# Monitoring RSS Feeds based on User Browsing Pattern

Ka Cheung Sia[*] and Junghoo Cho
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA
{kcsia,cho}@cs.ucla.edu

Koji Hino, Yun Chi, Shenghuo Zhu, and
Belle L. Tseng
NEC Laboratories America
10080 N. Wolfe Rd. SW3-350
Cupertino, CA 95014 USA
{hino,ychi,zsh,belle}@sv.nec-labs.com

## Abstract

RSS has been widely used to disseminate information on the Web over the years. With the help of RSS feed readers, a user may subscribe to the feeds that are published by her favorite blogs, news channels, or Websites, and access the most recent content from these information sources. However, when the size of the subscription list grows over time, it becomes less manageable for the user to catch up with the most up-to-date information. In this paper, we propose a *Personal Information Manager* that helps a user monitor the pool of information sources in her subscription list and recommends relevant articles based on her browsing history. In particular, in order for the manager to provide the most up-to-date content, we propose a retrieval scheduling algorithm that allocates limited system resources in an optimal way based on the user's previous access pattern. Experiments show that our scheduling algorithm significantly improves the freshness of content when compared to other scheduling algorithms which do not take into account a user's behavior.

## 1. Introduction

Weblogs (or Blogs) have been growing rapidly in the past several years. They have become information sources which are as important as the traditional mainstream media. Users often want to follow closely the new postings of their favorite blogs. With the help of RSS feed readers, users can subscribe to the RSS feeds published by their favorite blogs, news channels, or Websites and access the latest content through the reader. However, when the subscription list expands over time and contains hundreds of different sources, it becomes less manageable [1] for users to catch up with the latest news from their subscription pool.

To face this challenge, we have built the prototype of a *Personal Information Manager* that adds more intelligence to an ordinary RSS feed reader. The Personal Information Manager, whose high-level architecture is given in Figure 1, serves an individual user by collecting recent information from the user's subscription pool that matches her personal interest and presents the content in a summarized form. Our main motivation to build such a personal system, as opposed to a server-based system, is to preserve users' privacy. Although a server-based system may improve the relevance and accuracy of recommendations through collaborative filtering techniques, some users are skeptical about the privacy policy and are unwilling to reveal their interests to a centralized server.

In such a Personal Information Manager, there are constraints on the available system resources, such as limited network resources and disk storage. Under such constraints, the performance of the system, such as the scalability of the system and the content coverage of the summarization, relies heavily on the effectiveness of the RSS feeds' retrieval algorithm. As a result, designing an efficient scheduling algorithm for the *Crawler* module becomes a crucial issue. In this paper, we focus on the design of a scheduling algorithm that allocates the limited network resources efficiently so that the content can still be provided to the user in a timely fashion.

Our main contributions in this paper are the following:

- We give a high-level description of the Personal Information Manager that we have implemented, which assists users in managing their large pool of RSS feed subscriptions built over time.
- We develop an intelligent scheduling algorithm that optimizes the retrieval of RSS feeds based on a user's access pattern in order to provide timely content.
- We study the user access patterns and source posting patterns and show some of their general characteristics. We further demonstrate that by taking the user browsing patterns into account, our scheduling algorithm significantly improves the "freshness" of content provided to the user by the Personal Information Manager.

The rest of the paper is organized as follows. In Section 2, we describe the overall architecture of the Personal Information Manager. In Section 3, we give a mathematical model for measuring the performance of a scheduling algorithm. In Section 4, we develop the optimal retrieval scheduling algorithm that exploits both a user's access pattern and source posting patterns. In Section 5, we evaluate the performance of our algorithm by experimenting with a real data trace. In Section 6, we discuss related work in the literature and, in Section 7, we give conclusions.

## 2. Framework

Figure 1 shows the major components of our Personal Information Manager prototype. The system is comprised of four major modules: the *Web Access Recorder* module records every page browsed by the user; the *Learner* module discovers the user's interest and her access pattern; the *Crawler* module is responsible for downloading and archiving RSS feeds in the subscription list; the *Recommendation provider* module takes the browsing history and the content retrieved from the RSS feeds to generate a succinct summary specific to the user. More details of the four components are given below.

[*] This work is done when the author is in affiliation with NEC Laboratories America

- *Web Access Recorder* - This module records and stores the Webpage access history of the user into a file for further analysis by other modules. The information recorded includes the URL of the Webpages browsed by the user, the content of the Webpages, the time stamp of each access, the referring URL if it exists, etc.

- *Learner* - This module applies several existing text-mining algorithms, such as TF-IDF clustering and Latent Dirichlet Allocation (LDA) [2] on the set of browsed documents, and represents the user's interests as bags of words. Besides, it also maintains the user access pattern and the data posting patterns of each RSS feed being monitored, which will be explained in Section 4.3 in detail later.

- *Crawler* - Based on the user access pattern and the data posting patterns learned by the *Learner* module, the *Crawler* module schedules when to download the RSS feeds in the subscription pool. It also parses and indexes the downloaded content by using some open-source tools, such as Lucene [3], and stores them locally for later access.

- *Recommendation Provider* - Based on the user interests which have been learned, this module matches the collected RSS postings with the user profile (currently, this is done by querying the local index of the downloaded RSS postings with the bags of words found by the *Learner*), and retrieves relevant postings to be displayed.

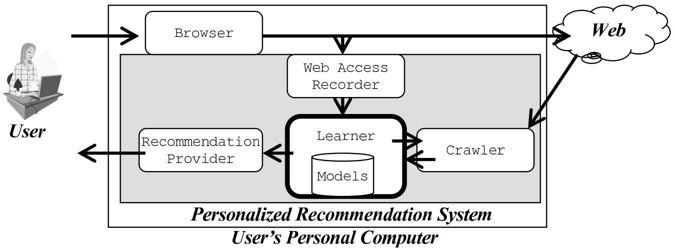

**Fig. 1:** *Overall architecture of the Personal Information Manager*

To seamlessly integrate these modules into the Personal Information Manager, we chose to implement and package the modules into a Firefox browser plugin. In this plugin, all four modules function in the background while the user browses the Web as usual. This Firefox browser plugin is implemented primarily in JavaScript and Java. The front-end uses JavaScript and XUL for browser object communication. The back-end consists of Java objects that use several open source tools including Lucene [3] for indexing browsed Webpages and crawled RSS feeds, ROME parser [4] for handling of XML content, and Jetty [5] as an embedded servlet container that serves recommendations to users.

With the system architecture given, in the rest of this paper, we will mainly focus on issues related to the *Crawler* module, including how to model and measure the performance of the *Crawler* module's scheduling methods, how to efficiently schedule the retrievals, and how the user access pattern and source posting patterns affect the scheduling.

## 3. Retrieval model

In this section, we propose a mathematical model to formalize the retrieval scheduling of RSS feeds into a constrained optimization problem.

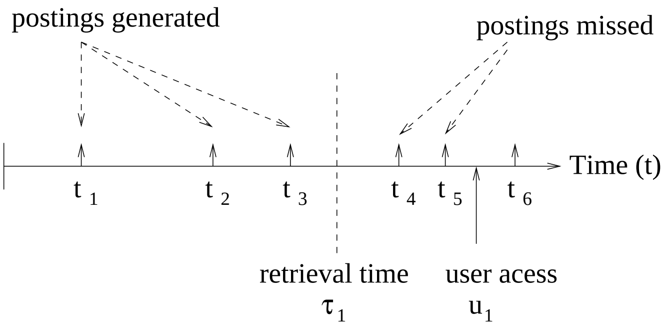

**Fig. 2:** *Illustration of relationship between number of articles missed, retrieval time and user access time*

Consider the scenario where a user has subscribed to a large number of RSS feeds[1] and she uses the Personal Information Manager to monitor this subscription pool. While the blogs, news channels, and Websites generate new information and assemble them into their RSS feeds (referred as data sources hereafter), the *Crawler* module (referred as the crawler hereafter) determines when to retrieve postings from these data sources to store them locally, which will be consumed by the user when she accesses the summary produced by the *Recommendation provider* module afterwards.

This task is similar to, yet inherently different from, the crawling problem faced by general Web search engines. This is because the Personal Information Manager runs on an ordinary desktop computer under an environment with very constrained network resources. Also, the effectiveness of crawling is very dependent on an individual user's access behavior; for example, suppose RSS feeds are retrieved at 3pm daily, two users may have very different experiences if one accesses the content right after 3pm and the other right before 3pm. As a consequence, we have to schedule retrievals of the RSS feeds in a way that is tailored for a specific user. Two of the questions we have to answer include: How to allocate resources among the RSS feeds in order to match the user's specific interest? How to optimize the crawl schedule based on the user's access pattern? In the following, we describe a mathematical model of the crawling task and we introduce several key concepts that lead us to derive an optimal crawl schedule.

### 3.1 Penalty

Figure 2 illustrates a scenario where a data source generates new postings at times $t_i's$, while the crawler retrieves them at time $\tau_1$, and the user accesses the local copies (as retrieved by the crawler) at time $u_1$. Apparently, the user will miss two recently generated postings ($t_4$ and $t_5$) at $u_1$ since the local copies were retrieved before $t_4$.

Under this scenario, we may consider several *penalty* metrics to evaluate the performance of the crawler. For example, the *delay* metric [6] measures the time elapsed between the generation of a posting and its retrieval by the crawler; alternatively, we may use a *miss* metric that considers the number of postings missed by the user when she looks at the local copies, since the crawling was done before these postings were generated; finally we may use a *miss-delay* metric that measures the difference between the user's access time and the article generation time only for the articles that are missed by

[1] Although the majority of RSS feed reader users subscribe to less than one hundred feeds, we expect such subscription lists may grow over time, similar to the case of social bookmarking, especially when the users are provided with a more efficient method to access content from the subscription.

the user. For example, their corresponding values in the scenario of Figure 2 are given as follows:

- *delay* : $(\tau_1 - t_1) + (\tau_1 - t_2) + (\tau_1 - t_3)$,
- *miss* : $|\{t_4, t_5\}| = 2$
- *miss-delay* : $(u_1 - t_4) + (u_1 - t_5)$.

In this paper, we choose *miss* as our penalty metric because it appears to represent a user's experience in a more intuitive sense for this Personal Information Manager scenario. We use the symbol $M(O, U)$ to represent this penalty metric — the number of postings missed by the user $U$ from the data source $O$.

### 3.2 Resource constraints and problem definition

Typically, the Personal Information Manager cannot make a large number of retrievals from each data source in order to keep the penalty zero, particularly when the subscription pool contains hundreds or thousands of RSS feeds; such constraints can be imposed both by the data sources and the user. This constraint is often expressed as a limit on the number of retrievals ($N$) to be made within a period of time (e.g. one day). Among these $N$ number of retrievals, more can be allocated to the data sources which generate more postings or to those in which the user is more interested. This can be mapped as a well-studied resource allocation problem where we need to determine how many retrievals should be allocated to each data source. In this paper, we assume that we have already decided on the number of retrievals to be made from each data source (using the algorithms in the literature [7, 6]), and focus on the problem of deciding the exact retrieval time for each source, so that we can minimize the penalty:

PROBLEM 1 Given the posting times $t_{i's}$, user access times $u_{k's}$ and a fixed number of retrievals $N$, find the retrieval times $\tau_{j's}$ such that the *penalty*, $M(O, U)$, experienced by the user is minimized. □

Intuitively, if we know when the user accesses the postings, we can reduce the penalty metric to zero by scheduling a retrieval right before every access by the user. For example, in Figure 2, we may schedule $\tau_1$ before $u_1$ but after $t_5$ to make the penalty zero. Unfortunately, when we have to make the retrieval decision, we do not know when the postings are generated at the source and when the user will access the local copies. The best that we can do, then, is trying to "predict" when these events may happen based on the past patterns of the posting generation and the user accesses and schedule the retrievals based on this prediction. In the next section, we describe how we model the posting generation and the user accesses for this prediction.

### 3.3 User access and posting pattern

To investigate whether we can observe a certain pattern in the user's access behavior, in Figure 3, we show a particular user's Web-page access activities during a 2-week time period. Each bar in the figure represents the number of Webpages accessed by the user within the corresponding 2-hour period. From the figure, we can observe certain periodicity: the user makes significantly more accesses during the day than in the night and during the weekdays than on the weekends. To illustrate this periodic fluctuation more clearly, in Figure 6 we show the average number of Webpages accessed per hour by each user within one day when we overlap the access history over 2 weeks. The graphs show fluctuations that seem to correspond to the user's work schedule (e.g. lunch and dinner breaks, sleeping habits, etc.). Such observations suggest that a periodic inhomogeneous Poisson process [8] may be a good approximation for such recurring and fluctuating pattern. Roughly, a periodic inhomogeneous Poisson process is a Poisson process with varying rate $\lambda(t)$, where $\lambda$ is a function of time $t$. Moreover, $\lambda$ is assumed to exhibit regular periodicity $T$ such that $\lambda(t) = \lambda(t - nT)$ for any integer $n$. Given the regularly recurring patterns in the user's access history, we believe this model is a good fit to describe user accesses.

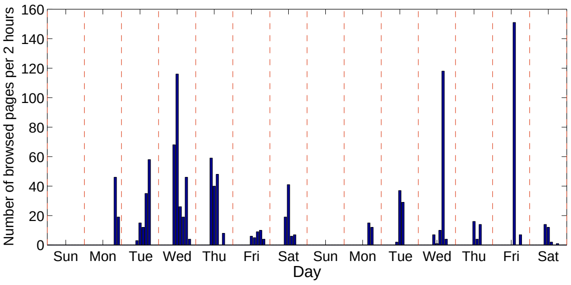

**Fig. 3:** *A sample 2-weeks user access pattern.*

For the generation of the postings at the sources, we also observe similar periodicity and fluctuation (as was reported in the literature [9, 6]). Therefore, we also use a periodic inhomogeneous Poisson process to model the generation of the postings.

### 3.4 Expected penalty

Since our model is probabilistic, we cannot know for sure the exact time of posting generation and user access. Therefore, to minimize the penalty, the best that we can do is to estimate the *expected penalty* for each scheduling policy and pick the one that minimizes this expected value. The following lemma shows how we can compute the expected penalty under our model:

LEMMA 1 *For a data source $O$ with posting rate $\lambda(t)$ and a user $U$ with access rate $u(t)$, assume we schedule the retrievals at time $\tau_{j-1}$ and $\tau_j$. Then the expected penalty experienced by the user during the time period within $[\tau_{j-1}, \tau_j]$ is as follows:*

$$\int_{\tau_{j-1}}^{\tau_j} u(t) \left( \int_{\tau_{j-1}}^{t} \lambda(x) dx \right) dt \qquad \square$$

PROOF. During a small time interval $dt$ at time $t$, there are $u(t)dt$ number of user accesses. Each access will miss $\int_{\tau_{j-1}}^{t} \lambda(x)dx$ number of posts. Therefore, the total expected *penalty* experienced by the user between $\tau_{j-1}$ and $\tau_j$ is $\int_{\tau_{j-1}}^{\tau_j} u(t)(\int_{\tau_{j-1}}^{t} \lambda(x)dx)dt$.

In the following sections, we use $\Lambda(t)$ and $U(t)$ to denote the integrals $\int_0^t \lambda(x)dx$ and $\int_0^t u(x)dx$ respectively.

## 4. Retrieval schedule

In the following subsections, we will describe the methods to derive optimal retrieval schedules based on the definitions and metrics we introduced in the previous section. We first analyze a simple case, where we can schedule only one retrieval per each period $T$, and generalize it to the case where we are allowed to schedule multiple retrievals per period.

### 4.1 Single retrieval

Consider a data source $O$ with the periodic posting rate $\lambda(t)$ and a user $U$ with a periodic access rate $u(t)$, where both share the same periodicity $T$ (e.g., $T$ may be a day). Assume that the postings from

$O$ can only be retrieved once in each period $T$. The following theorem suggests that the best retrieval time is when the instantaneous posting rate $\lambda(t)$ is proportional to the instantaneous user access rate $u(t)$ according to a value given by $\frac{U(T)}{\Lambda(T)}$.

THEOREM 1 *When a single retrieval is scheduled at time $\tau$, the expected penalty $M(O,U)$ is minimized when $\tau$ satisfies the following conditions:*

$$\frac{u(\tau)}{\lambda(\tau)} = \frac{U(T)}{\Lambda(T)} \qquad \left( \textit{and } \frac{u'(\tau)}{\lambda'(\tau)} > \frac{U(T)}{\Lambda(T)} \right) \tag{1}$$

□

PROOF. Without loss of generality, we consider only the user accesses within a single interval $[0,T]$. The notation $M(\tau)$ represents the expected number of postings missed by the user when the retrieval is scheduled at $\tau$, where $0 \le \tau \le T$. Between $[0,\tau]$, the user will miss the postings generated between $[\tau - T, t]$; between $[\tau, T]$, she will miss the postings generated between $[\tau, t]$, where $t$ is the exact time she accesses the local copies. Therefore, the *expected penalty* when we schedule one retrieval at time $\tau$ is,

$$\begin{aligned} M(\tau) &= \int_0^\tau u(t)(\int_\tau^T \lambda(x)dx + \int_0^t \lambda(x)dx)dt \\ &\quad + \int_\tau^T u(t)(\int_\tau^t \lambda(x)dx)dt \\ &= \int_0^T u(t)\Lambda(t)dt - \Lambda(\tau)U(T) + \Lambda(T)U(\tau) \end{aligned}$$

$M(\tau)$ is minimum when

$$\frac{dM(\tau)}{d\tau} = \Lambda(T)u(\tau) - U(T)\lambda(\tau) = 0$$

and $\frac{d^2M(\tau)}{d\tau^2} > 0$. After rearranging the expressions, we get Equation 1.

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

EXAMPLE 1 Figure 4 shows a data source (blue solid line) that undergoes a period of high posting activity between $t = [0,1]$ and a period of low posting activity between $t = [1,2]$. Similarly, the user (red dashed line) also undergoes a fluctuation in her access pattern but in a reverse way of the data source. According to the theorem, the best retrieval should be scheduled at $t = 1$. This solution matches the intuition that the crawler should retrieve right after a large number of new postings are generated and before the user accesses the local copies extensively in order to avoid missing too many postings. □

## 4.2 Multiple retrievals

Now, we generalize the scenario and consider the case when multiple retrievals are to be scheduled within one period.

THEOREM 2 *When we schedule $m$ retrievals at time $\tau_1, \ldots, \tau_m$ for a data source with posting rate $\lambda(t)$ based on a user access rate $u(t)$, where both have periodicity $T$. The expected penalty is minimized when all $\tau_{j's}$ satisfy the following equation:*

$$\frac{u(\tau_i)}{\lambda(\tau_i)} = \frac{\int_{\tau_i}^{\tau_{i+1}} u(t)dt}{\int_{\tau_{i-1}}^{\tau_i} \lambda(t)dt} \tag{2}$$

*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).*

□

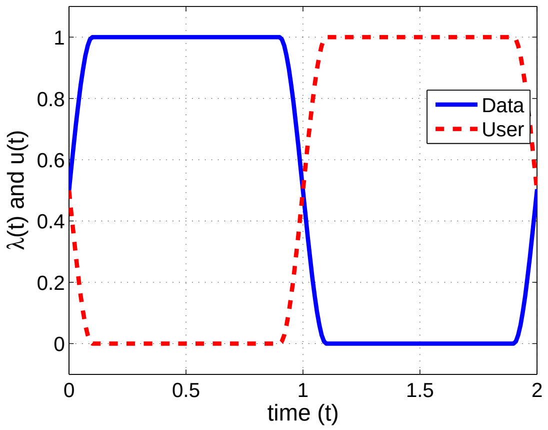

**Fig. 4:** *Example of the single optimal retrieval point.*

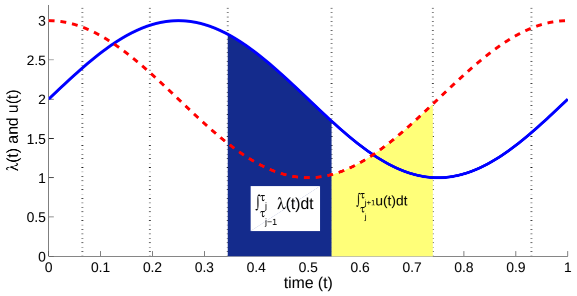

**Fig. 5:** *The optimal schedule for 6 retrievals per interval.*

PROOF. Without loss of generality, we consider the expected penalty of a user when she accesses the content between $\tau_1$ and $T + \tau_1$:

$$\begin{aligned} M(O,U) &= \sum_{i=1}^{m} \int_{\tau_i}^{\tau_{i+1}} u(t)(\int_{\tau_i}^{t} \lambda(x)dx)dt \\ &= \sum_{i=1}^{n} [(\int_{\tau_i}^{\tau_{i+1}} u(t)\Lambda(t)dt) - \Lambda(\tau_i)(U(\tau_{i+1}) - U(\tau_i))] \end{aligned}$$

The necessary condition for $M(O,U)$ to be minimum is when $\frac{dM(O,U)}{d\tau_i} = 0$ for every $\tau_i$. By rearranging the terms, we get Equation 2.

We illustrate the above theorem graphically using an example.

EXAMPLE 2 Figure 5 shows a data source (blue solid line) with the posting rate $\lambda(t) = 2 + sin(2\pi t)$ and a user access rate (red dashed line) of $u(t) = 2 + cos(2\pi t)$. Postings are retrieved from the source six times in one period. Assume that we have decided up to the $i^{th}$ retrieval point and need to determine the $(i+1)^{th}$ point. Note that the upper part of the right-hand side of Equation 2 is equivalent to the dark-shaded area in Figure 5, while the lower part of the right-hand side of Eq. 2 is equivalent to the light-shaded area of Figure 5. The theorem states that the expected penalty is minimized when $\tau_{i+1}$ is selected such that the two areas are in proportion to $\frac{u(\tau_i)}{\lambda(\tau_i)}$, which is the ratio of the instantaneous user access rate to the posting rate at time $\tau_i$. □

## 4.3 Computation of schedule

The above theorem only provides the analytical solutions to the optimal conditions. In practice, we may need to discretize the continuous time domain and schedule retrievals at discrete time points. One naive solution to find the optimal schedule is to enumerate all possible retrieval schedules and find the best one with the lowest *expected penalty*. In this section, we describe how to use the conditions derived previously to compute the optimal schedules efficiently.

Suppose we have discretized one day period into 1440 slots (assuming the resolution of retrieval time is set to be one minute). For the single retrieval case, one can use the method of bi-section to find the slot that satisfies Equation 1, which may converge to the solution faster than searching the slot linearly. For multiple retrievals per interval case, we illustrate how to utilize the condition to compute the solution as follows: Suppose we have picked the first-two retrieval points; we keep applying Equation 2 iteratively to determine the successive retrieval points. Once we have determined all retrieval points using the conditions, we then compute the *expected penalty* of this particular schedule. We repeat this procedure for all possible choices of the first-two retrieval points and choose the schedule with the lowest *expected penalty*; the pseudo-code for this computation is illustrated in Algorithm 1. This computation basically applies the necessary condition in Theorem 2 to prune out unnecessary search space rather than the method that enumerates all possible schedules and searches.

To learn the functions, $\lambda(t)$ and $u(t)$, from the past posting history and the user access history, we count the number of postings and the number of user accesses per *hour* and represent them as a 24-bin histogram. We then smooth out the 24-bin histogram by linear piecewise interpolation (or by some other smoothing functions) to obtain a discretized version of $\lambda(t)$ and $u(t)$.

We choose one hour as the bin size for the estimation of $\lambda(t)$ and $u(t)$ because when the bin size is much smaller (say, a minute), the histogram may contain many spikes, while when the bin size is much larger (say, 2 or 3 hours), the histogram may not capture the fluctuation precisely, and both will result in poor prediction accuracy.

**Algorithm 1** optimal schedule algorithm

```
smooth the input histogram with more number of bins.
for all possible position of $\tau_1$
  for all possible positions of $\tau_2$
    while desired number of retrievals not reached
      apply Eq. 2 to determine $\tau_{j+1}$
      if $\tau_{j+1}$ exceed one period then
        break
      end if
    end while
    if $\tau_{m-1}, \tau_m, \tau_1, \tau_2$ satisfy Eq. 2 then
      compute expected penalty of the schedule
    else
      continue
    end if
    compute expected penalty of the schedule
  end for
end for
select the schedule with smallest expected penalty
```

# 5. Experiment

In this section, we show some statistics of the user browsing patterns collected by the Personal Information Manager prototype and the data posting patterns of the set of RSS feeds we monitored. We then illustrate the performance improvement over existing scheduling algorithms based on this real data trace.

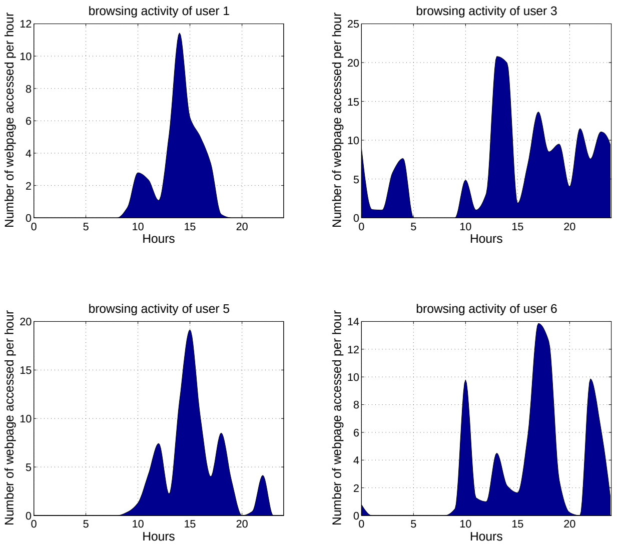

**Fig. 6:** *Samples of user browsing activity patterns*

## 5.1 Description of dataset

To obtain the browse history of users, we recruited nine users from the staff of the NEC Labs and the student of the UCLA CS department and ask them to install a trimmed-down version of the plugin described in Section 2, which records the time, the content, and the referring URL of the Webpages[2] a user has browsed. We collect their browsing activity for 3 consecutive weeks. During the same period of time, we monitor and download the postings from a collection of 1.5K frequently updating RSS feeds in the `blogger.com` domain to be used as the subscription pool and to learn the posting patterns and evaluate our scheduling algorithm. Although authors from `blogger.com` come from around the world and the posting update time specified in the RSS feeds are expressed in various timezone, in both the experiment and graphs shown in this section, they are normalized to the Pacific Daylight-savings time (PDT).

## 5.2 User access pattern

In order to acquire the "actual" user access pattern, the Personal Information Manager needs to be fully functional so it can capture the user access times of the local copies of RSS feeds. Since the prototype is still in development when we carry out the experiment, we assume, instead, a user's normal Webpage browsing activities to be her access pattern of local copies of RSS feeds.

### *Daily fluctuation*

Fig 6 shows four samples of the user access patterns obtained from the volunteers. They all demonstrate the periodicity as described in Section 3. The daily access activity overlapped over 2 weeks closely resembles a user's work schedule. Our algorithm can exploit such fluctuations to devise tailor-made crawl schedules to provide the user

[2] Certain Webpages, such as those start with *https* in the URL and those coming from well-know Web-based e-mail services, are not captured. Users are also given the option to exclude Webpages from certain domains from being captured due to privacy concern.

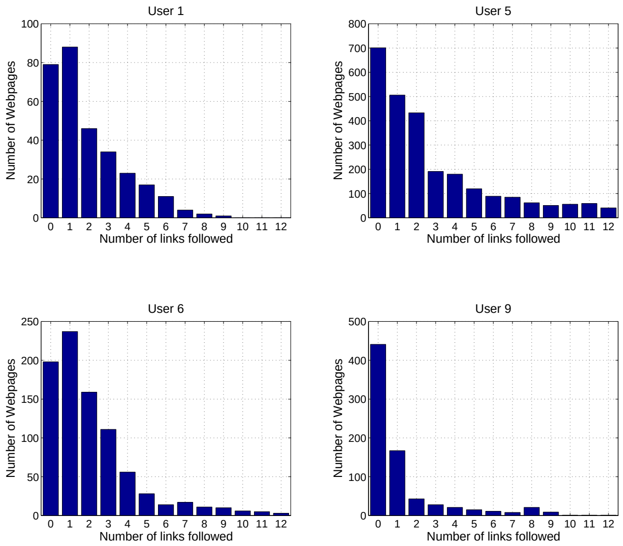

**Fig. 7:** *Samples of link following in user browsing history*

with fresh content.

### *Link following*

Other than the daily fluctuation pattern, we also investigate the link following pattern. For each page visited by the user, its referring URL, which indicates the previous URL that leads the user to the current page, is recorded.

Based on this information, we reconstruct the path preceding every page visited[3]. Figure 7 shows the distribution of pages that are accessed by following different numbers of links from four sample users. Although individual patterns vary, it indicates a general trend that the majority of the pages are accessed by following three links or less; this may be partly explained by the advance of search engines; users are becoming less likely to spend their effort browsing the Web by following links. On the other hand, for the Webpages being accessed by following a greater number of links, this may indicate that the user spends more effort to locate the information; thus, when we consider the relevance and importance of a Webpage to the user's interest, such pages may need to be weighted more.

### *Predictability of access pattern*

As the crawler optimizes retrieval schedules based on a user's access pattern, which is obtained from past history, it is important for the pattern to be predictable. To investigate this issue, we compare the correlation between the hourly number of Webpage accesses across two consecutive days in the 2 weeks browsing activities obtained. Figure 8 shows the correlation of all the 9 users' hourly Webpage access rate 24 hours apart. Suppose a user accesses 10 Webpages between 3pm to 4pm on day 1, and she accesses 15 Webpages between 3pm to 4pm on day 2, it will be recorded as $(10, 15)$ in the graph. The points are sorted according to their proximity to the diagonal line $x = y$, where the closer the points are to the diagonal, the more predictable the access pattern is. To better visualize the data, we show the strata of points that are within 50% and 90% proximity to the diagonal, and it indicates that user's access pattern is fairly predictable.

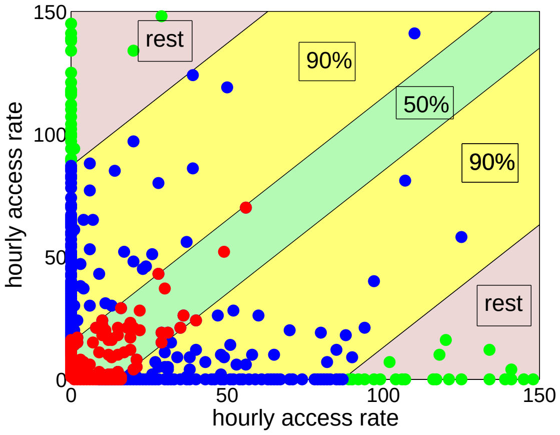

**Fig. 8:** *Correlation of user access rate in consecutive days*

## 5.3 Posting pattern

For posting pattern, each feed is represented by a 24-bins histogram which is the hourly posting rate averaged over the first two weeks. In our investigation, we observe that a large number of data sources share very similar posting patterns. To exploit this similarity, we decide to group the sources into a smaller number of clusters of similar posting patterns and compute their optimal retrieval schedules based on the class pattern, which is represented by the cluster centroid, they belong to. The effect of clustering can be two-folded: on one hand, it aligns data sources with less training instances to a more representative and common pattern instead of using a spiky and inaccurate pattern obtained from only one observation; also, it reduces the time spent to compute retrieval schedules for every source. On the other hand, it may reduce the accuracy of estimated posting patterns; thus giving a higher penalty.

The K-means method with Euclidean distance is used to cluster the posting patterns. Figure 9 shows the four major patterns found in our dataset. The graphs, to a certain extent, show different patterns of blogging activities: some blogs tend to generate postings around the clock (e.g. collaboration of several authors), while some blogs tend to have their posting time confined within a very specific time period (e.g. professional bloggers that write articles according to their regular schedule).

## 5.4 Performance evaluation

In this section, we evaluate the performance of our proposed method by comparing it with other scheduling algorithms that do not consider user access pattern as follows:

- *Uniform* - a simple method that schedules retrievals evenly spaced. For example, with 3 retrievals per day, each are spaced 8 hours apart. The majority of client-side RSS feed readers employ this policy.
- *Data only* - a method that optimizes retrieval schedules based on the *delay* [6] metric, which, in fact, is equivalent to our *penalty* metric while assuming the user access rate to be constant.
- *User+data* - our retrieval scheduling method that is optimized based on the *penalty* metric by considering both the posting patterns of data sources and the specific user access pattern.

[3] A precise and accurate reconstruction is inherently difficult because some pages are filtered due to the concern for user privacy.

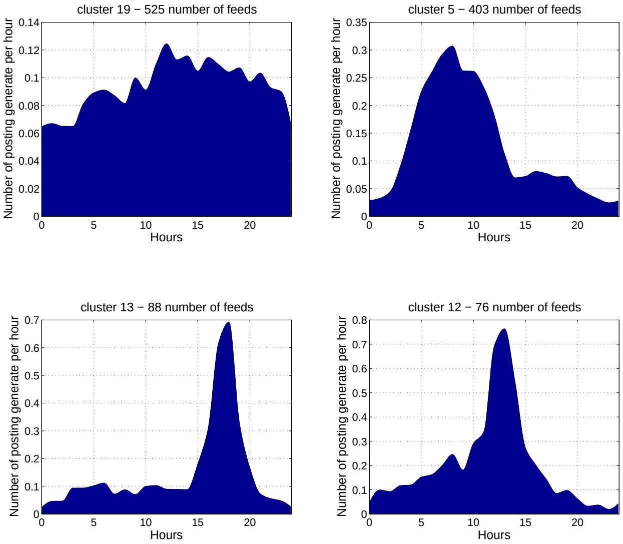

**Fig. 9:** *Samples of data posting patterns.*

We first compute the posting patterns and the user access pattern based on the data collected in the first two weeks. The posting patterns are first clustered into 20 groups and the class centroids are used as representatives to compute their optimal schedules. We then simulate the retrievals using the trace of user access times and posting times in the third week under different resource constraints (from 2 to 5 retrievals per RSS feed per day). For every user access, we compute the *penalty* (the number of postings missed in the 1.5k RSS feeds subscription pool) to evaluate the performance of each scheduling algorithm. Figure 10 shows the comparison of the three methods. To make it easier to comprehend the numbers, we scale the *penalty* metric with respect to the performance achieved by the *uniform* schedule and average the percentage reduction of *penalty* over 9 users. We observe that, in general, *user+data* reduces 40% of the number of postings missed as compared to *uniform*; when compared to *data only*, it reduces roughly 25% on average.

Another observation from the results is that the number of clusters used in grouping posting patterns does not affect the performance too much. As indicated in Figure 10, the *user+data* algorithm that uses 5 and 20 clusters shows comparable performance. This suggests that clustering posting patterns can reduce the computation of optimal schedules while not introducing too much degradation on the user's experience. This characteristic can be attributed to the fact that optimal retrieval schedules are more user-access-pattern oriented.

## 6. Related work

Web crawling is a well-studied research problem. The majority of them are oriented toward search engines that serve a large population of users [7, 10, 11, 12, 13]; thus the optimization objectives are more focused on the change characteristics of the Webpages or the collective behavior of search engine users. There are some web crawling algorithms that take the collective user behavior into account. In user-centric web crawling [14], Webpages are refreshed by considering their positions in the result list and the query frequency of their related keywords made by the users. Focused crawling [15] concentrates on how to obtain an initial crawl of the portion of the Web likely to be of interest to a particular community of users. In the model of [16], an embarrassment based metric is introduced to determine when to refresh a page by minimizing the likelihood of a user to access stale or irrelevant content in the search engine result page.

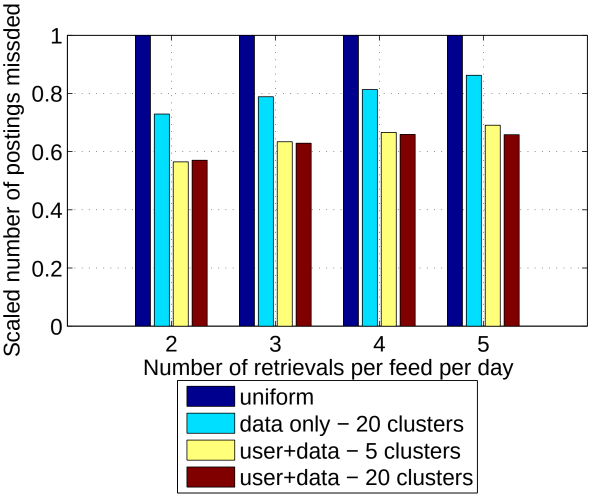

**Fig. 10:** *Comparison of three methods on the penalty metric*

The idea of a browsing assistant dates back 10 years. When following hyperlinks was the prevalent method of surfing the Web, as in contrast to the current search engine dominated era [17], Letizia[18] was a tool that recommended hyperlinks on the Webpage currently viewed by a user based on analyzing the user browsing history and pre-fetching the linked content in advance. Since then, many research thrusts [19, 20, 21] have proposed different methods to improve the relevance of recommendations to a user's browsing interest. The algorithm developed in our paper can be a good complement to them to further improve the freshness of recommendation items. Besides, a recent research in experiencing the Semantic Web [22] has benefited us in both the architectural design and implementation of the Personal Information Manager.

In terms of improving the user's browsing experience in the time perspective, pre-fetching is a technique commonly used to reduce the wait-time of loading Webpages. Such techniques can be deployed at different locations on the Web. In particular, when it is deployed on the client side [23], pre-fetching algorithms predict the links on the current page that are likely to be accessed by the user in the future and request them in advance; hence, it reduces the waiting time of the user when loading pages. In the case of deployment on the server-side[24, 25], pre-fetching works by analyzing the web access log for document access patterns; it then pre-fetches subsequent documents from disk into main memory to reduce the access time when they are requested by the clients afterwards. Similar techniques can also be deployed on proxy servers [26] which serve a collection of users within a subnet.

## 7. Conclusion

In this paper, we have proposed a Personal Information Manager that helps users to better manage the enormous amount of information that exists on the Web. In particular, we have developed a RSS feeds monitoring algorithm that exploits both the irregularity of data posting patterns and user access pattern to schedule retrievals efficiently under a network-resource constrained environment. Our results have demonstrated that the Personal Information Manager can provide the user with much more updated content when compared to existing approaches under the same resource constraints.

With the advance of artificial intelligence and easily available content publishing softwares, the increase in recommendation relevance and the diversity of content improve the usability of personal information assistants. We believe this to be a promising direction of accessing information from the Web other than the use of search engines in the foreseeable future. Also, existing RSS feed readers can also benefit from our proposed monitoring policy to deliver more up-to-date information to users while minimizing the amount of network resources used and imposing less of a burden on the RSS feed hosting sites.

## Acknowledgement

This work is partially supported by NSF grants, IIS-0534784 and IIS-0347993. 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.

## References

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

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research 3*, pages 993–1022, 2003.

[3] Lucene indexing package. `http://lucene.apache.org`.

[4] Rome: Rss and atom utilities. `http://wiki.java.net/bin/view/javawsxml/Rome`.

[5] Jetty embedded web server. `http://www.mortbay.org`.

[6] Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient Monitoring Algorithm for Fast News Alert. in IEEE Transaction of Knowledge and Data Engineering, to appear.

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

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

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

[10] Edward G. Coffman, Jr., Zhen Liu, and Richard R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1), 1998.

[11] Edith Cohen and Haim Kaplan. Refreshment Policies for Web Content Caches. In *INFOCOM Conference*, 2001.

[12] Jenny Edwards, Kevin McCurley, and John Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. In *WWW Conference*, 2000.

[13] Alexandros Labrinidis and Nick Roussopoulos. Update Propagation Strategies for Improving the Quality of Data on the Web. In *VLDB Conference*, 2001.

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

[15] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. In *WWW Conference*, 1999.

[16] 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.

[17] Junghoo Cho and Sourashis Roy. Impact of Web Search Engines on Page Popularity. In *WWW Conference*, 2004.

[18] Henry Lieberman. Letizia: An Agent That Assists Web Browsing. In *the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[19] L. Chen and K. Sycara. WebMate: A Personal Agent for Browsing and Searching. In *The second International Conference on Autonomous Agents and Multi Agent Systems*, 1998.

[20] Yoshinori Hijikata. Estimating a User's Degree of Interest in a Page during Web Browsing. In *Proc. of the Systems, Man, and Cybernetics Conference*, 1999.

[21] Marko Balabanovic. An Adaptive Web Page Recommendation Service. In *The first International Conference on Autonomous Agents*, 1997.

[22] David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *The International Semantic Web Conference*, 2005.

[23] Abdulmotaleb El-Saddik, Carsten Griwodz, and Ralf Steinmetz. Exploiting User Behavoiour in Prefetching WWW Documents. In *IDMS*, pages 302–311, 1998.

[24] E.P. Markatos. Main Memory Caching of Web Documents. In *WWW Conference*, 1996.

[25] Qiang Yang and Henry Hanning Zhang. Integrating Web Prefetching and Caching Using Prediction Models. *Journal of World Wide Web*, 4(4):299–321, 2001.

[26] Josep Domenech, Ana Pont, Julio Sahuquillo, and Jose A. Gil. An Experimental Framework for Testing Web Prefetching Techniques. In *the 30th EUROMICRO Conference*, 2004.