Tel Aviv University

The Raymond and Beverly Sackler

Faculty of Exact Sciences

Enriching Topic-Based Publish-Subscribe Systems with Related Content

Thesis submitted in partial fulfillment

of graduate requirements for the degree

"Master of Sciences" in Tel Aviv University

School of Computer Science

By

Rubi Boim

Prepared under the supervision of Prof. Tova Milo

April 2008

Acknowledgements

First, I would like to express my deep and sincere gratitude to Prof. Tova Milo, my supervisor. Her guidance throughout this work, her insightful advices and her constant encouragements made the completion of this thesis possible as well as fun. Thank you Tova.

To all my lab-mates from Tel-Aviv university, current and former, my thanks for their help, encouragement and good atmosphere in the lab.

For their unconditional love and support, a deep special thanks to my family: My Mother and Gerry, my siblings Itay, Adi and Idan, my Father and Sarit, my Grandmother Hagit, and lastly my girlfriend Dana. Thank you all!

Abstract

We consider in this thesis a class of Publish-Subscribe (pub-sub, for short) systems called *topic-based* systems, where users subscribe to topics and are notified on events that belong to those subscribed topics. With the recent flourishing of RSS news syndication, these systems are regaining popularity and are raising new challenging problems.

Topic-based pub-sub systems are fairly simple and consequently have very efficient implementations. This simplicity, however, has a price: A user subscribes to a set of topics (feeds) that he considers interesting. But related messages, possibly of great interest, may be published in other topics that he is not subscribed to (and may even be completely unaware of their existence). These relevant messages will never reach him. A possible solution is to move to a more sophisticated class of pub-sub systems such as *content-based* systems [46]. 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 work is to devise a method for identifying relevant messages, while at the same time retaining the *simplicity and efficiency* of topic-based P2P pub-sub systems. 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. Our solution is based on a 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.

We have implemented our solution in the `RMFinder` (Related Messages Finder) pub-sub system. `RMFinder` utilizes a standard popular topic-based pub-sub platform (*Scribe* [8]) to manage topics and users' subscriptions.

Contents

1 Introduction

The amount of information available for 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 a pub-sub system, subscribers express their interest in certain events (e.g. the appearance of new relevant data item), and are subsequently notified of any event, generated by a publisher (information provider), that belongs to their registered interest. Pub-sub systems have raised considerable interest in the research community over the years. In this thesis we focus on a relatively simple class of such systems, called *topic-based* systems, where users subscribe to *topics* and are notified on events that belong to those subscribed topics. With the recent flourishing of RSS news syndication, these systems are regaining popularity and are raising new challenging problems.

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. 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 thesis is to introduce 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. Our main algorithm 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. We then use these clusters to implement an effective search for related messages, within the corresponding topics according to the clusters formation.

Results The contributions of this thesis are the following:

- We present a dynamic distributed clustering algorithm that continuously adapts the topic-clusters according to their dynamic content. The algorithm employs local cluster updates to change the overall system configuration, where each of them is performed only when it is estimated to be (globally) cost effective.

- We present two different methods for retrieving related messages from within the clusters formation, created by our algorithm. Both methods requires minimal resources, and thus suitable for any P2P network.

- We have implemented the above ideas in `RMFinder` - a system that identifies related messages within a topic-based pub-sub systems. `RMFinder` uses a standard popular topic-based pub-sub platform (Scribe [8]) to manage topics, topic-clusters, and users' subscriptions.

- The `RMFinder` system described above will be demonstrated at the SIGMOD'08 conference. An accompanying demo paper will appear in the conference proceedings [3].

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, `RMFinder` can be deployed upon every pub-sub system simply by implementing its pub-sub interface.

The grouping of topics into sets has been previously proposed in the literature in a different context: To provide users with varying subscrip-

tion granularity it was suggested to group topics into sets forming a subset hierarchy [39]. 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.

Thesis outline Section 2 introduce the enriched topic-based pub-sub systems and describes it main properties. Section 3 presents our dynamic distributed clustering algorithm and section 4 describes two methods for finding related messages from within the clusters formation. Section 5 presents the system implementation as well as a through experimental study. In section 6 we overview related work, and we conclude in section 7 with future work. Finally, we have two Appendixes: at the first (A) we presents the step necessary for establishing the `RMFinder` development environment and at the second (B) we presents a GUI (graphical user interface) tour for the programs which utilize `RMFinder`.

2 Enriched topic-based pub-sub

We start by providing some background pub-sub systems in general, and topic-based ones in particular. We then describe the challenges encountered when trying to enrich them to deliver related content.

Background Pub-sub is a distributed computing paradigm that consists of three principal components: subscribers, publishers, and an infrastructure for event delivery. Figure 1 illustrates such a general pub-sub system. Subscribers express their interest in an event or a pattern of

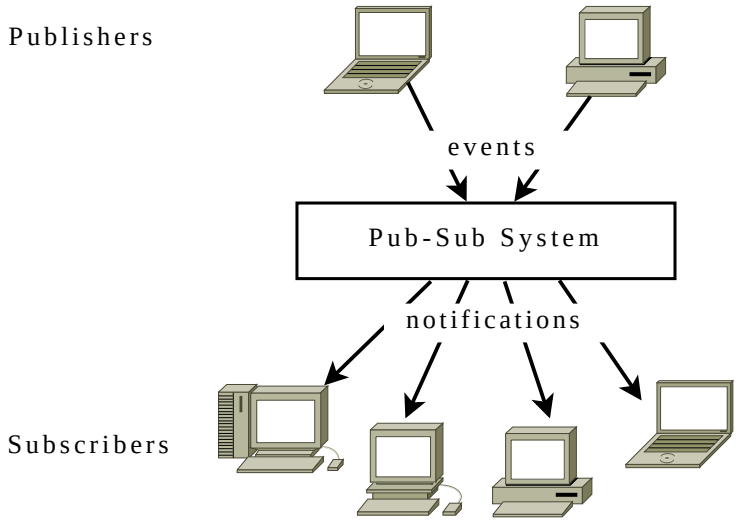

Figure 1: An overall illustration of the Pub-Sub system

events (say, the publication of new sports news). Publishers (e.g. CNN's sport department) generate events (e.g. post news). The infrastructure is responsible for matching events with the interests and sending proper notifications to the subscribers. Based on the way the subscribers specify their interest, pub-sub systems can be classified into two main categories: *topic-based* and *content-based*. In *topic-based* pub-sub systems,

subscribers specify their interest by subscribing to a *topic*, also known as feed, channel, subject, or group. Each event produced by the publisher is labeled with a topic and sent to all the subscribers of this topic. In other words, publishers and subscribers are connected together by a predefined topic. In *content-based* systems, on the other hand, subscribers specify their interest through event filters, which are functions of event contents. Published events are matched against the filters and sent to the subscribers if they match the specified filters.

Topic-based pub-sub is rather static and basic compare to a content-based one. Its simplicity however has the advantage of allowing for a very efficient implementation and a simple, intuitive, user interface. Content-based pub-sub typically requires more sophisticated protocols with higher runtime overhead, as well as more sophisticated user interaction. Because of this additional complexity, one generally prefers to use a topic-based pub-sub in contexts where events divide naturally into groups that correspond to users' interests.

To support scalability, most modern topic-based platforms [8, 48, 28, 32] are based on a Peer-to-Peer architecture. More specifically, they often run over Distributed Hash Table (DHT) systems [30, 34, 47, 24] and manage their operative layer using the DHT exported functions (API). Figure 2 depicts the structure of the three described layers. In that sense,

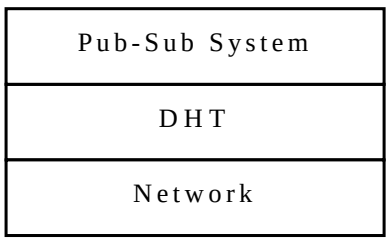

Figure 2: The topic-based pub-sub layers structure

the DHT acts as a mediator layer between the underneath network and the system core. For each topic the system allocates a dedicated distributed structure over the DHT which serves as medium for delivering the events. This structure consists of the (peers of the) topic subscribers as well as additional ”helper” peers assisting in the dissemination of the events. Since peers in the network may come and go frequently, the structure must be continuously maintained so that ”holes” do not disrupt the delivery of events.

Topic-based pub-sub The interfaces of typical P2P topic-based pub-sub systems share four common operations: CREATE, PUBLISH, SUBSCRIBE and UNSUBSCRIBE. They are implemented differently in different systems, but the usage scenario in the various systems is fairly similar: To send messages, the publishers first CREATE topics. Each topic is virtually represented by an individual peer (often called a channel, or simply ”the topic”), 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 impractical: 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 in 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, before a publisher passes a message to the topic‘s channel (to propagate it to the subscribers), the publisher first asks the topic's channel for all the clusters (and their member topics) which the topic belongs. The publisher then queries this set of topics (not necessarily the entire set, as it can be reduced by several methods) for relevant messages. The publisher then enriches the message with the most relevant messages (from all the queries), and only then passes it to the topic's channel to propagate it to all 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. These clusters should maximize the values of the following properties:

- **Coverage.** Most of the relevant messages from the entire network are retrieved.
- **Efficiency.** 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, depending on the season and/or the taste of the current reporter. A good solution thus must have a dynamic nature, continuously adapting the topics profiles 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 systems 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 profiles, should incur only very minimal overhead, not to harm the overall system performance.

3 Dynamic clustering

Clustering is the classification of objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait [41]. The objects we wish to cluster are all the topics within the topic-based pub-sub network. The trait we wish to share is content similarity - as two topics who share it are likely to publish related content.

The classic clustering problem refers to a static and centralized environment - one where one computer holds all the data, and it does not change. However, we are dealing with a dynamic and distributed environment, one which requires us to refer to the following properties: Adaptivity (the content of each topic is constantly changing), distribution (P2P network have a decentralized nature, where no central coordinator has full knowledge about the system) and low overhead (each peer should incur only minimal overhead).

To address these requirement, we use a novel dynamic distributed clustering algorithm that was developed recently in [38]. 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. This calculation is done by the formula F (to be described in section 3.1) which estimates the quality of a clusters formation. 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 [38] 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.

In the reminder of this section, we use the following notations: 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.

This section is structured as followed: In section 3.1 we introduce the formula F for estimating the quality of the clusters, in section 3.2 we describe how to assign the topics profiles, in section 3.3 we describe how to calculate the function for estimating the likelihood of two topics to publish related messages and in section 3.4 we describe our algorithm for creating and maintaining the dynamic clusters.

3.1 The formula F

The algorithm uses a formula F that estimates the quality of the current clusters. Only updates that are determined to be beneficial (i.e. increases the value of F) are performed. The formula F gives a grade (number) for a clusters set (C) at a given time. Calculating the formula at different times may yield different results, even if the set C (and each cluster $c \in C$ within it) does not change. This is due to the dynamic nature of the topics - they change focus on the subjects they cover constantly. Moreover, the "profile" which we build for each topic is designed to describe only the recent events of the topic, and not the previous ones (more on the "profile" on the section 3.2). Therefor, the basic element required by the formula is, given two topics t_1 and t_2, the estimation of the likelihood of t_1 and t_2 to publish related messages. We denote this function as $RM(t_1, t_2) \rightarrow [0:1]$ (we will explain how to calculate it in

section 3.3). This function may yield different results at different times, correspond to the current context of the topics.

Other properties the function should support are, as expected, the two required properties (from the clusters), mentioned in the previous sections: Coverage and Efficiency. We will describe each of them.

Coverage To maximize the coverage, we would like for every two topics, t_1 and t_2 ($t_1, t_2 \in T$) s.t. their $RM(t_1, t_2)$ value is relatively high, to both be members of some joint cluster $c \in C$. In other words, for every two topics defined above, the coverage will be "good" if $t_1 \in T_{t_2}$. Note that for every two topics, if $t_1 \in T_{t_2}$ then we can also conclude that $t_2 \in T_{t_1}$, because there exists some cluster c which both t_1 and t_2 are members with.

Efficiency As for every application used within a P2P network, efficiency is one of the most important aspects one should consider. We would like to enforce the following two properties:

- The first relates to the basic clusters formation. Assuming we would care only about the coverage, how the optimal set C would look? Recall we would like for every two topics, which their RM is relatively high, to be a member with some joint cluster. Therefor, if we create only one cluster c ($c \in C$) with all the topics (i.e. $c = T$), it would be the optimal - for every two topics t_1 and t_2, $t_1 \in T_{t_2}$. However, remember the reason we clustered related topics together: For every topic t' publishing a message, we query all the topics within $T_{t'}$. If one cluster c would contain all the topics ($c = T$), we would need to query all the topics in the network.

Clearly, it is impractical. Therefor, if two topics are unsimilar, they should not be members of a joint cluster.

- The second relates to overlapping clusters: A topic can be member in more than one cluster, and therefor overlapping clusters can be created. Unlike other classical clustering problems, the number of clusters is not given. Therefor, consider the following case: The cluster c_1 contains the topics t_1, t_2 and t_3, while the cluster c_2 contains only the topics t_1 and t_2 (i.e. $c_2 \subseteq c_1$). In this scenario, T_{t_1}, T_{t_2} and T_{t_3} would not change at all if we remove the cluster c_2. Recall that each cluster in the network requires maintenance (network traffic, cpu time), and therefor, we would like the formula to reduce the value for such scenarios. However, the clusters can be semi-overlapped (i.e t' can be member at both c_1 and c_2). This scenario can occur when a topic is focusing on several aspects, while other topics only focus on one. For example, consider a topic focusing on news in general, while other topics focus on only domestic news or foreign news. We do not want to create on big cluster for all the topics, as the chances that a related message from a domestic topic would be related to one from the foreign topics is minor. However, the topic focusing on news in general might have related message to both types. Therefor, we would like the formula to encourage the topic to be member at both clusters, the domestic one and the foreign one.

The formula F The formula depends on three variables: The set of topics (T), the set of clusters (C) and the current time calculated. Recall that for any two given topics t_1 and t_2, the $RM(t_1, t_2)$ function may return

different values when computed at different times. Therefor, the formula captures the situation at a given point in time (the current). Its structure is given below:

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

The first element in the formula is the outer sigma. As we can observe, it sums some calculation for each topic $t \in T$. This calculation is the value of the current topic t with regard to the current sets T and C, and it composed from two summands. 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 (i.e. T_t). 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.

Note that the formula, in its current formation, is "expensive" to calculate - each of the denominators requires going over all the topics in the domain (for each topic); clearly an expensive task in our distributed P2P setting. Therefor, we will use a "slimmer" version of the formula, which will be described later on section 3.4.

To understand the behavior of the formula more thoroughly, we demonstrate several scenarios showing how exactly the formula causes the clus-

ters the form correctly. Throughout these scenarios we use the following notation: t_{news1}, t_{news2} and t_{news3} will denote three topics focusing on news. Similarly, $t_{sports1}$, $t_{sports2}$ and $t_{sports3}$ are focusing on sports. To simplify the calculations, for every two topics with the same focus (e.g. t_{news1} and t_{news2}), their RM value is 1, while for every two with different focus (e.g. t_{news1} and $t_{sports1}$, their RM value is 0. In each of the following scenarios, we describe two different states of which the clusters could form. For each of them, we will calculate the corresponding formula value, and verify that the the higher value is indeed the preferred one.

Scenario 1 - Simple Behavior In this scenario, we wish to observe the behavior of the formula in a relatively simple situation. Figure 3 depicts the two formations of clusters. The first one formed one big cluster containing all the topics. The second one formed two clusters, one for each focus. Obviously, the preferred formation is the second one, as the efficiency of the state is better (in the first formation, we would query unrelevant topics as their focus does not match)

To verify that the formula behaves correctly, we calculated F's value for each formation. Recall that the formula sums the value of each topic in the current state. Note that in each formation, because of symmetry, the value of every topic is equal to any other. This is because we assumed the values of the RM function are 0 and 1. Therefor, it is sufficient to calculate the value of a single topic in each formation. Following are the calculation of the formula for each formation - $F(1)$ will denote the formula value for the first, while $F(2)$ denotes for the second.

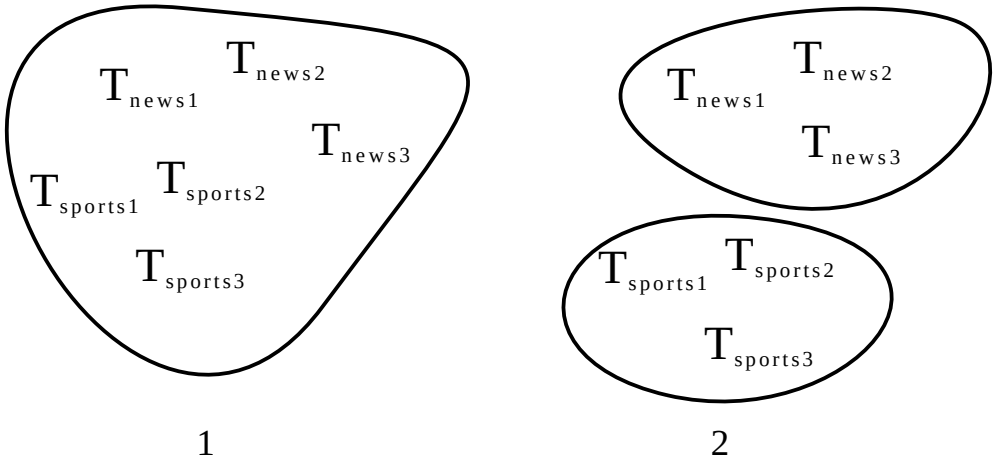

Figure 3: Simple clusters formation

$$F(1) = \sum_{t \in T} [\frac{1+1+0+0+0}{1+1+0+0+0} - w \frac{(1-1)+(1-1)+(1-0)+(1-0)+(1-0)}{(1-1)+(1-1)+(1-0)+(1-0)+(1-0)}]$$
$$= 6[\frac{2}{2} - w \frac{3}{3}] = 6[1-w]$$

$$F(2) = \sum_{t \in T} [\frac{1+1}{1+1+0+0+0} - w \frac{(1-1)+(1-1)}{(1-1)+(1-1)+(1-0)+(1-0)+(1-0)}]$$
$$= 6[\frac{2}{2} - w \frac{0}{3}] = 6$$

Recall that $w > 0$ and therefor $F(2) > F(1)$ as required. The first summand of the formula is similar for both formations, and has the same value. This is reasonable as both formations indeed managed to retrieve all the related messages from all the topics (for every topic, all its related topics can be reached). However, the difference between the values is due to the second summand - the tightness. Note that in the first forma-

tion, there is plenty of waste as topics of different focus are in the same cluster, and therefor we loose value for it. In the second formation, the tightness of the cluster is much better (even optimal), and therefor its value is higher.

Scenario 2 - Overlapped Clusters In this scenario, we wish to observe the behavior of the formula in a fully overlapped situation (i.e. one cluster is fully contained in another cluster). Note that in this scenario we use only three topics with the same focus (news), and the value of the RM function for each pair of topics is 0.9. Figure 4 depicts the two formations. The first formation consists of two clusters: The first cluster contains all three topics, while the second cluster contains only two. The second formation consist of a single cluster, containing all three topics. The preferred formation is the second one, again due to efficiency. However, while in scenario 1 the efficiency lacked because of unrelated topics clustered together, in this scenario the efficiency is lacked due to fully overlapped clusters, which does not give us any additional insight.

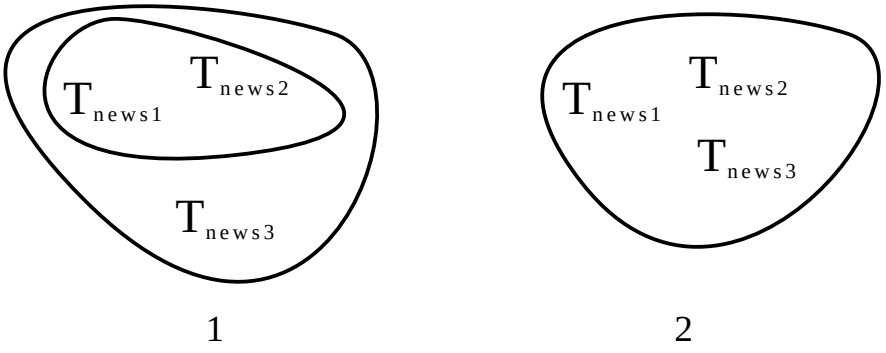

Figure 4: Overlapped clusters

To verify that the formula behaves correctly, we calculated F's value

for each formation again. However, for the first formation, the value for each topic is not equal for all - two topics (t_{news1} and t_{news2}) do have the same value (by symmetry), but t_{news3} has a different one. Therefor, we would calculate the value for t_{news1} and multiple it by a factor of 2. The value of each topic in the second formation however is (again) equal due to symmetry, and thus we need to calculate it for a single topic. Like previous calculations, $F(1)$ will denote the formula value for the first formation, while $F(2)$ denotes for the second.

$$\begin{aligned} F(1) &= 2[Value(t_{news1})] + Value(t_{news3}) \\ &= 2[\frac{0.9+0.9}{0.9+0.9} - w\ \frac{(1-0.9)+(1-0.9)+(1-0.9)}{(1-0.9)+(1-0.9)}] \\ &\quad + \frac{0.9+0.9}{0.9+0.9} - w\ \frac{(1-0.9)+(1-0.9)}{(1-0.9)+(1-0.9)} \\ &= 2[\frac{1.8}{1.8} - w\ \frac{0.3}{0.2}] + \frac{1.8}{1.8} - w\ \frac{0.2}{0.2} = 2 - 3w + 1 - w = 3 - 4w \end{aligned}$$

$$\begin{aligned} F(2) &= 3[Value(t_{news1})] \\ &= 3[\frac{0.9+0.9}{0.9+0.9} - w\ \frac{(1-0.9)+(1-0.9)}{(1-0.9)+(1-0.9)}] = 3[\frac{1.8}{1.8} - w\ \frac{0.2}{0.2}] = 3 - 3w \end{aligned}$$

Recall that $w > 0$ and therefor $F(2) > F(1)$ as required. Like the first scenario, the first summand of the formula is similar for both forms, and has the same value, while the second summand is different. If we look closely, we see that in the first formation, for t_{news1} (and thus for t_{news2}), we added the value of $(1 - 0.9)$ three times - one for t_{news3} but twice for t_{news2} (or t_{news1} respectively for t_{news2}). This is due to the overlapping of the two clusters, and thus we subtract the value twice, as

some kind of penalty. Note that if the RM function would return a value of 1 (perfect), the penalty would be 0, and therefor useless. However, as we would see on the section 3.2, the probability to return 1 for two different topics is negligible.

3.2 Topic profile

To measure the quality of the clustering, we need to estimate the likelihood for t_1 and t_2 to publish related messages (The RM function). For that, we would like to build for each topic a "profile", which will describe the current content of the topic. This profile should have to following properties:

- **Descriptive.** The profile should represents the current focus of a topic (i.e. sports, news..). This information will be later used for comparison with other topics profiles.
- **Adaptive.** Each topic has a dynamic nature - the type of messages may change over time. For instance, a sports channel may focus at different times on baseball, basketball or soccer, depending on the season and/or the taste of the current reporter. The profile should continuously adapt to the topic current focus.
- **Low overhead.** Our network is a distributed P2P, and as such decentralized system, one of the main requirements is low overhead (of the peers). The profile would be used to implement the RM function, which will be constantly used by the dynamic clustering algorithm. Therefor, sending the profile between different peers will be a common operation. We would not want to send a large amount of data constantly, and therefor, in order to keep the prop-

erties of the decentralized network, the profile's size should be relatively small.

To learn the focus of a topic, one would have to look at its published messages. In our context (i.e. topics-based pub-sub systems), a message is simply two short sets of features (words) - the title and the body. We will take these features, put them into a set (the topic's describing features set), and it will be the topic's profile. However, simply adding all the messages (and thus all the features) into the set will not satisfy all the properties mention above. Moreover, the size of this set will be extremely large. Therefor, in order to achieve all the properties above, and to reduce the set to a small constant size one, we first apply a set of rules. These rules are described on the following paragraphs:

Feature extraction If we could assign a weight for each feature, which will describe its current importance with regard to any other one, we could then choose only the ones with the highest weight to be members of our describing features set. But how to assign these weights? The most popular way is with no doubt the TF-IDF (term frequency - inverse document frequency), which was first introduced at [31]. It is the statistical measure used to evaluate how important a word is to a document in a collection. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the collection [43]. The TF-IDF is the product of two factors:

- **TF (term frequency).** The term frequency is a measure of the importance of a term (feature) s_i within a particular document d_j. It is simply the number of times a given term appears in that document.

This count is usually normalized to prevent a bias towards longer documents. We denote it as $tf_{i,j}$ and it is calculated as followed:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_{s_k \in d_j} n_{k,j}}$$

Where $n_{i,j}$ is the number of occurrences of the term s_i within the document d_j and the denominator is the number of occurrences of all terms in d_j.

- **IDF (inverse document frequency).** The inverse document frequency is a measure of the general importance of the term. If a term is very common, we would like to reduce its weight, and not set it too high. The reason is simply because that common term will not help us to distinguish between the documents (it is shared by all of them, and thus will have the same value). Note that unlike the TF value which is the value of a term within a specific document, the IDF value is for the entire collection for a given term.

  To calculate the IDF of a specific term s_i, we divide the number of all documents by the number of documents containing the term, and then we take the logarithm of that quotient. We denote it as idf_i and it is calculated as followed:

$$idf_i = \log \frac{|D|}{|\{d_j | s_i \in d_j\}|}$$

  Where D is the entire collection of documents and $d_j \in D$ is a document within the collection.

The final value of a term s_i within a specific document d_j is given by:

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

Note that in our context, each document is a published message, and a term is a feature within that message. Also recall the we do not wish to weight a feature within a specific message, but within the entire topic (collection). To do so, we create a main document which contains all the documents within the collection (just imagine we concatenate all the documents, one by one, to form one big document). For each term, we take the TF from within that document. To calculate the IDF however, we use the normal way defined above (recall that the IDF is given for a specific term within the entire collection).

Sliding window Two topics are similar/related by our definition if they are likely to publish related messages. Therefor, we are not interested in the entire history of messages published by the topic, but only with the ones recently published. More specifically, for each topic we consider a sliding window that includes its recent messages of a given time interval. As time passes, the window slides, leaving older messages behind, while adding the new ones. Deciding the length of the window is not trivial. Intuitively, a small window will yield bad results because it will be hard to identify the important terms, as the TF-IDF does not work well on very small set of documents. On the other hand, a big window will yields bad results also because the dictionary might not hold the correct features (for example, it might hold older features which describe interests which the topic used to discuss but not anymore). The algorithm should set the appropriate value, according to the current data.

Note that this is done not just to improve the similarity, but also to reduce the bandwidth as the use of the sliding window will reduce the number of messages the topics should keep.

Stemming Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form - generally a written word form [42]. It was first introduced at [23]. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. For example, a stemmer should identify the strings ”fishing”, ”fished” and ”fisher” as based on the root ”fish”. This algorithm is done in order to improve both the similarity and to reduce the bandwidth.

Stopwords Even after using the previous described methods, the number of features is relatively high. If we look at the extracted features, we can find a lot of ”connection” words which can be discard, for example: ”a”, ”be”, ”and” etc. These words are called ”stopwords”. Note that there is no definite list of stopwords which all natural language processing tools incorporate. However, removing all the stopwords can cause problems as doing so might result with context loose. For example, if we look at the phrase ”Vitamin A”, the letter ”A” will be recognized as a stopwords and therefor will be ignored, although it does not serve as a stopword in the phrase. Another example is from the famous tragedy ”Hamlet” (by William Shakespeare) - ”To be or not to be”. The entire phrase is constructed from stopwords, and therefor will be totaly ignored. However, despite these problems, filtering stopwords had been proven to be a very useful tool, and is in use in all major search engine today (with a

special care for common problems such as the ”hamlet”). Like before, filtering out stopwords is done in order to improve both the similarity and to reduce the bandwidth.

Title vs body While the body is not always available (some topics provide only the title as the nature of the messages is short updates), the title is always provided, and from is definition, the features within it provide a short description of the message, and therefor, they are ”more important” than the ones from the body. Recall that all the rules describes in the paragraphs above are done (one of the reasons) in order to reduce the size of the features set. They assumed they are given one field of text (message), which is not divided. However, we can apply all these rules separately to both the title and the body, and not once together. This way, we can first choose all the features from the title (as we assumed they are with greater importance), and in the remaining of the space (the features set's size is constant) we add the most important ones from the body.

Each of the above rules (feature extraction, stemming..) is relatively simple. Nevertheless, our dynamic algorithm (described in section 3.4) yields impressive results. But clearly, if desired, more sophisticated versions of the rules can easily be plugged in.

3.3 The RM function

Given two topics, t_1 and t_2, we would like to estimate the likelihood of t_1 and t_2 to publish related messages. We denote this function as $RM(t_1, t_2) \rightarrow [0:1]$, where a value near 1 represents a very high probability of publishing related content, while a value near 0 represents the opposite.

To implement this function, we use the topics profiles. As mentioned before, each topic has a dynamic profile which consists of a set of describing features. If two topics share a significant amount of these features, we can conclude that there is a good chance the topics are focusing on the same (or similar) subjects, and therefor there is a good chance they will publish related messages. We define the RM function for two topics t_1 and t_2, their corresponding features sets S_1 and S_2, and the constant feature set size k as followed:

$$RM(t_1, t_2) = \frac{|S_1 \cap S_2|}{k}$$

Calculating the RM function for the same two topics t_1 and t_2, but at different times, may yield different results - according to the constantly changing topics profiles. Note that returning the value of 1 (maximum value) by RM would required from any two topics to have the exact same profile (i.e. the exact same features set). Obviously, the probability for this event to occur is extremely low.

3.4 The algorithm

The P2P nature of the pub-sub system, where no central coordinator has a full knowledge about the system's state, calls for a corresponding distributed clustering algorithm. We use a novel dynamic distributed clustering algorithm that was developed recently in [38]. It was originally introduced 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.

The clustering algorithm is based on a set of local cluster update op-

erations, performed by individual channels (of clusters) consulting only a relatively small neighborhood. These operations includes: 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 a topic from a cluster and the destruction of clusters.

The algorithm uses the formula F described at section 3.1 to estimate the quality of the current formation of the clusters. Only updates that are determined to be beneficial (i.e. increase the value of F) are performed. In other words, prior of making a change in the formation (for example the merge of two clusters), the formula's value, denoted by F_{old}, is calculated and saved. Then, the algorithm calculates the value of the formula if indeed the operation would take place, denoted by F_{new}, and only if $F_{new} > F_{old}$ the operation is indeed executed.

Tweaking the formula F It should be stressed that it is impractical to try and evaluate the exact value of the formula F for each update - this required gathering information about the full network state (all the topics and all the clusters); clearly an expensive task in our distributed P2P setting. Even if the formation of the clusters does not change, the value of the formula can, as it relies on the RM function, which as described before, may return different values for the same two topics when calculated at different times. Moreover, since all the cluster update operations are local in nature, to evaluate the potential benefit of an update, there is no reason to evaluate the full formula but only the changes entailed to the small neighborhood involving the updated clusters. Therefor, we would like to change the formula to be sensitive to local updates. For

example, when merging two clusters, the effect on the formula would be only from the topics within these clusters, and therefor, to calculate the change we could look only on a small neighborhood of the entire network. The updated formula is defined as:

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

Where $T_{neighborhood}$ is the set of all topics involved in the operation. For example, for a merge between two clusters c_1 and c_2, the set will be $T_{neighborhood} = c_1 \bigcup c_2$.

Note that we removed the denominators from the two elements. These elements were supposed to calculate the relevant and irrelevant values within all the topics in the network. Thus, the value calculated now for each element is not the appropriate ratio (relevant/irrelevant) but the actual value of the current formation. The omitted ration however can be captured by adjusting the weight w.

Distributed Algorithm Recall the nature of a P2P network - distribution; No central coordinator has a full knowledge about the system's state. As a result, our algorithm should run at multiple peers, where each one behaves individually, without the guidance of some central peer; More specifically, each cluster will run an instance of the distributed algorithm.

At the initial phase, for each topic, a corresponding cluster is created, which contains only that topic. The algorithm is based on only two operation: The merge of two clusters and the removal of topics from a cluster. Note that all other operations described before are implemented by these two operation: To group two individual topics to form a new

cluster we simply merge the two corresponding clusters of these topics; to add a topic to an existing cluster we simply merge this existing cluster with the topic's corresponding cluster.

For both operations, the idea is the same: calculate the value of the formula F prior and post the change and prefer the higher valued one. If a cluster containing only one topic is merged with some other cluster, a new corresponding cluster containing only that topic is created. This is done to allow more clusters to add that topic, and not the whole new cluster it is now a member with. Note that the algorithm also supports exclusive clustering, where each topic is a member in only one cluster. As one can guess, if this option is chosen, the corresponding clusters are created only once.

To complete the picture, two main issues needs to be explained: The first is how each cluster chooses the candidates clusters for merge and the second is what triggers these operation (in other words, when do we run the algorithm at each peer). We describe these points on the following paragraphes.

Locating candidates Choosing which clusters will be the candidates is one of the most important components of the algorithm. We face here the following problems:

- The number of topics is enormous, hence it is unrealistic to traverse all of them (clusters are reachable by their member topics). We would like to select ones with a higher probability for a successive merge. Note that we do not have any information regarding the cluster (tags, classification), and we can not use their member topics profiles as we simply do not have this information (if we would

contact the cluster/topic for this information it will be the same as checking for merge).

- The topics (and thus the clusters) have a dynamic nature, and thus if two clusters were checked for merged and found not to be beneficial, it does not mean that in the future it will be the same case. Therefor, we might need to check the same clusters several times (over a period of time), which increases even more the number of candidates.

- Most pub-sub topic-based systems do not implement an API for traversing all the topics; the user must know the exact name of the topic he wishes to subscribe. The cluster therefor must implement a dedicated mechanism to get a random topic (and thus a cluster).

To address all these problems we use the following observation: Users that subscribe to a given topic are likely to subscribe also to related topics as well. Therefor, to retrieve a list of candidates clusters, where each one has a higher probability to be merged, we ask the subscribes of the cluster (that is, the subscribes of the member topics of the cluster) for all the topics they are also subscribed. For each such topic we ask for all his member clusters, and we repeat this way for several users (we are doing some kind of a poll). For each cluster we count the number of times it was ”discovered”, and finally we select the ones with the highest count value.

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). Therefor, the cluster should keep a short history of each of its member topics profiles, so it could compare them to determine if indeed there were a change in one of them. Similarly, after a new topic is created and some of its messages are published, the benefit of adding the topic to existing clusters needs to be estimated.

4 Finding related messages

In the previous section we considered the clustering of topics according to their content. We next consider the retrieval of related messages based on these clusters. More specifically, for a given published message and its corresponding topic, we would compare all the recent published messages of all the topics who share a cluster with the corresponding topic. For each such pair (the published message and a recent message from another topic) we assign a value, called their *related value*, that represents how related they are, and return only messages with highest such related values.

The problem of deciding if two messages are related (namely assigning a related value) has been studied in various domains, including in particular Information Retrieval. In our context however it has a particular flavor as we would like to exploit the knowledge obtained in the clustering phase for efficient solution with high accuracy. In this section we will describe two different methods for comparing (and computing the related value of) two given messages: On section 4.1 we introduce the D-VSM method and on section 4.2 we introduce the SA (simple appearances) method.

4.1 D-VSM

The first comparison method we use is a variation we made for the vector space model (VSM), which was introduced at [15]. Before we detail this variation, we give a short description of the original VSM.

VSM Vector space model (or term vector model) is an algebraic model for representing text documents (and any objects, in general) as vectors of terms, where each dimension corresponds to a separate term, ordered lexicographic by their name. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed. One of the best known schemes is TF-IDF weighting, which was described at section 3.2. Similarity between documents can be calculated by comparing the deviation of angles between each the documents vectors. In practice, it is easier to calculate the cosine of the angle between the vectors instead of the angle. If we denote these vectors as v_1 and v_2, it can be calculated by:

$$\cos\Theta = \frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|}$$

The VSM is widely used in Information Retrieval systems, and therefor the following properties are not considered as drawbacks:

- The system has a static nature - all documents must be retrieved prior to the calculation.
- A large features set must be computed, as each term (feature) which appeared must be saved in order to calculated the appropriate TF-IDF value for it.
- The size of each representative vector (of a document) is extremely large, as the dimension size is the number of unique terms within all the documents.

As one can imagine, these properties are not acceptable for our dis-

tributed P2P network. Therefor, we introduce the D-VSM (Dynamic VSM) which eliminates these drawbacks properties. We first describe it for a single topic (section 4.1.1), next we describe its difficulties (section 4.1.2) and finally we describe its generalize version for clusters (section 4.1.3).

4.1.1 Topic-based approach

The dynamic VSM is a variation of both the VSM and the topics profile described at section 3.2. To simplify, we start with the static case - we assume we have a topic t and a set of corresponding published messages D, ordered by their published date. We start by applying all the rules used for calculating the topics profiles: Filtering out stopwords, applying stemming, choosing only the recent messages published (sliding window), selecting only the most important feature (feature extraction) and prefering the features from the title (title vs body). Note that this process can be done once, and be used by both the topic profile and the dynamic VSM, and therefor we use the same notation to describe this set - S. We then apply the VSM with S as the set of all features, where for every document, we ignore features which are not part of this features set.

Using a smaller set of features reduces the dimension of the VSM. If we look at the representative vector of a message at this dimension, it will be partially represented (comparing to the traditional VSM) as only the features within the reduced features set appear in it - all other (which appear in the message but not in the reduced set) are discarded. At first, this might look as a major drawback, as we loose significant data from each message. However, we did not selected the reduced set randomly,

but we chose the most significant ones - therefor, the discarded features have a lower value, and therefor it is acceptable to ignore them.

With the reduced set we "solve" two of the drawback properties mention above: the size of the features set and the size of the representative vectors. Although we "lost" some information for every message, we gain the ability to use the VSM in a distributed environment, as the memory required now for implementing is acceptable for such P2P systems. For convenience, for the rest of the section we will refer the reduced features set as simply the feature set.

We continue our description of the D-VSM and introduced the dynamic case: We assume we have a continuance stream of documents (messages) published by the topic. To simplify the notation with the time variable, we assume a document $d_i \in D$ was published at time i. For example, if $i < j$ then the document d_i was published prior to d_j (at time i and j respectively).

Recall the sliding window mechanism used to create the topic profile, which "slides" as more messages are published leaving older messages behind. This method produces an updated topic profile in a continuance stream of messages. We would like to use the same method to enhance the dynamic VSM, to support streams of messages. Therefor, for every message (document) published d_i, we would use its corresponding feature set S_i, calculated according to the sliding window at that specific time. The message representative vector will be constructed according to the dimension of the current feature set.

4.1.2 Difficulties

Using the D-VSM (the dynamic version) as described above, raises two questions:

- When we wish to compare two vectors v_1 and v_2, constructed at different times (time 1 and 2 respectively), we can not simply apply the cosine method to calculate the distance between them - the i' element in each vector represents the i' feature at each corresponding feature set, which (at high probability) is not a matching feature. How then we compare messages with different dimensions?
- Calculating the weights at different times yields different results. This is due to the TF-IDF method, or more specifically the IDF value. There are two options: The first is to calculate the weight at the time published, while the second is at the time compared. Which one to choose? What are the differences?

Dimension fix We start with the first issue. To compare two vectors of different dimensions, we first need to bring them to a joined one, and then compare them using the desire method (cosine for example). This joined dimension can be the union, intersection or any other combination. We chose the intersection of the two dimensions as the joined one, as it requires no additional weighting of the features (if for example we would choose the union, then for each feature not shared, its weight at one vector will be 0, and thus will be useless unless we weight it). A simple way to do so, it to set the joined dimension to be the one of either of them, while transforming the dimension of the second one to the correspond to the first. The way to transform a dimension is simply to

rearrange the weights (the dimension is set by the order of the member features) within it to correspond to the new dimension. For missing features (if all the features were the same, the dimensions were identical), we simply give the value of 0 as the weight. To illustrate, let us consider the following example:

$$S_1 = \{a, b, c\} \quad v_1 = \{0.1, 0.2, 0.3\} \quad S_2 = \{b, c, d\} \quad v_2 = \{0.5, 0.6, 0.7\}$$

To compare v_1 and v_2, we need to transform v_2 to a dimension corresponding to S_1 - we need to order the features of S_2 to match the ones of S_1. If we denote the transformed dimension as $S_{2\to1}$, then we get:

$$S_{2\to1} = \{0, b, c\} \quad v2 = \{0, 0.5, 0.6\}$$

Note that the transform can not be done simply by both vectors v_1 and v_2, but must also be provided with the features sets S_1 and S_2.

TF-IDF at different times We next consider the second issue discussed above - when to calculate the TF-IDF. To understand the difference options, we introduce the following scenario: Lets assume we want to compare two documents, d_i and d_j at some time $query$ (can be i, j or any other time). We are interested in calculating the describing vectors v_i and v_j using the TF-IDF scheme. To calculate the TF we have only one option, as it depends solely on the documents itself, meaning that the time variable has no affect. Obviously, the best way will be to calculate it ones (probably when the document is published), and store the result. The IDF however is more complicated, as it depends on the entire collection D. As a results, calculating the IDF at different times may

yield different outcomes. Continuing our scenario from before, giving the two documents d_i and d_j (published at time i and j respectively) and a query at time $query$, when should we calculate the IDF? We have two options:

- **On query** - calculate the IDF whenever we run the query (I.E. at time $query$).
- **On published** - calculate the IDF when the documents is published and store its value for future queries (I.E. for document d_i calculate the IDF at time i).

We would like to choose the option which is superior with respect to the following two parameters: Running time (minimize the CPU/bandwidth) and comparison quality. On the following paragraphs we examine each of them, and decide which option suites us the most.

Running time Recall the definition of the IDF of a feature $s_i \in d'$:

$$idf_i = \log \frac{|D|}{|\{d_j | s_i \in d_j\}|}$$

To calculate the IDF one needs to maintain a set of features, where for each one, the number of documents it appears in is saved. With this data structure, we can calculate the IDF in $\log(|S|)$ steps (in a binary tree), and because the size of the feature set S is fixed, it can be calculated in a constant time. However, the number of calculation is not small: in a normal scenario, we will compare a newly published message against the entire domain, and for each message within it we need to calculate the IDF for each feature; clearly a large number of calculation, and recalling the nature of

a P2P network, where each peer wishes to minimize his work, we would prefer the "on published" method which calculates the value only once.

Query quality In general one may distinguish between two notions of similarly, often refer to as syntactic and semantics. Semantic similarity accounts for the semantics of the words, hence the same word appearing in different context would not be considered the same. The typical example is Jaguar (for a car) and Jaguar (for the animal). While the notion of similarity that we use here is syntactic, it turns out that by choosing the weights time properly, some semantics is naturally introduced. We explain this next.

To achieve these requirements we use the time variable with the IDF. Recall our scenario from before - we are given two documents, d_i and d_j (published at time i and j respectively), and a query (at time $query$). If we choose to use the "On query" method, then for every term $s_k \in d_i \bigcap d_j$, its IDF value will be the same for both document as we calculate it at the same time ($query$). While at first look this might look good, we are actually loosing a significant amount of information. To understand more exactly what we loose, we first need to understand when a term s_k would get a different IDF value: We continue our example from before, and add the following notations: w_k^i and w_k^j will denote the weight that the term s_k received in document d_i and d_j respectively (we are using now the "on published" method, therefor their value differ). Also note that w_k^i was given at time i and w_k^j at time j. Lets assume that $w_k^i < w_k^j$. We can conclude from the inequality that at time i, the

term s_k was more common than in time j. In other words, at time t_i there were a relatively big group of document containing w_k. Because we know that all these documents were published at around the same time (i), we can conclude that they only relate to the term s_k and do not focus on it, as it is very unlikely that a large group of documents will focus on the same term and be published at around the same time. Therefor, the term w_k should not be weight with a high value (nor very low) at time i. However, at time j, there is a better chance that the focus was indeed on w_k as fewer document published it, and therefor, we would like to give it a higher value.

The example described above illustrate us how we can use the time variable with the "on published" method to achieve, although in a small way, semantic data, while using the same low resources as used in a simple syntactic comparison. Therefor we prefer to use the "on published" method instead of the "on query".

As both properties (running time and query quality) prefer the "on published" method, it is no surprise our preferred method is the same.

4.1.3 Cluster-based approach

In the previous section we introduced the D-VSM for a single topic (domain) - we had a single topic which provided us a stream of data (messages) and each two messages we compared were within that stream. We would like to generalize this case to the global one, where multiple sources of stream are involved (cluster). For example, consider the case where several topics are clustered together and we wish to compare messages between all of them. We denote that cluster as c ($|c| > 1$), and we

will use this notation during the entire subsection.

Straight forward We start by describing the straight forward way to generalize the D-VSM. For each topic $t \in c$ we implement a local version of the D-VSM, as if he was the only topic in the network. When we wish to compare two messages, v_i and v_j, from different topics, t_i and t_j respectively, we are facing again the problem of different dimensions (the dimension of t_i was defined according to S_i - the features set of t_i, and the one of t_j according to S_j). To solve it, we use the same method introduced at section 4.1 (”Dimension fix”). Note that although it is the same problem, the causes are different: In the current case, it is due to the difference between S_i and S_j (the features set of different topics). In the previous case, it is due to the time variable; if we set t, than S_1 and S_2 are features set of the same topic t, but at different times (time 1 and 2 respectively).

The straight forward approach, although easy to implement, have one main problem: The quality of the queries. Recall that the weighting of the representative vector of the published message will be done by the features set of the published topic, and only it. More specifically, the IDF value will be calculated according to all the messages published only by the publisher topic. Because we ”extend” our domain to all the member topics of c, it will be more effective to calculate the IDF value with respect to all the topics in the ”extended” domain (i.e. all the member topics with the cluster).

Cluster features set To solve this problem, we use the following approach: For each cluster c, we create its corresponding features set S_c.

This set will be constructed from all the messages published by all the member topics of the cluster. It can be also calculated by the actual topic profile (features set) of each topic. The dimension of the D-VSM of each of the member topics will be defined by the cluster's features set (instead of the local topic's features set). Note that as a result, in order to weight a representative vector of a message, the topic should contact the cluster. With the cluster's features set we use the same D-VSM for all the member topics, and we process the data from all the topics together, and thus the IDF will be more accurate.

Topics can be member in more than one cluster. When a topic is publishing a message, or more specifically when a topic wants to represent a message by a vector, it can have more than one representation - one for each cluster it is a member off. Therefor, a feature within a message might have two different weight values (from each cluster). This pose a problem when the topic joins a new cluster - new topics will query the topic (for its recent messages), and they will have two weights to choose from. To solve it, instead of saving multiple weights for each feature, we save only the maximum weight from all the member cluster. This solution also reduce the memory required by the representative topic, as only one weight value is saved for each feature.

4.2 SA (Simple appearances)

The D-VSM method described above produce semantic knowledge while using low resources corresponding to syntactic knowledge. It does however have two drawbacks:

- Many features are discarded, as we use a small size features set to

reduce our dimension. Although we keep the most important ones, some discarded features may still provide us more insight.

- Using a cluster features set requires additional maintenance by the P2P networks (topics needs to be constantly synchronized with their corresponding clusters). Although this occur anyway by the clustering algorithm, it does add additional work.

We next introduce an alternative method, called simple appearances (SA) that avoids these problems. This method is base on the following intuition: The more common features two messages share, the more similar they are. If we compare it to the TF-IDF method (and thus to the D-VSM), then SA is simply the TF (term frequency) value, neglecting the domain frequency. The number of common features is also divided by the total message size to prevent bias towards longer messages.

At first this method might seem too simple, but we believe that the nature of our environment, where messages are shorter and thus more descriptive than ”standard” documents, is suitable for such a straight forward approach. Moreover, we believe this method should work because of the way we cluster the topics together - by their current dynamic content - which in a sense is a part of the actual comparison of the messages (it is only done prior for the actual comparison).

Note this method is extremely easy to implement, even compared to the previous method (D-VSM): It does not requires maintenance of any data structure (compared to the cluster features set before), and therefor is more suitable to our P2P nature. It also has the advantage that it utilize all the features, as none are discarded. The tradeoff however is the lack of weights, which reduces the quality of the comparisons. In section

5.2.2 we ran several experiments to evaluate this tradeoff, and to decide which method is indeed preferred.

5 Implementation & experiments

In order to validate our approach we implemented the algorithms and ran extensive experiments. Our implementation demonstrate the richer service provided by `RMFinder`, relative to traditional topic-based pub-sub, where users can automatically receive all messages related to those in the topics to which they are subscribed. Following, section 5.1 summarize the main features of our implementation and section 5.2 highlights our main experimental results.

5.1 Implementation

We have implemented the algorithms described in the previous sections in `RMFinder` - a system that identifies related messages within a topic-based pub-sub system. `RMFinder` was developed in Java and it can be deployed on any topic-based pub-sub environment. Figure 5 illustrated the layered architecture that we adopted in our design: At the highest level we can find the application the utilizes `RMFinder` through its API. Following, we can find `RMFinder`'s main application and the pub-sub interface, which connects with the desired pub-sub network and the corresponding DHT. At the bottom we find the network itself.

To deploy `RMFinder`, one would need to implement the pub-sub interface, which communicates between `RMFinder` and the pub-sub network. We provide such implementation for the popular topic-based pub-sub platform Scribe [8], which is itself built on top of the DHT system Pastry [30]. We used FreePastry's implementation [40] for both Scribe and Pastry, which is the most popular one available. In addition, to simulate long period scenarios and to test advanced features of the

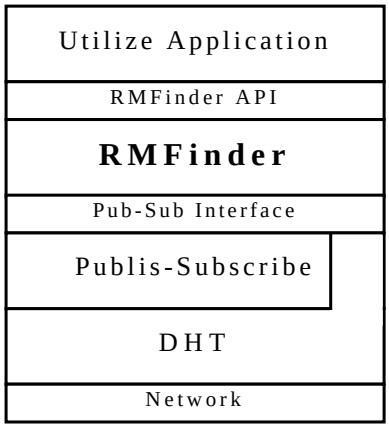

Figure 5: `RMFinder`network hierarchy

system, we provided an additional implementation of the pub-sub interface - a local one. This environment is much faster as it runs on a single computer where each hop between entities (topics, clusters) is done by a single pointer.

`RMFinder` exposes the same API as any other topic-based pub-sub system for defining topics, publishing events and (un)subscribing. Although it can operate with the these standard methods, to take full advantage of the additional properties, two new method were added to its API:

- **fetch(topic, message_id)** `RMFinder` enriches each published message with related content, which in order to reduced the bandwidth, is only the header of the message and a unique id to identify it within its corresponding topic. Therefor, we added the fetch function to retrieve this messages, and thus to support web ”surfing” navigation between them.

- **getClusters(topic)** With this function, one can view all the current clusters the given topic is member with, and with them, discover

new related topics which may share the same subjects.

Utilize `RMFinder` To test and demonstrate the various features of `RMFinder`, we needed to utilize its API. We therefor created two different GUI (graphical user interface) applications: The first, which we referred as the `Client`, supports all the obvious features one could expect from such a program (e.g. creation of topics, publishing messages, etc.). In addition, it also supports a web ”surfing” style navigation for traversing the related messages in an intuitive manner. The second application which utilize `RMFinder` is designed for extensive tests and demonstrations, and therefor includes (in addition to the ones of the `Client`) features for tracing and manipulating complex scenarios which could occur in the real world. For example, it can show at any time the current state of the clusters and the exact features set of each topic/cluster. It also supports an automatic mechanism for publishing massive amount of messages, which let us observe the behavior of the system over a long period of time. Note that we used real world messages, which were prior retrieved and kept in a database (more information at section 5.2). We refer this application as the `Simulator`. For additional information, appendix B provides a visual GUI tour of `RMFinder`'s GUI applications.

The layered design which we used to separate the different models gives us the freedom to decide exactly how to deploy the final program. For example, if we choose to run the `Simulator`, we can use it on any network which implements the pub-sub interface (in our case, either local or Scribe). Figure 6 illustrated the different options available by our implementations to run `RMFinder`. Level 1 describes the program

the utilize `RMFinder` while level 3 describes the the preferred pub-sub network. Note that the figure only represents the implementations we provide, and not entire availability options - any corresponding model (for example some other pub-sub network) can be replaced.

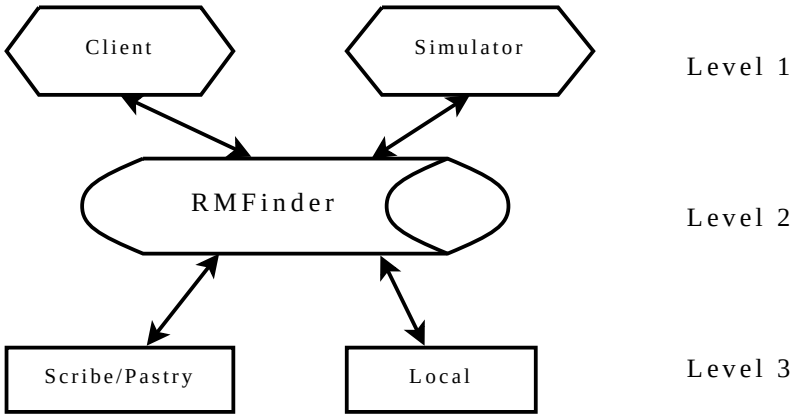

Figure 6: Different options for deploying the final program

`RMFinder` **packages** The implementation of `RMFinder` is encapsulated by one main package, naturally called ”rmfinder”. The inner package hierarchy is illustrated in figure 7. For simplicity, the figure shows only the main packages (which are important for understanding the design), and not the entire implementation. Following, we provide short description for each such one:

rmfinder.app holds the implementations of the programs who utilize `RMFinder`, and therefor, is the parent of two inner packages: ”.client” and ”.simulator” (for the `Client` and the `Simulator` respectively). Note that both programs will implement the client interface of `RMFinder` which resides in ”rmfinder.api”.

rmfinder.api holds all the classes required for one who wishes to utilize

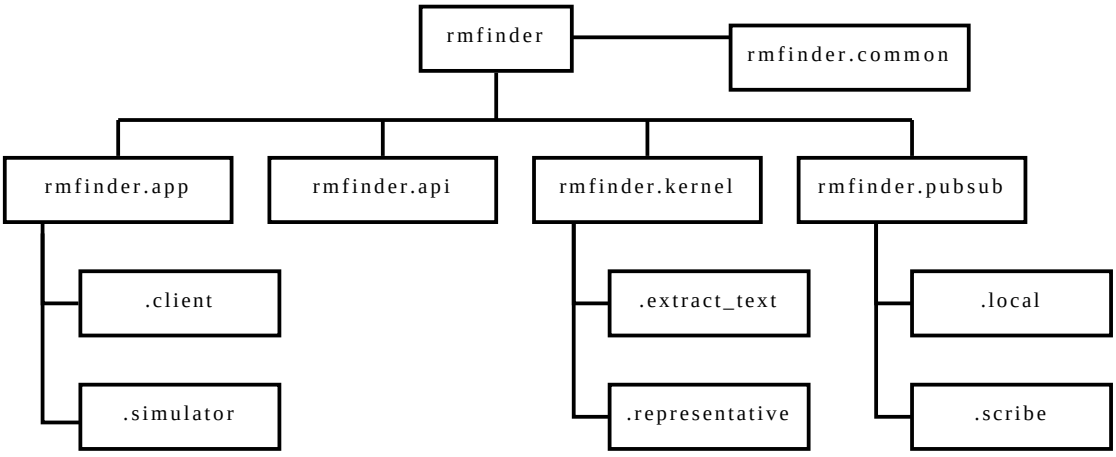

Figure 7: RMFinder package hierarchy (partial)

`RMFinder`. For example, this package holds the client interface (for `RMFinder` to notify about events), `RMFinder`'s API (for operating it) and all the entities required for operation (e.g. a topic, a message, etc.).

rmfinder.kernel holds all the kernel classes for operating `RMFinder`. For example, the main kernel package ("rmfinder.kernel") holds the classes for managing the user's (i.e. the utilize program) requests, managing all the threads and managing the container for all the representatives (topics/clusters) which the current instance (peer) represent. In addition, it holds two important inner classes: The first one is the ".extract_text", which holds all the logic about text extraction (from a message), stopwords, stemming and TF-IDF weighting. The second one is the ".representative", which holds all the logic about a topic and a cluster (for example, the profile, the archive, the related messages finder, etc.)

rmfinder.pubsub holds the interface for connecting a topic-based pub-

sub network. It also has to inner packages, ".local" and ".scribe", which holds the implementations for the local network and for Scribe, which we provide.

rmfinder.common holds all the common object for all the different packages.

Multi requests support For a simpler integration, `RMFinder` supports a non blocking interface: When the utilize program requests an operation from `RMFinder`, a unique request id number is returned. `RMFinder` then process the request (for example, publishing a message), and when its done, an appropriate function is called on the utilize program (via the client interface) with the corresponding request id number. During this time (some request might take some time, for instance the publishing request is now more complex due to the need to locate related messages, and therefor takes more time), additional request can be made with `RMFinder`.

Deadlock avoidance As one can predict, different peers may invoke `RMFinder`'s methods simultaneously. For example, lets assume our instance of `RMFinder` is the representative of some topic, and at the same time, two different users publish a message within that topic. If the pub-sub network supports (and normally it does) multiple threads, our representative topic will be invoked twice with the new published messages. `RMFinder` do supports this scenarios, and it even handle them simultaneously and will not put them in a queue for sequential processing. To support it, `RMFinder` enforces a strick locking mechanism; a representative (topic/cluster) will be locked for a minimal time, just to read or update

the data. While most operations fit perfectly to this scheme, there are two operation which requires the retrieval of information from other peers before they can successfully complete, and thus requires locking for a longer period. This behavior might lead to a deadlock - specific condition when two or more processes (peers) are each waiting for another to release a resource. Following, we describe these operations, the reasons which led for the deadlock, and finally the avoidance method.

- The first operation is the publishing of a new message. When a client (peer) wishes to publish a message, he send it to topic representative. In a regular topic-based system, the representative simply broadcast the messages via its corresponding multicast tree, rooted by him. In `RMFinder` however, the topic representative first contacts all its member clusters, weights the message, and then query all corresponding topics. If the topic representative will lock itself for the entire process, other new request might be postpone. While at first it does not seems too terrible, this action might cause a deadlock in the network, as two topic representatives might contact each other at the same time. To overcome it, we move the logic of publishing a new message from the topic representative to the publishing client (peer). This will cause the peer to wait for the responses from the other peers (representatives), and thus eliminate the need for the lock by the topic. This operation also reduce the bandwidth/cpu time required by the representative topic, and thus create a better scalability for the network.

- The second operation is the expending of a cluster. When one cluster tries to merge with a second one, it sends a request regarding

the possible merge, and then waits for a reply. During this time, the first cluster might receive requests for possible merge from other clusters. If the cluster will simply put these request in a queue, and handle them after it will finish the current request, a deadlock might occur. If for example, the second cluster will also send a merge for the first one, at the same time as the first sends, they will both enter a deadlock. Although this scenario can be easily be recognized, a more complex one, with the same principles can occur. Imaging there are three clusters involved, c_1, c_2 and c_3, and each one is trying to merge an other one; c_1 with c_2, c_2 with c_3 and c_3 with c_1. We get a circle of deadlocks, where each cluster waits for the other one. This example can of course be generalize to involved even more clusters, and thus the recognition of such a scenario is impossible. Therefor, to avoid these deadlocks, when a cluster sends a merge request, it will decline any incoming merge request from other clusters. Note that this is a special decline, and thus the requesting cluster will be aware of the reason the cluster had declined, and therefor will try again after some randomly period of time (enough for letting the cluster to finish its merge process).

`RMFinder` **Web site** For an easy installation and deployment, we provide a designated web site for `RMFinder`. The location of this site can be found at [2]. This web site will hold all the necessary files for running `RMFinder`, as well as a comprehensive documentation regarding the deployment of the development environment (see also appendix A). The web site also provide a Javadoc documentation [36] of the classes and methods structure.

5.2 Experiments

We ran two sets of experiments: The first group of experiments (section 5.2.1) concerns with the clustering algorithm and all its aspects (adaptivity, quality, etc.) The second group of experiments (section 5.2.2) compares the two method for finding related messages (D-VSM and SA).

Real life data To test our system in a realistic setting, we used real life data which we accumulate over a period of six months. More specifically, we retrieved over fifteen different popular RSS feeds, which were divided into three groups according to their content: NBA (basketball), News (world and US) and entertainment. The feeds origin are: CNN, BBC, CBS, FOX, Yahoo!, NY Times, ESPN, Sports Illustrated, Yahoo!, USA TODAY and E!. For a fast and easy access, the data was kept in a MySQL [37] database. Note that this data will be used by both groups of experiments.

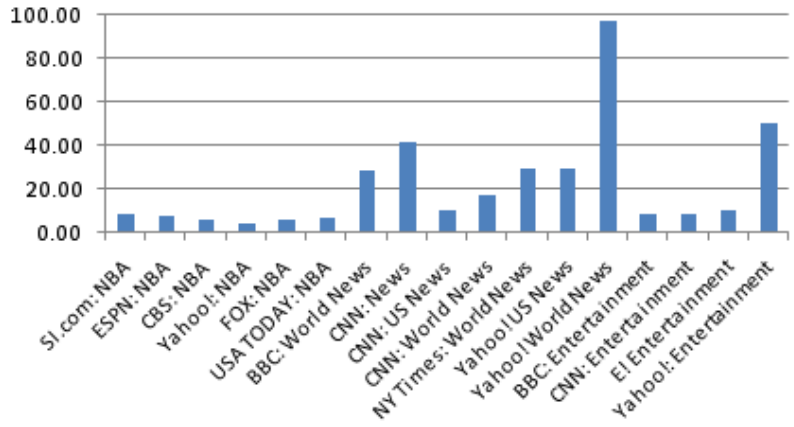

Figure 8: Average messages published per day

Despite the different coverage area of each topic, another variable

which will influence the results is the frequency of messages in a given topic (i.e. the number of messages published by a topic per day). Therefor, to add an additional validation to our experiments, we would like these frequencies to differ between the different feeds. Figure 8 depict the average number of messages published per day for feed. As one can see, they already differ, and therefor we leave them untouched.

5.2.1 Cluster evaluation

In this section we focus on evaluating the clusters formation, created by our algorithm, implemented by `RMFinder`. More specifically, we would like to examine two main aspects: First, the quality of the clusters created, and second the adaptivity of the clusters when changes occur. We describe each of them on the following paragraphs.

Clusters quality According to the data used to simulate the realistic settings, one would expect from the algorithms to produce three clusters, one for each group of interest (NBA, news and entertainment). Indeed, `RMFinder` successfully produced these three cluster, where each of them contained the exact corresponding topics according to our initial division. Despite these results, we would like to asses the quality of the clusters formation compared to a standard **centralized** algorithm. We chose K-means [21] as that algorithm, as it both has an easy implemenation and is one of the most (if not the most) popular clustering algorithm available today. Note that K-means requires its participates objects (topics, clusters) to form a vector space. Naturally, the topics use their profile as their describing vector (where the space is the one of all the features). The clusters will use their cluster features set, which we initially cre-

ated for weighting the messages published within the cluster, as their describing vector.

We conducted the experiment as followed: We used the real life data which was accumulated prior, to simulate a period of almost 6 months of regular use (of `RMFinder`). After every four days simulated, we ran K-means algorithm (with a value of K=3) on the simulated topics (which dynamically adapts their topics profiles according to their published messages) and compared the formula's value (F) calculated once for the formation created by `RMFinder` and ones for the formation created by the K-means algorithm. Note that a higher value of F means better utilization of the clusters, and thus better results. Figure 9 depict F's value for each algorithm.

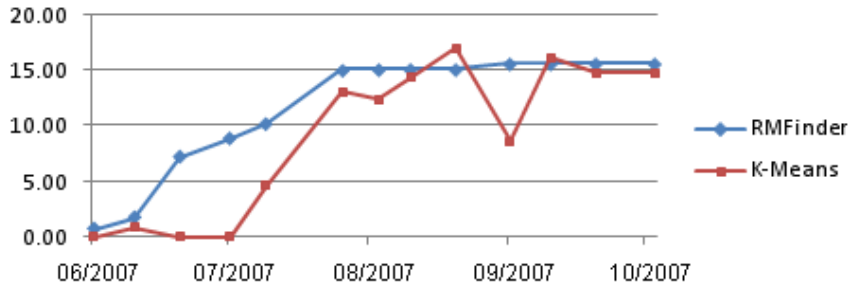

Figure 9: RMFinder VS K-Means (formula value)

As can be expected, both algorithms needs some "training time" (needed by the topics profiles to describe their corresponding topics accordingly), and thus F's value starts low for both, but as time passes by, it rises correspondingly, reaching a similar high value. However, while `RMFinder`'s value reaches a stable state, K-means fails to do so, and has several drops. We believe these drops origin from the fact that the clusters features set is less sensitive to the dynamic nature of the topics, compared to

the topics profiles used by `RMFinder`. We can see that overall, the clustering quality of `RMFinder` is close to that of the centralized K-Means, and most of the times even superior.

Adaptive clustering While previously we assessed the quality of the clusters, on the following experiment we focus on the dynamic aspects of `RMFinder` and evaluate the adaptivity of the clusters formation.

We conducted the experiment as followed: Like before, we used the real life data which was accumulated prior, between May 2007 to October 2007. However, in order to test the adaptivity, we simulated some changes in the topics focuses: First, we randomly chose three topics (one from each group). For all other topics, we republished their accumulated messages continuously (i.e. for November 2007 we published the messages retrieved at May 2007 and for April 2008 we published the messages retrieved at October 2007). For the remaining three selected topics, we switch their content in circle, and similarly republished their accumulated messages continuously (i.e. for three topics - A, B and C - for November 2007, A published B's May 2007 messages, B published C's May 2007 messages and C published A's May 2007 messages). Note that this change is considered to be dramatic, as it is unlikely that a topic will change its focus completely. To identify this change, we recorded the formula's value at the end of each day (compared to every four days at the previous experiment). Figure 10 depict F's value over a simulation of almost a year.

Like before, during the "training time", the formula's value rises up until it reach a stable state. Until October 2007, as can expected, the formula's value remains at this state. However, following this date we do expect a

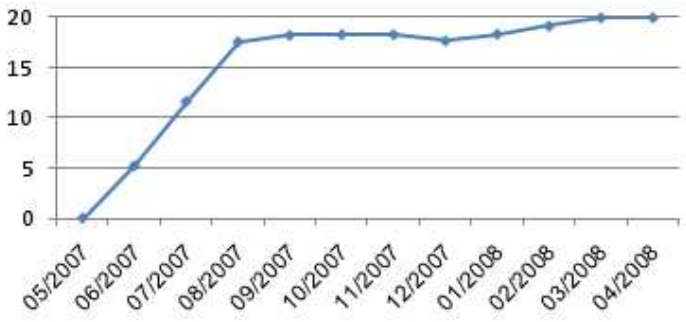

Figure 10: Adaptive simulation for RMFinder (formula value)

change, as three topics changed their content. Indeed, the decrease starts about a month after the change, and quickly after it rises back to a level even a little higher than before. The experiment highlights the adaptivity of `RMFinder` to changes in the topics focus.

5.2.2 Message similarity evaluation

In this section we focus on evaluating the two methods for finding related messages (D-VSM and SA), described at section 4. We assume we already have a stable environment, were all the topics are clustered correctly (by their content). The goals of the experiments described in this subsection are (1) to compare how well the two methods operates on real life data, and (2) examine the reasons for the differences in their behavior. This was tested by two sets of experiments, described below.

Main survey To compare the quality of the related messages found by the two methods, we ran the two algorithms and showed the results to a group consisting of 40 people. We then conducted a survey where each person was asked to grade the results according to two notions of similarity, denoted *Related* and *Similar*. Two messages are *related* if

they share a common focus; for example, the first message describes the new apple's iPhone, while the second describes the new blackberry: they share the focus of cell phones. Two messages are *similar* when they describe the same information (just put in different phrasing), as the same main headlines are published in each corresponding feed; for example, the first message can be "Apple releases its new iPhone", while the second can be "The iPhone is finally released by apple".

The survey was conducted as follows: Each user was given 15 randomly chosen messages, with each message being attached two sets of related messages (of size 3), discovered by `RMFinder` by the two methods (D-VSM and SA). The user was then asked to rate each set of related messages separately, according to the two notions (*related* and *similar*) discussed above. The ranking was on a scale of 0 to 5, as follows:

- 0 if non of the messages were related/similar.
- 1-2 if one message was related/similar.
- 2-4 if two messages were related/similar.
- 5 if all three messages were related/similar.

Through all the surveys, `RMFinder`'s configuration was set to the standard one: the number of messages kept for comparison was set to 200 (the 200 newest messages) and the size of the features set was set to 250. This number can affect the D-VSM results as we ignore all the features within a message which are not member with the features set. Recall that we can not increase this number too much as we want to keep the low overhead property of a P2P network.

The two graphs in figures 11 and 12 depict the average rank given

by the users to the D-VSM and SA methods respectively. Note that although only the published topics of the randomly select messages appear in the graphs, all the topic did participate in the experiment as they supplied the related/similar messages. Each graph contains the value for both the related and the similarity properties. As can be expected, in both methods the related property got a higher value than the similarity. This is due to the fact that the related property is essential for similarity, and thus the users could not rank the similarity with a higher value than the related. Figure 13 depict the average rank of all the topics for each

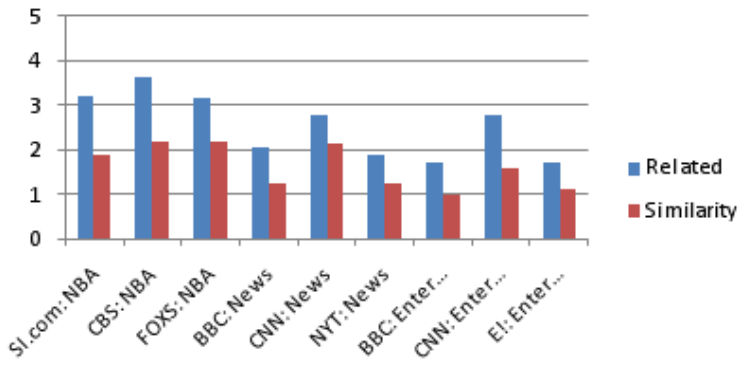

Figure 11: Average rank (by D-VSM method)

method separately. Both methods produced impressive results - the average related value for each method is high, and although the similarity value is lower, its still surprisingly good, as our system is not particularly targeted to retrieve such messages.

If we compare the two methods, we can observe that SA outranks the D-VSM method in both properties. This is somewhat surprising as we would expect the D-VSM methods to overcome the simpler SA method. To understand the reasons for these results, on the following paragraphs

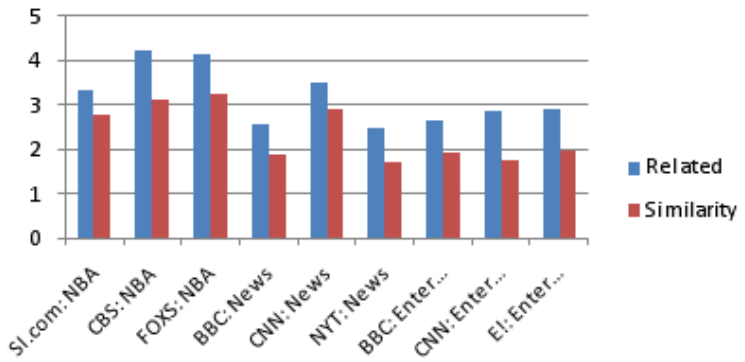

Figure 12: Average rank (by SA method)

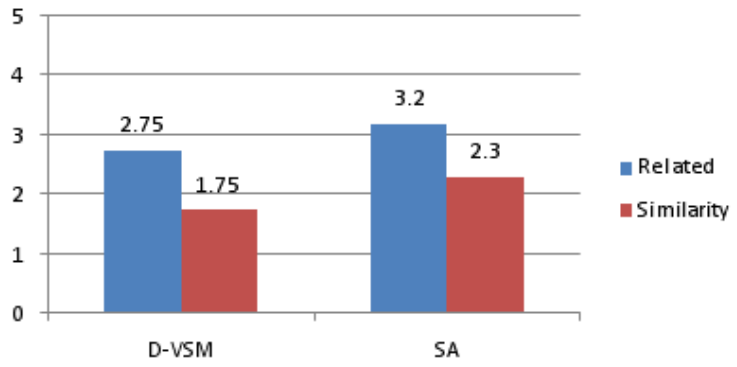

Figure 13: D-VSM VS SA

we examine the differences between the methods more carefully.

D-VSM vs SA Our first step was to measure the differences in the results obtained by the two methods. For that we ran the following experiment: On the ”regular simulation” (the publication of messages during a period of 6 months), for each topic request for finding related messages, we searched the topic's history by the two methods, and compared the results sets. Note that we compared the sets themselves, and not their internal order (i.e. if two sets contain the same messages but in different

order, they are considered to be the same). We repeated this experiment four times, each time with a different result set size (1, 5, 10 and 20). Figure 14 depict the average percentage of shared messages between the sets.

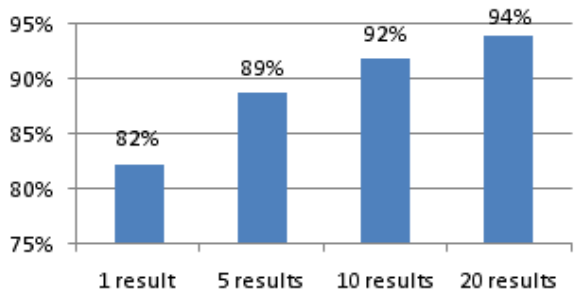

Figure 14: D-VSM method and SA shared results comparison

For a result set of size 1 (i.e. the most appropriate message available), the two methods have an average of 82%, that is for more than 8 messages out of 10, both methods returned the same message as their top related result. As we increase the size of the results set, the percentage increases respectively, where sets of size 20 produce an impressive 94% of similarity. To understand how these numbers match our surprising results from before (where the SA outranked the D-VSM), we first need to recall the way these methods operates: Both methods select a subset of messages (from within the set of comparison messages), and for each one they calculate the similarity value with the given message. The ones with the highest value are returned as the result. The difference is the way these subsets of messages are chosen: In the D-VSM method, this is done by selecting all the messages which share one or more feature with the given message **and the cluster's features set** (all others are dismissed). Note that we dismiss them because we do not have their rel-

ative weight, given by the cluster. In the SA however, there is no cluster features set, and the selection is done only according to the given message. Thus, the subset of messages can be more comprehensive. Note that the calculation is also different as the weights of the features are different in each method; in D-VSM it is set by the TF-IDF method, while in SA the weight is fixed and set to 1.

According to figure 14, the two methods indeed found the same set of messages, they just ordered them (ranked them) differently. As explained before, this may be due to two reasons: Either the TF-IDF weighting methods does not behave well in our environment (and thus weights the features incorrectly), or the features set is not comprehensive enough (and thus features are given the weight of 0 - i.e. are dismissed). The first reason does not seem to apply here, because we have seen previously that the clusters structure is adequate, and it does form correctly (section 5.2.1). Thus, the second reason seems to cause the above phenomenon. To reinforce this hypothesis, we have conducted an additional experiment as follows: For each published message and its corresponding result set (of size 5), constructed by the SA method (which by previous experiments is better than the D-VSM results set), we compared the features usage of each of the methods. That is, for SA we counted the number of common features between the given message and each messages from within the result set. For D-VSM, we counted the shared features which had a weight bigger than 0 (and thus, members within the cluster's features set). Figure 15 depict the average features usage by the two methods for the most related messages, discovered by the SA method. This average covers almost 600,000 comparison. As we can observe, the SA uses almost twice as much features compared to the

D-VSM, which proves our hypothesis. Another interesting point to note, is the big gap between the number of features and the number of actual usage features.

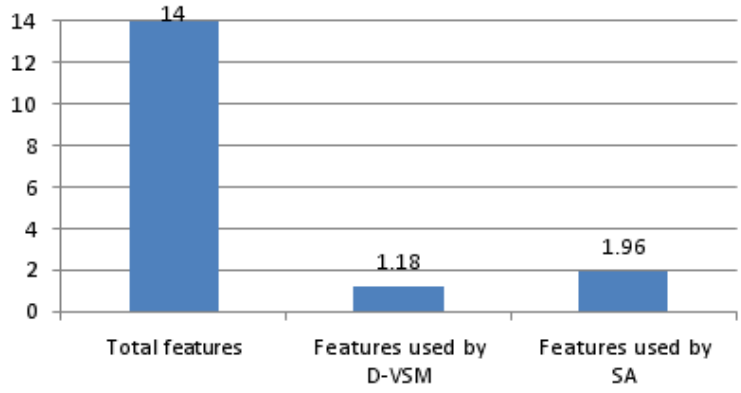

Figure 15: Average features usage of each method

These experiments shows that although the features set holds all the ”important” features, supplying additional ”less important” features is more valuable than simply weighting the ”important” ones. We believe there are two reasons which led to this conclusion: First, we deal with short descriptive messages, therefor each feature is more important compared to a traditional feature within a document. Second, prior for comparison we cluster the topics by their content, and thus the messages already have some related aspects between them. We can however imagine an environment, where these conditions do not apply (at least the first, where the average message length can be much longer), and thus this conclusion might not hold. In such environments, the D-VSM method might overcome the SA method.

6 Related work

The publish-subscribe paradigm have received much attention in recent years. Research on topic-based pub-sub led to the development of several systems, such as Scribe[8] and Bayteux[48]. While different in their implementation, they share similar API and structure, and may employ the clustering technique developed in this work to enrich messages with related content. Complementary to our work are projects like Green [33] and SplitStream[7]. Green introduce a middle-ware to support flexible pub-sub on top of diverse network types (e.g. Internet vs. mobile networks), while SplitStream aimed to high-bandwidth data dissemination application (e.g. video), and addressed the difficulty of interior nodes (in multicast trees) that carry most of the load of forwarding large messages. More additional work influenced by the flourishing of RSS news syndication are Corona [28] and FeedTree [32]. They are designed to alleviate the load on RSS feed providers and allow for better scalability.

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 subset hierarchy [39]. 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 stable good results even when the type of the messages published by the topics changes significantly.

Much research has been devoted in the database community over the last ten years to content delivery and data dissemination. See [11, 12, 1, 9] for a very small sample. In this paper we focus on a rela-

tively simple class of such systems, that disseminate events of predefined topics. The relative simplicity of these systems is precisely what makes them attractive for simple applications. As mentioned in Section 1, content-based pub-sub allows for more flexible subscription, based on the content of messages. Some example systems include JEDI[10], SEINA[6], Kyra[5], Gryphon[35] and Herald[4]. The added flexibility requires more sophisticated protocols, typically with higher runtime overhead. The use of clustering algorithms for enhancing the performance of content-based pub sub has been considered in [29]. However, all the algorithms considered there are centralized and the dynamic aspect of subscription is not addressed. A distributed clustering algorithm for content-based pub-sub has been proposed in HYPER[46]. It assumes however the use of a central coordinator. Such a coordinator may not exist in a fully distributed P2P environment as the one considered in our work.

Several distributed clustering algorithms appear in the literature, e.g. [26, 20, 17, 19], but assume static input and do not account for changes in the data. Adaptive clustering is considered in [18, 25]. But these works assume constant dimension for the clustered objects. This is not the case in our context: In a dynamic pub-sub system, topics may be created/demolished at any time, causing the dimension (of the objects) to change with time. Our work builds on a novel distributed dynamic clustering algorithm, presented in [38], which allows to continuously adapts the topic-clusters to the current system's state. The clustering algorithm was originally introduced in [38] 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, with

minimal overhead.

Feature selection is the technique, commonly used in machine learning, of selecting a subset of relevant features for building robust learning models. In this work we used a simple implementation of this technique - we weighted the features by the popular TF-IDF method [31] and selected the top ranked ones. However, more sophisticated algorithms can be used, such as [27, 44, 22, 45]. These techniques can be enhanced even more by feature generation based on domain-specific and common-sense knowledge [16].

7 Conclusion and future work

In this thesis, we studied topic-based pub-sub systems and proposed a novel technique for enriching these systems with related content. Our solution is based on 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. To locate within these clusters the corresponding messages, we introduced two different methods: D-VSM (dynamic version of the classical vector space model) and SA (simple appearances). Although the D-VSM is more sophisticated, we saw that, in our context, the SA techniques provides superior results. This due to the short descriptive nature of the messages and the prior clustering algorithm, which in fact is a part of the messages comparison.

We have implemented our solution in `RMFinder` pub-sub system, and showed experimentally its effectiveness. Some interesting issues are still open and left for future research. We briefly describe some of them:

Distributed message archive `RMFinder` enriches a message with the headers of the most related messages it can locate. A typical header contains the origin topic of the related message, its title, and most important its unique ID. With this ID, the user can invoke the ”fetch(topic, message_id)” method and receive the full version of the related message. Currently, each topic representatives holds the archive containing all previously published messages and thus the ”fetch” method received its data by the corresponding topic representative. This architecture is not ideal, as it is not scalable for massive

amount of users. To solve it, this archive can be saved in a distributed manner, removing the bottleneck from the topic representative. For example, some variation of [13] might give the desired result.

Reduce the number of message comparisons Currently when a message is published, it is searched for related messages by all member topics which share a cluster with the publisher topic. An optimization however, can select only an adequate subset of the member topics (according to their profile and the message published) and thus reduce the number of comparisons.

Tuning the system variables There are several variables which we set prior for running our algorithm: the size of the sliding window, the number of recent messages saved for comparison and the weight w used by the formula F. Although our algorithm achieves impressive results with these pre set values, tuning these values in a dynamic manner may be more adequate for non uniform/unstable environments.

Expanded API Recall that to provide richer service `RMFinder` expended the standard API of topic-based pub-sub systems. Further additional expansions are possible. For example, a user could register to all the topic-clusters of a specific topic, and thus receive even more messages, thereby expending his initial subscription request. Furthermore, we can filter the messages sent to a given user and avoid sending messages that are *similar* (not to confuse with *related*) with the messages that he had already received, thus prevent the re-dissemination of redundant information by multiple topics.

References

[1] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. Dbis toolkit: Adaptable middleware for large scale data delivery. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 544–546. ACM Press, 1999.

[2] R. Boim. Rmfinder site. http://www.cs.tau.ac.il/ boim/projects/rmfinder/.

[3] R. Boim and T. Milo. Enriching topic-based publish-subscribe systems with related content. In *Proc. SIGMOD*, 2008.

[4] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 87–94, Elmau, Germany. IEEE Computer Society, May 2001.

[5] Fengyun Cao and Jaswinder Pal Singh. Efficient event routing in content-based publish-subscribe service networks. *IEEE INFOCOM*, 2004.

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

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. 2003.

[8] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[9] Badrish Chandramouli, Junyi Xie, and Jun Yang. On the database/network interface in large-scale publish/subscribe systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 587–598. International Conference on Management of Data, 2006.

[10] G. Cugola, E. Di 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(9):827–850.

[11] Y. Diao, P. Fischer, M.J. Franklin, and R. To. Yfilter: Efficient and scalable filtering of xml documents. In *Proc. of the 18th Intl. Conf. on Data Engineering*, pages 341–342. ICDE, February 2002.

[12] Yanlei Diao, Shariq Rizvi, and Michael J. Franklin. Towards an internet-scale xml dissemination service. In *Proceedings of the 30th VLDB Conference*, pages 612–623, 2004.

[13] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. *HotOS VIII*, 2001.

[14] Eclipse. Eclipse home page. http://www.eclipse.org/.

[15] A. Wong G. Salton and C.S. Yang. A vector space model for automatic indexing. pages 613–620, 1975.

[16] E. Gabrilovich and S. Markovitch. Feature generation for text categorization using world knowledge. *Internationl Conference on Artificial Intelligence*, 2005.

[17] Joydeep Ghosh and Srujana Merugu. Privacy-preserving distributed clustering using generative models. In *The Third IEEE International Conference on Data Mining(ICDM'03)*, Melbourne, FL, November 2003.

[18] Y. Gourhant, S. Louboutin, V. Cahill, A. Condon, G. Starovic, and B. Tangney. Dynamic clustering in an object-oriented distributed system. In *Proceedings of OLDA-II (Objects in Large Distributed Applications)*, Ottawa, Canada, 1992.

[19] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM2003)*, Melbourne, FL, 2003.

[20] Ruoming Jin, Anjan Goswami, and Gagan Agrawal. Fast and exact out-of-core and distributed k-means clustering. *Knowledge and Information Systems*, 10(1):17–40, 2006.

[21] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions*, 24(7):881–892, 2002.

[22] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 7:491–502, 2005.

[23] J.B. Lovins. Development of a stemming algorithm. In *Mechanical Translation and Computational Linguistics*, volume 11, pages 22–31, 1968.

[24] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. IPTPS*, pages 53–65, 2002.

[25] Steiner Moritz and Biersack Ernst W. DDC: A dynamic and distributed clustering algorithm for networked virtual environments based on p2p networks. In *Proceedings of CoNEXT'05*, Toulouse, France, 2005.

[26] Elth Ogston, Benno Overeinder, Maarten van Steen, and Frances Brazier. A method for decentralized clustering in large multi-agent systems. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 789–796, New York, NY, USA, 2003. ACM Press.

[27] S. Perkins and J. Theiler. Online feature selection using grafting. *International Conference on Machine Learning*, 2003.

[28] V. Ramasubramanian, R. Peterson, and Emin Gun Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *Proc. of Networked System Design and Implementation*, 2006.

[29] Anton Riabov, Zhen Liu, Joel L. Wolf, Philip S. Yu, and Li Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 133. ICDCS, 2002.

[30] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. 2001.

[31] G. Salton and C.S. Yang. On the specification of term values in automatic indexing. *J. Documents*, 29(4):351–372, 1973.

[32] D. Sandler, A. Mislove, A. Post, and P. Druschel. FeedTree: Sharing web micronews with peer-to-peer event notification. In *Proc. Int. Workshop on Peer-to-Peer Systems (IPTPS05)*, New York, 2005.

[33] Thirunavukkarasu Sivaharan, Gordon Blair, and Geoff Coulson. GREEN: A configurable and re-configurable publish-subscribe middleware for pervasive computing. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA05)*, Agia Napa, Cyprus, 2005.

[34] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, pages 149–160, 2001.

[35] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. In *International Symposium on Software Reliability Engineering (ISSRE 98)*, 1998.

[36] Sun. Javadoc tool home page. http://java.sun.com/j2se/javadoc/.

[37] Sun. Mysql home page. http://www.mysql.com/.

[38] T. Zur T. Milo and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *Proc. SIGMOD*, pages 749–760, 2007.

[39] E. Patrick Th., G. Rachid, and S. Joe. Type-Based Publish/Subscribe. Technical report, 2000. LPD.

[40] Rice University. Freepastry. http://freepastry.org/FreePastry/.

[41] Wikipedia. Cluster analysis. http://en.wikipedia.org/wiki/Cluster_analysis.

[42] Wikipedia. Stemming. http://en.wikipedia.org/wiki/Stemming.

[43] Wikipedia. tf-idf. http://en.wikipedia.org/wiki/Tfidf.

[44] J. Liu Y. Li, Z. Wu and Y. Tang. Efficient feature selection for high-dimensional data using two lever filter. *Internationl Conference on Machine Learning*, 2004.

[45] Z. Zaho and H. Liu. Searching for interacting featues. *Internationl Conference on Artificial Intelligence*, 2007.

[46] Rongmei Zhang and Y. Charlie Hu. Hyper: A hybrid approach to efficient content-based publish/subscribe. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 427–436. ICDCS, 2005.

[47] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[48] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. Int. workshop on Network and OS support for digital audio and video*, pages 11–20. ACM, 2001.

Appendix

A Establishing the RMFinder development environment

This Appendix includes step by step instructions for establishing the `RMFinder`'s development environment. Before we can follow them, we first need to have the following platform and sources:

1. *Eclipse* SDK 3.2. Can be download from [14].
2. `RMFinder`'s development package ("RMFinderEnv1.0.rar"). Can be downloaded form [2]. Note that this package includes `RMFinder` 1.0 source files (rmfinder1.0.rar) and FreePastry 2.0_03 source files (FreePastry-2.0_03-source.zip). Other version of FreePastry can be found at [40] and used similarly.
3. MySQL 5.0 server. Can be download from [37]. The data used by RMFinder's simulator ("data.rar") can be downloaded from [2].

We are now ready to establish `RMFinder`'s development environment. For that, the detailed instruction steps below should be follow.

1. Run "Eclipse" and in the "Workspace Launcher" opening window set a new (and not existing yet) path. For example, "C:\ RMFinderEnv". Figure 16 illustrate this selection. Since it is a new workspace, you will have to choose your next step before entering the workspace itself. For that, press the "Workbench" link button (it is also titled: "Go to the workbench"), and you will finally be in the workspace. In your file system you will notice that you have now a new created folder - "C:\RMFinderEnv".

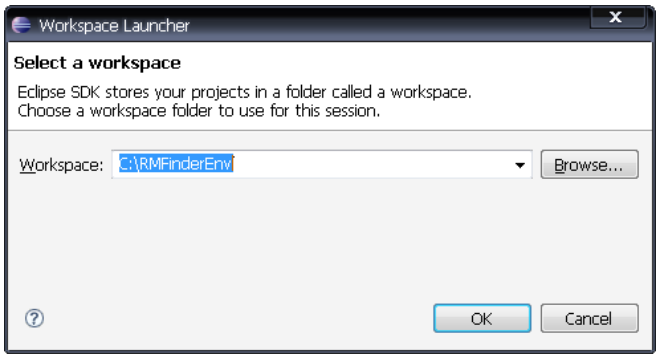

Figure 16: Eclipse ”Workspace Launcher”

2. In File→New→Project choose the ”Java Project” option, and press ”Next”. Set the name ”RMFinder1.0” in the ”Project name” field, make sure that in the ”Contents” selection area the option ”Create new project in workspace” is on, and press ”Finish”. In your file system you will notice that under the folder ”C:\RMFinderEnv” you have now a new folder named ”RMFinder1.0”.

3. Copy the following FreePastry source folders (from ”FreePastry-2.0_03-source.zip”) and their content into your new folder ”C:\RMFinderEnv\RMFinder1.0\”

   (a) src

   (b) jars

   (c) lib

   Copy the following `RMFinder` source folders (from ”rmfinder1.0.rar”) and their content into your new folder ”C:\RMFinderEnv\RMFinder1.0\”:

   (a) src

   (b) jars

   Note that when you are done, the ”src” folder will have now two subfolders: ”rmfinder” and ”rice”

4. In Eclipse, refresh the workspace (you will get many compilation errors - ignore them).

   On the left window, mark the new project ”RMFinder1.0” and open Project→Properties. First choose ”Java Compiler” and make sure the ”Compiler compliance level” is 5.0 (by enabling project specific settings if necessary). Then choose the ”Java Build Path” and follow the instructions bellow:

   (a) Choose the ”Source” tab. In the list of ”Source folders on build path” you will have one path of ”RMFinder1.0”. Mark it and ”Remove” it. Press ”Add Folder” and choose two: ”src” and ”freepastry” (located under ”jars”). Notice: do not choose ”jars”, although it has only one folder underneath. See figure 17. Press ”OK”. The two folders will appear in the ”Source

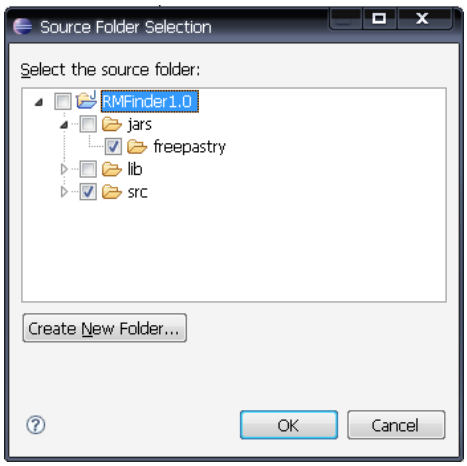

Figure 17: Eclipse folder selection

   folders on build path:” window. Mark the ”Allow output folders for source folders” check box. And in the ”Default output folder:” filed, set: ”RMFinder1.0/classes”. The ”Source” tab should look like figure 18.

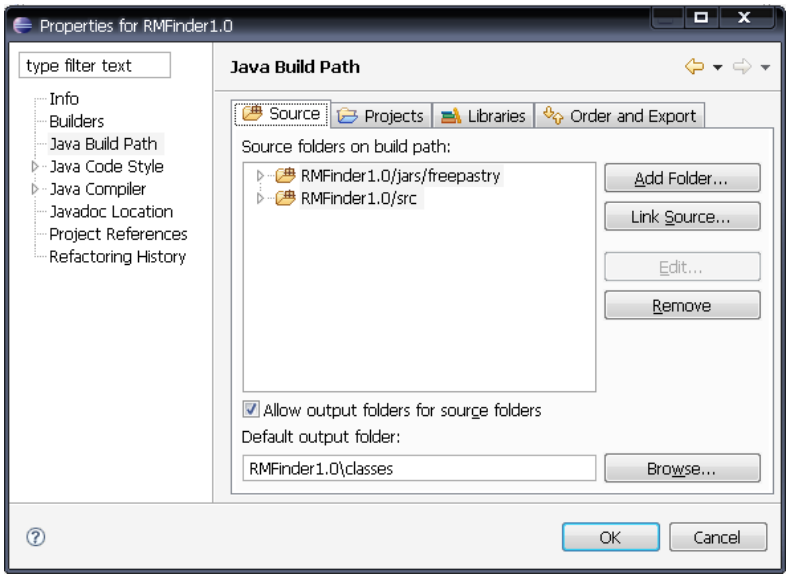

Figure 18: Eclipse source tab settings

(b) Choose the ”Libraries” tab. Press ”Add JARs...” and from the window selection choose the seven FreePastry JAR files: (1) bouncycastle.jar, (2) commons-jxpath-1.1.jar, (3) commons-logging.jar, (4) sbbi-upnplib-1.0.3.jar, (5) sbbi-upnplib-1.0.4.jar, (6) xmlpull_1_1_3_4a.jar and (7) xpp3-1.1.3.4d_b2.jar. Notice that these JAR files may have other (but similar) names, depends the FreePastry version you downloaded. Next, choose the ”Add External JARS...” and select the ”mysql-connector-java-3.1.11-bin.jar” from the ”jars” folders. The ”Libraries” tab should look like figure 19.

Press ”OK”.

You may receive a request window as shown in figure 20. Simply press ”OK”.

5. Install MySQL 5.0 database (or use an existing one). Following,

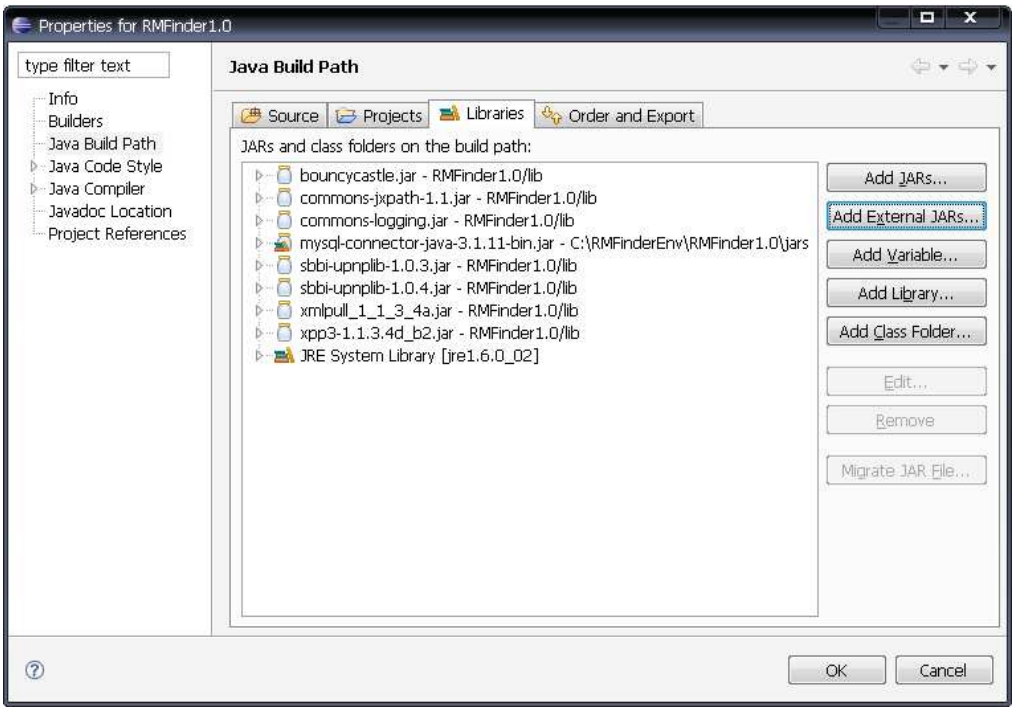

Figure 19: Eclipse libraries tab setting

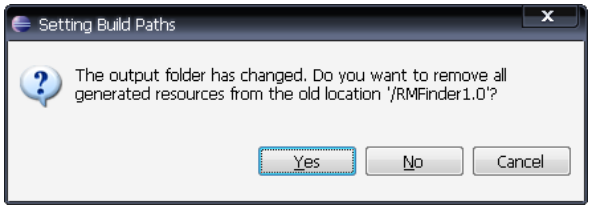

Figure 20: Eclipse setting build path warning

open the ”MySQL Administrator” program and choose ”Restore”. Then, select the ”data.sql” file (from ”data.rar”) as the ”File to restore”, ”rmfinder” as the ”Target schema” and click the ”Open Backup File”. The database should now be restored.

Congratulations , the `RMFinder` development environment is now established.

- To run the `Client` application, select the ”main” function located under ”rmfinder.app.client.RunClient.java”.
- To run the `Simulator` application, select the ”main” function located under ”rmfinder.app.simulator.RunSimulator.java”.

Note that the configuration file for all `RMFinder`'s parameters can be found at ”rmfinder.api.Consts.java”. Additional properties regarding the simulator can be found at ”rmfinder.app.simulator.SimConsts.java”.

B `RMFinder` GUI tour

In this section we present several screenshots taken from the `Client` and the `Simulator` programs. Note that some of the classes (for example the messages browser and the message window) are shared by both programs, and thus some of the screenshots correspond for both.

We start with basic menu used by the programs. Figure 21 shows the pub-sub menu used by both programs. Note the additional ”Get Clus-

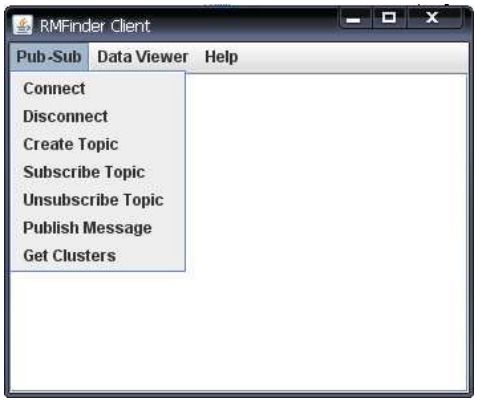

Figure 21: The pub-pub menu options

ters” option added by `RMFinder` for retrieving the current clusters formation for a given topic.

For managing all the published messages each peer received, we use the ”message browser” window. As we can observe from figure 22, on the left column we first choose the desired topic. We then receive on the middle column all the dates which contains a message published by the selected topic, and finally we choose the desired message from all the ones published at the selected date.

We continue our tour with the most most important window - the ”message window”. Figure 23 shows a typical message published in

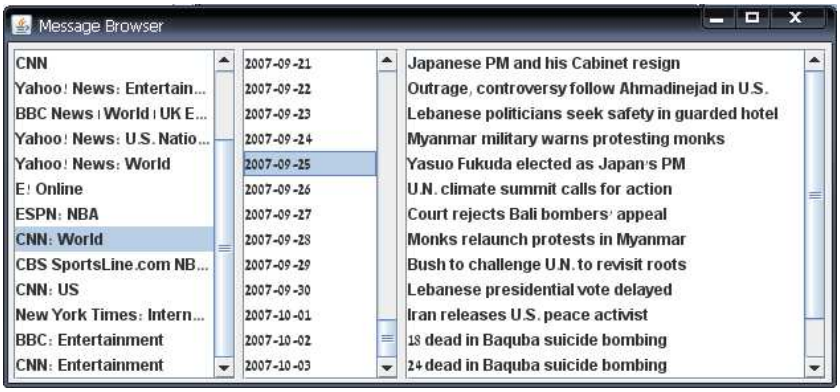

Figure 22: The message browser

Figure 23: The message window

`RMFinder`. On the top part of the window we can find all the typical data published such as title, body, origin (published topic) and date. On the lower part of the window however, we can find the enriched related messages found by `RMFinder` for the specific message, ordered by their similarity value. On the bottom right we can find two additional options for the enriched data: The first is the ”order by” option (by similarity or date) of the enriched content, and the second is the option to

filter out related messages published by the same topic as the one who published the given message.

On figure 24 we can find the simulator menu, with all the options for simulating all our experiments. These options include the DB and simulator management, external data load (topics, dates) and k-means algorithm.

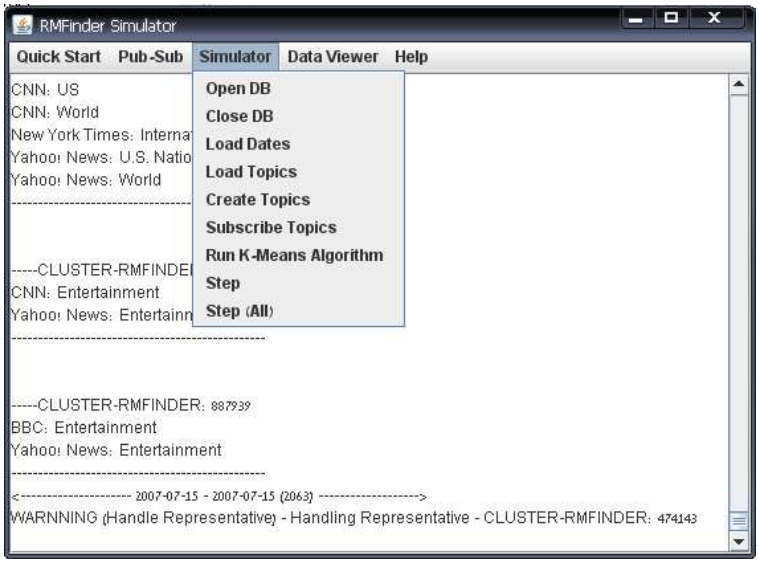

Figure 24: The simulator menu options

To conclude this GUI tour, we show on figure 25 the ”dictionary viewer”. With this tool we can observe and compare the specific features sets of any topic or topic-cluster desired.

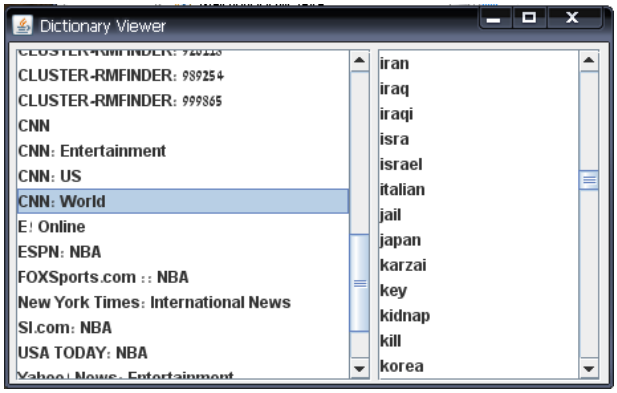

Figure 25: The dictionary viewer