CMS-ToPSS: Efficient Dissemination of RSS Documents

Milenko Petrovic Haifeng Liu Hans-Arno Jacobsen

University of Toronto
petrovi@eecg.toronto.edu hfliu@cs.toronto.edu jacobsen@eecg.toronto.edu

1 Introduction

Recent years have seen a rise in the number of unconventional publishing tools on the Internet. Tools such as wikis, blogs, discussion forums, and web-based content management systems have experienced tremendous rise in popularity and use; primarily because they provide something traditional tools do not: easy of use for non computer-oriented users and they are based on the idea of "collaboration." It is estimated, by `pewinternet.org`, that 32 million people in the US read blogs (which represents 27% of the estimated 120 million US Internet users) while 8 million people have said that they have created blogs.

Web-based collaboration is the common idea for this new breed of content-management tools. The center piece of such a tool is a web page that is being used as an area where multiple users participate in content creation. More significantly, the collaboration enabling tool used is the web page itself (accessed through the all-pervasive web browser).

With these new web applications, there rouse a need for users to stay informed about changes to the content. In general, users want to be updated about daily news headlines of interest to them, or be notified when there is a reply in a discussion they participate in, or their favorite web personality has updated his/her blog (online diary etc.).

RDF Site Summary (RSS), a RDF-based language for expressing content changes, is an application on the Web that has grown considerably in popularity. The RSS specification is so flexible, that any kind of changes can be described starting from blog updates, source code changes, forum discussions to database content changes and others, but it is up to the end user application to "understand" what the content is. Being based on RDF helps RSS to this end, since RDF is a language designed for expressing metadata.

RSS[1] is quickly becoming the dominant way to disseminate content update notifications on the Internet. `pewinternet.org` reports that 6 million people in the US use RSS aggregators (a service/application that monitors large numbers of RSS feeds).[2]

Web-based content management systems (CMS) have also grown in popularity mainly because they are based on the publishing tools just described, but also because they are much easier to use and maintain than traditional CMS.[3] Like traditional CMS systems, they provide content access control, user profiles, persistent storage, web access, RSS authoring, advanced content management, content routing and taxonomic content classification.

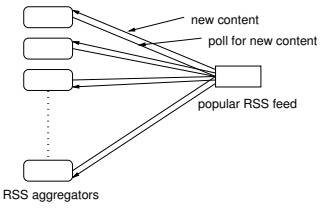

Figure 1: Current RSS dissemination architecture

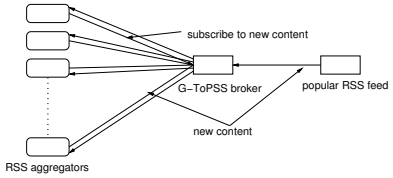

Figure 2: CMS-ToPSS RSS dissemination architecture

Existing RSS systems do not scale well. Anecdotal evidence suggests that websites hosting popular RSS feeds can be significantly overloaded with useless network traffic due to the architecture of the current

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

[1] `web.resource.org/rss/1.0/spec`

[2] Reported by Pew Internet & American Life Project (www.pewinternet.org), an organization that produces reports that explore the impact of the Internet on families, communities, the daily life. Also reported by "RSS at Harvard Law" (blogs.law.harvard.edu/tech/)

[3] Mid Market Web CMS Vendors Pull Ahead. Brice Dunwoodie. `CMSwire.com`

RSS delivery systems. Figure 1 illustrates the scalability problem. Multiple RSS aggregators continuously *poll* numerous RSS feed sites. Anecdotal evidence suggests that the way RSS dissemination is currently done can severely affect the performance of websites hosting popular RSS feeds.[4]

The challenge is to build a data management architecture that avoids the inherent polling-based design, supports the rich, on graphs based RDF meta-data model and query languages, supports efficient filtering of this kind of data, and offers scalability, as more users publish and expect to stay current with changes submitted by others.

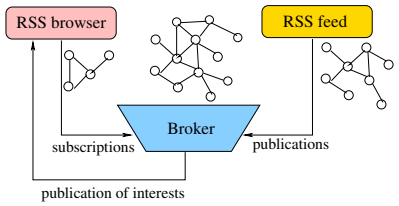

Figure 3: RDF Site Summary Dissemination System based on CMS-ToPSS

In this paper, we describe CMS-ToPSS, a novel extension to content management systems, for scalable dissemination of RSS documents, based on the publish/subscribe model. The extension provides fast routing and filtering of RSS documents as well as timely delivery of publications in a scalable manner using a novel graph-based meta-data filtering algorithm [11]. Figure 3 shows the architecture of CMS-ToPSS. The new information system architecture significantly reduces the number of unnecessary polls of RSS feed sites (see Figure 2).

To illustrate the effectiveness of the system, we extend an existing open-source web-based content-management system, Drupal (`drupal.org`) to use CMS-ToPSS in a manner that is transparent to end users, yet providing an efficient content-routing architecture.

2 Related Work

Use of the publish/subscribe communication model for selective information dissemination has been studied extensively. Existing publish/subscribe systems [8, 1, 5, 3] use attribute-value pairs to represent publications, while conjunctions of predicates with standard relational operators are used to represent subscriptions. Systems such as those described in [2, 6] use XML to express publications and XPath as the subscription language. CMS-ToPSS uses a new graph-based matching algorithm [11] that allows fast filtering of graph-based subscription languages such as those based on RDF which are necessary for filtering of RSS documents.

OPS [12] uses a very general subgraph isomorphism algorithm for matching over RDF graphs. However, this approach, as we have shown [11], unnecessarily increases the matching complexity.

Racer [9] is a publish/subscribe system based on a description logics inference engine. Racer does not scale as well as CMS-ToPSS (matching times are in the order of 10s of seconds even for very simple subscriptions vs. 10s of milliseconds for G-ToPSS), but it does have more powerful inference capabilities.

CREAM [4] is an event-based middleware platform for distributed heterogeneous event-based applications. Its event dissemination service is based on the publish/subscribe model. Similar to other publish/subscribe systems, the subscription and publication model in CREAM, is based on attribute-value pairs. Attributes and values can be associated with semantic information from an ontology. Unlike CMS-ToPSS, which uses RDF, ontology and data are represented in a CREAM-specific data model.

3 System Architecture

CMS-ToPSS employs the publish/subscribe, data-centric communication model [7]. There are three main entities in this model: publishers, subscribers and brokers. Publishers send all data to a broker (or a network of brokers). Subscribers register with the broker their interest in receiving specific data. The role of the broker is to mediate communication between the publishers and the subscribers by matching the published data with the interests of the subscribers. This way the subscribers do not need to know who is publishing the data, as long as the data meets their specific interest, and the publishers do not need to know who are the ultimate receivers of their publications. This provides decoupling of senders and receivers of data both in space and time, which makes the publish/subscribe paradigm particularly well suited for structuring of large and dynamic distributed systems such as RSS feed dissemination for example.

A CMS-ToPSS broker relies on G-ToPSS [11] to perform RDF filtering. G-ToPSS is based on the following data model consisting of three components: publications, subscriptions and an ontology.

A publication is represented as a directed labeled graph. We interpret the graph as a set of RDF triples $(subject, property, object)$. Each triple is represented by a node-edge-node link (as shown in Figure 4). *Subject* and *property* are URI references, while *object* is either an URI reference or a literal. In all, a publication is a directed graph where the vertices represent *subject*s and *object*s and edges between them represent *properties*.

The following illustrates a publication about one paper published in the 2001 SIGMOD conference expressed in G-ToPSS publication language (a simplified

[4]InfoWorld RSS growing pains. Chad Dickerson. *CTO Connection*, `infoworld.com`

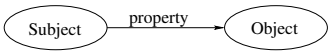

Figure 4: RDF triple graph

RDF serialization):

```
BEGIN ID Publication1
(uri:ArnoPaper,uri:author,lit:Jacobsen)
(uri:ArnoPaper,uri:title,lit:FastFilter)
(uri:ArnoPaper,uri:conference,uri:SIGMOD)
(uri:SIGMOD,uri:location,lit:California)
(uri:SIGMOD,uri:year,lit:2001)
```

A subscription is a directed graph *pattern* specifying the structure of the publication graph with optional constraints on some vertices. A subscription is represented by a set of 5-tuples (*subject, property, object, constraintSet(subject), constraintSet(object)*). Constraint sets can be empty.

Similar to the publication data model, each subscription 5-tuple can be represented as a link starting from *subject* node and ending at the *object* node with the *property* as its label. From the publication data model, we know that each node is labeled with a specific value. However, in a subscription, we also allow *subject* and *object* to be either a constrained or unconstrained variable. The unconstrained variable matches any specific value of the publication, in the entire domain while the constraint variable matches only values satisfying the constraint. A constraint is represented as a predicate of the form $(?x, op, v)$ where $?x$ is the variable, *op* is an operator and v is a value.

For example, the following illustrates a subscription that specifies interest in a paper published at the SIGMOD conference after the year 2000 expressed in the G-ToPSS SQL-like subscription language (GQL), based on RDQL [10]:

```
ID Subscription1
SELECT ?z WHERE
(uri:ArnoPaper,uri:author,lit:Jacobsen)
(uri:ArnoPaper,uri:conference,uri:SIGMOD)
(uri:SIGMOD,uri:year,?z)
SUCHTHAT (?z > 2000)
```

This type of constraint is used for literal value filtering. Similarly, the following subscription is looking for Arno's publication in a conference after 1999:

```
ID Subscription2
SELECT ?x, ?y WHERE
(?y,uri:author,lit:Jacobsen)
(?y,uri:conference,uri:SIGMOD)
(uri:SIGMOD,uri:year,?x)
SUCHTHAT (?y <= Publication) AND (?x > 1999)
```

There are two variables; the one constraining the year is a literal value filter; the other is a semantic constraint which uses a class ontology (see Figure 5). Only an instance in the publication that is a descendant of the "Publication" class is going to match.

For each 5-tuple (*subject, property, object, constraintSet(subject), constraintSet(object)*) in subscription graph G_S, if there is at least one triple (*subject, property, object*) in publication G_P such that the *subject* and *object* nodes are matched and linked by the same *property* edge then G_P matches G_S. The nodes that match are either the same (i.e., their labels are lexicographically equal) or the node in G_S is a variable for which the value of the node in G_P satisfies all constraints associated with the variable.

For example, the `Subscription1` is matched by the `Publication1` since the publication contains the same RDF triples (*ArnoPaper, author, Jacobsen*), (*(ArnoPaper, conference, SIGMOD*), and (*2001 > 2000*), thus (*SIGMOD, year,?x(?x > 2000)*) is satisfied.

As illustrated in `Subscription2`, G-ToPSS allows the use of an ontology to specify taxonomy constraints on *subject*s and *object*s. G-ToPSS uses RDF's class taxonomy with the *is-a* relationship to represent semantic information about a *subject* or an *object* that is available in the ontology. Multiple inheritance is allowed and the only restriction on the taxonomy is that it must be acyclic.

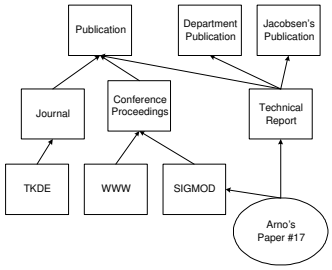

Figure 5: Example Ontology

In Figure 5, we show an example of a class taxonomy about an academic bibliography system. Class "Publications" includes three subclasses: "Journal", "Conference Proceeding" and "Technical Report". The document instance "Arno's paper ♯17" belongs to both "Jacobsen's Publications" and "SIGMOD" proceedings.

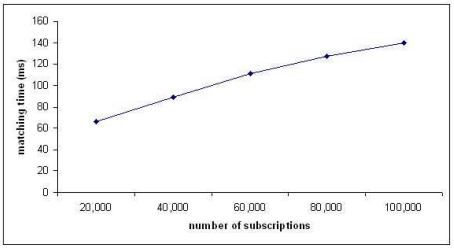

Figure 6: Matching performance

To demonstrate the scalability of CMS-ToPSS, in Figure 6, we measure the matching time with the increasing number of subscriptions. These results underline the scalability of our approach.

4 Demonstration

The demonstration system consists of two main components: Drupal module (a conent management system) and the G-ToPSS filtering service. The overall architecture is shown in Figure 7. The filtering service reads RSS feeds and GQL queries using XML-RPC over HTTP. The Drupal module acts as a client to the filtering service. The module does not require any changes to Drupal, and any Drupal installation can experience the benefits of CMS-ToPSS by simply retrieving and installing the module.

G-ToPSS filtering service is accessible via XML-RPC and can be accessed by any XML-RPC client. G-ToPSS inputs are publications and subscriptions serialized as RSS feeds. G-ToPSS outputs are notifications which are also serialized as RSS feeds. Each subscription that a user submits is, in fact, a distinct RSS feed (containing items matching the user's subscription).

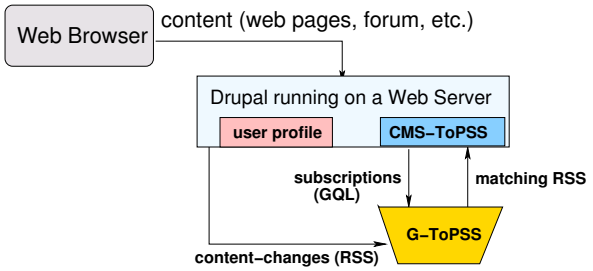

Figure 7: CMS-ToPSS system architecture

The Drupal module performs both subscribing and publishing based on user interaction with Drupal CMS. The module serializes all content changes in Drupal using RSS and sends them to the G-ToPSS filter service. The filtering service forwards the document to the interested clients which could be other XML-RPC clients as well as other Drupal modules. Note that G-ToPSS filtering service can serve multiple Drupal sites.

In addition to publishing all content changes in RSS, Drupal module also extends different kinds of Drupal content with change notification capabilities. For example, users can subscribe to receive notifications when they have replies on the discussion forum, or when a certain web page in Drupal has been updated. The Drupal module registers these kinds of subscriptions with the G-ToPSS filtering service transparently to the user.

In the demonstration, a user, using a web browser, accesses a Drupal site that is extended with the module described in this paper. The user can choose to receive notifications for content of her choice (e.g., discussion forum replies, web page updates etc.) Drupal supports convenient taxonomic content classification, which can be directly mapped to a G-ToPSS ontology. In this case, the user will get notifications only when both the content and taxonomic constraints of her subscription are satisfied. The users can also create content (e.g., participate in a discussion form or create/update a web page) to trigger notifications. The users' subscriptions are stored as part of their Drupal profile. Via the profile web page, users can review their notification requests as well as see all notifications received for those requests.

To demonstrate the query capabilities of G-ToPSS, we also allow users to subscribe directly on the RSS content by expressing their subscriptions in G-ToPSS's SQL-like subscription language (GQL). The subscriptions and their results are also shown as part of the user profile.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, pages 53–61, 1999.

[2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB*, 2000.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[4] M. Cilia, C. Bornhoevd, and A. P. Buchmann. CREAM: An Infrastructure for Distributed Heterogeneous Event-based Applications. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 482–502, 2003.

[5] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27:827–850, sep 2001.

[6] Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *ICDE*, 2002.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[8] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, 2001.

[9] V. Haarslev and R. Moller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proceedings of the International Workshop on Description Logics*, 2003.

[10] L. Mille, A. Seaborne, and A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *Proceedings of ISWC*, 2002.

[11] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS - fast filtering of graph-based metadata. In *the 14th International World Wide Web Conference*, Chiba, Japan, May 2005.

[12] J. Wang, B. Jin, and J. Li. An Ontology-Based Publish/Subscribe System. In *Middleware*, 2004.