Efficient RSS Feed Polling using Rolling Curl

Neeraj Kumar
Project Manager – Technology (2020MSL)

Abstract
RSS feeds are still one of the most popular source of consuming information from Internet. Still there is no standard protocol defined for feed fetching. Most software rely on inefficient algorithms for polling and fetching feeds from Internet. In this paper, I am proposing an efficient method of feed polling based on Moving Average. Proposed method is more efficient than traditional approaches. It is easy to implement, wastes less CPU cycles and consumes much less bandwidth. When evaluated, proposed method was ~500% to ~700% faster than traditional sequential approach. For implementing this, I have used more than 15,000 real and unique RSS feeds from different sources like online newspapers, magazines blogs etc.

Keywords
Feed Polling, php, Rolling Curl, Moving Average

I. Introduction

The main object of interest of this paper is RSS feeds polling. RSS (originally RDF Site Summary, often dubbed Really Simple Syndication) is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document (which is called a "feed", "web feed", or "channel") includes full or summarized text, plus meta-data such as publishing dates and authorship [1]. RSS is transmitted in XML format over Internet. Feeds are used by news sites, blogs, and social media portals to announce new content to everyone interested.

RSS is useful for people who regularly use web. It allows them to easily retrieve the latest content from the sites. It also saves times by removing the need of visiting website again to read content. And with the web growing faster than ever, the number of sites offering RSS feeds is growing rapidly. Generally, RSS feeds are fetched using software called feed reader. These software, available both on-line and off-line, poll feeds to retrieve latest content from the sites offering RSS feeds.

Unfortunately, the technology for consuming RSS feeds relies heavily on polling. Due to which, RSS feed readers have to poll RSS feed servers and check for updated content at regular intervals. This method brings up couple of problems. When a RSS server is polled for updates two cases can occur, first, that there is no new content available and second, if the server is polled at large intervals we might miss some items. Also, every feed behaves differently. Some RSS feeds may get updated too often and some may get updated as late as after one year. This leads to wastage of CPU cycles and bandwidth.

In this paper, I am proposing a method of feed polling using cURL module. cURL is a computer software project providing a library and command-line tool for transferring data using various protocols. The cURL project produces two products, libcurl and cURL. It was first released in 1997 [2].

I have used cURL as a software of choice, because it supports easy implementation and data transmission over Internet using protocols like supporting FTP, FTPS, Gopher, HTTP, HTTPS, SCP, SFTP, TFTP, Telnet, etc. Also, cURL is available as a library on many different platforms like Windows, Linux and Mac.

I have also used PHP for implementing and evaluation of algorithms, because it's one of the most popular language on web for programming [4] and the results gained would be more relevant to web programmers because of the popularity of language. Nonetheless, algorithms and pseudo-code provided can be applied to other languages as well.

II. Update Strategies

A. Fix Update

In this method, we set a fixed time interval for polling feed. For eg. if the fix interval is set at 1 hour, then we poll the feeds at every 1 hour interval.

The most common fixed intervals chosen by most feed readers is 1 hour or one day. In my implementation, I have used 1 hour time interval.

To find the next update interval t_{n+1}, we add c to current time t_n:

$$t_{n+1} = t_n + c \quad (1)$$

where c is our fix update interval i.e. 60 mins

B. Moving Average

In statistics, a Moving Average, also called rolling average, rolling mean or running average, is a type of finite impulse response filter used to analyze a set of datum points by creating a series of averages of different subsets of the full data set [5].

Moving Average is a better than fix update in a way that it updates the next update interval continuously and can adapt to the changing frequency of article publishing. Also, intervals at with previously published articles are a good predictor for future article.

To find t_{n+1} using Moving Average we first calculate the average of the post time

$$p_{avg} = (p_0 + p_1 + p_3 + \ldots + p_n) \div n$$

$$p_{avg} = \left[\sum_{i=0}^{n} p_i \right] \div n \quad (2)$$

$$t_{n+1} = t_n + p_{avg} \quad (3)$$

where p_{avg} is average post interval

Above equations are efficient enough to give us a rough estimate of next update interval. But there is a scope for improving the precision.

Suppose, if we poll a feed early only to find that there are no new articles published, then using eq. will yield same t_n. Therefore, we introduce a virtual item v at current time t and calculate the new average publishing interval.

$$p_{avg} = \left[\sum_{i=0}^{n} + v \right] \div (n+1) \quad (3)$$

$$t_{n+1} = t_n + p_{avg} \quad (4)$$

By introducing a virtual item, we can increase the update interval and skip unnecessary polling and wastage of CPU cycles and bandwidth.

C. Moving Average with Rolling Curl

As suggested by other authors (David Urbansky, Sandro Reichert, Klemens Muthmann, Daniel Schuster, Alexander Schill) [8] Moving Average is efficient enough to provide us an efficient update interval for our feeds. Still this method does not tell us how decrease the execution time taken by a program using this method.

As an improvement, I am suggesting an enhancement to above method by introducing rolling polling method. This method helps us in improving the algorithms by increasing the speed of feed fetching by issuing parallel requests to feed servers.

1. Limitations of Traditional Polling

Traditional method of polling requires us to poll every feed in our corpus sequentially. This method is not efficient as a lot of CPU cycles are wasted and we can't poll next feed until we have fetched previous one. Therefore the total time taken to fetch feeds becomes directly proportional to sum of time taken by each feed.

$$T = T_0 + T_1 + T_2 + \ldots + T_n$$

$$T = \sum_{i=0}^{n} T_i \qquad (5)$$

2. Enhanced Feed Polling Using Parallel Fetching

cURL provides us an easy way of querying servers with much different kind of protocols. cURL provides us two ways of firing requests to any server, one is in a sequential mode other is in parallel batch mode.

For firing requests, we use curl_multi(). curl_multi is a great to process multiple HTTP requests in parallel in PHP. It can be used to fetch large number of RSS feeds at one time. Unfortunately the documentation about the curl_multi is limited and therefore, not everyone knows how to use multi_curl efficiently. As a result, most of the algorithms are either inefficient or fail entirely when asked to handle more than a few hundred requests.

A traditional approach while working with curl_multi is that we have to wait for each set of requests to complete before processing. The problem is that most implementations of curl_multi wait for each set of requests to complete before processing them. If there are too many requests to process at once, they usually get broken into groups that are then processed one at a time. The problem with this is that each group has to wait for the slowest request to download. In a group of 100 requests, all it takes is one slow one to delay the processing of 99 others. The larger the number of requests you are dealing with, the more noticeable this latency becomes.

This can easily be eliminated by implementing a queuing system similar to Shortest Processing Time (SPT) queuing system. In this, we process each request as soon as it is completed in a rolling queue. This primarily eliminates the wasted CPU cycles spent in waiting state. The result is a faster and more efficient way of processing large quantities of cURL requests in parallel.

$$T = max(T_0, T_1, T_2, \ldots, T_n) \qquad (6)$$

where T_i is total time taken by a set of feed to complete

III. Pseudo Code

Following is the pseudo code implementation of moving average with rolling curl algorithm.

Initialize corpus C to a set of feeds to be fetched next
Initialize rolling queue Q to $\{U \mid U \subseteq C\}$
while $n[Q]>0$ do:
  if any u finishes
    calculate next time interval using eq. 4 and remove u from Q
  if $n[C]>0$
    add a $\{u \mid u \in C\}$ to queue Q
end while
where,
C is a set of feeds to be fetched next from whole set of feeds
Q is rolling queue
U is subset of C, $n[U]$ is constant
u is a single feed URL

IV. Evaluation

I evaluated the update strategies where goal is to minimize the data transfer and wait state of CPU to complete a batch of requests. I compared the data for 2 weeks for all the update strategies mentioned in this paper.

A. Number of Polls

I also compared the number of polls made to servers over a period of 2 weeks. These results clearly shows that that fix update strategy, as expected, fires most request at constant rate. On the other hand Moving Average and enhanced Moving Average strategies produces similar results but adapt themselves over time.

B. Network Traffic

Under this evaluation, I compared the amount of network traffic transferred by these update strategies for a period of 2 weeks.

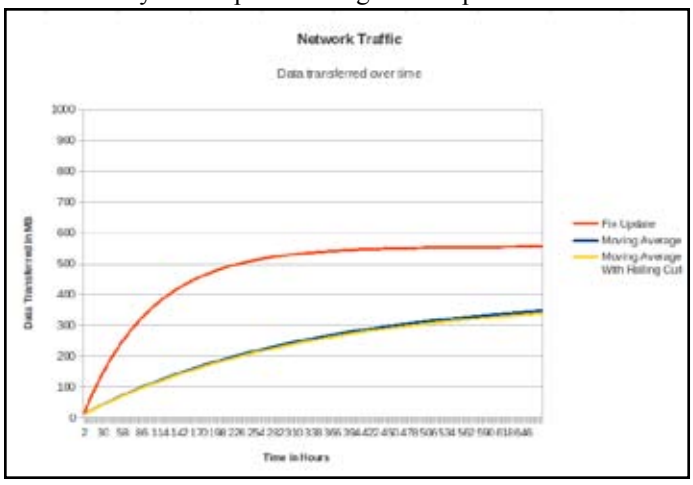

Fig. 1: Network Traffic

As we can see that fix update transfers the maximum amount of data over time. Whereas, other two strategies adapt themselves and perform better over the time. Here we can see that enhanced Moving Average performs slightly better than Moving Average and transfers slightly lesser amount of data over time.

C. Time Taken to Finish a Set of Request

Under this analysis, we compared the total time taken by all three strategies to finish the processing of a set of requests over time. It can clearly be seen that time taken by fix update is very high compared to other strategies. Also, the graph is not smooth, which is because of the nature of sequential polling.

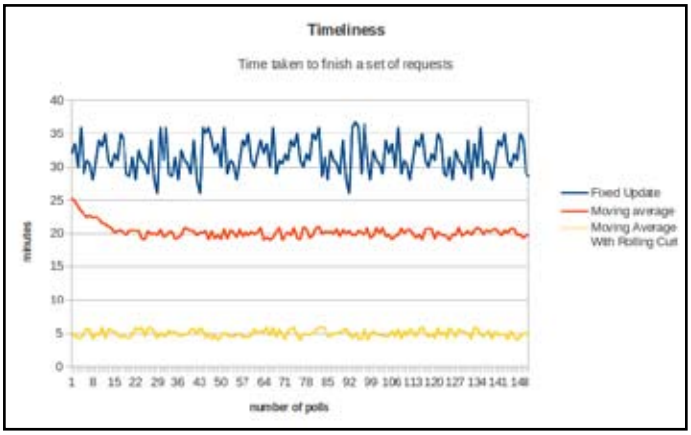

Fig. 2: Timeliness

Table 1: Average Time Taken by Strategies

Strategy	Average Time
Fix Update	31.59
Moving Average	20.26
Moving Average with Rolling Curl	4.99

Here, we also note that that time taken by enhanced Moving Average is much less than the Moving Average and fix update strategy. And it can be clearly inferred from fig. 2 and Table 1 that enhanced Moving Average takes less time to finish processing of a set which saves a lot of CPU cycles.

V. Conclusion and Future

In this research I have evaluated all the update strategies proposed in the paper. And it can easily be inferred from the results and fig. 2 and Table 1 that time taken by Moving Average with rolling curl has shown significant improvement over algorithm based only on Moving Average and fix update. The Moving Average with Rolling curl is ~700% faster than Fix Update and ~500% faster than Moving Average.

This technique can also be easily extended to build an efficient sentiment analysis engine. Since the proposed technique is fast and more accurate, one can for example build a sentiment analysis engine for a brand and monitor its health on-line.

VI. References

[1] Wikipedia (2012). "RSS" [Online] Available: http://www.en.wikipedia.org/wiki/RSS.

[2] Wikipedia (2012). "cURL" [Online] Available: http://www.en.wikipedia.org/wiki/CURL.

[3] Wikipedia (2012). "PHP" [Online] Available: http://www.en.wikipedia.org/wiki/PHP

[4] TIOBE (2012). "TIOBE Programming Community Index for July 2012" [Online] Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[5] Wikipedia (2012). "Moving Average" [Online] Available: http://www.en.wikipedia.org/wiki/Moving_average

[6] Google (2010). "rolling-curl - Project Hosted on Google Code", [Online] Available: http://www.code.google.com/p/rolling-curl/

[7] Google (2010). "pubsubhubbub - Project Hosted on Google Code", [Online] Available: http://www.code.google.com/p/pubsubhubbub/

[8] Urbansky, D., Reichert, S., Muthmann, K., Schuster, D., and Schill,"An Optimized Web Feed Aggregation Approach for Generic Feed Types", Proceedings of the 5th International AAAI Conference on Weblogs and Social Media, 638 641, 2011.

Neeraj Kumar received his B.Tech degree in Computer Science and Engineering from Haryana Engineering College, Haryana, India in 2011. He was an Executive Software Engineer with Virtuos Solutions in 2008 - 2009. Currently, he is working as a Project Manager (Technology) with 2020MSL - a leading Social Media company in India. His research interests include artificial intelligence, machine learning and cognitive sciences.