Friticores: A RSS Feed Monitoring and Dissemination System

Mesaac Makpangou, Bassirou Ngom, Samba Ndiaye

Friticores: A RSS Feed Monitoring and Dissemination System

Mesaac Makpangou[1], Bassirou Ngom[1,2], and Samba Ndiaye[2]

[1] REGAL Team
INRIA / LIP6 (UPMC)
Paris, France
firstname.lastname@lip6.fr
[2] Départment Mathématiques-Informatique, Faculté des sciences et techniques
Université Cheikh Anta Diop
Dakar, Senegal
firstname.lastname@ucad.edu.sn

Abstract. *RSS feeds is a simple information medium that permits to reduce the information discovery delay. It requires some effort to consumers to find suitable information especially if they ignore providers that could provide it. Providers that lack an established reputation are marginalized by users. This paper presents Friticores, a RSS feed monitoring and dissemination system that addresses both issues. With respect to providers, it stands as a dissemination system that makes each information update visible to all interested users. With respect to consumers, Friticores extends the traditional RSS feed service with filtering service and virtual feeds. The key contributions presented comprise the architecture of the system and the services offered to both consumers and providers, a partitioned persistent representation of feeds, and a distributed index scheme. As for the status of the work, the design is complete and the development of a simulation prototype is ongoing.*

Keywords: RSS feed, content indexing, over-dht index, keyword-based search

1 Introduction

With the development of web 1.0 and web 2.0 applications (e.g., e-store, blogs, social networks, wikis and mashups) Internet users who were passive consumers became content providers. Nowadays, number of users are interested in discovering without delay new updates from their favorite providers. RSS feeds allow contents providers to publish announcements of new information updates available at their sites; then users interested in receiving announcements from a particular provider, subscribe to his feed. Once subscribed to a feed, they receive all announcements that flow through that feed. Hence, RSS feeds provide a simple information dissemination mechanism that permits to reduce the delay between the occurrence of a new content and its discovery by interested users.

While this mechanism is a good match for consumers interested in contents from a few providers, it is less convenient for users interested in a large number of providers. Consider for instance a researcher interested in new scientific publications covering his research topics. A priori, one affordable solution is to subscribe to feeds provided by leading research groups, organizations supporting major conferences and journals of his domain, universities and research institutes, and major digital libraries. However these providers produce or distribute contents that concern the particular topics of this researcher and many others. Hence, this researcher needs support to filter out relevant information from the huge volume of received updates.A number of filtering facilities [22, 4, 24] already exist. One limitation for these solutions is bandwidth consumption: users still have to retrieve useless updates. For users with limited resources, wasting bandwidth to download useless updates might be an issue. Another problem that faces feeds users is the the difficulty to find all suitable feeds that treat their favorite subjects. Search engines such as syndic8 [27] allow users to find relevant feeds that match their requirements. However, these engines rely on static descriptions of existing feeds. Unfortunately, the actual contents provided over time by a provider often diverge from the initial static description of his feed. Such mismatch might lead to poor quality of service for search engines exploiting static descriptions of feeds. Beside, a continuous monitoring by users themselves would be too expensive in term of time, bandwidth, storage and processing.

On the provider's side, the main issue is the lack of visibility and reputation for certain providers. Consider for instance the case of the universities and research laboratories in Africa whose the quality of research are ignored by many users. This lack of visibility prevents users to subscribe to their feeds. The consequence is that the feeds will be released in a limited way, and the content posted by these sites will have limited impact. Moreover, the maintenance requires a significant investment to provide an acceptable quality of service. Insecurity, limited resources available, and the low impact of RSS on the universities's reputation lead them to sacrifice the RSS using and perpetuate the marginalization of the content of these universities. For such a universities, we need a dissemination system that gives them a fair chance to let interested users discover their publications, while devoting a minimum of resources to this service. A P2P network for the dissemination of RSS feeds to RSS users is proposed in [20, 21]. While this proposal balance the access load over participating servers, it does not help users discover relevant information, regardless their providers.

In this paper we propose *Friticores*, a RSS Feed Monitoring and Dissemination System designed to help researchers from underdeveloped Countries. *Friticores* allows a user to subscribe and to be notified in real-time when updates that match his subscriptions are available. Our scheme is based on a set of peers that collaborate to retrieve and disseminate RSS feeds. First, the system stands as a content-based publish/subscription system for RSS feeds that eliminates the provider's burden, that filters content before sending it to users who can set the content in which they wish to subscribe. Second, it consists of a keyword-based information retrieval system that permits users to catch up missed updates and

to discover new feeds that concern their desired contents. This gives a fair chance to each provider to have his updates notified to all interested subscribers.

The main contribution of the paper is three-fold. First, a simple partitioned representation of RSS feeds that permits to efficiently store and retrieve feeds within a structured peer-to-peer storage network. A feed is represented as a set of immutable data, plus a list of keys that can only extend. Second, an index scheme that permits to offer to users keyword-based queries and subscriptions. This scheme relies on Bloom filter [1] to enforce a compact encoding of keyword sets as well as efficient matching operations. Third the system proposes a number of novel services and implements an architecture that permits to achieve them.

The rest of the paper are organized as follows. Section 2 discusses relate work. Section 3 presents the overview of the system, then Section 4 details some key aspects of the system. Finally, Section 5 points some future work and concludes the paper.

2 Related Work

Many proposed mechanisms [5, 17–19, 25] exploit XML document structure to handle complex and semantic queries expressed with XPath language. Our work differs from these proposals in that we consider keyword-based queries. Our work is more closely related to indexing schemes in P2P systems and to RSS feed filtering and dissemination systems.

2.1 Indexing Schemes

Distributed Bloom Filter. Jamard et al [19] index XML documents and paths by introducing Distributed Bloom Filters (DBF). Bloom filters are divided into segments of equal size and segments are distributed in the network. Segments with a high probability of false positive are replaced by larger segments. For each segment number i, there exists a controller node responsible on which all i^{th} segments are stored. Bloom filters are associated with topics and a shared catalog lists all available topics. Query processing takes place in three phases: the first step accesses all controllers that are responsible for a segment; the second step contacts peer servers; and the final step eliminates false positives. Additional maintenance such as segment replacement and shared catalog update introduce index overhead. Moreover the index depends on a hight availability of the node controller since it is the responsible of all i^{th} segments and its failure breaks all the system. Our system splits content keys into chunks, in the same way Bloom filters are decomposed here into segments. One difference is that we intend to use our index to perform broad semantic matchings.

Prefix Tries Indexes. Data search on DHTs is made deterministically. To support complex queries, many over-DHT indexes have been proposed[3]. PHT[15]

[3] A detailed survey of complex query supports in DHT based P2P systems can be found in [16]

is an index structure that uses a prefix tree built over a DHT. Nodes are identified by a D-bit prefix. PHT stores objects at the tree leaf nodes. The trie structure is distributed over the same structured P2P storage system as the indexed contents. PHT maintains a double list that connects all leaf nodes and supports object insertion and deletion. A leaf node that already stores more than B objects must be split before inserting a new key. One challenging issue with PHT is performance, especially if one has to split nodes very often.

Tang et al propose LIGHT[6] to improve over-DHT indexes performance. LIGHT is also based on prefix tree and consists of a trie structure, a data structure called "leaf bucket" and a naming function. When LIGHT split a node, it keeps the half data on the local node (split node), while the remaining half is transferred to the remote node. Given a key, LIGHT-lookup returns the corresponding DHT-key. For this, the label of the "leaf bucket" that covers the key must be found, then applying the naming function allows to get data DHT-key. LIGHT consumes less bandwidth than PHT.

However, PHT and LIGHT indexes yield low performance when performing an inexact query search (such as matching keyword queries) since its search mechanism uses key prefix which yields poor performance for a content-based routing scheme.

Inverted list indexes. An inverted list is a set of pairs (w, O), where w is a keyword, and O is the set of objects containing that keyword. A keyword search method in a P2P networks can use a distributed index based on the inverted list [10, 11]. The naive approach is to distribute the entire structure in the network so that each keyword is assigned to a node that indexes objects having this keyword. This naive approach has a number of drawbacks: high bandwidth consumption, uneven load of nodes, redundant storage of popular keywords, and the weak filtering of information when querying with popular keywords.

Wang et al [13] use counting Bloom Filter to resolve "AND" and "OR" queries in P2P networks. Intermediate results intersection are avoided while satisfying a multiple keywords based query. A node that receives an "AND" query sends its filter to the other nodes having the query keywords. Each node does locally the intersection with its local filter and return the result. The requesting node performs locally received results and returns the response. A node that receives an "OR" query sends its local stored keywords that matches the query to the requesting node at the same time it sends the filter to other nodes, once a node receive the filter, it makes the difference between its local keywords and all received keywords, then it returns the result directly to the requesting node.

A survey of solutions that improve the naive inverted list solution can be found in [8, 7].

Hypercube-based Index Joung et al [12] addresses a number of drawbacks inherent to the use of keyword inverted lists. They propose to rely on a r-dimension hypercube $H_r(V, E)$ to build the index. Two types of searches can be performed: pin search and superset search. A pin search find the sets of entries for a given set of keywords. A superset search aims to return a set of objects that can describe

a given keyword set. Superset search is conducted with a spanning tree. Browsing that tree allows to find all supersets describing the keyword set. Superset search is expensive since several nodes can be visited unnecessarily, this increases the number of messages if the query contains few keywords. To eliminate these problems, Szekeres et al [9] propose indexing and searching algorithms based on ad-hoc spanning trees. They construct paths from object's keyword-set to all the subsets of this keyword-set. Keywords are lexicographically ordered. However this requirement is not easily feasible in a content retrieval system where users use different writing style.

2.2 RSS Feeds Filtering and Dissemination

Zhou et al propose SoSpace [22] a search engine for RSS feeds. A node in SoSpace can either be a provider or a RSS aggregator. In addition to the feeds, Each node keeps both the RSS feeds and an inverted list of all the terms that appear on them. An item in a RSS feed is treated by SoSpace as a document. The query processing follows three steps: initially the requesting node sends messages to nodes responsible for the query terms asking posting list or bloom filter of each term, then the requesting node collects and combines responses and finally the requesting node gets the documents description.

Weixiong et al [2] allow users to subscribe to their favorite content using keywords. The system consists of a service of RSS feeds aggregation and a services of feeds distribution which send the most k relevant feeds to the user. A user registers its keywords-based subscriptions (query or filter) on the alert agent which creates an URL for him. The user periodically polls the URL as in the actual case. The relevance between a document and a query is based on "rendezvous" of terms. If a node responsible of a document term receives a document, it forwards it to all registered alert agent. Each alert agent maintains a set of k documents candidates. Techniques such as terms selection and documents pruning are used to increase the performance.

Our proposal differs from these two solutions mainly on the indexing scheme. They rely inverted list (or equivalent scheme) while we implements an over-dht index.

3 Friticores Overview

Friticores is a feed monitoring and dissemination system. A feed is statically characterized by its location and a description of information that it provides. From a dynamic point of view, a feed models a continuous flow of resource's advertisements (called `items`). Each `item` comprises: a title, the summary of the advertised resource, the `URL` where one can access this resource. We assume that both feeds' descriptions and items summaries are given as sets of keywords.[4]

Friticores provides to its users four basic services.

[4] If this were not the case, number of tools exist that can help extract keywords from full text.

– registerFeed(FeedLocation url, FeedDescription desc);
– publishItems(FeedLocation url, ItemList items);
– ItemRefList search(FeedLocation url, Set<Keyword> keywords);
– subscribe(FeedLocation url, Set<Keyword> keywords);

To providers the possibility to register feeds and to publish items; to consumers, the possibility to receive items published within a particular feed or related to a particular topic (the feed location is set to null); consumers can either formulate queries or subscribe for asynchronous notifications of suitable items when available. To request a topic-based query or subscription, one simply specifies a null URL as feed location.

The system is structured in three layers. Figure **??** shows the main components of the system and how they interact with one another.

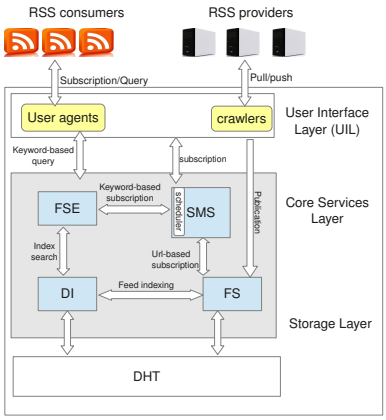

Fig. 1. Friticores architecture

The top layer implements the system API while the lowest layer is the storage system. The intermediary layer comprises the components that implement the services offered to users. These include the feed store (FS), the feed search engine (FSE), and the subscription management system (sms). The fourth component is used by other components to achieve their service.

FS component implements primitives offered to providers: `registerFeed()` that permits to store feeds' descriptions and`publishItems()` which is used to publish information updates (or items). The FS component provides an additional primitive that permits to retrieve items (or updates) published at a feed.

FS stores feeds' descriptions and updates in the underlying DHT-based storage overlay network. FS cooperates with distributed index component to index stored feeds.

The feed search engine component solves each query and returns feeds or items that meet the query criteria to the requester. FSE relies on the Index component to determine the feeds that match the query criteria.

The subscription management component manages users' subscriptions. One virtual feed is associated with each subscription. The heart of this component is a scheduler that repeats periodically, for each virtual feed, the corresponding query to the FSE component; returned feeds are notified to subscribers.

The distributed index component maintains the index structure over the DHT-based storage system. It hides detailed representation of the index to other companion components. It provides them two basic primitives: one permits to add new keys in the index, and the other permits to lookup keys of contents that match the query criteria.

4 System Functioning

This section describes the details some key points of the system, in particular the system api, partitioned representation of feeds, and the index scheme.

4.1 System Api

Users post their demands (queries or subscriptions) through Users' Agents (UA). A demand can be a url-based/topic-based subscription or query.

Url-based subscriptions permit users to obtain the current functioning of RSS. Users may specify a keyword-based filter such as to prevent notifications of unsuitable feeds.

When an user wants to do an foresight on a given topic, he can not subscribe to all providers that treat the topic. The UA component allows him to post a set of keywords that describes the desired topic. We call such subscriptions topic-based subscriptions.

For each subscription (topic-based or url-based), the UA creates a virtual feed composed by all items matching the subscription. When a new item matches a virtual feed description, his subscriber will be notified.

If an user wants to find existing items, he posts a query request rather than a subscription. Again, the system supports both url-based and topic-based queries.

The difference between a subscription and a query is that the former is a lasting query while the latter is an instantaneous search.

UAs rely on the underlying Search Engine to handle queries and on the subscription management system to handle subscriptions.

On the providers' side, crawlers monitor the web and register new feeds. They are also responsible of monitoring information updates and to publish them.

4.2 Partitioned Representations of Feeds

We use Bloom filter to code feeds' descriptions and updates' summaries as well.

We partition the sequence of items published by a feed according to their publication date. We define a sub-feed as a set of items published at the same date by the same feed; it may contain any number of items. We associate to each sub-feed an identifier that distinguishes it from other sub-feeds of the same feed, and the union Bloom filter of Bloom filters characterizing its contained items.

Each sub-feed is stored within the storage network using its characteristic Bloom filter as key. To represent the whole feed, we store an object that contains the keys that identify its subfields. We use the feed Bloom filter as the global feed object's key.

Overall, the persistent representation of a feed is one feed object and a set of *item containers* (one per sub-feed).

4.3 Friticores Index Scheme

The propose index scheme is designed specially to solve matching requests sent by the users agents or by the subscription management system. It ensures the broad semantic matching of received keyword-based queries against stored objects represented by their descriptions given as a keyword set.

The baseline of our solution is three-fold. Firstly, we build an index of Bloom filters used as storage keys to store feeds and sub-feeds. Secondly, given a set of keywords that define a query, we compute the Bloom filter corresponding to these keywords,relying on the same set of hash functions that were used to compute the Bloom filters used as storage keys. Finally to locate items that match a query, we search for indexed keys that match the query Bloom filter. The matched keys are then used to retrieve suitable feeds.

Bloom filter matching operation, per say, is simple and efficient. The issue here is to build a distributed key index, stored on the DHT and which supports efficiently the matching process (which can be viewed as a form of complex query). A number of existing solutions to support complex queries rely on `PHT` [15] or on hypercube [12]. We have considered using these data structure; however, it turns out that they did not fit well for our case. In particular, our Bloom filters are bitmaps of large size. Using a `PHT` or an `hypercube`-like structure to represent our index yield poor performance for the matching operation.

The solution we have adopted is to split each key of size m into k chunks and to assign a common unique group identifier g_i to all chunks of the same key. Each chunk is characterized by its position pos (from 0 to $k-1$) and its value bf (a substring of the initial Bloom filter). We index chunks instead of entire keys. Each posting in the index structure is then a 3-tuple ($< bf, position, g_i >$). A chunk ($< bf, position, g_i >$) is stored using bf as key. To lookup contents that match a wry, we follow the same process:(i) computation of the Bloom filter that characterizes this request; (ii) splitting this filter in chunks relying on the splitting algorithm; (iii) lookup individual chunks; (iv) merge different returned chunks to reconstitute matched contents keys.

The detailed description of the index structure and of the add and lookup protocols are out of the scope of this paper. One can easily convince himself that chunks of different positions and possibly from different keys could have the same storage key. Hence the chunk index structure and the lookup require careful design.

5 Conclusion

We have presented Friticores, a RSS feed monitoring and Dissemination system. Friticores is primary destined to feed users (consumers and providers) with limited resources or willing to avoid wasting of resources. Friticores also aims to give each provider a fair chance to improve its visibility with respect to users that could be interested in his publications, thanks to the provision of topic-based queries and subscriptions. We highlighted the a partitioned representation of feeds that eases feed storage within a dht-based storage system, and our index scheme that permits efficient matching of keyword-based queries against feeds' descriptions. We detailed the system architecture and how components interact to achieve both novel and the traditional feed services.

We are implementing a simulation prototype in oder to evaluate Fritcores performances. In a near future, we plain to setup a monitoring system such as to collect real data that will be used to defined a benchmark for our evaluation.

References

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM. 13, 422–426 (1970).
2. Rao, W., Chen, L.: A distributed full-text top-k document dissemination system in distributed hash tables. World Wide Web. 14, 545–572 (2011)
3. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11, 17–32 (2003)
4. Yuan, M.W., Jiang, P., Zhu, J., Wang, X.N.: Sensing Semantics of RSS Feeds by Fuzzy Matchmaking. Intelligent Information Management. 2, 110–119 (2010)
5. He, W.: Bloom filter-based keyword search over XML data in structured Peer-to-Peer systems. Science and Information Technology ICCSIT. (2010)
6. Tang, Y., Zhou, S., Xu, J.: LIGHT: A Query-Efficient Yet Low-Maintenance Indexing Scheme over DHTs. IEEE Trans. on Knowl. and Data Eng. 22, 59–75 (2010)
7. Passarella, A.: Review: A survey on content-centric technologies for the current Internet: CDN and P2P solutions. J. Comput. Commun. 35, 1–32 (2012)
8. Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: search methods. J. Comput. Netw. 50, 3485–3521 (2006)
9. Szekeres, A., Baranga, S.H., Dobre, C., Cristea, V.: A keyword search algorithm for structured peer-to-peer networks. J. Grid Util. Comput. 2, 204–214 (2011)
10. Gnawali, O.D.:A Keyword Set Search System for Peer-to-Peer Networks. Massachusetts Institute of Technology, (2002)

11. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: ACM/IFIP/USENIX 2003 International Conference on Middleware, pp. 21–40. Springer-Verlag New York Inc, New York (2003)
12. Joung, Y., Fang, C., Yang, L.: Keyword Search in DHT-Based Peer-to-Peer Networks. In: 25th IEEE International Conference on Distributed Computing Systems, pp. 339–348. IEEE Computer Society, Washington (2005)
13. Wang, M., Zhang, D., Tian, X., Bi, X., Zeng, B.: Multi-keyword search over P2P based on Counting Bloom Filter. In: 2nd International Conference on Networking and Information Technology IPCSIT(2011)
14. Liu, H., Ramasubramanian, V., Sirer, E.G.: Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In: 5th ACM SIGCOMM conference on Internet Measurement, pp. 3–3. USENIX Association, Berkeley (2005)
15. Sriram, R., Sylvia, R., Joseph M.H., Scott, S.: Brief Announcement: Prefix Hash Tree. In: 23rd ACM Symposium on Principles of Distributed Computing. St John's, Newfoundland (2004)
16. Hidalgo, N.: Towards an Efficient Support for Complex Queries on Structured Peer-to-Peer Networks. Université Pierre et Marie Curie - Paris 6. (2011)
17. Bonifati, A., Matrangolo, U., Cuzzocrea, A., Jain, M.: XPath lookup queries in P2P networks. In: 6th annual ACM international workshop on Web information and data management, pp. 48–55. ACM, New York (2004)
18. Koloniari, G., Pitoura, E.: Content-based routing of path queries in Peer-to-Peer systems. In: 9th International Conference Extending Database Technology (EDBT'04)
19. Jamard, C., Gardarin, G., Yeh, L.: Indexing textual XML in P2P networks using distributed bloom filters. In: 12th international conference on Database systems for advanced applications, pp. 1007–1012. Springer-Verlag, Berlin, Heidelberg (2007)
20. Sandler, D., Mislove, A., Post, A., Druschel, P.: FeedTree: sharing web micronews with peer-to-peer event notification. In: 4th international conference on Peer-to-Peer Systems, pp. 141–151. Springer-Verlag, Berlin, Heidelberg (2005)
21. Jun, S., Ahamad, M.: FeedEx: collaborative exchange of news feeds. In: 15th international conference on World Wide Web, pp. 113–122. ACM, New York (2006)
22. Ying, Z., Xin, C., Chen, W.: A Self-Organizing Search Engine for RSS Syndicated Web Contents. In: Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on. (2006)
23. Taddesse, F.G., Chbeir, R.: Semantic aware RSS query algebra. In: 12th International Conference on Information Integration and Web-based Applications & Services, pp. 291–298. ACM, Paris, France (2010)
24. Tomàs, J. C., Amann, B., Travers, N., Vodislav, D.: RoSeS: a continuous content-based query engine for RSS feeds. In: 22nd international conference on Database and expert systems applications - Volume Part II, pp. 203–218. Springer-Verlag, Toulouse, France (2011)
25. Xiaochuan, Y., Alvin, C.T.S.: A Time/Space Efficient XML Filtering System for Mobile Environment. In: the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01, pp. 184–193. IEEE Computer Society, Washington (2011)
26. Rao, W., Fu, A.W. Chen, L., Chen, H.: STAIRS: Towards Efficient Full-Text Filtering and Dissemination in a DHT Environment. In: the 2009 IEEE International Conference on Data Engineering, pp. 198–209. IEEE Computer Society, Washington (2009)
27. Syndic8, `http://www.syndic8.com/`