modeling the publication of new postings as *modifications* of a data source. For example, the freshness, $F(O;t)$, and age, $A(O;t)$, of a data source O at time instance t can be defined as

$$F(O;t) = \begin{cases} 0 & \text{if } \exists t_i \in [\tau_j, t] \\ 1 & \text{otherwise} \end{cases}$$

$$A(O;t) = \begin{cases} t - t_m & \text{if } \exists t_i \in [\tau_j, t] \\ 0 & \text{otherwise} \end{cases}$$

where τ_j is the most recent retrieval time and t_m is the minimum of all t_i's within $[\tau_j, t]$.

For illustration, we show an example evolution of delay, freshness and age in Figures 2(a), (b), and (c), respectively. The data source generates five postings at $t_1, \ldots, t_5$ (marked by dashed lines). Two retrievals are scheduled by the aggregator at τ_1 and τ_2 (marked by solid lines). The vertical axes represent the delay, freshness, and age associated with the data source. Note that after the generation of t_2, the delay metric increases twice as rapidly as before, because two new postings, t_1 and t_2, are pending at the source. In contrast, the age metric does not take into account that two pending postings and still increases at the constant same rate as before. Thus, we may consider our delay metric as an improved version of the age metric that takes into account multiple postings pending at a source, which we believe is more appropriate in the context of RSS feeds.

When multiple sources generate new postings, it may be more important to minimize the delay from one source than others. For example, if a source has more subscribers than others, it may be more beneficial to minimize the delay for this source. This difference in importance is captured in the following weighted definition:

Definition 2 We assume each source O_i is associated with weight w_i. Then the total weighted delay observed by the aggregator, $D(A)$, is defined as

$$D(A) = \sum_{i=1}^{n} w_i \, D(O_i) \quad \square$$

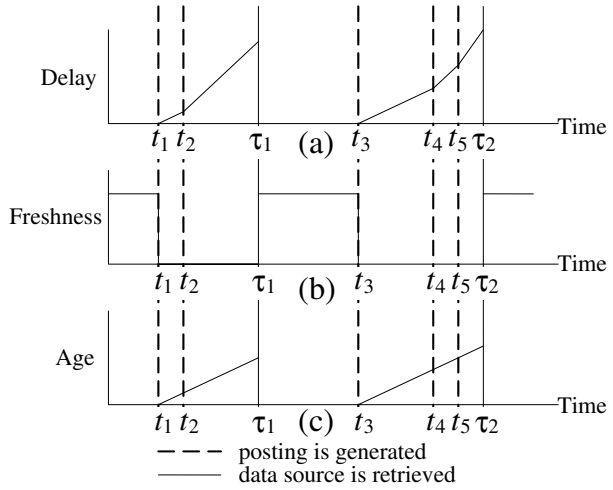
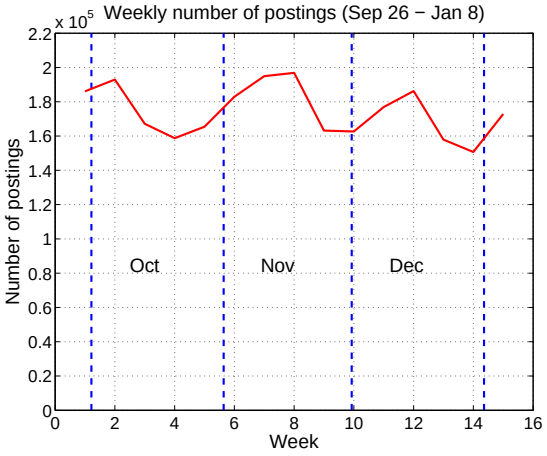
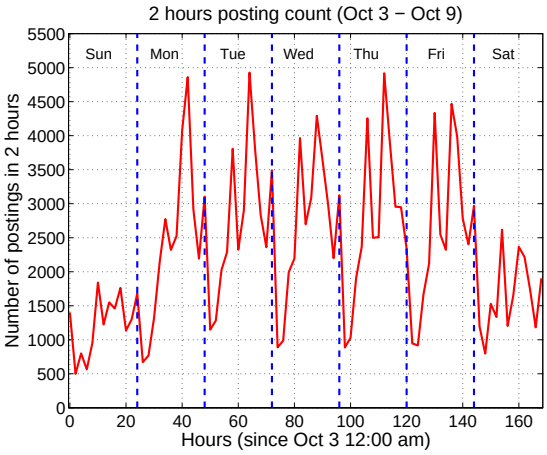


Fig. 2. Illustration of the delay, freshness, and age metrics



(a) Weekly posting rate



(b) Hourly posting rate

Fig. 3. Posting rate at different resolution.

Delay minimization problem: We use t_{ij} to represent the jth posting generation time at O_i and τ_{ij} to refer to the time of the jth retrieval from O_i by the aggregator. Given the definitions, we can formalize the problem of delay minimization as follows:

PROBLEM 1 Given the posting generation times t_{ij}'s, find the retrieval times τ_{ij}'s that minimize the total delay $D(A) = \sum_{i=1}^{n} w_i D(O_i)$ under the constraint that the aggregator can schedule a total of M retrievals. □

A. Posting generation model

In practice, the aggregator does not know the future posting generation times t_{ij}'s. Therefore, to solve the delay minimization problem, the aggregator has to *guess* the future posting times based on the *past* posting pattern of each source.

In the context of general Web search engines, researchers have proposed that a *homogeneous* Poisson process with a rate λ is a good model to be used [7], [10]. Roughly, a homogeneous Poisson process is a stateless and time-independent random process, where new postings always appear at a *constant* rate λ regardless of the time [37]. A number of studies [7], [10] show that this model is appropriate especially when the time granularity is longer than one month. For example, Figure 3(a) shows the total number of postings appearing in roughly 10,000 RSS feeds that we monitored (more details of this dataset is described in the experiment section). The horizontal axis is the time, and the vertical axis shows the number of postings appearing in each week of the monitoring period. While there are small fluctuations, the total number of new postings in each week is reasonably stable at roughly 180,000 postings, which matches with the homogeneous Poisson assumption. Formally, this assumption can be stated as $\lambda(t) = \lambda$, where the posting generation rate at time t, $\lambda(t)$, is constant and independent of time t. Based on this homogeneous model, researchers have derived the optimal re-download algorithms for Web crawlers [10], [13].

Unfortunately, when the time granularity is much shorter than one month, there exists strong evidence that the homogeneous Poisson model is no longer valid [4], [17], [20]. In Figure 3(b), for example, we show the total number of postings appearing in the same RSS feeds when we count the number at a granularity of two hours. From the figure, it is clear that at this time granularity, the time-independence property of the homogeneous Poisson model does not hold. The posting rate goes through wide fluctuation depending on the time of the day and the day of the week. The graph also shows a certain level of periodicity in the posting rates. During the day, there are a significantly higher number of postings than at night. Similarly, there are more activities during the weekdays than on weekends. Based on this observation, we propose to use an *inhomogeneous* Poisson model, where the posting rate $\lambda(t)$ changes over time. Depending on whether similar patterns of $\lambda(t)$ values are repeated over time, this model can be further classified into one of the following:

1) *Periodic inhomogeneous Poisson model*: We assume that the same $\lambda(t)$ values are repeated over time with a period

of T. That is, $\lambda(t) = \lambda(t - nT)$ for $n = 1, 2, \ldots$. This model may be a good approximation when similar rate patterns are repeated over time, such as the burst of activities during the day followed by a period of inactivity at night.

2) *Non-periodic inhomogeneous Poisson model*: This is the most general model where no assumption is made about the periodicity in the changes of $\lambda(t)$. That is, there exists no T that satisfies $\lambda(t) = \lambda(t - nT)$.

Given the periodicity that we observe in the RSS posting pattern and the strict temporal requirement for the retrieval of new postings from RSS feeds, we mainly use the periodic inhomogeneous Poisson model in the rest of this paper.

B. Expected retrieval delay

Since the aggregator does not know the exact times at which new postings are generated, it can only estimate the *expected* delay based on the posting generation model of a source. In general, the expected delay can be computed as follows under the general inhomogeneous Poisson model:

LEMMA 1 *For a data source O with the rate $\lambda(t)$, the total expected delay for the postings generated within $[\tau_{j-1}, \tau_j]$ are as follows:*

$$\int_{\tau_{j-1}}^{\tau_j} \lambda(t)(\tau_j - t)dt. \qquad \square$$

Proof: During a small time interval dt at time t, $\lambda(t)dt$ postings are generated. Since these postings are retrieved at time τ_j, their associated delays are $\tau_j - t$. Therefore, the total delay of the postings generated between τ_{j-1} and τ_j is $\int_{\tau_{j-1}}^{\tau_j} \lambda(t)(\tau_j - t)dt$. ■

For the simpler homogeneous Poisson model, the above formula is simplified to the following formula.

COROLLARY 1 *When the posting rate remains constant at λ within the time period $[\tau_{j-1}, \tau_j]$, the total expected delay for postings generated within this period is*

$$\frac{\lambda(\tau_j - \tau_{j-1})^2}{2}. \qquad \square$$

The expected delays computed above will be used in the next section when we investigate optimal retrieval policy used by the aggregator.

III. RETRIEVAL POLICY

We now study how the aggregator should schedule the M retrieval points τ_{ij}'s to minimize the total expected delay. We approach this scheduling problem in two steps:

- *Resource allocation*: Given n data sources and a total of M retrievals per period T, the aggregator first decides *how many times* it will contact individual source O_i. This decision should be made based on how frequently new postings appear in each source and how important each source is.
- *Retrieval scheduling*: After the aggregator decides how many times it will contact O_i per T, it decides exactly *at what times* it will contact O_i. For example, if the aggregator has decided to contact O_1 twice a day, it may either schedule the two retrieval points at uniform intervals (say, one at midnight and one at noon) or it may schedule both retrievals during the day when there are likely to be more new postings.

In Section III-A, we start with the resource-allocation problem. We then study the retrieval scheduling problem in Section III-B. As far as we know, our work is the first study to develop optimal solutions for the retrieval-scheduling problem for Web sources, while similar resource-allocation problems have been studied before (e.g., [7], [9], [13]) albeit under a different metric. Finally in Section III-C, we go over techniques to obtain accurate estimate of posting rates and patterns from past history.

A. Resource-allocation policy

Our task in this section is to allocate the M retrievals among the data sources to minimize the total expected delay. For this task, we use the simple homogeneous Poisson process model because the resource allocation is done based on the *average posting generation rate* and the *weight of each source*, both of which are adequately captured by the homogeneous Poisson model. The more complex inhomogeneous model will be used later when we consider the retrieval-scheduling problem.

Our main result for this resource-allocation problem is summarized in the following theorem, which shows that the optimal allocation of resources to a source O_i should be proportional to the square root of the product of its posting rate λ_i and its importance w_i.

THEOREM 1 *Consider data sources $O_1, \ldots, O_n$, where O_i has the posting rate λ_i and the importance weight w_i. The aggregator performs a total of M retrievals per each period T.*

Under this scenario, the weighted total delay of postings, $D(A) = \sum_{i=1}^{n} w_i D(O_i)$, becomes minimum when the source O_i is contacted at a frequency proportional to $\sqrt{w_i \lambda_i}$. That is, m_i, the optimal number of retrievals per each period for O_i, is given by

$$m_i = k\sqrt{w_i \lambda_i} \tag{1}$$

where k is a constant that satisfies $\sum_{i=1}^{n} k\sqrt{w_i \lambda_i} = M$. $\square$

Proof: We consider the data source O_i that is retrieved m_i times per day. Under the homogeneous Poisson model, we can show that $D(O_i)$, the total delay of postings from O_i, is minimum when the retrievals are scheduled at the uniform interval.[3] In this case, $D(O_i) = \frac{\lambda_i T^2}{2m_i}$, and the total weighted delay, $D(A)$, is

$$D(A) = \sum_{i=1}^{n} \frac{\lambda_i w_i T^2}{2m_i}.$$

$D(A)$ can be minimized by using the Lagrange multiplier method.

$$\frac{\partial D(A)}{\partial m_i} = -\frac{\lambda_i w_i T^2}{2m_i^2} = -\mu.$$

[3]This proof follows from a special case of the Cauchy's inequality stating that sum of squares are always less then square of sums and equality holds when all numbers are equal.

If we rearrange the above equation, we get

$$m_i = \sqrt{\lambda_i w_i T^2/2\mu} = k\sqrt{\lambda_i w_i}.$$

■

As we can see from the solution, the optimal resource allocation can be computed simply by multiplying the posting rate of each source with k, which can be computed from w_i's and λ_i's. Therefore, the complexity of computing the optimal resource-allocation policy is linear with the number of data sources.

B. Retrieval-scheduling policy

We have just discussed how to allocate resources to data sources based on their weights and average posting rates. Assuming that postings are retrieved m times from the source O, we now discuss exactly at what times we should schedule the m retrievals. Clearly, this decision should be based on what time of the day the source is expected to generate the largest number of postings, so we now use the periodic inhomogeneous Poisson model to capture the daily fluctuation in the posting generation rate.

To make our discussion easy to follow, we start with a simple case when only one retrieval is allocated per period in Section III-B.1. We then extend our analysis to a more general case in Section III-B.2.

1) Single retrieval per period: Consider a data source O at the periodic posting rate $\lambda(t) = \lambda(t - nT)$. The postings from O are retrieved only once in each period T. The following theorem shows that the best retrieval time is when the instantaneous posting rate $\lambda(t)$ equals the average posting rate over the period T.

THEOREM 2 *A single retrieval is scheduled at time τ for a data source with the posting rate $\lambda(t)$ of period T. Then, when the total delay from the source $D(O)$ is minimized, τ satisfies the following condition:*

$$\lambda(\tau) = \frac{1}{T}\int_0^T \lambda(t)dt \qquad \left(\textit{and } \frac{d\lambda(\tau)}{d\tau} < 0\right). \qquad (2) \quad \square$$

Proof: Without loss of generality, we consider only the postings generated within a single interval $[0, T]$. We use the notation $D(\tau)$ to represent the delay when the retrieval is scheduled at τ. The postings generated between $[0, \tau]$ are retrieved at τ, so their delay is $\int_0^\tau \lambda(t)(\tau - t)dt$. The postings generated between $[\tau, T]$ are retrieved in the next interval at time $T + \tau$, so their delay is $\int_\tau^T \lambda(t)(T + \tau - t)dt$. Therefore,

$$\begin{aligned} D(\tau) &= \int_0^\tau \lambda(t)(\tau - t)dt + \int_\tau^T \lambda(t)(T + \tau - t)dt \\ &= T\int_\tau^T \lambda(t)dt + \int_0^T \lambda(t)(\tau - t)dt. \end{aligned}$$

$D(\tau)$ is minimum when

$$\frac{dD(\tau)}{d\tau} = -T\,\lambda(\tau) + \int_0^T \lambda(t)dt = 0$$

and $\frac{d^2D(\tau)}{d\tau^2} = -T\frac{d\lambda(\tau)}{d\tau} > 0$. After rearranging the expressions, we get Equation 2. ■

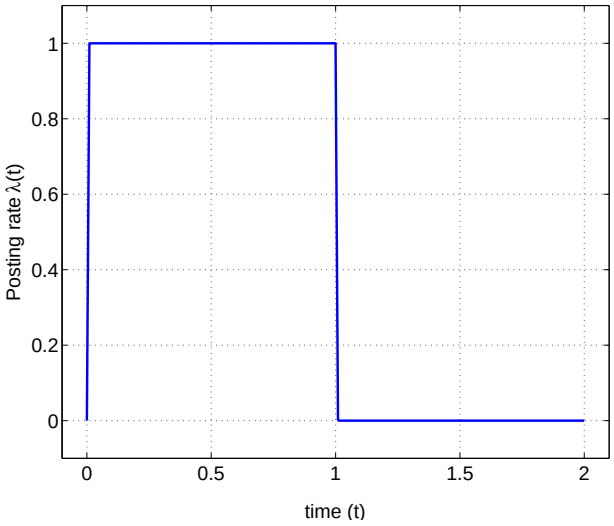


Fig. 4. A data source going through periods of high activity and low activity

We illustrate the implication of the theorem using a simple example.

EXAMPLE 1 Figure 4 shows a data source that goes through a period of high activity, $\lambda(t) = 1$, during $t \in [0, 1]$ and a period of low activity, $\lambda(t) = 0$, during $t \in [1, 2]$. The same pattern is repeated after $t = 2$. Its postings are retrieved once in each period.

According to our theorem, the retrieval should be scheduled at $t = 1$ when the $\lambda(t)$ changes from 1 to 0 and takes the average value $\lambda(t) = 0.5$. This result matches our intuition that the retrieval should be scheduled right after a period of high activity. The expected total delay in this case is $\frac{1}{2}$. Compared to the worst case when the retrieval is scheduled right before a period of high activity (i.e., $\tau = 0$) we get a factor of 3 improvement. Compared to the average case, we get a factor of 2 improvement. □

2) Multiple retrievals per period: Now, we generalize the scenario and consider the case when multiple retrievals are scheduled within one period.

THEOREM 3 *We schedule m retrievals at time $\tau_1, \ldots, \tau_m$ for a data source with the posting rate $\lambda(t)$ and periodicity T. When the total delay is minimized, the τ_j's satisfy the following equation:*

$$\lambda(\tau_j)(\tau_{j+1} - \tau_j) = \int_{\tau_{j-1}}^{\tau_j} \lambda(t)dt, \qquad (3)$$

where $\tau_{m+1} = T + \tau_1$ (the first retrieval point in the next interval) and $\tau_0 = \tau_m - T$ (the last retrieval point in the previous interval). □

Proof: Without loss of generality, we consider the expected total delay in postings generated between τ_1 and $T + \tau_1$:

$$D(O) = \sum_{i=1}^{m} \int_{\tau_i}^{\tau_{i+1}} \lambda(t)(\tau_{i+1} - t)dt$$

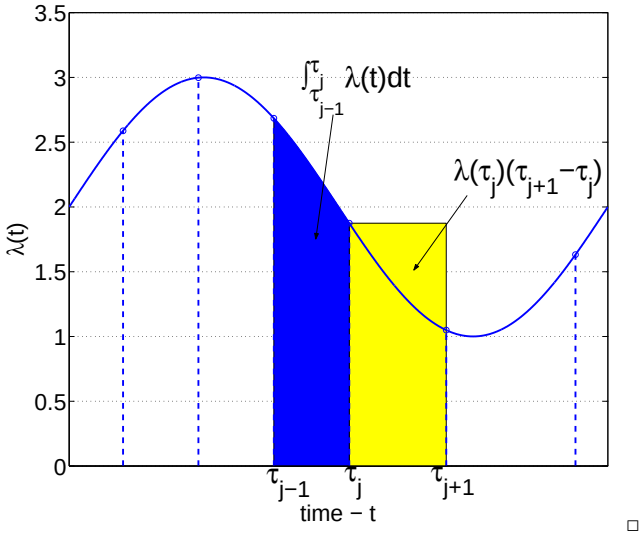


Fig. 5. The optimal schedule for 6 retrievals per period for data source with posting rate $\lambda(t) = 2 + 2\sin(2\pi t)$.

$$= \sum_{i=1}^{m} \left(\tau_{i+1} \int_{\tau_i}^{\tau_{i+1}} \lambda(t)dt \right) - \int_{\tau_1}^{T+\tau_1} \lambda(t)tdt$$
$$= \sum_{i=1}^{m} \left(\tau_{i+1} \int_{\tau_i}^{\tau_{i+1}} \lambda(t)dt \right) - \int_{0}^{T} \lambda(t)tdt.$$

Then $D(O)$ is minimum when $\frac{\partial D}{\partial \tau_j}$ for every τ_j:

$$\frac{\partial D}{\partial \tau_j} = \int_{\tau_{j-1}}^{\tau_j} \lambda(t)dt + \tau_j\lambda(\tau_j) - \tau_{j+1}\lambda(\tau_j) = 0.$$

By rearranging the above expression, we get Equation 3. ■

We illustrate the graphical meaning of the theorem using an example.

EXAMPLE 2 Figure 5 shows a data source with the posting rate $\lambda(t) = 2 + 2\sin(2\pi t)$. Postings are retrieved from the source 6 times in one period. We assume that we have decided up to the j^{th} retrieval point, and need to determine the $(j+1)^{th}$ point. Note that the right-hand side of Equation 3 corresponds to the dark-shaded area in Figure 5. The left-hand side of the equation corresponds to the light-shaded area of Figure 5. The theorem states that the total delay is minimized when τ_{j+1} is selected such that the two areas are the same.

The above theorem states the necessary conditions of the optimal solution. Based on this result, we may use one of the following two methods in computing the optimal schedule once we know the $\lambda(t)$ of a source.

1) *Exhaustive search with pruning*: Once the first two retrieval points are determined, the remaining retrieval points are derived automatically from Equation 3. Therefore, all possible plans are evaluated by exhaustively trying all choices for the first two retrieval points (assuming a certain level of discretization in the time). We can then choose the plan with the minimum delay from only the combinations of all first two retrieval points. Assuming that the the time is discretized into k intervals, the cost of exhaustively searching all possible combination of two initial points is $O(k^2)$.
2) *Iterative refinement*: Initially, we place all retrieval points at uniform intervals. We then iteratively adjust the retrieval points by comparing the areas below the graph. For example, if the dark area in Figure 5 is larger than the light area, we move τ_j slightly to the left to compensate for it. We repeat the adjustment until the retrieval points converge or subsequent adjustments are below a certain threshold.[4]

In our experiments, we find that both methods lead to reasonable performance in finding the optimal retrieval points when a time granularity of 5 minutes is used for exhaustive search with pruning. For the experiments described afterwards, we compute the optimal retrieval schedule by the exhaustive search with pruning method.

C. Learning posting rates and patterns

In order to implement the resource allocation and retrieval scheduling policies, the aggregator has to learn the average posting rate and the posting pattern $\lambda(t)$ of each source. Assuming that they do not change rapidly over time, we may estimate them by observing the source for a period of time and use the estimation in determining the optimal monitoring policies.

Measuring the posting rate can be done simply by counting the total number of postings generated within a particular learning period and dividing it by the length of the period. Learning the continuous posting pattern $\lambda(t)$ is more difficult, because we are observing discrete events of posting generation. Therefore, we first count the number of hourly postings at every source and build a daily histogram of hourly postings for the sources. We then overlap the daily histograms for k-week data for each source and obtain a graph similar to Figure 11. This discrete posting histogram is then approximated by a continuous function of $\lambda(t)$ through interpolation by using, say, a i^{th} degree polynomial function.

Note that there exists a clear tradeoff in deciding how long we choose to monitor a source to estimate $\lambda(t)$; if the length is too short, the estimated $\lambda(t)$ may be inaccurate due to the randomness in the posting generation. However, if it is too long and if the posting pattern itself changes over time, the estimated $\lambda(t)$ will become obsolete by the time we obtain it (making the monitoring policy based on the estimated $\lambda(t)$ ineffective). Later in the experiment section, we evaluate the impact of the length of estimation on the effectiveness of the policies and empirically determine the optimal estimation period.

IV. EXPERIMENTS

In this section, we show some statistics of the collected RSS feeds data and the result from the performance evaluation of our retrieval policies.

[4]More precise formulations on how much we need to shift the retrieval points are given in the extended version of this paper [36].

A. Description of dataset

RSS feeds are essentially XML documents published by Web sites, news agents, or bloggers to ease syndication of their Web site's contents to subscribers. Figure 6 shows a typical RSS feed. It contains different postings in the ⟨**item**⟩ tag and summaries in the ⟨**description**⟩ tag. Each posting is associated with a timestamp ⟨**dc:date**⟩, stating when it was generated. The postings are arranged in the reverse chronological order where new postings are prepended in the front and old postings are pushed downwards and removed. For the majority of current implementations, an RSS feed contains the most recent 10 or 15 postings. New postings are added to the feed at any time without notifying their subscribers; thus, the subscribers have to poll the RSS feeds regularly and check for updates. The set of data used comes from a list of 12K RSS feeds listed in `http://www.syndic8.com` that include time of posting within the RSS. They were downloaded 4 times a day between September 2004 and December 2004. Out of the 12K feeds, 9,634 (about 80%) feeds have at least one posting within this monitoring period. We focus on this subset of 9,634 RSS feeds in the following experiments. The range of the topics covered by this set of RSS feeds is quite diverse and the feeds are originating from about five thousand domains. Table I shows some of the frequent domains where these RSS feeds originate from.

```
- <rdf:RDF>
    <channel rdf:about="http://slashdot.org/"/>
  - <image rdf:about="http://images.slashdot.org/topics/topicslashdot.gif">
      <title>Slashdot</title>
    - <url>
        http://images.slashdot.org/topics/topicslashdot.gif
      </url>
      <link>http://slashdot.org/</link>
    </image>
  - <item rdf:about="http://slashdot.org/article.pl?sid=05/06/21/2238256&from=rss">
      <title>Legal Music Downloads At 35%, Soon To Pass Piracy</title>
    - <link>
        http://slashdot.org/article.pl?sid=05/06/21/2238256&from=rss
      </link>
    - <description>
        bonch writes "Entertainment Media Research released a study stating that 35% of
        strategic milestone with the population of legal downloaders close to exceeding tha
      </description>
      <dc:creator>timothy</dc:creator>
      <dc:date>2005-06-22T02:00:00+00:00</dc:date>
      <dc:subject>music</dc:subject>
      <slash:department>cars-surpass-buggies</slash:department>
      <slash:section>mainpage</slash:section>
      <slash:hitparade>39,39,27,17,1,0,0</slash:hitparade>
      <slash:comments>39</slash:comments>
  </item>
```

Fig. 6. A sample RSS feed

In Figure 7 we show the distribution of posting rates among the 9,634 RSS feeds, with the x-axis being the number of postings generated within three months and the y-axis being the number of feeds at the given rate. Both axes are shown in log scale. Within the 3 months, 3,116 feeds have generated one or more postings per day on average. The distribution roughly follows a straight line in the log-log scale plot, which suggests that it follows a power-law distribution.[5]

[5]A curve fit of the data indicates the best matching power-law curve is $y = ax^b$, with $a \simeq 376$ and $b \simeq -0.78$.

Count	Domain
1133	`scotsman.com`
209	`www.rss-job-feeds.org`
154	`newsroom.cisco.com`
138	`www.employmentspot.com`
118	`blogs.msdn.com`
109	`radio.weblogs.com`
88	`feedster.com`
83	`www.computerworld.com`
79	`www.sportnetwork.net`
67	`abclocal.go.com`

TABLE I
TOP 10 DOMAINS HOSTING THE RSS FEEDS IN THE DATASET.

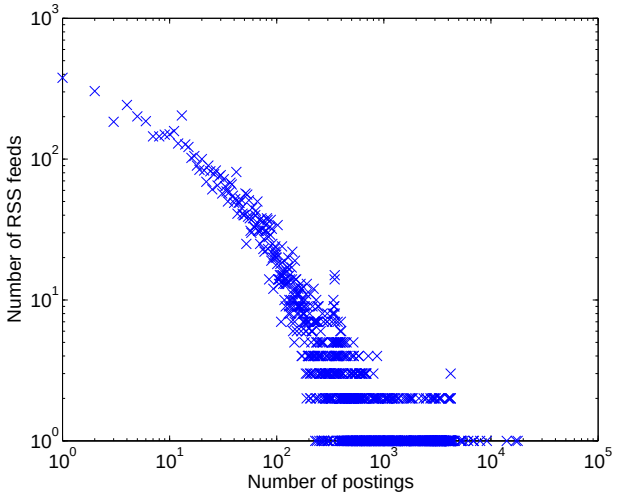


Fig. 7. Distribution of posting rate of 9,634 RSS feeds

B. Effectiveness of our policy

In this section, we evaluate the potential improvement of our proposed policy by comparing it against the best policies in the literature. In particular, we compare the *total weighted delay* $D(A)$ (Definition 2 in Section II) achieved by our policy against that of the age-based optimal crawling policy in [9].[6] Since both policies have to know the average posting rate of each source, we first learn the rates from the first two-week data and simulate the policies on the remaining 11-week data using the learned posting rates.[7] We assign equal weights to all sources because we want to evaluate the improvement from our accurate modeling of the posting generation at the sources, which is the main focus of this paper. For our policy, we employ both resource-allocation and retrieval-scheduling policies.

The results from these experiments are shown in Figure 8. The horizontal axis represents the resource constraint for a given experiment. More precisely, it shows the *average retrieval interval* per source (i.e., 11 weeks divided by M/n,

[6]Reference [9] describes two policies, one for the freshness metric and the other for the age metric. Since the result from the age-based policy outperforms the freshness-based policy by several orders of magnitude, we only show the age-based policy in our comparison.

[7]The choice of the two-week estimation window is explained later in Section IV-C.

where M is the total number of retrievals and n is the number of sources). Note that even when the average retrieval interval is the same, the actual retrieval points for each source are different under the two policies due to their different optimization approach.

The vertical axis represents the retrieval delay of postings under each policy at the given resource constraint. More formally, it shows the *average delay*, which is the total delay $D(A)$ divided by the total number of postings generated by all sources. We believe that reporting the average delay makes the numbers easier to interpret.

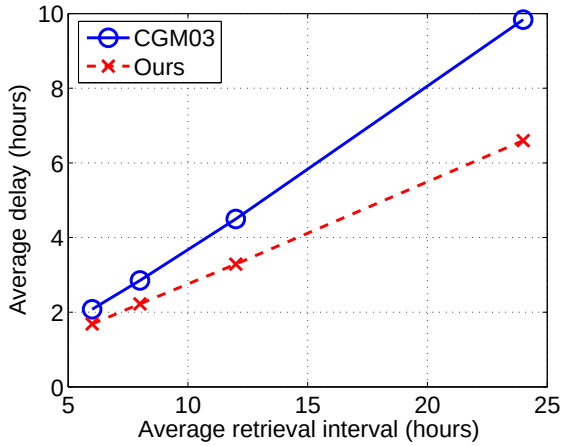


Fig. 8. Comparison with CGM03 policy

From the figure, we can see that our policy clearly outperforms CGM03; in general, CGM03 gives 35% longer delay than our policy. Also note that the average delay is significantly shorter than half of the average retrieval interval, which is the expected delay when no optimization is performed. For example, when the average retrieval interval is 10 hours, the average delay is less than 3 hours under our policy, which is more than 2 hours shorter than 5-hour expected delay with no optimization.

Contribution of individual policy: To investigate further how much improvement we get from each of our two optimizations (i.e., the resource-allocation policy in Section III-A and the retrieval-scheduling policy Section III-B), we now compare the average delay of the following four policies:

1) *Uniform scheduling*: We do not employ either our resource-allocation or the retrieval-scheduling policies. That is, all sources are retrieved the same number of times and the retrieval points are scheduled at uniform intervals. This can be considered as the baseline.
2) *Retrieval scheduling only*: We employ our retrieval-scheduling policy only. That is, all sources are retrieved the same number of times, but the retrieval points are optimized based on their posting patterns.
3) *Resource allocation only*: We employ our resource-allocation policy only. That is, we retrieve postings different numbers of times from the sources depending on their posting rates, but the retrieval points are scheduled at uniform intervals for each source.
4) *Combined*: Both of our policies are employed. The source are retrieved different numbers of times and the retrieval points are further optimized based on their posting patterns.

Again, we use the first 2-week data to learn the posting rates and the posting patterns, and use the remaining 11 weeks to simulate the retrieval policies and to compute the average delay. Every source is assigned an equal weight in this experiment.

Average retrieval interval	6hr	8hr	12hr	24hr
Uniform	180	256	352	645
Retrieval scheduling	159	211	310	518
Resource allocation	109	145	217	433
Combined	101	133	197	395

TABLE II
PERFORMANCE OF 4 RETRIEVAL POLICIES UNDER DIFFERENT RESOURCE CONSTRAINTS

Table II shows the average delays (reported in minutes) for the four policies under different resource constraints (from one retrieval per every 6 hours per source, up to one retrieval per every 24 hours per source). For example, the number 180 in the second column of the second row means that the average delay is 180 minutes under the uniform policy when the average retrieval interval per source is 6 hours.

From the table, we first note that the average delay under the uniform policy is close to the half of the average retrieval interval. For example, when the average retrieval interval is 6 hours, the average delay under the uniform policy is 180 minutes (or 3 hours). This result is expected because when the postings are retrieved every 6 hours from a source, the maximum delay will be 6 hours and the minimum delay will be 0 hour, with the average being 3 hours.

The results also show that both resource-allocation and retrieval-scheduling policies are effective in reducing the average delay. For example, when we retrieve new postings once every 24 hours on average (the last column), retrieval scheduling and resource allocation decreases the delay by 20% and by 32%, respectively, from the uniform policy. Combined together, we observe a 40% reduction in the average delay compared to the uniform policy.

While both resource-allocation and the retrieval-scheduling policies are effective in reducing the average delay, we note that the improvements are obtained through different mechanisms. Under the resource allocation policy, resources are taken away from the sources of low posting rates (or of low importance) and are allocated to the sources of high posting rates (or of high importance). Thus, while we decrease the *average* delay, we end up *increasing* the *maximum* delay of postings (for the sources of low posting rates). In contrast, the retrieval scheduling policy improves the delay simply by selecting the best retrieval time for a source without reallocating resources among the sources, so the maximum delay do not vary much among the sources under this policy. For example, under the resource constraint of one retrieval per day per source, the maximum delay of a posting was 1440 minutes for the retrieval-scheduling only policy, while the maximum delay was 9120 minutes for the resource-allocation policy. Given this result, we recommend employing only the retrieval-scheduling policy when a tight bound on the maximum delay is important.

C. Learning the posting rates and patterns

In Section III-C, we discussed that we use the past posting history of an RSS source to learn its average posting rate λ_i and its posting pattern $\lambda(t)$. We also discussed that if the length of the estimation period (which we refer to as the *estimation window*) for λ_i and $\lambda(t)$ is too short, the estimate may be unreliable, while if the window is too long, the estimate may be obsolete. In this section, we try to experimentally identify the optimal window length for our RSS sources.

Learning the posting rates: We first study the optimal window length for learning the average posting rate λ_i for source O_i. To investigate this, at the beginning of each day, we use the past k-day history data to estimate the posting rate of each source and decide the optimal number of retrievals per day for each source. We repeat this process over the entire 3-month data and measure the average delay at the end of the 3-month period.

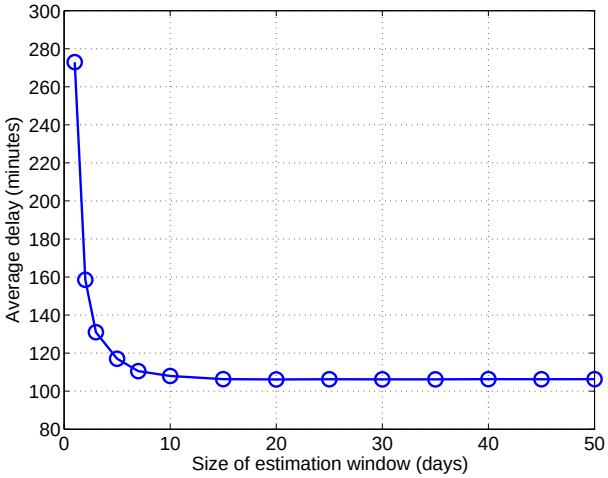


Fig. 9. The effect of estimation window width

Figure 9 shows the average delay of postings for different k values.[8] The graph shows that average delay decreases as the estimation window gets longer, which indicates more accurate estimation of the posting rates. However, there is no more improvement beyond k =14, which suggests that 14-day worth of data is adequate to smooth out fluctuations and get reasonable estimates.

In addition, the fact that the delay does not increase after $k = 14$ suggests that the posting rate of a source is stable and does not change significantly over time. To further investigate the change in the posting rate, we plot the following graph: We calculate the posting rate of each source using the first 14-day trace and use it as the x-coordinate. We then calculate the posting rate again based on the last 14-day trace and use it as the y-coordinate. Based on the two coordinates, we draw a x-y scatter plot in Figure 10. In this graph, if the posting rates are stable, all dots should be aligned along the diagonal line $y = x$. We use different colors for the dots depending on their proximity to the diagonal line.

- *Group A (darkest)*: the top 50% dots that are the closest to the diagonal,
- *Group B (lightest)*: the top 50%–90% dots closest to the diagonal, and
- *Group C (medium light)*: the rest

In the graph, we can see that most of the dots are very close to the line $y = x$; more than 90% of the dots are tightly clustered in a narrow band around $y = x$. This result indicates that the posting rates of most sources are stable, at least within the RSS sources that we monitored in our experiments.

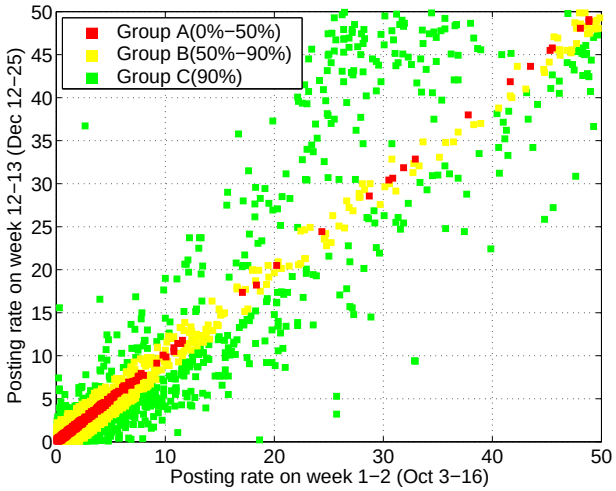


Fig. 10. Correlation between posting rate measured at different time

Learning posting pattern: We now study the optimal window size for learning the posting pattern $\lambda(t)$. For this task, as we described before, we count the number of hourly postings at every source and build a daily histogram of hourly postings. We then overlap the daily histograms for the k-week data for each source and obtain a graph similar to Figure 11, which we use as the $\lambda(t)$ graph of the source. Different k values are used to obtain this cumulative count graph. Our retrieval scheduling policy is then applied and the average delay is measured for each k value setting. The result of this experiment is shown in Figure 12 with x-axis showing the k value used to obtain the $\lambda(t)$ graph and the vertical axis showing the average delay at the given k value. The graph shows that the size of k does not affect the final delay too much, which indicates that the accuracy of the posting pattern estimation is not affected by the estimation window size much. Given this result and the result from the posting rate estimation, we conjecture that past 14-day history data is a good choice in learning both the posting rate and the pattern of each source.

D. Potential saving by push-based approaches

Other than the pull-based approach that we have mainly investigated in this paper, there can be a push-based approach where the data sources notify the aggregator whenever a new posting appears. Under this approach, the aggregator no longer needs to poll the sources regularly or maintain the posting-pattern profile of each source. Furthermore, since only new postings can be pushed to the aggregator, no resource will be wasted retrieving previously downloaded postings. In our dataset, it shows that, on average, each RSS feed contains

[8]The graph is obtained when postings are retrieved 4 times per day per feed on average. The results were similar when we use different numbers of retrievals per day.

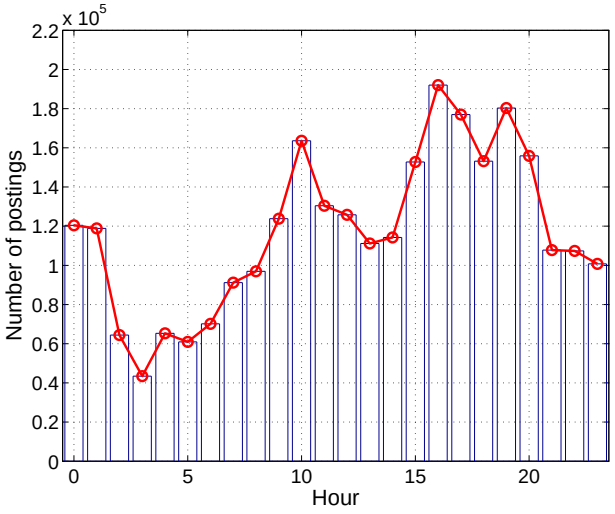


Fig. 11. Aggregated posting pattern of 5,566 RSS feeds

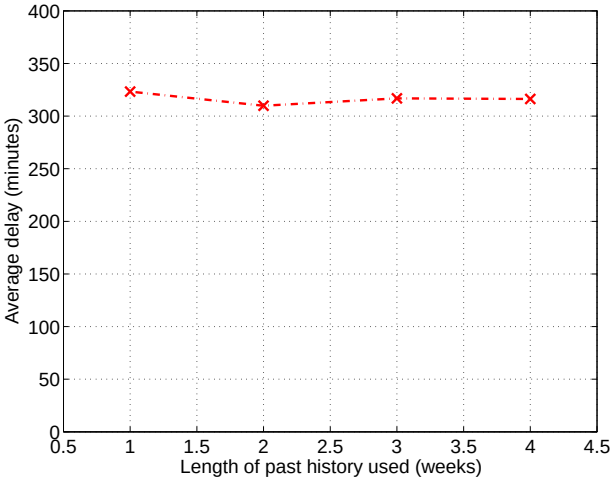


Fig. 12. Effect of different learning period of posting patterns

the 12 most recent postings on average.[9] However, each feed only generates 4.3 new postings per day on average; therefore, if the aggregator retrieves from the data sources once a day under the pull-based approach, about $7.7/12 = 64\%$ of the bandwidth will be wasted in retrieving previously downloaded postings, which will be saved when a push-based approach is employed. Furthermore, if implemented correctly, a push-based approach can potentially eliminate any noticeable delay of new postings at the aggregator, which is very difficult to achieve under the pull-based approach. For example, given our experimental results, if we want to achieve the average delay of less than an hour, the aggregator needs to contact the sources at the average rate of once every three hours under the pull-based approach, which corresponds to 8 retrievals per day. In comparison, a push-based approach will deliver a new posting to the aggregator only 4.3 times per day on average given their posting generation rate, which is roughly 50% reduction in the number of retrievals per day.

These estimates show that a push-based approach is clearly beneficial to the aggregator in both saving bandwidth and the number of retrievals. However, it remains to be seen how widely a push-based approach will be deployed on the Internet, given the dominant adoption of the existing pull-based protocols, and a number of potential problems that a push-based approach entails (such as the problem of spamming by certain RSS feeds that generate bogus spam postings in order to be shown more prominently at the aggregator and the problem of the additional cost at the sources for maintaining the list of subscribed aggregators and their preferences).

V. Related work

References [7], [9], [10], [11], [13], [14], [15], [17], [31] investigate the problem of maintaining a fresh copy of Web pages for search engines. While the general problem is similar, the exact model and overall goals are significantly different from ours. For example, references [7], [9], [10], [13], [31] assume the homogeneous Poisson model to describe Web-page changes (which does not consider the fluctuations in the change rate as discussed in Section II-A). References [5], [8] investigate the problem of minimizing the time to download *one snapshot* of the Web by efficiently distributing the downloading task to multiple distributed processes.

In more recent work [30], [40], researchers have proposed new crawling strategies to improve the user satisfaction for Web search engines by using more sophisticated goal metrics that incorporate the query load and the user click-through data. Since this body of work mainly focuses on getting improvement by exploiting the *user behavior* in the context of Web search, it still assumes a relatively simple model to predict the changes of Web pages. For example, reference [30] assumes that Web pages always change in the same manner after every refresh. We believe that a more sophisticated change model such as our periodic inhomogeneous Poisson process can further improve the result of these studies and is complementary to this body of work.

In the context of relational database, reference [17] has studied the use of periodic inhomogeneous Poisson process (referred to as Recurrent Piecewise-Constant Poisson process in the reference) to model record updates in a database. Due to the difference in the general goal and user requirements, however, its overall problem formulation and final solutions are significantly different from ours.

Researchers [14], [29] have also studied a source-cooperative approach where data sources actively *push* new changes to the aggregator. While these push-based approaches have significant benefits, it remains to be seen whether they will be widely adopted for the general Web.

There have been recent efforts to make Web crawling more efficient by improving the underlying protocol. For example, Google sitemap protocol [19] allows a site administrator to publish the list of pages available at her site at a predefined location together with the last modification dates of the pages. While this new protocol helps a crawler discover new Web pages and their changes more efficiently, it is still based on

[9]The majority of implementation is to return either the 10 or 15 latest postings.

the *pull* architecture, where a Web crawler is still responsible for periodically contacting the sites and downloading changes. Therefore, even if this protocol are widely deployed, our monitoring policy will be still helpful in reducing the retrieval delay of new postings.

Researchers have studied publisher-subscriber systems [1], [6], [16], [25], [34], [41] and proposed strategies for the efficient dissemination of information in these systems. This body of work mainly focuses on the efficient filtering of the overwhelming incoming data stream against a large pool of existing subscriber profiles, and the efficient data delivery method in the Internet scale; different from this body of work, our aggregator is not passively waiting for new data to come in; instead, the aggregator monitors and actively pulls from different data sources to collect new postings.

Researchers in the information retrieval communities have also tackled the similar problem of monitoring multiple data source but in the perspective of improving information relevance, which complements the information freshness issue addressed in this paper. Under the federated search and P2P search framework, references [26], [28], [35] have proposed algorithms for querying a subset of data sources while maintaining a high quality of search result. Reference [21] has provided a theoretical study on the trade-off between wait-time in querying the underlying data sources and providing reasonable quality of results to users. In terms of efficient content delivery, reference [39] has extended the publish/subscribe model on DHT network that considers data and language model in data placement process.

Google Alerts [18] and a number of blog aggregation services [2], [3] provide ways for users to subscribe to a set of news sources and get notified of any new articles. Unfortunately, the details of their implementation is a closely guarded secret. Besides, many interesting WWW applications, such as blogging and semantic web [23], semantic information retrieval based on XML data [12], and the recently emerged Mashup [27] technology can all benefit from a more efficient dissemination and exchange of content in the XML/RSS format. We believe that our work can be helpful to further improve these systems by providing the formal foundation and a disciplined solution to the delay minimization problem in order to provide timely service.

VI. Conclusion

In this paper we have investigated the problems related to an RSS aggregator that retrieves information from multiple RSS sources automatically. In particular, we have developed a new RSS monitoring algorithm that exploits the non-uniformity of the generation of new postings and is able to collect new data efficiently using minimal resources. Our results have demonstrated that the aggregator can provide news alerts significantly faster than the best existing approach under the same resource constraints. In addition, based on the analysis of the collected RSS data, it suggests that the posting rate follows a power-law distribution, and that the posting rate and pattern remain fairly stable over time. Finally, we have estimated the potential benefit of a push-based approach to the aggregator by measuring its saving in bandwidth and the number of retrievals per day.

The ability to provide timely information to Web users is of high commercial value to a Web service provider in both attracting user traffic and mining user behavior. We believe that existing RSS aggregators and blog search engines will benefit from the proposed monitoring policy to provide up-to-date information to users.

Acknowledgement

This work is partially supported by NSF (CNS-0626702, IIS-0534784, IIS-0347993) and ETRI. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding institutions. The authors would like to thank the reviewers for their valuable comments to improve this paper.

References

[1] Mehmet Altmel and J. Michael Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *VLDB Conference*, 2000.
[2] Bloglines. `http://www.bloglines.com`.
[3] Blogpulse. `http://www.blogpulse.com`.
[4] B.E. Brewington and G. Cybenko. How Dynamic is the Web. In *WWW Conference*, 2000.
[5] Andrei Z. Broder, Marc Najork, and Janet L. Wiener. Efficient URL Caching for World Wide Web Crawling. In *WWW Conference*, 2003.
[6] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD Conference*, 2000.
[7] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to Improve Freshness. In *SIGMOD Conference*, 2000.
[8] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *WWW Conference*, Honolulu, Hawaii, May 2002.
[9] Junghoo Cho and Hector Garcia-Molina. Effective Page Refresh Policies for Web Crawlers. *ACM TODS*, 28(4), 2003.
[10] Junghoo Cho and Hector Garcia-Molina. Estimating Frequency of Change. *ACM TOIT*, 3(3), August 2003.
[11] Junghoo Cho and Alexandros Ntoulas. Effective Change Detection Using Sampling. In *VLDB Conference*, 2002.
[12] Jennifer Chu-Carroll, John Prager, Krzysztof Czuba, David Ferrucci, and Pablo Duboue. Semantic Search via XML Fragments: A High-Precision Approach to IR. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 2006.
[13] Edward G. Coffman, Jr., Zhen Liu, and Richard R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1), 1998.
[14] Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. Adaptive Push-Pull: Disseminating Dynamic Web Data. In *WWW Conference*, 2001.
[15] Jenny Edwards, Kevin McCurley, and John Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. In *WWW Conference*, 2000.
[16] Francoise Fabret, Arno Jacobsen, Francois Llirbat, Joao Pereira, and Ken Ross. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *SIGMOD Conference*, 2001.
[17] Avigdor Gal and Jonathan Eckstein. Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach. *Journal of the ACM*, 48(6):1141–1183, 2001.
[18] Google Alerts. `http://www.google.com/alerts`.
[19] Google Sitemaps Protocol Beta. `https://www.google.com/webmasters/sitemaps/docs/en/about.html`.
[20] D. Gruhl, R. Guha, David Liben-Nowell, and A. Tomkins. Information Diffusion Through Blogspace. In *WWW Conference*, 2004.
[21] Kartik Hosanagar. A Utility Theoretic Approach to Determining Optimal Wait Time in Distributed Information Retrieval. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 2005.

[22] Thorsten Joachims. Optimizing Search Engines using Clickthrough Data. In *SIGKDD Conference*, 2002.

[23] David R. Karger and Dennis Quan. What Would It Mean to Blog on the Semantic Web. In *The Internaional Semantic Web Conference*, 2004.

[24] Ronny Lempel and Shlomo Moran. Preditive Caching and Prefetching of Query Results in Search Engines. In *WWW Conference*, 2003.

[25] Ling Liu, Calton Pu, and Wei Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE TKDE*, 11:610–628, 1999.

[26] Jie Lu and Jamie Callan. Federated search of tex-based digital libraraies in hierarchical peer-to-peer networks. In *Proceedings of ECIR*, 2005.

[27] Duane Merrill. Mashups: The new breed of Web app, 2006. `http://www-128.ibm.com/developerworks/library/x-mashups.html`.

[28] Henrik Nottelmann and Norbert Fuhr. Evaluating Different Methods of Estimating Retrieval Quality for Resource Selection. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 2003.

[29] Chris Olston and Jennifer Widom. Best-Effort Cache Synchronization with Source cooperation. In *SIGMOD Conference*, 2002.

[30] Sandeep Pandey and Christopher Olston. User-Centric Web Crawling. In *WWW Conference*, 2005.

[31] Sandeep Pandey, Krithi Ramamritham, and Soumen Chakrabarti. Monitoring the Dynamic Web to respond to Continuous Queries. In *WWW Conference*, 2003.

[32] RSS 2.0 Specification. `http://blogs.law.harvard.edu/tech/rss`.

[33] RSScache. `http://www.rsscache.com`.

[34] Shetal Shah, Shyamshankar Dharmarajan, and Krithi Ramamritham. An Efficient and Resilient Approach to Filtering and Disseminating Streaming Data. In *VLDB Conference*, 2003.

[35] Luo Si and Jamie Callan. Relevant Document Distribution Estimation Method for Resource Selection. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 2003.

[36] Ka Cheung Sia and Junghoo Cho. Efficient Monitoring Algorithm for Fast News Alert. Technical report, UCLA, 2005. `http://oak.cs.ucla.edu/~sia/pub/alert.pdf`.

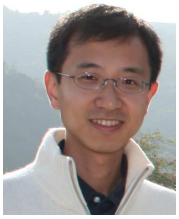
[37] H.M Taylor and S. Karlin. *An Introduction To Stochastic Modeling*. Academic Press, 3rd edition, 1998.

[38] Technorati. `http://www.technorati.com`.

[39] Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis. Publish/Subscribe Functionalit in IR Environments using Structured Overlay Networks. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 2005.

[40] J.L. Wolf, M.S. Squillante, P.S. Yu, J. Sethuraman, and L. Ozsen. Optimal Crawling Strategies for Web Search Engines. In *WWW Conference*, 2002.

[41] Tak Yan and Hector Garcia-Molina. The SIFT information dissemination system. *ACM TODS*, 24(4):529–565, 2000.



Ka Cheung Sia is currently a PhD student in the Department of Computer Science, University of California, Los Angeles. He received a MPhil degree in Computer Science and a BEng degree in Information Engineering both from the Chinese University of Hong Kong in 2003 and 2001 respectively. His current research is on web data management and mining, with a focus on weblog evoluation and infomration propagation among blog, storage and access method for concurrent document authoring.

Junghoo Cho is an assistant professor in the Department of Computer Science at University of California, Los Angeles. He received a Ph.D. degree in Computer Science from Stanford University in 2002 and a B.S. degree in physics from Seoul National University in 1996. His main research interests are in the study of the evolution, management, retrieval and mining of information on the World-Wide Web. He publishes research papers in major international journals and conference proceedings. He is an editor of IEEE Internet Computing and serves on program committees of top international conferences, including SIGMOD, VLDB and WWW. He is a recipient of the NSF CAREER Award, IBM Faculty Award, Okawa Research Award and Northrop Grunmann Excellence in Teaching Award.

Hyun-Kyn Cho is an Team Leader, Principle Member of Engineering Staff in the Fundamental Intelligent Robot Research Team at Electronic and Telecommunications Research Institute, Korea. He received a Ph.D. degree in Management Information System from Hannam University in 1997 and a M.A. degree in Management Information System from Korea University in 1990. He has conducted R&D about computer integrated manufacturing system and electronic commerce systems, intelligent e-business systems, semantic web services technology for the past 15 years. Now, his main research interests are in the study of the fundamental intelligent robotic technology in ubiquitous robot and web robot technology.