A New Aggregation Policy for RSS Services

Young Geun Han[1] Sang Ho Lee[1] Jae Hwi Kim[1] Yanggon Kim[2]

[1] School of Computing, Soongsil University
Seoul, Korea
{younggeun,shlee199,oasisdle}@gmail.com

[2] Dept. of Computer and Information Sciences, Towson University
Maryland, USA
ykim@towson.edu

ABSTRACT
RSS is the XML-based format for syndication of Web contents, and users aggregate RSS feeds with RSS feed aggregators. There are RSS aggregation policies that help aggregate RSS feeds effectively. In this paper, we first propose an aggregation policy to minimize the number of missing postings within an aggregation. Second, we analyze and compare our aggregation policy with existing aggregation policies. Our analysis reveals that our aggregation policy can reduce approximately 23% of the missing posts in comparison with an existing policy while it increases only 6% of the aggregation delay.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Online Information Services – *data sharing, web-based services.*

General Terms
Algorithms

Keywords
RSS, RSS Aggregator, Aggregation Policy

1. INTRODUCTION
As blogs have been growing rapidly in the past years, the use of RSS, which is the XML-based format for syndication and subscription of information, is diffused [5]. RSS stands for the Really Simple Syndication, Rich Site Summary, or RDF (Resource Description Framework) Site Summary. In particular, RSS is used to easily deliver up-to-date postings such as personal weblogs, news Web sites to their subscribers [9]. RSS 0.90 was designed by Netscape in 1999. After the RSS team at Netscape evaporated, RSS-DEV Working Group and the UserLand company have been working on standardization on RSS independently. RSS-DEV Working Group has designed RSS 1.0 based on RDF. UserLand has designed RSS 2.0 based on RSS 0.90 [5][9][12]. These two standards are currently available and our study is based on RSS 2.0.

Figure 1 shows the basic structure of RSS 2.0. The subordinate to the <rss> element is the <channel> element that contains metadata about RSS feeds. <item> is an element that has information on postings. A channel may contain any number of <item>s.

RSS is not operated under a push-based protocol but under a pull-based protocol in which individual subscribers are responsible for collecting information from Web sites. The basic syndication and subscription process of RSS feeds is shown in Figure 2. Publishers generate postings into RSS feeds. Subscribers register RSS feed addresses to an RSS feed aggregator. An RSS feed aggregator aggregates registered RSS feeds and shows postings to subscribers.

The role of RSS aggregator is becoming more important in Web services. As the number of RSS feed that users want to subscribe grows, the number of RSS feeds that an RSS aggregator has to aggregate grows. In addition, postings dynamically appear and disappear over time. As a result, it is crucial to have a good aggregation policy that enables us to efficiently aggregate postings that are generated. Aggregation policies may determine not only the number of aggregations for each RSS feed, but also schedule when aggregations take place. Sia and Cho [10][11] proposed the framework of an information aggregator, and proposed the scheduling and resource allocation algorithm that exploits posting patterns and user browsing patterns.

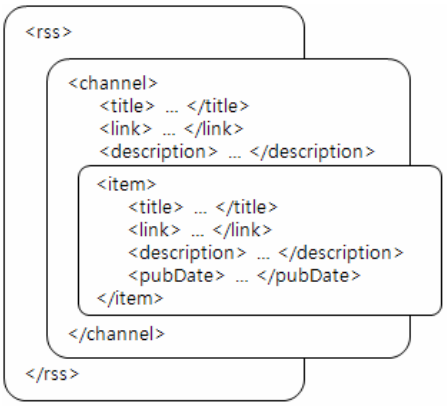

Figure 1. Basic structure of RSS 2.0

CSSSIA 2008, April 22, Beijing, China

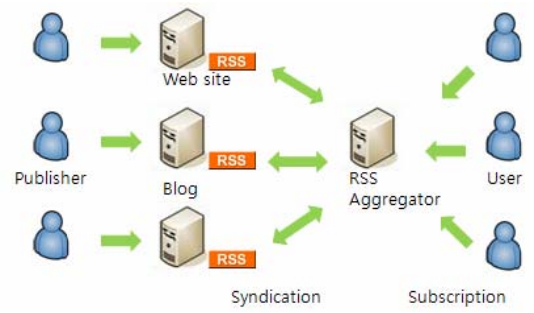

Figure 2. Basic RSS feed syndication and subscription process

RSS feeds are unlikely to store all generated postings. Instead, they store only a part of the most recently generated postings. It is quite possible for some postings in RSS feeds to be deleted (hence not to be aggregated by RSS aggregators). In this paper, we propose an aggregation policy to efficiently minimize those missing postings during the aggregation process. Our policy makes use of the number of postings that can be stored in each RSS feed to schedule aggregations. We also analyze and compare our aggregation policy with existing aggregation ones. Our results show that our aggregation policy can reduce approximately 23% of the postings being missed in comparison with the other aggregation policies, while it increases only 6% of the aggregation delay.

The rest of this paper is organized as follows: In Section 2, we review the previous work on existing aggregation policies, discuss problems in those polices, and present our policy. In Section 3, we compare the results from our experiments. Finally, we conclude in Section 4.

2. AGGREGATION POLICIES FOR RSS SERVICES

Let an RSS aggregator aggregate n RSS feeds. The simplest aggregation policy is to give each feed the same number of aggregations. We call this the uniform aggregation policy. The uniform aggregation policy is inefficient, because RSS feeds are aggregated equally no matter how many postings are stored in RSS feeds and how often postings are generated.

2.1 Minimum Delay Aggregation Policy

The aggregation delay is a widely-accepted metric that can evaluate an RSS aggregation policy [10]. The aggregation delay is the delay time between the generation time of a posting at an RSS feed and its aggregation time by an aggregator. Assume that an RSS feed F generates postings $p_1,\ldots,p_k$ at times $t_1,\ldots,t_k$ and the aggregator aggregates new postings from F at times $a_1,\ldots,a_m$. The delay associated with the posting p_i is defined as

$$D(p_i) = a_j - t_i$$

where a_j is the minimum value of $a_1,\cdots,a_m$ with a_j greater than posting time t_i ($a_j \geq t_i$). As can be seen in Figure 3, the delay time becomes bigger as a_j becomes more far away from t_i. The total aggregation delay for all postings $p_1,\cdots,p_k$ of an RSS feed F is defined as follows.

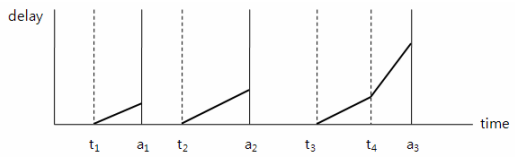

Figure 3. Aggregation delay

$$D(F) = \sum_{i=1}^{k} D(p_i) = \sum_{i=1}^{k} a_j - t_i, where\ t_i \in [a_{j-1}, a_j]\ .$$

We have the condition, i.e. $t_i \in [a_{j-1}, a_j]$, because the aggregation delay calculates the posting times that are generated between aggregation times a_j and a_{j-1}, where a_{j-1} denotes the immediately previous aggregation time of a_j. The total aggregation delay for all RSS feeds is defined as below:

$$D(A) = \sum_{i=1}^{n} D(F_i)\ .$$

EXAMPLE 1. As in Figure 4, postings $p_1,\ldots,p_5$ are generated at $t_1,\ldots,t_5$ in a RSS feed and an aggregator aggregates at times a_1 and a_2. $D(p_1)$ is the difference between a_1 and t_1, therefore 3.

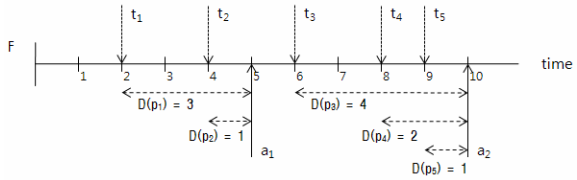

Figure 4. Example of aggregation delay

The aggregation delay is the difference between the aggregation time and the generation time. The total aggregation delay for all postings $p_1,\ldots,p_5$ of an RSS feed can be calculated:

$$D(F) = \sum_{i=1}^{5} D(p_i) = \sum_{i=1}^{5} a_j - t_i, where\ t_i \in [a_{j-1}, a_j]$$
$$= D(p_1) + D(p_2) + D(p_3) + D(p_4) + D(p_5)$$
$$= 11$$

■

In order to reduce the aggregation delay, Sia and Cho [10] proposed the minimum delay aggregation policy that allocates aggregation on the basis of the posting rate, which is the number of postings over a period of time. Let RSS feed F_i have the posting rate λ_i and the importance weight ω_i. Let M be the possibly maximum number of aggregations F_i can aggregate over time T. Then, the number of aggregation of F_i can be calculated as

$$m_i = k\sqrt{\omega_i \lambda_i}_i$$

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

EXAMPLE 2. Assume that there are RSS feeds F_1, F_2, F_3, F_4 that have the posting rates λ_1, λ_2, λ_3, λ_4 as 30, 30, 10, 10, respectively (see Figure 5). Also assume that all RSS feeds have the same importance weight ($\omega_i = 1$) and M is 8.

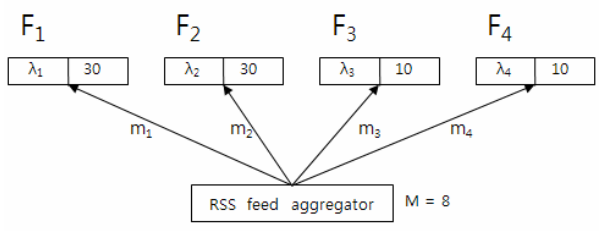

Figure 5. Feeds and posting rates

Table 1 shows the allocation result. Since $\sum_{i=1}^{*} k\sqrt{\lambda_i} = 8$, the value of constant k is approximately 0.46 and the values of m_1, m_2, m_3, m_4 are 3, 3, 1, 1, respectively. Note that the values of m for RSS feeds would be 2 in the uniform aggregation policy.

Table 1. Example of the minimum delay aggregation policy

F_i	λ_i	ω_i	$k\sqrt{\omega_i\lambda_i}$	m_i
F_1	30	1	2.535898385	3
F_2	30	1	2.535898385	3
F_3	10	1	1.464101615	1
F_4	10	1	1.464101615	1

■

2.2 Minimum Missing Aggregation Policy

In the previous approach, the performance criteria of RSS feeds is the total aggregation delay of RSS feeds. The algorithm is devised to minimize the total aggregation delay of RSS feeds. In this paper, we propose a new performance criterion, missing postings.

The RSS feeds are unlikely to store all generated postings. Instead, they are likely to store only a part of the most recently generated postings. It is quite possible for some postings in RSS feeds to be deleted (hence not to be aggregated by RSS aggregators).

Let S_i be a number of maximum postings that can be stored in the ith feed. Consider Figure 6. Let S_3 and S_4 be 10 and 5, respectively. All 10 postings are stored in F_3, so an aggregator can aggregate all postings in a single aggregation. However, postings $p_1,\ldots,p_5$ are deleted in F_4 while new postings $p_6,\ldots,p_{10}$ are generated, since S_4 is 5. An aggregator in a single aggregation can aggregate only 5 new postings, missing 5 postings that were previously generated.

We define postings that are generated in RSS feeds but are not aggregated (due to storage capacity of RSS feeds) by an aggregator as *missing postings*.

We denote the number of postings that are aggregated at the jth aggregation in the ith RSS feed as $PPA_{i,j}$ (postings per aggregation). The number of total missing postings in RSS feeds F_i, which we denote as $MP(F_i)$, is calculated as:

$$MP(F_i) = \lambda_i - \sum_{j=1}^{m_i} PPA_{i,j} \quad .$$

Then, the number of total missing postings in an RSS aggregator can be defined as

$$MP(A) = \sum_{i=1}^{n} MP(F_i) \quad .$$

We denote the number of the total postings that can be aggregated after the jth aggregation in the ith RSS feed as $TP_{i,(j+1)}$ (target postings). Then we have:

$$TP_{i,(j+1)} = TP_{i,j} - PPA_{i,j} \quad .$$

The total number of postings that can be aggregated before the aggregation (when $j = 0$) (i.e. $TP_{i,1}$) is λ_i, since the number of total postings generated in RSS feeds F_i is λ_i. The total number of target postings in RSS feed F_i when the aggregation is finished in RSS feeds F_i (when $j = m_i$) (i.e. $TP_{i,(mi+1)}$) is $MP(F_i)$, since $MP(F_i)$ is the difference of λ_i and summation of $PPA_{i,j}$.

The algorithm in Figure 8 is devised to minimize missing posting during aggregation. In this algorithm, the aggregation number of the feed that has maximum $PPA_{i,j}$ increases by one (see lines 18-20, in Figure 8) and this process is repeated for M times.

$PPA_{i,j}$, which is the number of postings aggregated in an aggregation, is set to S_i when $TP_{i,j}$ is greater than S_i. Otherwise it is set to $TP_{i,j}$ (see lines 11-16). Note that we assume that new postings (as many as S_i) are generated as early as possible before aggregation times.

There are cases in which the newly generated postings are less than S_i when an RSS aggregator aggregates F_i. That is, if all $TP_{i,(j+1)}$ are 0 (which means there are no postings to aggregate at the current aggregation time) even though there are more aggregations to take place (i.e. $\sum_{i=1}^{n} m_i < M$), set all $TP_{i,(j+1)}$ to λ_i (see lines 6-10).

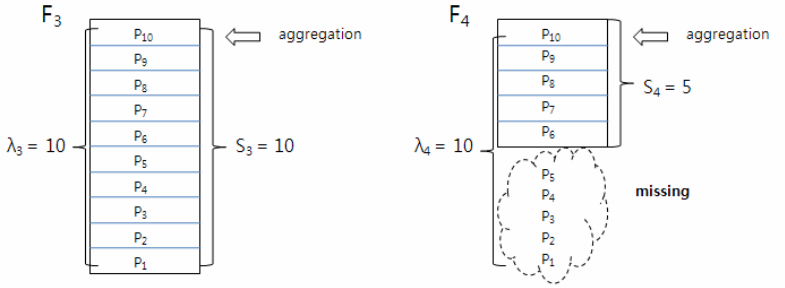

Figure 6. Example of missing postings in RSS feeds

Minimum missing aggregation policy algorithm

Input : M,
n,
$\lambda_i = \{\lambda_1, \ldots, \lambda_n\}$
$S_i = \{S_1, \ldots, S_n\}$

Procedure

[1] for i ← 1 to n
[2] $m_i = 0$, $TP_{i,1} = \lambda_i$
[3] end for
[4] total_TP = $\sum_{i=1}^{n} TP_{i,1}$
[5] for total_m ← 1 to M
[6] if (total_TP = 0) then
[7] for i ← 1 to n
[8] $j = m_i + 1$
[9] $TP_{i,j} = \lambda_i$
[10] end for
[11] for i ← 1 to n
[12] $j = m_i + 1$
[13] if ($TP_{i,j} \geq S_i$) then
[14] $PPA_{i,j} = S_i$
[15] else
[16] $PPA_{i,j} = TP_{i,j}$
[17] end for
[18] Find the maximum value of $PPA_{i,j}$
(where $1 \leq i \leq n$, $j = m_i + 1$)
[19] Let indexes i and j of maximum $PPA_{i,j}$ be maxI, maxJ, respectively
[20] $m_{maxI} = m_{maxI} + 1$
[21] $TP_{maxI,(maxJ+1)} = TP_{maxI,maxJ} - PPA_{maxI,maxJ}$
[22] total_TP = $\sum_{i=1}^{n} TP_{i,1}$
[23] end for

$TP_{i,(j+1)}$(target postings) : The number of total postings that can be aggregated after jth aggregation in ith RSS feed

$PPA_{i,j}$(postings per aggregation) : The number of postings that is aggregated at jth aggregation in ith RSS feed

Figure 8. Minimum missing aggregation policy algorithm

The posting rate may change over time, hence it needs to be updated. To keep it up to date, we recalculate the posting rate periodically.

EXAMPLE 3. Let RSS feed F_1, F_2, F_3, F_4 generate postings 30, 30, 10, 10 per day, and S_1, S_2, S_3, S_4 are 15, 10, 10, 5, respectively (see Figure 7). Assume that their importance weight is of the same ($\omega_i = 1$) and the RSS aggregator can aggregate as many as 8 times.

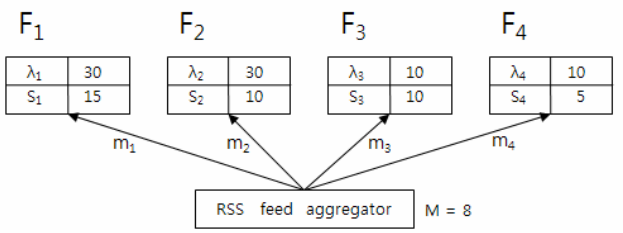

Figure 7. Feeds, posting rates, storage capacity

In our algorithm, calculate $PPA_{i,j}$ (see Table 3) and give an aggregation to the feed that has the maximum value of $PPA_{i,j}$, and repeat this process for 8 times, because M is 8.

■

Table 2 shows the comparison of the three different aggregation policies in terms of the number of aggregation and missing postings.

Table 2. Comparison of Policies

F_i	λ_i	S_i	uniform		minimum delay		minimum missing	
			m_i	$MP(A)$	m_i	$MP(A)$	m_i	$MP(A)$
F_1	30	15	2	0	3	0	2	0
F_1	30	10	2	10	3	0	3	0
F_1	10	10	2	0	1	0	1	0
F_1	10	5	2	0	1	5	2	0

For each aggregation policy, $MP(A)$ can be calculated as shown in Figure 9. In this figure, the numbers (1), (2) and (3) correspond the uniform, minimum delay, and minimum missing, respectively.

(1) $MP(A) = \sum_{i=1}^{4}(F_i) = \sum_{i=1}^{4}(\lambda_i - \sum_{j=1}^{m_i} PPA_{i,j})$
$= (30-(15+15)) + (30-(10+10)) + (10-(10+0)) + (10-(5+5)) = 10$
(2) $MP(A) = (30-(15+15+0)) + (30-(10+10+10)) + (10-10) + (10-5) = 5$
(3) $MP(A) = (30-(15+15)) + (30-(10+10+10)) + (10-10) + (10-(5+5)) = 0$

Figure 9. Calculation of missing postings

3. EXPERIMENTS

The architecture of the RSS aggregator for our experiment is shown in Figure 13. The URLs of RSS feeds are loaded into DB (Feed URL) by the Importer. URLs stored in DB are fetched into the Feed Queue by the Feed Fetcher. The Feed Reader gets URLs from the Feed Queue, read RSS feeds, parse elements, and store them into the Posts repository in DB. The Feed Monitor calculates λ_i and S_i and store them in Monitoring Data in DB.

Table 3. Example of allocating aggregations by calculating $PPA_{i,j}$

M	Computation to find the maximum $PPA_{i,j}$																m_i			
	F_1				F_2				F_3				F_4				m_1	m_2	m_3	m_4
1	$TP_{1,1}$	30	$PPA_{1,1}$	15	$TP_{2,1}$	30	$PPA_{2,1}$	10	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	1	0	0	0
2	$TP_{1,2}$	15	$PPA_{1,2}$	15	$TP_{2,1}$	30	$PPA_{2,1}$	10	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	0	0	0
3	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,1}$	30	$PPA_{2,1}$	10	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	1	0	0
4	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,2}$	20	$PPA_{2,2}$	10	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	2	0	0
5	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,3}$	10	$PPA_{2,3}$	10	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	3	0	0
6	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,4}$	0	$PPA_{2,4}$	0	$TP_{3,1}$	10	$PPA_{3,1}$	10	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	3	1	0
7	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,4}$	0	$PPA_{2,4}$	0	$TP_{3,2}$	0	$PPA_{3,2}$	0	$TP_{4,1}$	10	$PPA_{4,1}$	5	2	3	1	1
8	$TP_{1,3}$	0	$PPA_{1,3}$	0	$TP_{2,4}$	0	$PPA_{2,4}$	0	$TP_{3,2}$	0	$PPA_{3,2}$	0	$TP_{4,2}$	5	$PPA_{4,2}$	5	2	3	1	2

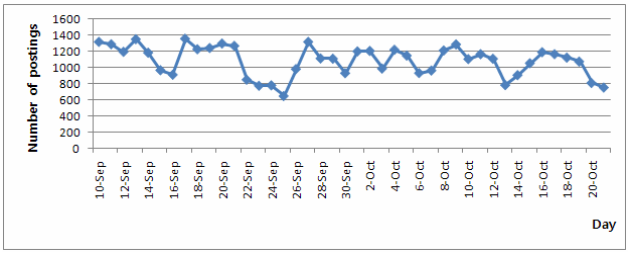

Figure 10. The number of daily postings in all RSS feeds

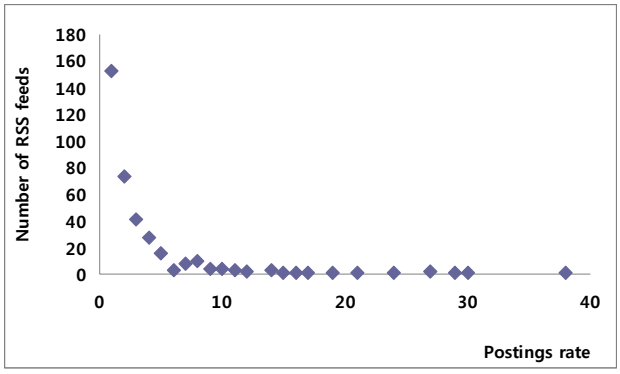

Figure 11. The distribution of daily posting rate

We randomly selected 1,000 RSS feeds from Allblog [1], which is a meta blog portal, to get experimental data. We aggregated 22,399 postings for 6 weeks. To check for missing postings, for each feed, we aggregated every 2 hours and compared aggregated postings to see if duplicated postings are found. When duplicated postings are founded, we know that new postings are not generated as much as S in the feed (i.e. there was no overflow in the feed). In this case, we know that no postings are being missed.

The number of daily postings in all RSS feeds is shown in Figure 10. The average of daily postings is 1083 (1,160 on weekdays, 892 on weekends). The number of postings generated daily ranges from 600 to 1400. The distribution of daily posting rate for total RSS feeds is shown in Figure 11. Approximately 94% of total RSS feeds generate less than 10 postings per day.

Figure 12 shows the distribution of the number of postings that can be stored at feeds. The storage capacity of the feeds is by no means uniform; feeds are different from each other in terms of storage capacity. Approximately 83% of the RSS feeds store 10 to 15 postings while the other RSS feeds store 1 to 60 postings. This observation implies that the consideration on the storage capacity of feeds pays off (cannot be negligible).

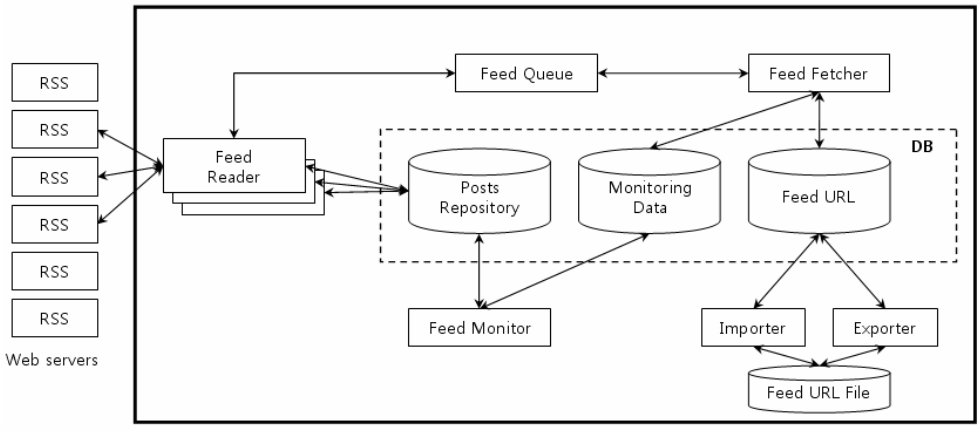

Figure 13. Architecture of the RSS aggregator

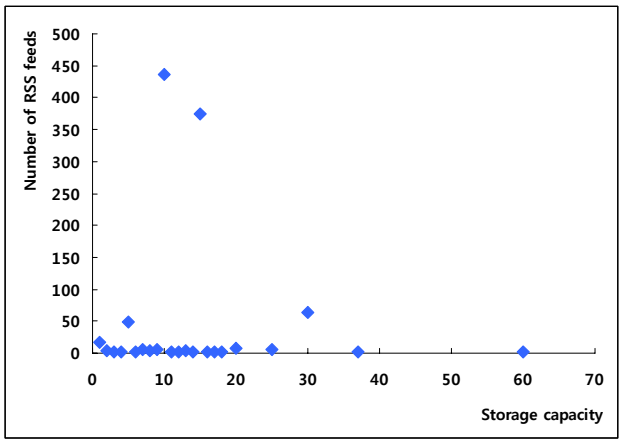

Figure 12. The distribution of storage capacity

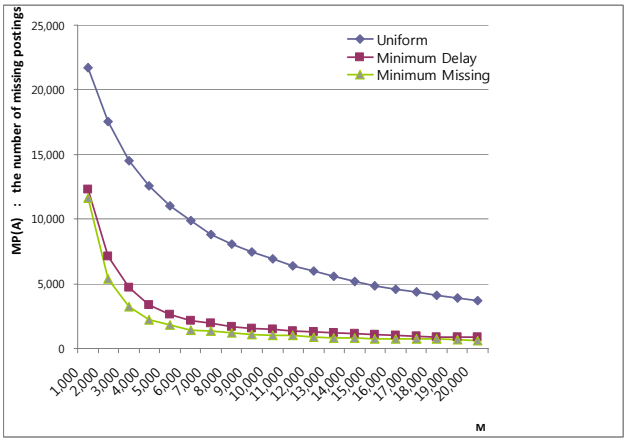

Figure 14. The number of missing postings

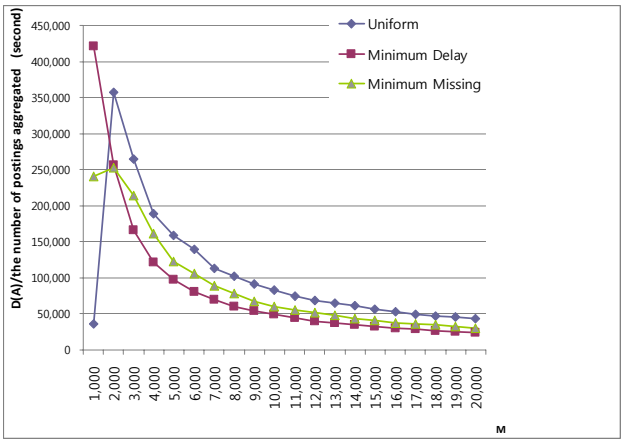

Figure 15. The aggregation delay

Table 4. Comparison of the number of aggregation and missing postings

	minimum missing aggregation policy	
	missing postings	***D(A)*/postings**
uniform aggregation policy	0.23	0.86
minimum delay aggregation policy	0.77	1.06

The objective of the experiments is to compare aggregation efficiency of the uniform aggregation policy, the minimum delay aggregation policy, and the minimum missing policy in perspective of the aggregation delay and the number of missing postings. We assumed that all RSS feeds have the same importance weight ($\omega_i = 1$). For our experiment, the value of M, which is the maximum number of aggregation, was ranging from 1,000 to 20,000. We determined λ_i and S_i from the experimental data of the first three weeks, and used them for our second-half experiments, in which the aggregation delay and missing postings were actually collected.

Figure 14 illustrates the results for missing postings. As M increases, the number of missing postings decreases in all policies. The minimum missing aggregation policy outperforms the rests in terms of minimizing the number of missing posts. The average rate of missing postings for the minimum missing policy is 8%, while the uniform aggregation policy and the minimum delay policy are 37% and 11%, respectively. When M is 20,000, the rate of missing postings for the minimum missing policy is 2% which is acceptable and may decrease as M increases.

Figure 15 shows the aggregation delays. When the value of M is greater than 2,000, the value of minimum delay was always the lowest in the minimum delay aggregation policy.

Table 4 summarizes our analysis. In terms of missing postings, 77% of missing postings of the minimum delay policy are missed in the minimum missing aggregation policy. In terms of aggregation delay, 6% of aggregation delay of the minimum delay policy is additionally added to the minimum missing aggregation policy. In other words, the minimum missing aggregation policy can reduce approximately 23% of the missing postings in comparison with the minimum delay aggregation policy, while it increases only 6% of the aggregation delay. As such, we may conclude that our proposed aggregation policy can efficiently reduce missing postings.

4. CONCLUSION

This paper presents a new aggregation algorithm, the minimum missing aggregation policy that efficiently reduces missing postings. We define a new performance criterion, missing posts, for RSS aggregators. We also conducted an experiment to show the performance of the proposed algorithm and compared with other existing methods. We believe that our study help users choose an appropriate aggregation policy of RSS services for their particular computing environment.

In our study, we assume that whenever the aggregator aggregates RSS feeds, new postings are generated as many as the number of postings that can be stored in each RSS feed. In practice, postings are generated in various patterns, hence more aggregations may be required. We run across cases in which we need to do more aggregations on RSS feeds than we expected.

As for the future work, we plan to explore a possibility that an aggregator may exploit the general posting pattern of RSS feeds based on their past posting history to schedule aggregations. Knowledge on the posting patterns could help us aggregate RSS feeds in an efficient way (for example, predicting the posting times).

5. ACKNOWLEDGMENTS

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD). (KRF-2006-005-J03803)

6. REFERENCES

[1] Allblog. http://www.allblog.net.

[2] Brewington, B., and Cybenko, G. How Dynamic is the Web?, in Proceedings of the 9th International World Wide Web Conference on Computer networks (Amsterdam Netherlands, June 2000), 257-276.

[3] Cho, J., and Garcia-Molina, H., Synchronizing a Database to Improve Freshness, in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (Dallas Texas, May 2000), 117-128.

[4] Cho, J., and Garcia-Molina, H., The Evolution of the Web and Implications for an Incremental Crawler, in Proceedings of the 26th International Conference on Very Large Data Bases (Cairo Egypt, September 2000), 200-209.

[5] Gill, K.E. Blogging, RSS and the Information Landscape: A Look At Online News, WWW 2005 2nd Annual Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics (Chiba Japan, May 2005).

[6] Kim, S.J., and Lee, S.H., An Empirical Study on the Change of Web Pages, in Proceedings of the Seventh Asia-Pacific Web Conference (Shanghai China, March 2005), 632-642.

[7] Kim, S.J., and Lee, S.H., Estimating the Change of Web Pages, in Proceedings of the International Conference on Computational Science 2007 (Beijing, China), 798-805.

[8] Ntoulas, A., Cho, J., and Olston, C., What's New on the Web? The Evolution of the Web from a Search Engine Perspective, in Proceedings of the 13th International World Wide Web Conference (New York NY, May 2004), 1-12.

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

[10] Sia, K.C., and Cho, J. Efficient Monitoring Algorithm for Fast News Alert. IEEE Transaction on Knowledge and Data Engineering, 19(7):950-961, July 2007.

[11] Sia, K.C., Cho, J., Hino, K., Chi, Y., Zhu, S., and Tseng, B.L., Monitoring RSS Feeds Based on User Browsing Pattern, in Proceedings of the International Conference on Weblogs and Social Media (Boulder Colorado, March 2007), 161-168.

[12] What is RSS? http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html.