Enriching Topic-Based Publish-Subscribe Systems with Related Content *

Rubi Boim and Tova Milo
School of Computer Science, Tel Aviv University
boim@post.tau.ac.il, milo@post.tau.ac.il

Abstract

This demonstration presents RMFinder (Related Messages Finder), a system that retains the *simplicity and efficiency* of topic-based P2P pub-sub, while providing a *richer service* where users can automatically receive all messages related to those in the topics to which they are subscribed. RMFinder is based on a novel, dynamic, distributed clustering algorithm, that takes advantage of similarities between topic messages to group topics together, into topic-clusters. The clusters adjust automatically to shifts in the focus of the messages published by the topics, as well as to changes in the users interest, and allow for an effective delivery of related messages with minimal overhead.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: [Distributed networks]
; C.2.4 [**Distributed Systems**]: [Distributed applications]
; H.3.3 [**Information Search and Retrieval**]: [Clustering]

General Terms

Algorithms, Performance, Experimentation

Keywords

Publish-Subscribe, P2P, Dynamic Clustering, Related Content

1. INTRODUCTION

The amount of information available to Internet users is increasing rapidly. The need of users to be constantly updated with an up-to-date, accurate, and relevant data, out of this ocean of information, makes the publish-subscribe interaction scheme (pub-sub, for short) particularly appealing. In this demo we focus on a relatively simple class of such systems, called topic-based systems, where users subscribe to topics and are notified on messages that belong to those subscribed topics [2, 3].

The main reason for the popularity of topic-based pub-sub systems is their simplicity. It allows for a simple intuitive user interface as well as a very efficient implementation, and is thus a perfect fit for application areas where messages divide naturally into groups that correspond to users interest. A typical example is the increasingly popular RSS news syndication. An RSS system is a simple topic-based pub-sub system. Publishers publish their news by putting them into an RSS feed and providing the URL for the feed on their website. It is interesting to note that many existing RSS applications rely on a rather primitive implementation where RSS readers poll the feeds periodically. But with the continuous dramatic increase in the number of RSS users, it is anticipated that, for scalability, future implementation will move to push-based peer-to-peer (P2P) platforms.

The simplicity of topic-based pub-sub, however, has a price: A user subscribes to a set of topics (feeds) that she considers interesting. But related messages, possibly of great interest, may be published in other topics that she is not subscribed to (and may even be completely unaware of their existence). These relevant messages will never reach her. Clearly, a possible solution is to move to a more sophisticated class of pub-sub systems such as *content-based* systems [8]. In such systems, subscribers specify their interest through message filters, which are boolean queries on the message content. This added flexibility however does not come for free - content-based pub-sub typically requires much more sophisticated protocols with higher runtime overhead, as well as more sophisticated user interaction.

The goal of this demonstration is to present RMFinder (Related Messages Finder), a system that retains the *simplicity and efficiency* of topic-based P2P pub-sub, while providing a *richer service* where users can automatically receive all messages related to those in the topics to which they are subscribed. RMFinder is based on a novel, dynamic, distributed clustering algorithm, that takes advantage of similarities between topic messages to group topics together, into topic-clusters. The clusters adjust automatically to shifts in the focus of the messages published by the topics, as well as to changes in the users interest, and allow for an effective delivery of related messages with minimal overhead.

It should be stressed that our aim here is not to invent yet another topic-based pub-sub system, but rather to present a generic novel technique for better utilization of existing platforms. Indeed, while our implementation uses the Scribe[1] pub-sub system to manage topics and user subscriptions, the technique that we propose can similarly be used on any such network.

Remark The grouping of topics into sets has been previously proposed in the literature in a different context: To provide users with varying subscription granularity it was suggested to group topics into sets forming a sub-set hierarchy[7]. A main difference with the present work is the static nature of that grouping. In contrast, our solution adapts continuously the topic-clusters to the actual correlations between the topics messages, guaranteeing, as we shall see, stable good results even when the type of the messages published by the topics changes significantly.

2. RMFINDER OVERVIEW

We start by providing some background on topic-based pub-sub

*The research has been partially supported by the European Project Mancoosi and the Israel Science Foundation.

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

systems and the challenges encountered when trying to enrich them to deliver related content. Next we present RMFinder.

Topic-based pub-sub. The interfaces of typical P2P topic-based pub-sub systems share four common operations: CREATE, PUBLISH, SUBSCRIBE and UNSUBSCRIBE. To send messages, the publishers first CREATE topics. Each topic is virtually represented by an individual peer (often called a channel), which is recognized by a unique ID (called a topic-ID), and serves as a mediator between the publishers' side and the subscribers' side. To publish a message for a given topic, the publisher calls the PUBLISH operation with a specific topic-ID. The message is passed to the appropriate channel and propagated from it to the topic subscribers. To become subscribers of a given topic, interested users call the SUBSCRIBE operation, with the appropriate topic-ID. The corresponding UNSUBSCRIBE operation removes the subscription.

Enriched topic-based pub-sub. In our enriched topic-based pub-sub, before a message is sent from the channel to the subscribers, it is enriched with information on related messages recently published by other topics. This may include just the identifiers of such related messages (the topic-ID and the message id within that topic), some message summary (e.g. title and date) or the full message text.

How can one find these related messages? Channels often store recent messages for a certain time interval to allow users that were not connected to catch up. But a naive approach that simply queries all the channels (e.g. using some search criteria that defines what qualifies as related messages) is clearly infeasible: The number of topics in a typical pub-sub system is very large. Furthermore, most topics are completely irrelevant to the message at hand. Thus, querying all topics each time that a message is sent is prohibitively wasteful. To avoid this, we cluster together topics that are likely to contain related messages. This is done as follows. We build for each topic a "profile" (features set) that concisely describes the nature of messages recently published by the topic. We then use these profiles to group topics with similar profiles into a set, which we call a topic-cluster. Just like individual topics, each topic-cluster is given a unique id and is represented by a channel (peer). When such a cluster is created, the channels of its topics are informed. (In general, the clusters are not required to be disjoint and a topic may belong to more than one cluster). Now, when a publisher passes a message to the topic's channel, before propagating the message to the subscribes, the peer first sends the message to the channels of all those clusters to which the topic belongs and awaits their response. The (cluster) channels then each match the message to the profiles of the topics in the cluster. They determine which topics are most likely to contain relevant messages and query only those topics. The message is then enriched with the retrieved data and returned to the topic channel to be sent to the subscribers. To efficiently retrieve relevant messages at the queried peers, each message is also assigned a profile (features set) and a dedicated query optimization technique is used to identify messages with similar profiles.

For optimal results, we would like to form the "best" topic-clusters, s.t. most of the relevant messages are indeed retrieved while the overhead of querying irrelevant topics is minimized. While this may appear to be a traditional clustering problem, there are three requirements, derived from the specific context, which together make the problem particularly challenging.

- **Adaptivity.** A P2P pub-sub environment has a dynamic nature: not only that topics and publishers may come and go, but also the focus and type of the messages published in each topic may change over time. For instance, a sports channel may focus at different times on baseball, basketball, or soccer. A good solution thus must have a dynamic nature, continuously adapting the topics profile and clusters to the current system state.
- **Distribution.** The decentralized P2P nature of pub-sub systems, where no central coordinator has full knowledge about the systemŠs state and the topics behavior, calls for a corresponding distributed clustering algorithm.
- **Low overhead.** Finally, the continuous clustering efforts, as well as the adjustment of the topics profile, should incur only very minimal overhead, not to harm the overall system performance.

To address these requirement, RMFinder uses a novel dynamic distributed clustering algorithm that we developed recently in [10]. The algorithm employs local cluster updates to change the overall system configuration. Each local update is performed only when it is estimated to be (globally) cost effective. Furthermore, to minimize the overhead involved in gain estimations, a probabilistic component is employed to guarantee that (with high probability) gain estimation are computed only for updates that are likely to be beneficial. The clustering algorithm was originally introduced in [10] as a technique for reducing communication overheads in topic-based pub-sub systems, and is adjusted in the present work to enable efficient retrieval of related messages.

3. DYNAMIC CLUSTERING

To complete the picture, let us go briefly over the main component of the clustering algorithm and see what is required to apply it in our context.

Clusters Quality. The clustering algorithm is based on a set of local "cluster update" operations, performed by individual channels (of topics and clusters) consulting only a relatively small neighborhood. These operations include: the grouping of two individual topics to form a new cluster; the addition of a topic to an existing cluster; the merge of two existing clusters into a single cluster; and conversely the removal of topics from a cluster and the destruction of clusters. The algorithm uses a formula F that estimates the quality of the current clusters. Only updates that are determined to be beneficial (i.e. increase the value of F) are performed.

The formula F used by RMFinder is given below. It uses the following notation. T denotes the set of all topics. C denotes the current set of clusters. For a topic $t \in T$, C_t denotes the set of clusters in which t is a member, and T_t denotes the set of all topics that are members of these clusters. For two topics t, t' $RM(t,t') \rightarrow [0:1]$ is a function that estimates the likelihood for t and t' to publish related messages. (RM will be discussed in more details below). Recall that a "good" clustering is one where most relevant messages are indeed retrieved while the overhead of querying irrelevant topics is minimized. To capture this, F considers each topic $t \in T$. For each topic, the first summand estimates which portion of the related messages, out of all related messages, will be found when considering only the topics in the clusters to which t belong. The second part of the formula measures the "tightness" of the clustering - it estimates how many topics irrelevant for t, out of all the potentially irrelevant topics, might be queried by the clusters. The relative importance of these two criteria is tuned using the weight (constant) w.

$$F(T,C) = \sum_{t \in T} [\frac{\sum_{t' \in T_t} RM(t,t')}{\sum_{t' \in T} RM(t,t')} - w \frac{\sum_{c \in C_t} \sum_{t' \in c} (1 - RM(t,t'))}{\sum_{t' \in T} (1 - RM(t',t))}]$$

It should be noted that since all the cluster update operations are local in nature, to evaluate the potential benefit of an update, there is no need to evaluate the full formula but only the changes entailed to the small neighborhood involving the updated topics/clusters.

Features extraction and relevance estimation. To measure the quality of the clustering, we need to estimate the likelihood for t and t' to publish related messages. (The function RM used above). For that, we build for each topic a "profile" (features set) that concisely describes the nature of messages recently published in this topic. More specifically, for each topic we consider a sliding window that includes its recent messages of a given time interval. From this set of messages we extract a features set of size k (for some predefined constant k). The features set is incrementally updated as time advances and the window slides. In our implementation we use a simple TFIDF features selection [9], where the selected features are words that appear in the messages with highest TFIDF score. Stop words are excluded and words appearing in the title get higher weight. Finally, for two topics t, t', the likelihood for them to contain related messages is estimated by the similarity of their feature sets s, s': $RM(t, t') = \frac{|s \cap s'|}{k}$.

We will show in the demonstration that even with these very simple feature extraction and relevance estimation, `RMFinder` yields impressive results. But clearly, if desired, more sophisticated features extraction/relevance estimation can easily be plugged in (including ones that take users' feedback into consideration).

Triggering cluster updates. To complete the picture, let us explain what triggers cluster updates. Looking at the formula F above we can see that when the features set of some topic changes (hence its relevance to other cluster topics decreases or increases), the formula needs to be re-evaluated to determine if the topic should be removed from existing clusters or be added to some other clusters (and if consequently some clusters should be merged). To determine which cluster and topic pairs should be considered, `RMFinder` uses a simple heuristics based on the observation that people that subscribe to a given topic are likely to subscribe also to related topics. Clusters thus search for new candidate topics (or clusters) by probing the subscribers of the cluster topics and checking to which other topics (clusters) they are subscribed (belong).

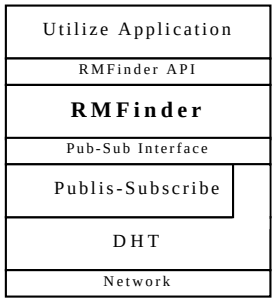

Figure 1: RMFinder architecture

4. DEMONSTRATION SCENARIO

`RMFinder` is developed in Java, designed to be deployed on any pub-sub network by simply implementing its pub-sub interface. We provide the implementation for the popular topic-based pub-sub platform Scribe [1]. Scribe itself is built over Pastry [4] – a DHT (Distributed Hash Table [5, 6]) that provides the overlay network services. `RMFinder` exposes to users the same API as Scribe for defining topics, publishing messages, and (un)subscribing. In addition, to support all `RMFinder` features, a few more functions have been added to its API. It uses the dynamic clustering algorithm described above to automatically group topics into clusters and to enrich messages with related content. Figure 1 illustrates the layered architecture that we adopted in our design, starting from the highest level - the application that utilizes `RMFinder` through its API, and ending with the Network itself.

To illustrate the power of `RMFinder` we will show how it can be used to enrich typical RSS feeds with related content. In our demonstration we will use real life RSS feeds whose messages we accumulated over the last six months from popular sites such as CNN, ESPN, and Yahoo. These include in particular fifteen different feeds from three main categories (news, basketball and entertainment). Throughout the demonstration we will follow the operation of the system from three angles: (1) we will show how the news publishers post messages, (2) we will see how the feed readers receive the (enriched) messages, and (3) we will follow what happens behind the scene using a surveillance panel. The panel will show the topics and their selected feature sets, the formed (and dynamically updated) clusters, and the message enrichment process.

The demonstration has two main parts. The first part aims to illustrate the daily system operation. We will show how standard RSS messages are posted and enriched before they are delivered to the readers. The importance of this enrichment will be illustrated by showing how critical related messages are automatically discovered and delivered to the users. The second part of the demonstration focuses on the dynamic aspects of `RMFinder`. It illustrates the adaptability of the clustering (and of the messages retrieval) to changes in the type of the messages, as well as to the addition (resp. removal) of feeds. For that we will simulate a scenario where the news reporters of some feeds switch to write for some other feeds (hence the style and content of the messages in these feeds will dramatically change). We will see how the selected features of the feeds (and consequently the clusters) adapt to the new setting, and how related messages are properly identified and sent to the users.

Finally, to conclude our demonstration we will show an application of `RMFinder` to RSS archives, where enriched messages are archived and a standard Web browser provides a simple, intuitive navigation between (transitively) related news items.

5. REFERENCES

[1] M. Castro and P. Druschel and A. Kermarrec and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in communications (JSAC) 2002

[2] V. Ramasubramanian and R. Peterson and Emin Gun Sirer. Corona: A High Performance Publish-Subscribe System for the World Wide Web. Proc. of Networked System Design and Implementation 2006

[3] D. Sandler and A. Mislove and A. Post and P. Druschel. FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification. Proc. Int. Workshop on Peer-to-Peer Systems (IPTPSŠ05) New York 2005

[4] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Proc. IFIP/ACM Middleware 2001

[5] I. Stoica and R. Morris and D. Karger and F. Kaashoek and H.Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for InternetApplications. Proc. ACM SIGCOMM 149–160 2001

[6] B. Y. Zhao and J. D. Kubiatowicz and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location andRouting. UC Berkeley 2001 UCB/CSD-01-1141 April

[7] E. Patrick Th. and G. Rachid and S. Joe. Type-Based Publish/Subscribe. 2000

[8] R. Zhang and Y. C. Hu. HYPER: A Hybrid Approach to Efficient Content-based Publish/Subscribe. ICDCS Proc. ICDCS 2005

[9] G. Salton and C.S. Yang. On the specification of term values in automatic indexing. J. Documents 1973

[10] T. Milo, T. Zur and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. Proc. SIGMOD 2007