Best-Effort Refresh Strategies for Content-Based RSS Feed Aggregation*

Roxana Horincar, Bernd Amann, and Thierry Artières

LIP6 - University Pierre et Marie Curie, Paris, France
{roxana.horincar,bernd.amann,thierry.artieres}@lip6.fr

Abstract. During the past several years RSS-based content syndication has become a standard technique for efficiently and timely disseminating information on the web. From a data processing perspective RSS feeds are standard XML resources which are periodically refreshed by feed aggregators for generating continuous streams of items. In this article, we study the problem of information loss in the context of a content-based feed aggregation system and we propose a new best-effort refresh strategy for RSS feeds under limited bandwidth. This strategy is evaluated experimentally and compared to other state-of-the-art crawling strategies for web pages.

1 Introduction

RSS [14] (and Atom [7]) have become the de-facto standards for efficient information dissemination and web sites providing RSS-based services face an increasing usage of their bandwidth for delivering RSS information to their clients. Web syndication standards are mainly concerned with the document format. From the point of view of a web server, there is no distinction between a RSS feed and a regular web resource, both of them being fetched by using the standard `http` protocol. Thereby web syndication can be considered as a particular publish-subscribe application where subscribers must regularly refresh the web feed sources in order to access the latest updates.

In this article we are interested in the issue of reducing bandwidth usage in the context of RoSeS (Really Open Simple and Efficient Syndication) [13], a content-based RSS feed aggregation system which allows individual users to create personalized feeds by defining content-based aggregation queries on selected collections of RSS feeds. Compared to centralized server-based feed aggregators like GoogleReader [6] and YahooPipes [17], RoSeS advocates a distributed client-based aggregation infrastructure which allows user to install and personalize their local feed aggregator. RoSeS has been designed as a pull-based system. This choice is first based on the fact that the great majority of RSS services and applications use the standard pull-based web protocol `http` where subscribers are the ones that regularly contact the feed sources to retrieve new postings. Alternative solutions based on push-based or hybrid pull/push-based protocols are presented in [1], [16].

* The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ROSES (ANR-07-MDCO-011) "Really Open, Simple and Efficient Syndication".

© Springer-Verlag Berlin Heidelberg 2010

Whereas RoSeS allows to formulate complex queries with joins and windows, in this article we consider only stateless continuous queries computing a filtered union of source feeds. We study the problem of information loss and we propose a new best-effort refresh strategy for RSS feeds under limited bandwidth.

In particular, we make the following contributions:

- a declarative feed aggregation model with a precise semantics for feeds and aggregation queries (Section 2),
- appropriate metrics (feed completeness) designed to measure the quality of newly produced aggregation feeds (Section 2),
- a best-effort refresh strategy to retrieve new postings from different feed sources which maintains an optimal level of aggregation quality (Section 3) and
- an experimental evaluation of our strategy (Section 5).

Additionally, we review related work in Section 4 and conclude in Section 6.

2 Content-Based Feed Aggregation and Feed Completeness

Feed aggregation queries are defined in a declarative RSS query language. The result of each query is a new feed that can be accessed locally and, if necessary, be published for other users. For example, a user wants to create a feed with news about volcano eruptions in Iceland fetched from "The Guardian" and images published by "The Big Picture" on the same topic. This can easily be translated into the following aggregation query which applies a simple disjunctive filtering condition on the union of the corresponding feeds:

```
CREATE FEED IcelandVolcanoEruption AS
RETURN "http://feeds.guardian.co.uk/theguardian/rss" |
       "http://www.boston.com/bigpicture/index.xml"
WHERE ITEM CONTAINS "iceland" OR "volcano" OR "eruption"
```

Aggregation Queries: Let $S = \{s_1, ..., s_m\}$ be a set of RSS feeds. An *aggregator* node n is defined by a set of *aggregation queries* $Q(n) = \{q_1, ...q_k\}$ on S. A query q applied on a source feed introduces a selectivity factor $sel(q) \in [0, 1]$: $sel(q) = 1$ if the aggregator query keeps all items published by the input feeds (simple union) and $sel(q) = 0$ if it filters out all items (empty result). The aggregator node n generates for each query $q \in Q(n)$ a query feed $f(q, n)$ and we will denote by $feeds(n)$ the set of feeds generated by n. We assume that S and $feeds(n)$ are disjoint.

Windows and Streams: There are mainly two different interpretations of a feed produced by some query q at some time instant t. In the first case, which corresponds to the standard document-based interpretation of RSS feeds, a feed is represented by a limited number of items available in a XML document at some time instant t. We will call this document a *publication window* of size W_s and denote it by $A(f, t)$. The publication process guarantees that for all time instants t and t' where $t < t'$ all items in $A(f, t') - A(f, t)$ have been published after all items in $A(f, t)$. This observation allows us to define the second interpretation which considers a feed as a continuous stream of items published until time instant t. This stream is called the *publication stream* of f and denoted $F(f, t)$.

Feed Completeness: There are two main reasons for losing items during the aggregation process that influence $F(f,t)$. On the back-end, the crawler might miss some items that have been published by the sources and never been read by the aggregator. This read loss depends on the publishing frequency of the sources, the available bandwidth b and the polling strategy. On the front-end, the query processing and publication threads might receive more items than they can consume. This is a well known data stream processing problem which can be solved in different ways (approximation, load shedding). This issue is considered to be outside the scope of this article.

The feed completeness of a query feed f at some time instant t is denoted $\mathcal{C}_F(f,t) = |F(f,t)|/|I(f,t)|$ and estimates the relative read loss of the aggregation process generating the feed by comparing the number of published items in $F(f,t)$ with the number of items in $I(f,t)$, where $I(f,t)$ represents an ideal stream generated by the aggregator with unlimited bandwidth and refresh frequency.

3 A Best-Effort Refresh Strategy for Feed Completeness

Window Divergence: We call *divergence* the degree to which a source feed differs from the feed published by an aggregator subscribed to that source. In our case, we define *window divergence* Div_A as the number of new items generated by the source feed s during time period $[t_0, t]$, available at time instant t (computed by $new(A(s,t))$) and that are relevant to query q, where t_0 denotes the time of the last refresh of source s: $Div_A(q,s,t) = |q(new(A(s,t)))|$. Observe that the divergence value depends on the publication frequency of s and $sel(q)$, the selectivity of q.

Since the number of relevant items present in the source publication window can decrease if they are replaced by new irrelevant ones, Div_A is a *non monotonic* function. This is illustrated in figure 1 representing the evolution of Div_A compared to the total number of new items published since the last refresh by some "ideal" source s publishing on average λ items per cycle with a uniform distribution of relevant items.

Both curves start at time t_0 when s is refreshed and evolve identically (*monotonically increasing*) until some time t' near $t = t_0 + W_s/\lambda$ where the number of new items reaches the capacity of the source publication window W_s. We will say that source s has *saturated* at time t'. At this moment, both curve values correspond to some value near $sel(q) \cdot W_s$ and from this moment on, window divergence Div_A can decrease since new irrelevant items might replace relevant items that have not been read yet (read loss).

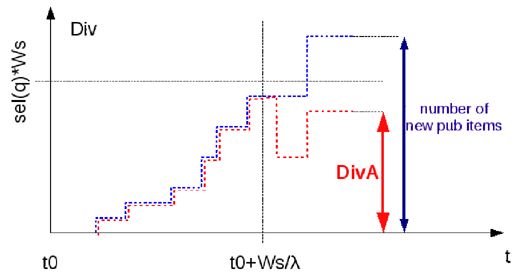


Fig. 1. Window divergence

Best-Effort Refresh Strategy for Unsaturated Sources: We adapt the web page approach presented in [9], where the authors define the utility gained by refreshing a web page based on a monotonic divergence function. In our case, we use the window divergence Div_A defined in Section 3, guaranteed to be monotonic for *unsaturated sources* to define $Uti_A(q_i, s_j, t)$, the utility of refreshing source s_j for the quality of query feed $f(q_i)$ at time instant t:

$$Uti_A(q_i, s_j, t) = (t - t_k) \cdot Div_A(q_i, s_j, t) - \int_{t_k}^{t} Div_A(q_i, s_j, x)dx \qquad (1)$$

Utility function $Uti_A(q_i, s_j, t)$ aggregates all divergence scores over the period $[t_k, t]$ where t_k is the last time when source s_j was refreshed $(Div(q_i, s_j, t_k) = Uti(q_i, s_j, t_k) = 0)$.

We then define the following refresh strategy for *unsaturated sources*. This strategy is *best-effort* in the sense of [9], since there exists no other strategy which obtains a better result within the same cost:

Let τ be any positive constant and Uti_A be the utility function defined in equation 1. Then, at each time t and for each query q_i, refresh those source feeds s_j that have:

$$Uti_A(q_i, s_j, t) \geq \tau \qquad (2)$$

τ is a refresh threshold which controls the global polling frequency of sources with respect to their divergence scores. For a high value of τ, the aggregator will refresh its sources at a rather low rate and for low values of τ, it will refresh at higher rates. The actual value of τ depends on the maximum number of sources that the aggregator is allowed to refresh at time t and on the update frequencies of the sources. τ converges to a non-negative value in case all these remain constant (see our experiments in Section 5).

Best-Effort Refresh Strategy for Saturated and Unsaturated Sources: Since window divergence monotonicity is guaranteed only for unsaturated sources, all sources which already have reached their saturation point must be handled separately. Therefore we define a two-step refresh strategy as described in the algorithm 1 conceived for the special case of RSS feeds. The first step will refresh the top-b *saturated* sources that have maximum positive window divergence Div_A. This criteria ensures maximum feed completeness for saturated sources. The following step handles the rest of the sources (the *unsaturated* ones) and refreshes only those that meet the condition in equation 2. Since the refresh threshold τ in equation 2 corresponds to an average limited bandwidth of b refreshed sources, threshold τ must be adjusted to the fraction of the bandwidth available after the first step.

Estimating the Refresh Threshold τ: Since in the context of content-based feed aggregation the update frequency of items relevant to query q and the available resources b of the node might change, there is no single value for τ that is optimal all the time. If the system parameters change, the algorithm dynamically adjusts the value of τ so it converges to a new best value.

We consider a τ value specific to each aggregator n that can be set to any random initial value. We suppose that if n has selected b' sources to refresh, it will (i) increase

Algorithm 1. Refresh strategy in 2 steps

```
Input: $b$, $\tau$, $Div_A(n, s_j, t)$, $Uti_A(n, s_j, t)$
  $R(n,t) := \emptyset$, $Sat(n,t) := \emptyset$, $b_{sat} = 0$
  {First step: refresh up to $b$ saturated sources ordered by divergence}
  for all $s_j \in S(n)$ do
    if $s_j$ is at saturation point and $Div_A(n, s_j, t) > 0$ then
      $Sat(n,t) := Sat(n,t) \cup \{s_j\}$
  $R(n,t) := top_b(Sat(n,t))$
  {Second step: refresh unsaturated sources using utility threshold}
  $b_{sat} = size(R(n,t))$
  if $b_{sat} \leq b$ then
    for all $s_j \in S(n)$ do
      if $Uti_A(n, s_j, t) \geq \frac{b}{b - b_{sat}} \cdot \tau$ then
        $R(n,t) := R(n,t) \cup \{s_j\}$
  estimate new threshold $\tau$
  refresh sources in $R(n,t)$
```

τ with a factor β if more resources than available were used ($b' > b$) and it will (ii) decrease it with α if the available bandwidth has been exploited below some percentage p ($b' < p\%b$). α controls how aggressively n gives priority for more feed sources to be refreshed and β reflects how quickly it slows down the sources refresh rate. τ doesn't change otherwise ($p\%b \leq b' \leq b$). During experiments, we made τ vary with 5% since big variations may prevent τ from converging.

4 Related Work

The problem of efficient polling policies for web resources was largely studied in the context of web page crawling [2,3,4,5,8,9,10,11]. For example, Garcia-Molina and Cho [3,2,4] develop synchronization strategies to optimize web page freshness and age. In our paper, we study a measure more appropriate to our context: feed completeness. Another approach proposed in [10] introduces more elaborated quality measures: web page freshness perceived by users. This metric is optimized based on query load and click-through data. [11] proposes CAM (Continuous Adaptive Monitoring), a resource allocation algorithm which allocates limited monitoring resources across pages so as to minimize the information loss compared to an ideal monitoring algorithm which can monitor every change of the page. Whereas this setting is similar to ours, we explore the problem in a different direction by adapting existing best-effort refresh strategies for web pages.

Reference [9] proposes different types of divergence metrics, lag being similar to our window divergence. The authors introduce a best-effort synchronization scheduling policy that exploits cooperation between data sources and the cache in a push architecture, where data sources actively notify the cache of any changes. In our case, we assume a pull context with passive data sources that are periodically contacted by aggregator nodes. A best-effort strategy which focuses on the lifetime of content fragments that appear and disappear from the web pages over time (information longevity) is presented in [8].

Closer to our problem setting, reference [15] proposes a pull based aggregator architecture that monitors RSS data sources and quickly retrieves new postings, minimizing the delay between the appearance of a posting at the source and its retrieval by the aggregator. They propose a periodic inhomogeneous Poisson process to model the generation of RSS feed items. Whereas their approach is efficient for refreshing RSS feeds, we believe that content-based filtering destroys periodicity and we follow the idea of [9] which dynamically adjusts τ to the real workload.

5 Experiments

Parameter Settings: The simulation environment used to perform our experiments is PeerSim [12], a Java-based engine for testing peer-to-peer protocols. We constructed a cycle-based environment with an aggregator node that applies a query q computing a filtered union on a set of feed sources that publish following a Poisson process of rate parameter λ_i. The values chosen for the input parameters (summarized in table 1) are representative for the obtained experimental results.

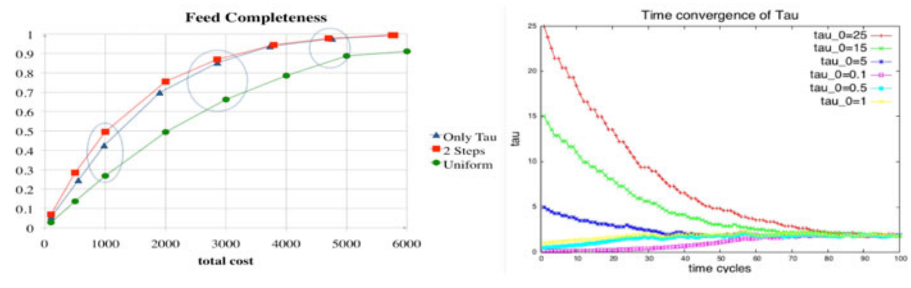
Table 1. Input parameter values

No. of aggregator nodes	1	No. of cycles in a simulation	100
No. of source feeds	100	λ_i uniformly distributed in	[0, 6.5]
W_s	10	b	(0,100]
α	0.95	β	1.05
$p\%$	90%		

Comparing Strategies: In the following sections we compare the optimal refresh strategy presented in Section 3 (that we will identify as the $2steps$ strategy) with five other strategies. The considered strategies are all offline, based on a-priori knowledge of the divergence function and the average publication frequency λ_i of source feed s_i:

- $onlySat$ represents the first step of our optimal $2steps$ refresh strategy, taken separately. It refreshes only those b feed sources that have maximum window divergence Div_A among the saturated sources.
- $onlyTau$ represents the second step of the $2steps$ refresh strategy, considered separately. It refreshes only those feed sources s_i for which $Uti_A(n, s_i, t) \geq \tau$.
- $topKUtility$ refreshes b sources that have maximum utility Uti_A.
- $uniform$ refreshes every source feed at the same frequency b/m, where m represents the total number of feed sources.

Strategy Effectiveness: Besides measuring the feed completeness $\mathcal{C}_F$ introduced in Section 2, we also consider the *window freshness* $\mathcal{F}_W$, that compares the number of relevant items available both on the source feeds and on the query feed $f(q)$ in the same time. $\mathcal{F}_W$ is defined as a weighted average of the single-sourced window freshness



(a) **(b)**

Fig. 2. Feed Completeness and Tau Convergence

$\mathcal{F}_W(f/s,t) = |A(f,t) \cap q(A(s,t))|/|q(A(s,t))|$, where $s \in S(q)$. Both $\mathcal{C}_F$ and $\mathcal{F}_W$ estimate the quality of the used refresh strategy in terms of item loss, where the former represents a refresh quality metric for long-term feed aggregation tasks (archiving) and the latter reflects the refresh quality for short-term feed aggregation tasks (news reader).

In figure 2a we plotted the feed completeness obtained for different values of b (on the abscissa we plotted the total cost, measured as the total number of refresh requests done during a simulation). In order to make the figure more clear, we present only the $2steps$, $onlyTau$ and $uniform$ strategies. The results for the complete set of strategies are presented in table 2, where we give the experimental values both for feed completeness $\mathcal{C}_F$ and window freshness $\mathcal{F}_W$, for three representative values of b. All the presented results are obtained after a warm-up period when the value of τ has converged.

When the aggregator is restricted to refresh few sources (for relatively small values of $b \in (0, 22]$), very many sources that were not refreshed in time become quickly saturated. And thus, giving priority to refreshing the saturated sources (as the optimal $2steps$ or $onlySat$ strategies do) generates significantly better feed completeness values than all the other strategies, as shown in table 2 for $b = 10$. However, refreshing the saturated sources does not guarantee fetching many relevant items and thus, the strategies that refresh based on their utility ($onlyTau$, $topKUtility$) obtain better results for the window freshness.

Table 2. Feed completeness and Window freshness

	$b = 10$			$b = 30$			$b = 50$		
	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$
2steps	**0.4954**	0.3671	1000	**0.8691**	**0.7279**	**2854**	**0.9781**	**0.8762**	**4700**
onlyTau	0.4252	**0.4197**	982	0.8524	0.7048	2860	**0.9759**	**0.8731**	4762
topKUtility	0.4160	**0.4220**	1000	**0.8671**	**0.7200**	3000	**0.9793**	**0.8868**	5000
onlySat	**0.4934**	0.3671	1000	0.8082	0.6590	2516	0.8091	0.6625	2524
uniform	0.2682	0.3620	1000	0.6625	0.6398	3000	0.8871	0.7863	5000

If the aggregator is allowed to refresh more sources (b takes values in $[22, 40]$), there are fewer sources that reach the saturation point. Since the optimal $2steps$ strategy takes advantage both of refreshing saturated sources (as $onlySat$) and sources with best utility (as $onlyTau$), it obtains the best results for a lower cost, as presented in table 2 for $b = 30$.

For more permissive constraints ($b > 40$), the optimal $2steps$ strategy manages to refresh all the sources based on their utility before they reach saturation, just as the $onlyTau$ and $topKUtility$ strategies do, but at a lower cost.

Strategy Robustness: Using a self adaptive threshold-setting algorithm for finding the optimal value of τ in the second step of our $2steps$ refresh strategy makes our policy highly adjustable to the changes that may occur in real world. As discussed in Section 3, not only the average source publication frequency λ may vary in time, but also the update frequency of items relevant to some query q. Since τ depends on the available bandwidth and the divergence rates of the news feeds, there is no single best value that works well all the time. In order to show the high adaptability of the τ threshold (and consequently, of our optimal $2steps$ refresh policy), we show a study on the convergence of τ in figure 2b.

We plotted the evolution of τ for the optimal $2steps$ refresh strategy when the average number of refresh sources by time cycle is set to $b = 30$, for different τ initial values. From figure 2b, we can see that τ always converges to the optimal value. This convergence assures the robustness of our refresh strategy to different changes that may occur in real world.

6 Conclusion

In this article we have studied the problem of refreshing RSS feeds in a content-based feed aggregation context. We proposed a declarative feed aggregation model with precise semantics for feeds and aggregation queries. We defined a quality metric (feed completeness) and a two-step polling strategy optimizing this metric in time. This strategy has been evaluated experimentally by comparing its behavior to different other strategies on a synthetically generated setting.

As future work, we want to test our refresh strategy in a real world setting with real data sources and online estimation functions for the source publication frequencies and the divergence values. Besides that, we want to adapt our feed aggregation approach to a distributed context where users create personalized feeds aggregating feeds generated by other users.

References

1. Acharya, S., Franklin, M., Zdonik, S.: Balancing push and pull for data broadcast. In: SIGMOD 1997: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, Tucson, Arizona, United States, pp. 183–194. ACM, New York (1997)
2. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. SIGMOD Rec. 29(2), 117–128 (2000)

3. Cho, J., Garcia-Molina, H.: Effective page refresh policies for web crawlers. ACM Trans. Database Syst. 28(4), 390–426 (2003)
4. Cho, J., Garcia-Molina, H.: Estimating frequency of change. ACM Trans. Interent Techonol. 3(3), 256–290 (2003)
5. Cho, J., Ntoulas, A.: Effective change detection using sampling. In: VLDB 2002: Proceedings of the 28th international conference on Very Large Data Bases, Hong Kong, China, pp. 514–525. ACM, New York (2002)
6. google_reader, `http://www.google.com/reader`
7. Network Working Group: The atom publishing protocol, `http://tools.ietf.org/html/rfc5023`
8. Olston, C., Pandey, S.: Recrawl scheduling based on information longevity. In: WWW 2008: Proceeding of the 17th international conference on World Wide Web, Beijing, China, pp. 437–446. ACM, New York (2008)
9. Olston, C., Widom, J.: Best-effort cache synchronization with source cooperation. In: SIGMOD 2002: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, pp. 73–84. ACM, New York (2002)
10. Pandey, S., Olston, C.: User-centric Web Crawling. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, pp. 401–411. ACM, New York (2005)
11. Pandey, S., Ramamritham, K., Chakrabarti, S.: Monitoring the dynamic web to respond to continuous queries. In: WWW 2003: Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, pp. 659–668. ACM, New York (2003)
12. peersim, `http://peersim.sourceforge.net/`
13. roses, `http://www-bd.lip6.fr/roses/doku.php`
14. rss, `http://www.rssboard.org/`
15. Sia, K.C., Cho, J., Cho, H.-K.: Efficient Monitoring Algorithm for Fast News Alerts. IEEE Trans. on Knowl. and Data Eng. 19(7), 950–961 (2007)
16. Silberstein, A., Terrace, J., Cooper, B.F., Ramakrishnan, R.: Feeding frenzy: selectively materializing users' event feeds. In: SIGMOD 2010: Proceedings of the 2010 international conference on Management of data, Indianapolis, Indiana, USA, pp. 831–842. ACM, New York (2010)
17. yahoo_pipes, `http://pipes.yahoo.com/pipes/`