# Optimized Retrieval Algorithms for Personalized Content Aggregation

Dan He
IBM T.J. Watson Research
Yorktown Heights, NY
dhe@us.ibm.com

Douglass S. Parker
UCLA Computer Science Dept.
Los Angeles, CA 90095
stott@cs.ucla.edu

## Abstract

*Personalized content aggregation methods, such as for news aggregation, are an emerging technology. The growth of mobile devices has only increased demand for timely updates on online information. To reduce traffic or bandwidth, efficient retrieval scheduling strategies have been developed to monitor new postings. Most of these methods, however, do not take user access patterns into consideration. For example, the strategy for a user who checks news once a day should be different from the strategy for a user who checks news ten times a day.*

*In this paper, we propose a personalized content aggregation model in which delay time depends not only on the retrieval time and posting time, but also on user access patterns. With total expected delay as the objective, we derive a resource allocation strategy and retrieval scheduling strategy that is optimal when postings are Poisson. To our knowledge, this is the first personalized aggregation model on multiple data sources.*

## 1 Introduction

Content aggregation methods such as news aggregators have been widely used to retrieve new postings in a timely manner by subscribers of personal blogs, news sources, forums, and so forth [6] [12] [15] [18]. Recently, *personalized content aggregation* has emerged as an important class of applications on mobile devices. These apps collects new postings from established sites automatically, following some scheduling strategy. The strategy typically retrieves new postings according to the sites' posting rates, in a way that is designed to minimize retrieval delay.

There are many benefits of content aggregation. One major benefit is that users automatically obtain new information of interest, instead of having to check for updates. Another is that automatic retrieval can improve over time as users' interests are inferred from their access patterns.

By following a scheduling strategy, aggregators also can avoid unnecessary retrievals from an originating web site, which often leads to waste of computational resources. Due to its inherent scalability, aggregation is also able to reduce traffic. Nowadays certain blogs and web sites provide notifications when new content is available. This can greatly reduce unnecessary contacts from the subscribers and still keep the new content up-to-date with subscribers.

Although there has been a great deal of work on optimizing retrieval scheduling strategies for content aggregation, much less attention has been focused on personalized content aggregation. Existing scheduling algorithms for aggregators usually do not consider unique user access patterns. The objectives of personalized aggregation are centered more on user convenience, rather than on timeliness of update — pushing so as to minimize delay may not be perceived as beneficial by some users. However, if other users browse the news frequently, minimizing delay and showing updates in a timely manner can be important.

Personalized aggregation has even greater potential on mobile devices. Retrieval of information can not only lead to higher cost to mobile users, but also consume mobile device power and bandwidth. A personalized aggregator that is able to optimally schedule retrieval according to the browsing patterns of the user therefore has potentially vast importance. Recently a large number of personalized mobile aggregators have emerged, such as Trove [7], etc. These aggregators cover a rapidly increasing number of 'lifestreams' — permitting people to stay on top of multiple social networks, automate their daily routine and business interactions, monitor information (including with dashboards and mashups), etc. The basic concept of a 'news feed' is expanding in hundreds of directions.

The development of cloud services such as *Google News* [4] introduced an approach in which a single crawl serves a very large number of users. User access patterns become less important for the crawling strategy as the number of users grows. However, to satisfy requirements of certain users to see new postings as soon as possible, the crawling must be conducted quite often. No centralized service or strategy can be optimal for all users; ultimately they will need personalized aggregators.

There has been little published work on scheduling policies for personalized content aggregators. The work of Sia et al. [19] proposed a retrieval policy based on user browsing patterns. However, this work considered single data sources, where content aggregators involve multiple data sources. For retrieval scheduling algorithm on a single source, their penalty metric is the number of missing posts between a retrieval and the nearest user access afterwards. This penalty metric ignores the delay time of the missing posts, which might be important for some users.

For example, suppose we conduct two retrievals and the number of missing posts is one for both retrievals, and both missing posts involve an important opportunity with a very short information lifetime. Suppose furthermore that one missing post is a day old for one retrieval, and the other missing post is a minute old for the other retrieval. The first missed post is clearly a failure since the opportunity has been missed, but the second may not lead to any loss to the user. Another example is that users with low frequency access may not care whether news of the last ten minutes is posted or not. However if news of a few hours ago is not posted, even these users may get serious misimpressions. Therefore, delay times of missing posts can be important for any retrieval policy.

As we will show later, another advantage of minimizing the total delay of missing posts is that we are able to derive an optimal resource allocation strategy for multiple sources. It is not clear that an optimal resource allocation strategy can be derived when the objective function is to minimize the number of missing posts. Therefore, we believe the delay of missing posts has advantages as an objective function for personalized content aggregation.

The main contributions of this paper can be summarized as follows:

- We propose a personalized content aggregation model that tries to minimize total delay time of missing posts in order to capture older posts.

- We consider multiple data sources. Besides an optimal scheduling algorithm, we also propose an optimal resource allocation strategy that determines how many retrievals should be allocated to each data source (from multiple data sources, given a fixed total number of retrievals). To our knowledge, this is the first personalized aggregation model on multiple data sources.

When the frequency of a user checking the new posting is involved, we propose a user access-based retrieval delay model, where the delay time for a post is the difference between the posting time and the time the user accesses the post (we call it *user access time*), where there is no retrieval in between. (If there were a retrieval between, the delay time for the post would be zero.) This is different from the traditional retrieval delay models where the delay time is the difference between the posting time and the closest retrieval time after the posting.

With this user access-based delay model, we propose an optimal retrieval policy to minimize total expected delay for a single data source. When multiple data sources are considered, we derive an optimal resource allocation strategy for minimizing the sum of expected delays for all sources, and show the strategy is indeed effective and efficient.

## 2 Related Work

Content aggregation has attracted a great deal of attention recently due to its potential importance for mobile devices, for example in news aggregators [5]. The popularity of content aggregation has triggered tremendous research on designing more efficient retrieval algorithms. Rose et al. [17] proposed a system Cobra (Content-Based RSS Aggregator), to crawl, filter and aggregate a large number of RSS feeds. Sia et al. [18] proposed the *Retrieval Delay* metric and proposed an optimal retrieval scheduling strategy to minimize the total expected delay time for multiple sources, given limits on the number of retrievals. They assumed a pull architecture, where passive data sources are periodically contacted by clients. Horincar et al. [14] proposed a best-effort refresh strategy for content-based feed aggregation to reduce the bandwidth usage. Similar to feed aggregation, blog aggregation [1] [2] [3] also allows users to subscribe to news sources to obtain up-to-date news articles automatically.

The problem of retrieval strategy in web-crawler research differs from the feed aggregation problem here. Existing work on web crawlers [8] [10] [11] [13] has focused on strategies that maintain fresh copies of web pages for search engines. Typically the goal is to optimize metrics involving freshness or age of existing web pages [9]. Homogeneous Poisson models for web page changes are common assumptions.

There is a great deal of other work for web search engines aimed at improving user satisfaction. For example, the work of [16] and [20] studied more sophisticated metrics involving user behavior, such as query load and user click-through data. Their metrics still assume a relatively simple model, such as constant refresh rate of web pages. The periodic inhomogeneous Poisson model is more complicated and is shown below to be complementary to this work.

Retrieval strategies for feed aggregation are usually *pull-based*, i.e., the aggregator actively pulls new changes from data sources. There is also work on *push-based* methods [12] [15]. Olston and Widom [15] studied an alternative optimization problem in which the architecture is source-cooperative, so that data sources actively notify clients of changes. Push-based methods are more effective than pull-based methods in that push-based methods are able to notify clients right after changes, without delay. By contrast, pull-based methods rely on retrieval scheduling algorithms and usually incur delay. However, pull-based methods are still more widely used than push-based methods given the special requirement on the data sources to be able to deploy push-based methods.

## 3 Posting Generation Model

To design optimal scheduling strategies for new postings, we first need to see how posts are generated. Two posting generation models have been widely used:

- *Homogeneous Poisson Model* [8]: a stateless and time-independent random process, where new postings are posted at a constant rate regardless of the time.
- *Inhomogeneous Poisson Model* [18]: a process in which posting rate changes over time, with two further sub-models:
  - Periodic Inhomogeneous Poisson Model: the same rate $\lambda(t)$ is assumed to repeat over time with period $T$, namely $\lambda(t) = \lambda(t - nT)$ for $n = 1, 2, \ldots$.
  - Non-periodic Inhomogeneous Poisson Model: the most generic model, in which the periodicity in of $\lambda(t)$ does not necessarily hold.

It has been shown that which model is most appropriate depends on the time granularity used for retrievals [18]. If the granularity is month-scale, the homogeneous model is more appropriate. If the granularity is day-scale, the inhomogeneous model is more appropriate. The periodic inhomogeneous Poisson model is a good approximation for activities that have bursts during the day and are inactive at night.

## 4 Modeling User Access Patterns

Based on a particular user's web-page access activities during a 2-week time period, [19] showed that there is a clear periodic pattern for user access: the user makes significantly more accesses during the daytime than late at night, and during the weekdays than on the weekends. They claim a periodic inhomogeneous Poisson process might be a good approximation to the user access pattern. Therefore, in this work, we assume periodic inhomogeneous Poisson process for both user access and posting generation.

## 5 Metrics

In this work, we assume a retrieval captures all the updates before it and the retrieval delay of an access depends only on the updates between the previous retrieval and the access. Therefore, given an access at time $a$ and a posting at time $t$, we define the *user-access retrieval delay* for the access as:

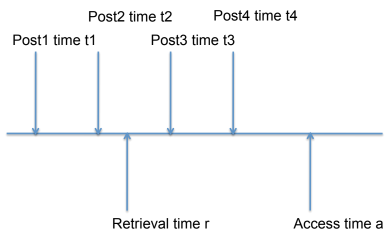


**Figure 1. Example of user-access retrieval delay.**

$$D = \begin{cases} a - t & if\ t \leq a\ and\ \nexists r \in [t, a] \\ 0 & \text{otherwise} \end{cases}$$

where $r$ is the retrieval time. For illustration purposes, an example is in Figure 1. For the retrieval, the penalty metric of [19] is 2 since there are two missing posts, post 3 and post 4. However, according to our user-access retrieval delay metric, the penalty is $(a - t_3) + (a - t_4)$. Above we've discussed cases where our metric improves on that in [19], as our metric can better take care of relatively old posts. We will show later that another advantage of our metric is that it admits an optimal resource allocation strategy, which cannot be achieved using the metric of [19].

## 6 Models

Based on the user-access retrieval delay metric, we have the following lemma:

**Lemma 1** *For a data source $O$ with posting rate $\lambda_p(t)$, assume the retrieval times are $\tau_1, \tau_2, \ldots, \tau_m$ and the access rate of a user is $\lambda_c(t)$. Then the total expected delay for the postings generated within $[\tau_{j-1}, \tau_j]$ is*

$$\int_{\tau_{j-1}}^{\tau_j} \lambda_c(t) \int_{\tau_{j-1}}^{t} (t - f)\, \lambda_p(f)\ df\ dt \qquad (1)$$

**Proof:** For a small time interval $df$ at time $f$, the number of posts is $\lambda_p(f)\, df$, thus the total number of posts from time $\tau_{j-1}$ to time $t$ is $\int_{\tau_{j-1}}^{t} \lambda_p(f)\, df$. The number of checks for a small time interval $dt$ at time $t$ is $\lambda_c(t)\, dt$, for each check at time $t$, the delay time for each post at time $f$ is $t - f$. Therefore, the total expected delay for the posting generated within $[\tau_{j-1}, \tau_j]$ is $\int_{\tau_{j-1}}^{\tau_j} \lambda_c(t) \int_{\tau_{j-1}}^{t} (t - f)\, \lambda_p(f)\, df\, dt$. ■

In the homogeneous Poisson model, the posting rate and access rate remain constant — as $\lambda_p$ and $\lambda_c$ — during the period $[\tau_{j-1}, \tau_j]$. Thus formula (1) for total expected delay for postings in this interval simplifies to

$$\frac{\lambda_c \lambda_p (\tau_j - \tau_{j-1})^3}{6}. \tag{2}$$

## 6.1 User-Access Retrieval Policy

For simplicity, assume first that our retrieval policy is under the constraint that the total number of retrievals is $M$. We then need to answer two basic questions: (1) *Resource allocation*: given $n$ sources and a time period $T$, how many times should the aggregator contact each source $O_i$? (2) *Retrieval Scheduling*: at what times should the aggregator contact the source $O_i$?

### 6.1.1 Resource Allocation

For resource allocation, we use an averaged daily posting rate and daily access rate for estimation. This naturally suggests a homogeneous Poisson process model, which assumes the posting rate and access rate are constant.

Inspired by Cauchy's inequality — in which the sum of squares is never greater than the square of sums, and equality holds when all numbers are equal — we make the same assumption for our total expected delay formula (2). We can now prove that the total expected delay for postings generated within time period $T$ is minimized when retrievals are scheduled at uniform intervals:

**Lemma 2** *Under the homogeneous Poisson model, the total expected delay for postings generated within period $T$ is minimized when retrievals are scheduled at uniform intervals.*

**Proof:** We prove this by induction. In the case where there is only one retrieval, retrieval should be conducted at time $\frac{T}{2}$, where the total expected delay is $(\frac{T}{2})^3 + (\frac{T}{2})^3$. Shifting the retrieval time point by a period of $d$, to either an earlier time or a later time, increases this delay. For example, shifting the retrieval to an earlier time yields a greater total expected delay (for $d > 0,\ T > 0$):

$$\left(\frac{T}{2} - d\right)^3 + \left(\frac{T}{2} + d\right)^3 \;=\; 2\left(\frac{T}{2}\right)^3 + 6d^2\frac{T}{2} \;>\; 2\left(\frac{T}{2}\right)^3.$$

Thus uniform retrieval is optimal in the base case. For the induction step, assume that uniform retrieval is optimal for $n-2$ retrievals. Then for $n-1$ retrievals, without losing generality, we can assume the new retrieval is the final one. For uniform retrieval, the total expected delay is $(\frac{T}{n})^3\ n = \frac{T^3}{n^2}$. If we shift the final retrieval time by a period of $d$ to an earlier time, then the $n-2$-th retrieval time point is $\frac{(n-1)}{n}\,T - d$. By induction uniform retrieval is optimal for $n-2$ retrievals, so we perform them uniformly for a time period of $\frac{(n-1)}{n}\,T - d$. The total expected delay for these $n-2$ retrievals is $(\frac{(n-1)}{n}\,T - d)^3 \,/\, (n-1)^2$. Thus the best total expected delay for the $n-1$ retrievals is:

$$\frac{(\frac{(n-1)}{n}\,T - d)^3}{(n-1)^2} + \left(\frac{T}{n} + d\right)^3. \tag{3}$$

We can show the above formula is greater than the total expected delay $\frac{T^3}{n^2}$ from uniform retrieval when:

$$\begin{aligned} & d > 0,\ n > 2,\ T > 0 \\ \textit{or}\quad & d > 0,\ 1 < n \le 2,\ T > \frac{2dn - dn^2}{3(n-1)}. \end{aligned}$$

Thus we conclude, for all $n \ge 1$, that the total expected delay is minimized by the uniform retrieval policy. ■

A user may weight multiple sources differently according to their interests in each source, or the importance of updates from each source, or some other criteria. Thus the total expected delay is a weighted linear combination of the expected delays of all sources. Under the homogeneous Poisson model, given $m$ sources $O_1, O_2, \ldots, O_m$, where $O_i$ has posting rate $\lambda_{p_i}$ and importance weight $w_i$, each source is contacted by the aggregator $n_i$ times per period $T$. If the user browses the source $O_i$ with rate $\lambda_{c_i}$, the weighted total expected delay of postings $D(A)$ is defined as:

$$D(A) = \sum_{i=1}^{m} w_i\, D(O_i) = \sum_{i=1}^{m} \frac{\lambda_{c_i}\lambda_{p_i} w_i T^3}{6n_i^2}. \tag{4}$$

To minimize $D(A)$, we use Lagrange multipliers and compute the derivative of $D(A)$ for each $n_i$ as:

$$\frac{\partial D(A)}{\partial n_i} = -\frac{\lambda_{c_i}\lambda_{p_i} w_i T^3}{3n_i^3} = -\mu.$$

Solving this equation, we obtain:

$$n_i = \left(\frac{\lambda_{c_i}\lambda_{p_i} w_i T^3}{3\mu}\right)^{\frac{1}{3}}.$$

We can then solve $\mu$ via the following equation:

$$\sum_{i=1}^{m} n_i = M \iff \sum_{i=1}^{m}\left(\frac{\lambda_{c_i}\lambda_{p_i} w_i T^3}{3\mu}\right)^{\frac{1}{3}} = M.$$

Thus for $\mu$ satisfying the above equation, we can obtain $n_i$ for each source $O_i$, which is proportional to the product of the access rate, posting rate, and the weight of the source.

### 6.1.2 Retrieval Scheduling

Once the number of retrievals is determined for a source, we need to decide upon the exact times for the retrievals.

Under the inhomogeneous Poisson model, assume that the periodicity is $T$ for source $O$. According to formula (1), for $m$ retrievals at times $\tau_1, \tau_2, \ldots, \tau_m$, the total expected user-access delay is:

$$D(O) \;=\; \sum_{i=1}^{m} \int_{\tau_i}^{\tau_{i+1}} \lambda_c(t) \int_{\tau_i}^{t} (t-f)\, \lambda_p(f)\, df\, dt. \quad (5)$$

**Lemma 3** *For source $O$, periodicity $T$, posting rate $\lambda_p(t)$, user access rate $\lambda_c(t)$ and $m$ retrievals $\tau_1, \tau_2, \ldots, \tau_m$, the expected delay $D(O)$ is minimized when all $\tau_i$'s satisfy*

$$\begin{aligned} &\lambda_c(\tau_i) \int_{\tau_{i-1}}^{\tau_i} (\tau_i - t)\, \lambda_p(t)\, dt \\ =\;& \lambda_p(\tau_i) \int_{\tau_i}^{\tau_{i+1}} (\tau_i - t)\, \lambda_c(t)\, dt \end{aligned} \quad (6)$$

*where $\tau_{m+1} = T + \tau_1$ and $\tau_0 = \tau_m - T$, $\tau_1, \tau_{m+1}$ are the first retrieval point in the current and the next interval, respectively, $\tau_0, \tau_m$ are the final retrieval point in the previous and the current interval, respectively.*

**Proof:** To minimize $D(O)$, we compute $\frac{\partial D(O)}{\partial \tau_i} = 0$ and we obtain the above equation for each $\tau_i$. ■

According to the above equation, once $\tau_{i-1}, \tau_i$ are determined, $\tau_{i+1}$ can be determined. Thus we can apply an exhaustive search for the first two retrieval points during the period $T$. For every pair of retrieval points, the remaining retrieval points can be determined consequently. Thus for every pair of retrieval points, we can obtain a delay time. We then select the pair that yields minimum delay time. We can discretize the points and consider only retrieval points with certain time granularity, such as every five minutes. Given any first two retrieval points, the remaining retrieval points can be computed according to equation (6). Due to periodicity, assume $m$ retrievals $\tau_1, \ldots, \tau_m$ are scheduled in the period $T$. The cases for the first retrieval in the next interval $\tau_{m+1}$ and the final retrieval in the previous interval $\tau_m$ which violates the above equation are pruned. Therefore the optimal schedules can be computed efficiently. For any source, once we have determined the optimal number of retrievals $m$ with the threshold model, we can apply the above strategy to locate retrieval time points.

One problem of our algorithm is in learning the posting rate $\lambda_p(t)$ and user access rate $\lambda_c(t)$ from past posting history and user access history. Sia et al. [19] suggested counting the number of postings and the number of user accesses per *hour*, and represented them as a 24-bin histogram. The bin size was set as one hour because the histogram may contain spikes when the bin size is much smaller (for example, a minute), and on the contrary the histogram may not capture the fluctuation precisely when the bin size is much larger (for example, 2 or 3 hours). They claim a one-hour bin size leads to good estimation. Therefore in this work we apply the same one-hour bin size. The 24-bin histogram is smoothed out by linear piecewise interpolation to obtain a discretized version of both rates. Our optimal scheduling algorithm is then conducted on the two smoothed histograms.

```
Algorithm Optimized Schedule
Input: Time interval $T$, posting histogram, user access histogram, number of retrievals $m$
Output: The optimal schedule with the minimum expected delay
Optimized Schedule
1. Smooth the input histograms with more number of bins
2. $N \leftarrow$ total number of bins
3. for $0 \leq \tau_1 \leq N$
4.     for $\tau_1 + 1 \leq \tau_2 \leq N$
5.         while $\tau_j \leq N$
6.             compute $\tau_{j+1}$ according to equation (6)
7.             if $\tau_{j+1} > N$ then break
8.         end while
9.         if $\tau_{m-1}, \tau_m, \tau_1, \tau_2$ satisfy equation (6)
10.             compute expected delay of the schedule according
                to equation (5)
11.         end if
12.     end for
13. end for
14. select the schedule with the smallest expected delay time
```

**Figure 2. Greedy algorithm to find the optimal number of retrievals for each source.**

We summarize the algorithm in Figure 2. As we can see on line 6, we need to calculate $\tau_{j+1}$ from $\tau_{j-1}$ and $\tau_j$ according to equation (6). In the implementation of the algorithm, $\tau_{j+1}$ cannot be computed directly. We thus consider different values for $\tau_{j+1}$ and find one such that the difference of the two sides of equation (6) is minimized. On line 7, we conduct the pruning once we observe the current scheduling is suboptimal. On line 9, we validate the periodicity of the current scheduling. In the implementation, we do not require equation (6) to be exactly satisfied. Instead, if the difference of the two sides of the equation is below a small threshold, we treat the current scheduling as valid.

## 7 Experimental Results

In this work, our experiments on content aggregation focus on RSS feed aggregation. We evaluate the performance of our retrieval strategies on real data collected from RSS feeds and we compare the performance of our strategy with state-of-the-art strategies.

### 7.1 Experimental Setup

We used the same dataset used by Sia et al. [19], which comes from a collection of 1.5K frequently updating RSS feeds in the blogger.com domain. The postings from these feeds were monitored for 3 consecutive weeks. Since the authors from blogger.com came from all over the world, the posting update times were normalized to the Pacific Daylight-savings time (PDT). In the meanwhile, the browsing activities of real users that subscribed to these feeds were recorded for the same 3 consecutive weeks. We clustered the browsing activities of the real users and selected 6 representative real users. The posting patterns and user access patterns were then learned from the data. We assigned equal weights to all the feed sources.

## 7.2 Learning Posting Rates and Posting Patterns

Since our resource allocation and retrieval scheduling strategies require knowing the feeds posting rates and posting patterns, we split the 3 weeks' data into the first 2 weeks and 1 remaining week. The feeds posting rates and patterns were learned from the first 2 weeks. We assumed the feeds posting rate and patterns would not have dramatic changes over this period and therefore applied the same rates and patterns learned from the first 2 weeks to the remaining 1 week. The period $T$ for the inhomogeneous Poisson model was set to one day since significant daily repetitive patterns were observed. Therefore the posting rate was a daily rate, and we computed the average number of postings per day for each feed source in the first 2 weeks. To learn the posting patterns, we counted the number of hourly postings for each source and built a daily histogram of hourly postings. Thus the average histogram of hourly postings in the first 2 weeks was considered as the posting pattern. The retrieval scheduling algorithm was then conducted on this pattern to schedule retrieval time points. We show the sample hourly posting histogram for four feeds in Figure 3(a). As we can see, the RSS feeds generated many new postings every day and instead of generating many postings at a specific time, they tended to generate postings at different times of the day.

## 7.3 Learning User Access Patterns

We show the sample hourly histograms for the access pattern of four users in Figure 3(b). Again they are based on the first 2 weeks of data. As we can see, most of the user accesses fell in the range of early morning, late evening and around noon.

## 7.4 Effectiveness Evaluation

We conduct two sets of experiments, one for the effectiveness of the resource allocation policy and one for the effectiveness of the retrieval scheduling algorithm.

### 7.4.1 Resource Allocation

Since the work [19] only considers retrieval scheduling for a single source and didn't propose a resource allocation policy for multiple sources, in this experiment, we only compare our method with [18], where an optimal resource allocation policy was proposed that was is not personalized and was based on only the posting pattern (we call this method *posting-only*). We also consider a baseline method, which allocates even number of retrievals to each feed (we call this method *baseline*). We call our method *user-access-delay* to differentiate from other user access based method [19] based on the number of missing posts (we call this method *user-access-missing*). To compare resource allocation policies, we first applied the three aforementioned policies and then conducted the same retrieval scheduling algorithm for each allocation such that only the resource allocation policy matters for the delay time. We fixed the total number of retrievals as 4,500 such that each of the 1.5K RSS feeds is allocated 3 retrievals for the baseline method. We assumed all the feeds were equally weighted. We computed the average total delay time for each method and show the results in Table 1.

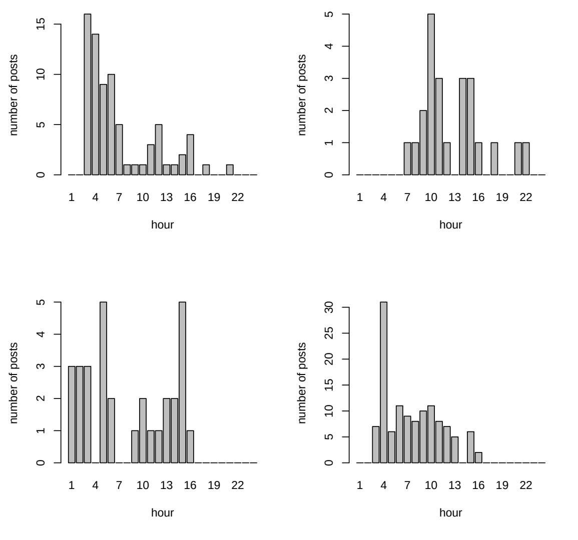
| method | user 1 | user 2 | user 3 | user 4 | user 5 | user 6 |
|---|---|---|---|---|---|---|
| *baseline* | 66.17 | 48.55 | 42.79 | 27.4 | 47.32 | 27.87 |
| *posting* | 19.98 | 29.01 | 28.85 | 21.53 | 29.73 | 19.54 |
| *delay* | 17.95 | 28.66 | 27.11 | 20.13 | 23.02 | 15.15 |

**Table 1. Comparison of delay (hours) for different resource allocation policies (baseline, posting-only [18], user-access-delay) with the same retrieval scheduling algorithm.**

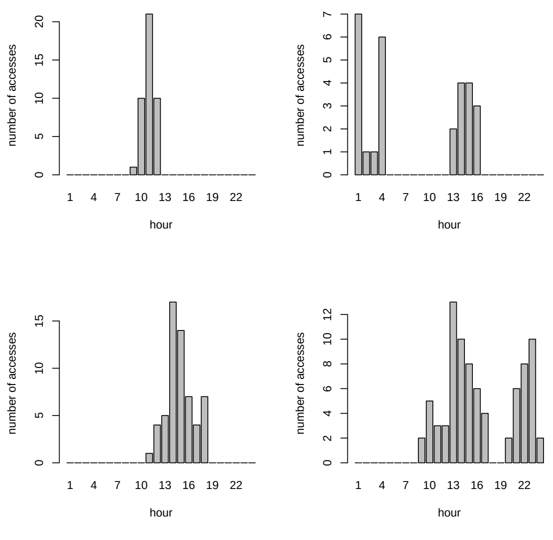
As we can see, the baseline resource allocation method clearly performed worst, since it doesn't depend on either the posting rate or the user access rate. Notice that delay is accumulated for every possible pair of post and access. The number of posts and accesses can be potentially large, such as a few hundred. Therefore the accumulated delay time can be large as well. The *posting-only* policy achieves better performance than the *user-access-delay* policy, indicating our allocation policy is indeed more effective.

### 7.4.2 Retrieval Scheduling

To illustrate the effectiveness of our retrieval scheduling algorithm, we conducted our resource allocation algorithm and then fixed the number of retrievals for each RSS feed according to our algorithm for the remaining experiments. For comparison, we conducted both algorithms *posting-only* [18] and *user-access-missing* [19] (notice now they refer to retrievals scheduling algorithms). We also considered a baseline method where for each feed we scheduled the retrievals at uniform intervals during each time period. Since the number of retrievals for each RSS feed is the same for all methods, only the scheduling algorithm matters for delay time. Again we fixed the total number of retrievals at 4,500 and assumed all feeds were equally weighted. Table 2 shows the resulting average total delay for each method.



(a)



(b)

**Figure 3. Hourly posting patterns (a) and access patterns (b) for four randomly selected feeds.**

| method | user 1 | user 2 | user 3 | user 4 | user 5 | user 6 |
|---|---|---|---|---|---|---|
| *baseline* | 402 | 273 | 534.58 | 280.84 | 161.36 | 324.31 |
| *posting* | 378 | 245.93 | 499.38 | 278.25 | 121.02 | 220.21 |
| *missing* | 19.98 | 31.63 | 41.97 | 29.52 | 25.13 | 18.87 |
| *delay* | 17.95 | 28.66 | 27.11 | 20.13 | 23.02 | 15.15 |

**Table 2. Comparison of delay (hours) for different retrieval scheduling algorithms (baseline, posting-only [18], user-access-missing [19], user-access-delay) with the same resource allocation policy.**

| method | user 1 | user 2 | user 3 | user 4 | user 5 | user 6 |
|---|---|---|---|---|---|---|
| *baseline* | 209.1 | 144.65 | 277.55 | 148.55 | 114.1 | 172.7 |
| *posting* | 176 | 127.55 | 251.4 | 139.8 | 94.45 | 121.95 |
| *missing* | 33.65 | 47.84 | 96.56 | 46.76 | 40.1 | 35.45 |
| *delay* | 35.45 | 49.53 | 100.31 | 47.58 | 41.65 | 37.55 |

**Table 3. Comparison of the number of missing posts for different retrieval scheduling algorithms (baseline, posting-only [18], user-access-missing [19], user-access-delay) with the same resource allocation policy.**

As we can see, the performance of the baseline method is poor for obvious reasons. Again delay time is accumulated for every possible pair of post and access and thus can be large. The algorithm *posting-only* also has poor performance since it tends to schedule retrievals to intervals where the posting rate is high, even though there is no user access. The two user access-based algorithms achieve much better performance. What's more, our algorithm is more effective in minimizing delay time. Our algorithm achieves around 10% or greater improvement in minimizing delay over the algorithm where only the number of missing posts is considered. For users 3 and 4, the improvements were 35% and 32%, respectively. Therefore, our algorithm is less likely to miss older posts, which is another benefit besides the optimal resource allocation policy.

### 7.4.3 Number of Missing Posts

We can compare the number of missing posts for the four different retrieval scheduling algorithms (*baseline, posting-only [18], user-access-missing [19], user-access-delay*) with the same number of retrievals allocated for each feed. We show the results in Table 3. As we can see, the number of missing posts for the baseline algorithm and *posting-only* are much larger compared with the other two algorithms. The numbers of missing posts for our algorithm are very similar to those for the *user-access-missing* algorithm, indicating our algorithm is able to minimize delay time of the missing posts and in the meanwhile, to maintain low numbers of missing posts.

Notice that the *user-access-missing* algorithm is a little bit more effective to minimize the number of missing posts. However, even with more missing posts, our algorithm still minimimizes delay time. This is because with the same number of retrievals, our algorithm penalizes missing old posts more than missing recent posts. Thus it may select different retrieval time points and may even sacrifice some recent posts for an old post. However, missing too many recent posts may also lead to poor performance and our algorithm is able to avoid this situation. That is why the number of missing posts for our algorithm is very close to the number of missing posts for the *user-access-missing* algorithm.

## 8 Conclusions

The growth of mobile devices is making personalized content aggregation increasingly important. Not only is there increasing demand for timely updates on online information, but power and bandwidth limitations of the devices will require novel retrieval strategies. Retrieval scheduling algorithms could become basic infrastructure supporting the rapidly increasing number of personalized aggregation apps.

This paper proposes a personalized content aggregation model that considers both the new content posting rate and user access rate. Furthermore, our model considers not only the number of missing posts, but also delay time for these missing posts. With this model, assuming Poisson postings, we develop an optimal resource allocation algorithm, as well as an optimal retrieval scheduling algorithm to minimize the expected delay time of aggregation over multiple data sources. To our knowledge, this is the first personalized aggregation model on multiple data sources. We evaluated our strategies on real RSS feed data; the comparison between our strategies and state-of-the-art strategies demonstrate that our methods are not only effective but also efficient.

## References

[1] Bloglines, http://www.bloglines.com, 2011.

[2] Blogpulse, http://www.blogpulse.com, 2011.

[3] Google Alerts, http://www.google.com/alerts, 2011.

[4] Google News, http://news.google.com/, 2011.

[5] News aggregators, http://www.newsonfeeds.com/faq/aggregators, 2011.

[6] RSS, http://www.rssboard.org, 2011.

[7] Trove, http://www.trove.com/m, 2011.

[8] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *ACM SIGMOD Record*, 29(2):117–128, 2000.

[9] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)*, 28(4):390–426, 2003.

[10] J. Cho and A. Ntoulas. Effective change detection using sampling. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 514–525. VLDB Endowment, 2002.

[11] E. Coffman Jr, Z. Liu, and R. Weber. Optimal robot scheduling for web search engines. *Journal of scheduling*, 1(1):15–29, 1998.

[12] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *Proceedings of the 10th international conference on World Wide Web*, pages 265–274. ACM, 2001.

[13] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th international conference on World Wide Web*, pages 106–113. ACM, 2001.

[14] R. Horincar, B. Amann, and T. Artières. Best-Effort Refresh Strategies for Content-Based RSS Feed Aggregation. *Web Information Systems Engineering–WISE 2010*, pages 262–270, 2010.

[15] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 73–84. ACM, 2002.

[16] S. Pandey and C. Olston. User-centric web crawling. In *Proceedings of the 14th international conference on World Wide Web*, pages 401–411. ACM, 2005.

[17] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: Content-based filtering and aggregation of blogs and RSS feeds. In *Proc. of the Symposium on Networked Systems Design and Implementation, Boston, MA*, pages 29–42, 2007.

[18] K. Sia, J. Cho, and H. Cho. Efficient monitoring algorithm for fast news alerts. *IEEE Transactions on Knowledge and Data Engineering*, pages 950–961, 2007.

[19] K. Sia, J. Cho, K. Hino, Y. Chi, S. Zhu, and B. Tseng. Monitoring rss feeds based on user browsing pattern. In *Proceedings of the International Conference on Weblogs and Social Media*. Citeseer, 2007.

[20] J. Wolf, M. Squillante, P. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the 11th international conference on World Wide Web*, page 147. ACM, 2002.