Efficient Monitoring Algorithm for Fast News Alert

Ka Cheung Sia Junghoo Cho
{kcsia,cho}@cs.ucla.edu
UCLA Computer Science Department
Los Angeles, CA 90095

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. While the subscribers rely on news feeders to regularly pull articles from the Web sites, the aggregated effect by all news feeders puts an enormous load on many sites. In this paper, we propose a blog aggregator approach where a central aggregator monitors and retrieves new postings from different data sources and subsequently disseminates them to the subscribers to alleviate such a problem.

We study how the blog aggregator should monitor the data sources to quickly retrieve new postings using minimal resources and to provide its subscribers with fast news alert. Our studies on a collection of 10K RSS feeds show that, with proper resource allocation and scheduling, the blog aggregator provides news 50% faster than the best existing approach and also reduces the load on the monitored data sources by a significant amount.

1. 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 [20]. In essence, RSS is a *pull*-based protocol, where individual subscribers have to regularly contact Web sites to retrieve new postings, using programs such as *news feeders*.

As the popularity of the RSS feeds and news feeders grows, however, they have started to put an enormous load on many sites, endangering the future of RSS-based services [23, 24]. That is, since each news feeder contacts every subscribed Web site constantly – often more than once every hour – many sites have experienced enormously high loads. In several instances, some sites had to shut down their RSS feeds completely due to the increased traffic that they could not handle [21, 22].

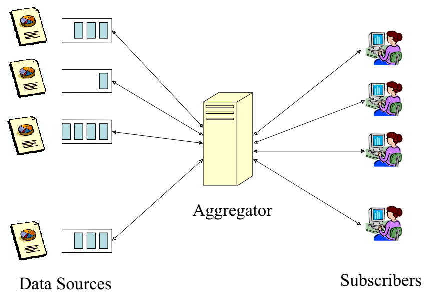


Figure 1. Framework of an information aggregator.

This surge in the traffic may be handled simply by increasing the bandwidth for a small number of sites operated by big organizations. However, for the majority of sites that are operated by individuals or small organizations, this solution is unrealistic because they do not have technical expertise or financial resources to manage a large-scale Web site. In fact, the growing popularity of personal weblogs (often called blogs) is due to the fact that *individuals* could have posted many interesting articles even before traditional media caught on to the news. These personal sites, by their nature, are not designed to handle a large volume of traffic and often fail when the traffic exceeds a certain level.

1.1. Blog aggregator

As a potential solution to this problem, we propose and study an *aggregator approach* shown in Figure 1, where a central aggregator collects new postings from the original RSS feeds and users retrieve new postings indirectly from the central aggregator. This approach has a number of advantages over the existing architecture:

1. *Offloading traffic from the sites*: The underlying sites are shielded from the direct user traffic and do not suffer from increased popularity or "Slash-

dot effects" [27]. Only the aggregator has to scale as more users subscribe. Upgrading a single aggregator will clearly be easier and more manageable than upgrading hundreds of thousands of sites managed by independent individuals. Having delegated scalability issues to the aggregator, the individual sites can focus on generating high-quality content.

2. *Collecting important user statistics*: It is possible to collect statistics on how users access and use the RSS feeds, which can be used to improve the overall experience of RSS feed users. For example, the aggregator may monitor the popularity of individual blogs and the set of users who subscribe to them, and use this information to recommend users with a few "suggestions" for the "hot" RSS feeds that may be interesting to users.

1.2. Challenges and contributions

While clearly beneficial, one important challenge for an aggregator is being able to quickly retrieve new postings from the sites to minimize the delay from the publication of a posting at the site to its appearance at the aggregator; otherwise, users would rather go directly to the original sites to obtain the most recent postings. In this paper, we study how we can minimize this delay without incurring excessive overload to the sites.

The problem of delay minimization is similar to the traditional Web-crawling problem in the literature, where Web crawlers have to maintain an up-to-date local copy of remote Web pages. However, there are two main differences that distinguish the current problem from the crawling problem: (1) Our goal is to retrieve *new* postings early while the goal of a Web crawler is to maintain *existing* copies of Web pages "fresh". This difference makes the overall optimization goal distinct, leading to a significantly different retrieval policy. (2) The expected retrieval intervals are significantly different between the two systems; For traditional Web crawlers, it is acceptable to index a new Web page within, say, a month of its creation, but for many applications based on the RSS protocol (such as personal weblogs or newsfeeds), it is imperative to retrieve new postings within hours, if not minutes, of its publication. As we will see later, this difference fundamentally changes how we should model the generation of new postings and provides new opportunities for improvement.

In this paper, we investigate the potential of our proposed aggregator approach for RSS feeds. In particular, we make the following contributions in this paper:

- In Section 2, we describe a formal framework to this problem. In particular, we propose a periodic inhomogeneous Poisson process to model the generation of postings at the RSS feeds.
- In Section 3, we investigate the optimal ways to retrieve new postings from individual RSS feeds.
- In Section 4, we examine the general characteristics of the RSS feeds available on the Web using data collected over three months from 10K RSS feeds. We also evaluate the effectiveness of our retrieval policies using the experimental data. Our experiments show that our policy significantly reduces the retrieval delay compared to the best existing policies.

2. Framework

The primary goal of this paper is to develop mechanisms that allow users to access new postings quickly from RSS feeds without overloading the Web sites. Potentially, there exist two ways to approach this problem:

1. *Extend the RSS protocol*: The current RSS protocol does not provide an efficient mechanism for a subscriber to retrieve only new postings since her last retrieval. By extending the protocol to make this possible (e.g., *"Return everything since 2:00PM May 10th, 2005"*), we may significantly reduce redundant downloads.[1] Even further, we may adopt a new *push-based* protocol, where the sources actively notify the subscribers of new postings, so that we can entirely avoid the periodic checking of the Web sites by subscribers.

2. *Build a new layer of service*: Using the existing protocol, we may build a new layer of service that can alleviate the problem. As we described in the introduction, for example, we may build an RSS-feed aggregator that collects new postings from Web sites and lets users access them centrally.

As it is for any changes to a widely adopted standard, changing the existing RSS protocol is practically very difficult, especially because under the push-based protocol, the Web sites have to take the additional responsibility of keeping track of the last visit time of their subscribers and the items the users are subscribed to. In the rest of this paper, therefore, we primarily focus on the aggregator approach. Later in the experiment section, however, we will also measure the potential savings from the described changes of the RSS protocol to see whether the changes are worth the effort.

1 For the general HTTP protocol, a similar extension has been proposed to help Web caches, but it is not being widely used due to limited browser supports.

2.1. Architecture and terminology

As shown in Figure 1, we consider a distributed information system that consists of n *data sources*, a single *aggregator* and a number of *subscribers*. The data sources constantly generate new pieces of information referred to as new *postings*. The aggregator periodically collects the most recent k postings from each source.[2] A subscriber, in turn, retrieves the 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. 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.) Note that under this model, each retrieval is assumed to use the same amount of resources. It is straightforward to extend our model to the variable cost case.

Retrieval delay The primary goal of the aggregator is to minimize the delay between the appearance of a posting at the source and its retrieval by the aggregator. The notion of retrieval delay can be formalized 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

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

□

For illustration, we show an example evolution of the delay in Figure 2(a). (Ignore other subfigures in Figure 2 for now.) 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 figure shows the delay associated with the data source over time. Note that after the generation of t_2, the delay increases twice as rapidly as before. This increase in delay is because two new postings, t_1 and t_2, are pending at the source until they are retrieved by the aggregator.

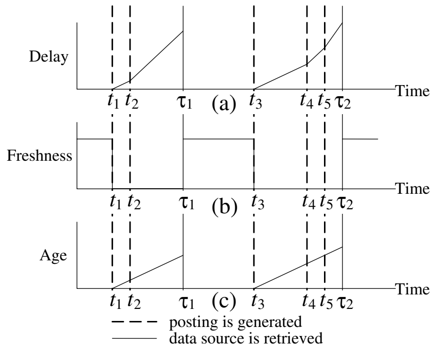


Figure 2. Illustration of the delay, freshness, and age metrics

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

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

Delay minimization problem Given the definitions, we can formalize the problem of delay minimization as follows. The notation t_{ij} is used for the jth posting generation time at O_i and τ_{ij} for the jth retrieval time from O_i by the aggregator.

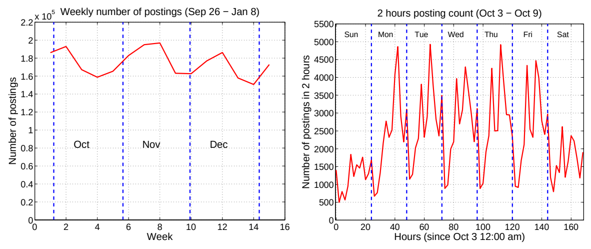
PROBLEM 1 Given the posting generation times t_{ij}'s, find the retrieval times τ_{ij}'s that minimize the total delay $D(A) = \sum_{i=1}^{n} w_i \, D(O_i)$ under the constraint that the aggregator can schedule a total of M retrievals. □

2.2. Posting generation model

Note that 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 *learn* the general posting pattern of each source based on its past history and *guess* the future posting times from the pattern.

In the context of Web crawlers, researchers have proposed that a *homogeneous* Poisson process with a rate

2 k is typically in the range of 10–15



(a) homogeneous Poisson model

(b) periodic inhomogeneous Poisson model

Figure 3. Posting rate at different resolution.

λ is a good model to be used in this context [4, 6]. Roughly, a homogeneous Poisson process is a stateless and time-independent random process where events occur with the same probability (or rate) λ at every time point [28]. In our context, we may apply this model by assuming that a data source O_i generates a new posting at the same rate λ_i at every time instance. That is, $\lambda(t) = \lambda_i$ at any t for O_i.

Researchers have shown that this model is appropriate especially when the time granularity is longer than one month [4, 6]. 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 our 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 postings is reasonably stable at roughly 180,000 postings per week, which matches with the homogeneous Poisson assumption. Based on this homogeneous model, researchers have derived the optimal re-download algorithms for Web crawlers [6, 8].

Unfortunately, when the time granularity is much shorter than one month, there exists strong evidence that the homogeneous Poisson model is no longer valid [2, 14]. 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 λ 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 generic 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, we mainly use the periodic inhomogeneous Poisson model in the rest of this paper.

2.3. Comparison with previous crawler research

We briefly compare our delay metric to other metrics in the literature used for Web crawlers [4, 7, 6, 5, 8, 16, 19]. For Web-crawler optimization, some commonly used metrics are *freshness* and *age* [4, 7]. *Freshness* is a zero-one metric indicating whether a local copy of a page is same as the one in the original Web sites. *Age* is a monotonic increasing metric indicating the time elapsed since the first modification that is not reflected in the local copy. The main difference between these metrics and our delay metric is that other metrics are mainly concerned about changes to *existing* Web pages, while our delay metric is about the retrieval of *new* postings. Assuming that the publication of a new posting corresponds to a change to an existing Web page, Figure 2(a), (b), and (c) show the time evolution of delay, freshness, and age metrics, respectively. Note that our delay metric increases more rapidly as new postings are published, while the age metric increased just linearly as time goes on. This difference, combined with the fundamental difference in our posting generation model, allows significant improvement as we will see later.

More recent metrics, such as the quality metric in [19], try to model the freshness *perceived by the users* to improve the effectiveness of search results. These metrics, however, are not directly applicable to our context because they are specifically optimized for the Web-search context.

2.4. 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*

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

3. 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 quickly 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, once at midnight and once at noon) or it may schedule both retrievals during the day when there are likely to be more new postings.

In Section 3.1, we first study the resource allocation problem. In Section 3.2, we then investigate the retrieval-scheduling problem.

3.1. 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.

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

$$m_i = k\sqrt{w_i \lambda_i} \qquad (1)$$

where k is the proportionality constant satisfying $\sum_{i=1}^{n} k\sqrt{w_i \lambda_i} = M$. □

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, $D(O_i) = \frac{\lambda_i T}{2m_i}$, and the total weighted delay, $D(A)$, is

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

$D(A)$ can be minimized by using the Le'grange multiplier method.

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

If we rearrange the above equation, we get

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

3.2. Retrieval scheduling

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

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.

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.

We start our discussion with the simple case when only one retrieval is allocated per period in Section 3.2.1. We then extend our analysis to more general cases in Section 3.2.2.

3.2.1. 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 is minimized, τ satisfies the following condition:*

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

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 equations, we get

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

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.

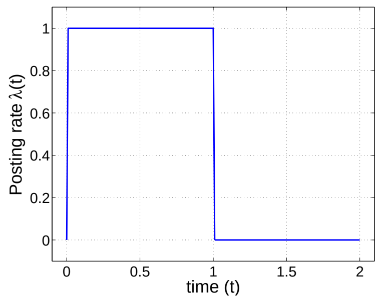


Figure 4. A data source going through periods of high activity and low activity

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., $\tau = 0$, which makes the delay $\frac{3}{2}$), we get a factor of 3 improvement. $\square$

3.2.2. 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:*

$$\lambda(\tau_j)(\tau_{j+1} - \tau_j) = \int_{\tau_{j-1}}^{\tau_j} \lambda(t)dt, \qquad (2)$$

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). $\square$

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

$$\begin{aligned} D(O) &= \sum_{i=1}^{m}\int_{\tau_i}^{\tau_{i+1}} \lambda(t)(\tau_{i+1} - t)dt \\ &= \sum_{i=1}^{m}\left(\tau_{i+1}\int_{\tau_i}^{\tau_{i+1}} \lambda(t)dt\right) - \int_{\tau_1}^{T+\tau_1} \lambda(t)tdt \\ &= \sum_{i=1}^{m}\left(\tau_{i+1}\int_{\tau_i}^{\tau_{i+1}} \lambda(t)dt\right) - \int_0^T \lambda(t)tdt. \end{aligned}$$

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

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

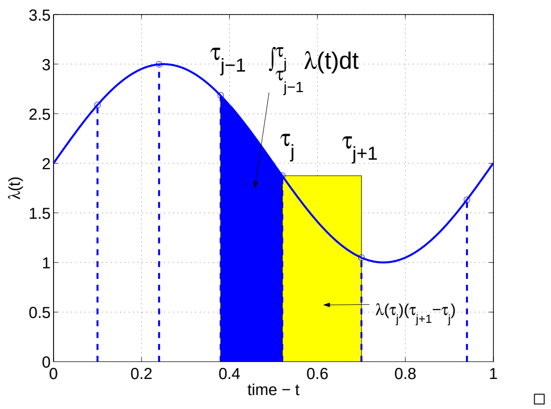


Figure 5. The optimal schedule for 6 retrievals per period for data source with posting rate $\lambda(t) = 2 + 2\sin(2\pi t)$.

By rearranging the equation, we get

$$\lambda(\tau_j)(\tau_{j+1} - \tau_j) = \int_{\tau_{j-1}}^{\tau_j} \lambda(t)dt. \quad \blacksquare$$

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 m 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 2 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 example suggests two methods for computing the optimal retrieval points.

1. *Exhaustive search with pruning*: Once the first two retrieval points are determined, the remaining retrieval points are derived automatically from Equation 2. 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.
2. *Iterative refinement*: Initially, we place the 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_4 slightly to the left to compensate for it. (More precise formulations on how much we need to shift the retrieval points are given in the extended version of this paper [26].)

In our experiments, we find that both methods lead to reasonable performance in finding the optimal retrieval points when a time granularity of 30 minutes is used.

4. Experiments

In this section, we evaluate the performance of our retrieval policies based on real data collected from RSS feeds.

4.1. 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. Consistent with the architecture mentioned above, new postings are added to the feed at any time without notifying the subscribers; thus, the subscribers have to poll the RSS feeds regularly and check for updates. We have started archiving a list of 12K RSS feeds collected from the Web since September 2004 by downloading them 4 times a day. Out of the 12K feeds, 9,634 (about 80%) have at least one posting within the three-month period between September 2004 and December 2004. We focus on this subset of 9,634 RSS feeds in the following experiments.

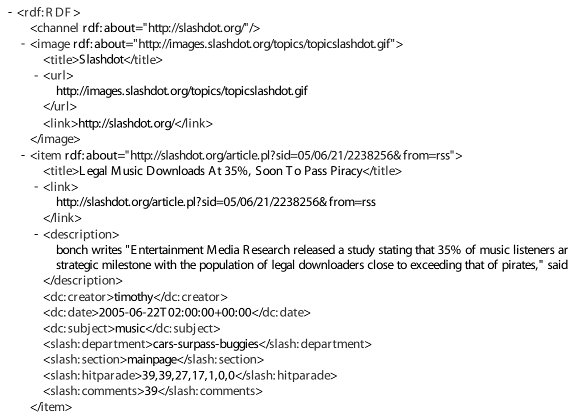

```
- <rdf:RDF>
    <channel rdf:about="http://slashdot.org/"/>
  - <image rdf:about="http://images.slashdot.org/topics/topicslashdot.gif">
      <title>Slashdot</title>
    - <url>
        http://images.slashdot.org/topics/topicslashdot.gif
      </url>
      <link>http://slashdot.org/</link>
    </image>
  - <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 music listeners ar
        strategic milestone with the population of legal downloaders close to exceeding that of pirates," said
      </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>
```

Figure 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.[4]

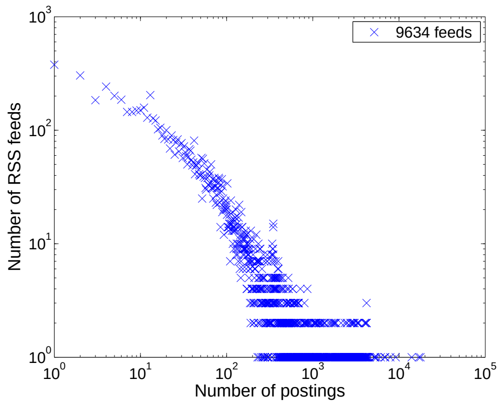


Figure 7. Distribution of posting rate of 9,634 RSS feeds

4.2. Learning posting rates

In order to implement our resource allocation policy, the aggregator has to estimate the average posting rate λ_i of each source. Intuitively, the posting rate can be estimated by observing how many postings are generated by a source within a particular period of time. We refer to this period of estimation as the *estimation window*.

Clearly, there exists a tradeoff in choosing the size of the estimation window; if the window is very small, the estimated rate may be inaccurate due to the randomness in the posting generation, but if the window is very large and if the posting rate itself changes over time, the estimated rate from the past history may be different from the current posting rate.

To explore this tradeoff and learn the optimal estimation window length, we run the following experiment: 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.

Figure 8 shows the average delay of postings for different k values.[5] The graph shows that as the estimation

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

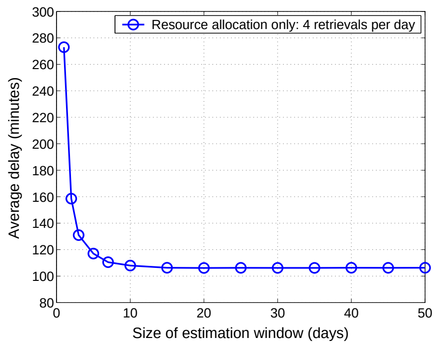


Figure 8. The effect of estimation window width.

window gets longer, the average delay decreases. We believe this improvement is due to the increased accuracy of the estimated posting rate. Beyond the window size of 14 days, however, we do not observe any improvement, which suggests that we achieve a reasonably accurate estimation of the rate from the 14-day data. The fact that delay does not increase after 14-day window suggests that the posting rate of a source does not change significantly over time.

To further investigate changes in the posting rate, we plot the following two graphs:

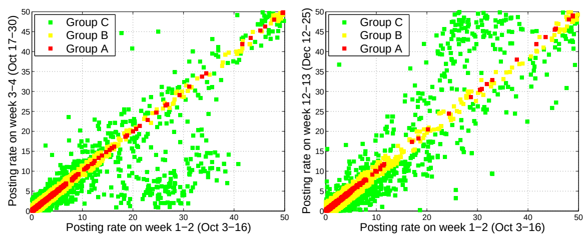
- We calculate the posting rate of each source using the first 14-day trace and use it as the x-coordinate. We then calculate the posting rate again based on the succeeding 14-day trace and use it as the y-coordinate. Based on the two coordinates, we draw a x-y scatter plot. If the posting rate remains the same between the two 14-day intervals, all dots should be aligned along the line $y = x$. Figure 9 (a) shows the graph.
- We select the first and the last 14-day traces and draw a similar x-y scatter plot (Figure 9 (b)). This graph shows the stability of posting rates during the 3-month period of our experimental data.

In the figures, we use different colors for the dots depending on their proximity to the diagonal line.

- *Group A (dark red)*: the top 50% dots closest to the diagonal,
- *Group B (light yellow)*: the top 50%–90% dots closest to the diagonal, and
- *Group C (green)*: the rest

The two graphs in Figure 9 show 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

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



(a) two consecutive periods

(b) periods separated by 2 months

Figure 9. Correlation between posting rate measured at different time.

$y = x$. This result indicates that the posting rates of most sources are stable, at least within the 3-month period of our data.

4.3. Learning the posting patterns

In order to implement our resource scheduling policy, the aggregator has to learn the posting pattern of each source (more precisely, the shape of $\lambda(t)$ of each source). In Section 2.2, we showed that similar posting patterns are repeated every day as a result of daily periodicity of people's activity. Given this, we use $T = 1$ day as the period of the posting pattern.[6]

Again, the posting pattern of a source should be learned based on its past history. To learn the pattern, we overlap the hourly posting counts everyday for k-week data of each source and obtain a cumulative hourly-posting graph similar to the one shown in Figure 10. We then use this cumulative count graph as the $\lambda(t)$ of the source.

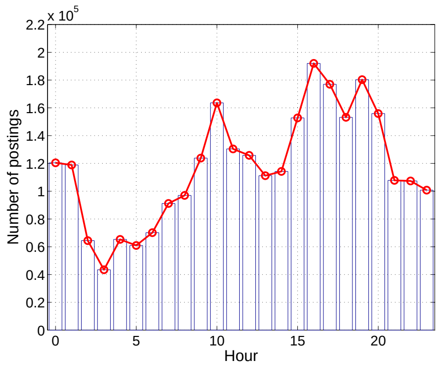


Figure 10. Aggregated posting pattern of 5,566 RSS feeds.

6 We also observe weekly fluctuation of posting rates, but we mainly focus on the daily pattern in this section.

Similar to the estimation of average posting rate, there may exist similar tradeoffs in deciding how much data to overlap; a small k value may lead to inaccuracy, while a large k value may not reflect changes in posting patterns. Again, to address this issue, we use different k values to obtain the cumulative graph, apply our retrieval scheduling policy, and measure the average delay at the end of our experiments. The result of this experiment is shown in Figure 11. The graph shows that the size of k does not impact the final delay too much; The delay does not change significantly for $k = 1, 2, \ldots, 4$. 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.

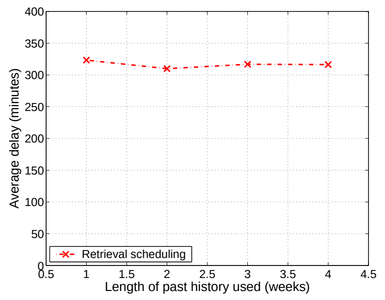


Figure 11. Effect of different learning period of posting patterns.

4.3.1. Posting pattern clusters From our investigation, we also find that a large number of sources have very similar posting patterns. To exploit this similarity, we decide to group the sources into a small number of clusters of similar posting patterns and find the optimal retrieval scheduling based on the cluster that a source belongs to.

K-means method is used to cluster the posting patterns. First, the first 2-week data are used to construct individual posting pattern of every feed as we did for Figure 10. Each feed is then represented by a 24-dimension vector, where each dimension represents the percentage of daily postings generated within that particular hour. The K-means clustering algorithm is applied on this dataset and cluster centroids are used as the representative pattern of the cluster. We test on different K values and find that $K = 12$ is a good choice because most of the patterns found beyond 12 clusters tend to be similar to others.

The most frequently occurring 6 out of 12 pattern are shown in Figure 12. The horizontal axis shows the time of the day (in hour) and the vertical axis shows the fraction of postings generated in each hourly period. The result shows that quite diverse posting patterns exist in our RSS collection. For example, the locations of the peaks are quite different among the clusters, sometimes occurring in the morning, sometimes in the afternoon. Moreover, it shows that some feeds show a bursty behavior in the posting generation within a very limited time window, like 10AM–12PM and 11PM–1AM.

These 12 posting-pattern clusters are used in our experiments in the next section to determine the optimal retrieval schedule of each RSS feed.

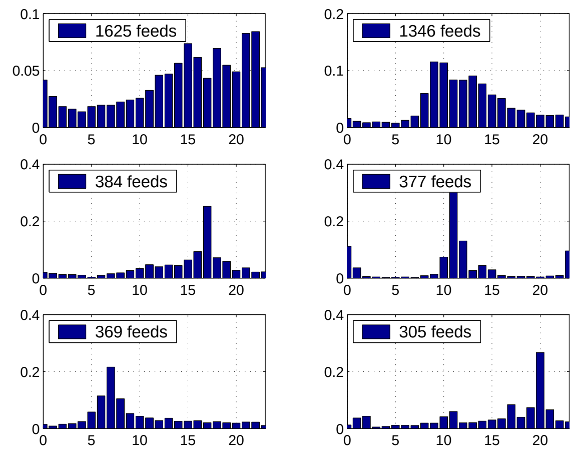


Figure 12. Six major posting patterns.

4.4. Effectiveness of retrieval policy

We now study the effectiveness of our proposed retrieval policies. To measure the improvement from individual retrieval decisions, we compare the performance of the following 4 retrieval policies:

1. *Uniform scheduling*: All sources are retrieved the same number of times and the retrieval points are scheduled at uniform intervals. The result from this policy can be considered as the baseline.
2. *Retrieval scheduling only*: All sources are retrieved the same number of times, but the retrieval points are optimized based on our scheduling algorithm.
3. *Resource allocation only*: We retrieve postings different numbers of times depending on the source, but the retrieval points are scheduled evenly.
4. *Combined*: The sources are retrieved different number of times. The retrieval points are also optimized using our scheduling algorithm.

Based on our earlier results, the first 2-week data are used to learn the posting rates and posting patterns, and the remaining 76 days are used to simulate the retrievals and to compute the average delay under different resource constraints. The results are shown in Figure 13. The horizontal axis shows the average number of retrievals per day and the vertical axis shows the overall average delay at the given resource constraint.

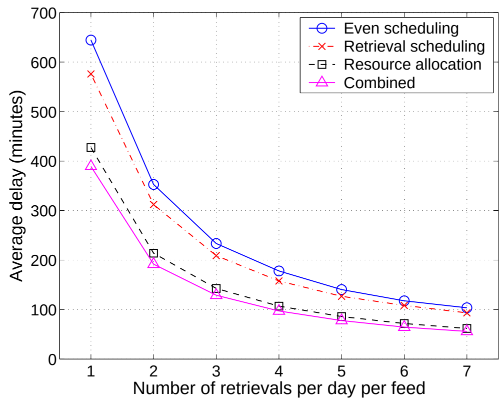


Figure 13. Performance of 4 retrieval policies under different resource constraints.

This result shows that the retrieval scheduling (2) alone reduces the delay by 12% compared to the uniform scheduling (1). The resource allocation (3) alone reduces the delay by 33%. When combined together (4), we observe about 40% reduction in delay.

While the resource allocation and the retrieval scheduling policies are both 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 the sources of low importance) and 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 for the sources of low posting rates under this policy. In contrast, the retrieval scheduling policy improves the delay simply by selecting the best retrieval time without reallocating resources, so the maximum delay is not affected by this policy. To illustrate this point, Table 1 shows the average and the maximum delays for the previous four strategies assuming one retrieval per day per source on average. We can see that the maximum delay of strategy 2 (retrieval scheduling only) is the same as that of strategy 1 (uniform), while strategy 3 (resource allocation) shows a significant increase in the maximum delay. Given this result, when it is important to keep a tight bound on the maximum delay, we may decide to employ the retrieval scheduling policy only.

4.5. Comparison with prior work

In this section, we compare the result from our retrieval policy against the policies proposed in the literature. In particular, we compare against the two op-

strategy	1	2	3	4
average delay (in min)	645	581	433	395
max delay (in min)	1440	1440	9120	10073
standard deviation	392	405	542	560

Table 1. Statistics breakdown of posting delay using one retrieval per day.

timal crawling policies described in [5]. For the comparison, we implement the two policies in [5], measure the resulting overall average delay, and show the results in Table 2. In obtaining the delays, we assume one retrieval per day per feed. The two rows CGM03(Age) and CGM03(Freshness) show the delays from the optimal age policy and the optimal freshness policy in the paper, respectively. For ours, we use the *Combined* policy from the previous section.

strategy	average delay (in min)
Ours	395
CGM03(Age)	590
CGM03(Freshness)	40,105

Table 2. Comparison with CGM03 policy

From the table, we can see that the optimal freshness policy shows significantly worse delay than the other two policies. This high delay is because the freshness policy decides to ignore the sites with high posting rates, as is well documented in [5]. The optimal age policy shows significantly better delay than the freshness policy, but still shows 50% more delay than our *Combined* policy. The improvement of our policy comes from both the fact that it can exploit the daily fluctuation of posting rates[7] and that it is specifically optimized for the delay metric.

4.6. Savings from protocol changes

We now briefly investigate the potential savings in bandwidth from a protocol change that allows the retrieval of new postings since the user's last visit. Our data shows that the average size of a posting is 560 bytes, and each RSS feed returns 12 most recent postings on average. Out of the 12 postings from each feed, we find that only about 4.3 postings are new after one day on average. Therefore, if users retrieve new postings once a day, the protocol change can avoid downloading 7.7 postings on average, reducing the bandwidth consumption by $7.7/12 = 64\%$.[8] Clearly, the savings will be more significant if users make more retrievals per day. This estimate shows the clear benefit of this protocol change and suggests that this change may be worthwhile to pursue.

5. Related work

There exists a large body of literature on Web-crawler research [7, 9, 10, 15, 12, 4, 6, 5]. In spirit, the problem setting of the crawler research is similar to ours, but the exact models and the overall goals are significantly different. For example, reference [4, 6, 5] assume the homogeneous Poisson model to describe Web-page changes (which does not consider the fluctuations in the change rate as discussed in Section 2.2) and develop strategies to optimize freshness or age of existing Web pages. In this paper, we propose the periodic inhomogeneous Poisson model to capture daily fluctuations in the generation of new postings and study the problem of delay optimization, which is more appropriate in our context.

In recent work [19, 29], more sophisticated goal metrics have been proposed to improve the freshness *perceived by users*; this is done by carefully optimizing the crawling strategy based on the query load and user click-through data. Since these studies are specifically designed for search engines, however, their goal metrics and crawling strategies are not directly applicable to our context. Also, in these studies, a page is assumed to change identically after every download; we believe that a more sophisticated change model, such as our periodic inhomogeneous Poisson model, can further improve the results of these studies.

Reference [18] proposes the *divergence* metric, which is similar to our delay metric; interestingly, the final optimization ends up quite different from ours because of the fundamental difference in the underlying architecture. The reference assumes a source-cooperative architecture, where data sources actively notify the clients of any changes, while we assume a *pull* architecture, where *passive* data sources are periodically contacted by clients.

Researchers have also studied publisher-subscriber systems [1, 3, 11, 17, 25, 30] and proposed strategies for the efficient dissemination of information in these systems. This body of work mainly focuses on the efficient filtering of incoming data stream against a large pool of existing subscriber profiles; differently from this body of work, our aggregator is not passively waiting for new data to come in; instead, the aggregator actively pulls from different data sources to collect new postings.

7 The policies in [5] is derived based on the assumption that the posting rate remains the same over time for each source

8 We assuming that the increased size of the protocol header is negligible compared to the size of the postings.

Google Alerts [13] provides ways for users to subscribe to a set of news sources and get notified of any new articles through an email. Unfortunately, the details of Google's implementation are closely guarded secret; we believe our work provides the formal foundation to the delay minimization problem and investigates important issues in this context in the open literature.

6. Conclusion

In this paper we have proposed and investigated the problems related to an RSS aggregator that retrieves information from multiple RSS sources automatically. It off-loads the bandwidth consumed at the RSS sites and allows users a central access to new information. 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 alert significantly faster than the best existing approach under the same resource constraints. In addition, an empirical analysis has shown that 2 weeks worth of data is good enough to learn and predict the characteristics of data generation in existing RSS feeds. It also shows that incorporating the if-modified-since mechanism in RSS avoids retrieval of redundant postings and significantly reduces the bandwidth consumption.

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 providing an aggregated information portal is a promising direction to pursue given the growth of both information and users on the Internet.

References

[1] M. Altmel and J. M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *VLDB Conference*, 2000.

[2] B. Brewington and G. Cybenko. How Dynamic is the Web. In *WWW Conference*, 2000.

[3] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD Conference*, 2000.

[4] J. Cho and H. Garcia-Molina. Synchronizing a database to Improve Freshness. In *SIGMOD Conference*, 2000.

[5] J. Cho and H. Garcia-Molina. Effective Page Refresh Policies for Web Crawlers. *ACM TODS*, 28(4), 2003.

[6] J. Cho and H. Garcia-Molina. Estimating Frequency of Change. *ACM TOIT*, 3(3), August 2003.

[7] J. Cho and A. Ntoulas. Effective Change Detection Using Sampling. In *VLDB Conference*, 2002.

[8] E. G. Coffman, Jr., Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1), 1998.

[9] E. Cohen and H. Kaplan. Refreshment Policies for Web Content Caches. In *INFOCOM Conference*, 2001.

[10] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive Push-Pull: Disseminating Dynamic Web Data. In *WWW Conference*, 2001.

[11] F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, and K. Ross. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *SIGMOD Conference*, 2001.

[12] A. Gal and J. Eckstein. Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach. *Journal of the ACM*, 48(6):1141–1183, 2001.

[13] Google Alerts. `http://www.google.com/alerts`.

[14] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information Diffusion Through Blogspace. In *WWW Conference*, 2004.

[15] A. Labrinidis and N. Roussopoulos. Update Propagation Strategies for Improving the Quality of Data on the Web. In *VLDB Conference*, 2001.

[16] A. Labrinidis and N. Roussopoulos. Balancing Performance and Data Freshness in Web Database Servers. In *VLDB Conference*, 2003.

[17] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE TKDE*, 11:610–628, 1999.

[18] C. Olston and J. Widom. Best-Effort Cache Synchronization with Source cooperation. In *SIGMOD Conference*, 2002.

[19] S. Pandey and C. Olston. User-Centric Web Crawling. In *WWW Conference*, 2005.

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

[21] RSS growing pains. `http://www.infoworld.com/article/04/07/16/29OPconnection_1.html`.

[22] Microsoft flip-flop may signal blog clog. `http://news.com.com/Microsoft+flip-flop+may+signal+blog+clog/2100-1032\%_3-5368454.html?tag=nefd.lede`.

[23] PubSub. `http://www.pubsub.com`.

[24] My Yahoo! `http://my.yahoo.com`.

[25] S. Shah, S. Dharmarajan, and K. Ramamritham. An Efficient and Resilient Approach to Filtering and Disseminating Streaming Data. In *VLDB Conference*, 2003.

[26] K. C. Sia and J. Cho. Efficient Monitoring Algorithm for Fast News Alert. Technical report, UCLA, 2005.

[27] Slashdot Effect. `http://en.wikipedia.org/wiki/Slashdot_effect`.

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

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

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