Subscription Subsumption Evaluation for Content-based Publish/Subscribe Systems

Hojjat Jafarpour, Bijit Hore, Sharad Mehrotra and Nalini Venkatasubramanian

Department of Computer Science, University of California at Irvine
{hjafarpo,bhore,sharad,nalini}@ics.uci.edu

Key words: Publish/Subscribe, Subscription Subsumption, Message-oriented middleware

Abstract. In this paper we address the problem of subsumption checking for subscriptions in pub/sub systems. We develop a novel approach based on negative space representation for subsumption checking and provide efficient algorithms for subscription forwarding in a dynamic pub/sub environment. We then provide heuristics for approximate subsumption checking that greatly enhance the performance without compromising the correct execution of the system and only adding incremental cost in terms of extra computation in brokers. We illustrate the advantages of this novel approach by carrying out extensive experimentation.

1 Introduction

Content-based Publish/Subscribe (pub/sub) is a customized many-to-many communication model that can satisfy requirements of many modern distributed applications [1]. In a pub/sub scheme, subscribers express their interest in content by issuing a subscription (query). Whenever some content is produced, it is delivered to the subscribers whose query parameters are satisfied by the content in the publication. By decoupling communication parties, a pub/sub system provides anonymous and asynchronous communication making it an attractive communication infrastructure for many applications. Such applications include selective information dissemination, location-based services, and workload management [1].

In order to distribute the load of publications and subscriptions a distributed content-based pub/sub system uses a set of network brokers (nodes). Different architectures have been proposed for connecting brokers in a pub/sub network [2, 11, 12]. Routing protocols for publications and subscriptions in brokers aim to reduce network traffic that results from transferring publications and subscriptions between nodes. One way in which publication traffic can be reduced is by enabling filtering of publications close to their sources. This can be achieved by flooding each subscription to all brokers in the network. However, such a naive approach significantly increases subscription dissemination traffic. Broadcasting all subscriptions over the network also increases subscription

table size at nodes which makes content matching more expensive during publication dissemination. An optimization for reducing subscription dissemination traffic is to exploit "covering" relationship between a pair of subscriptions. In this case, if a new subscription is covered by an existing subscription, it is not forwarded. Applying subscription covering in every broker in a pub/sub overlay network can greatly reduce the subscription dissemination traffic. Most of the existing pub/sub systems like SIENA [2], REBECA [6] and PADRES [4] implement *pair-wise* subscription cover checking to reduce redundancy in subscription forwarding. Efficient techniques for checking pair-wise subscription covering have also been proposed in the other works [5, 7].

A more efficient approach for reducing subscription dissemination traffic and subscription table size is to exploit subscription subsumption relationship between a new subscription and a set of existing active subscriptions. In this case, if a new subscription is completely covered (subsumed) by the set of existing subscriptions, then it is not forwarded. Clearly, since subscription covering is a special case of subscription subsumption checking for the latter results in greater efficiencies both in terms of reducing traffic between nodes and reducing size of the subscription table at nodes. Efficient subscription subsumption checking is not a trivial task. The subsumption checking problem where subscriptions can be represented as convex polyhedra has been shown to be co-NP complete [10] [1]. To the best of our knowledge the only work that considers subscription subsumption in pub/sub systems is a 'Monte Carlo type' algorithm for probabilistic subsumption checking proposed by Ouksel *et al.* [3]. However, this technique may falsely determine a subscription to be subsumed by a set of existing subscriptions when in fact it isn't. This may result in false negatives in publication dissemination meaning that a publication may not be delivered to subscribers with matching subscription. In fact not forwarding some of subscriptions changes behavior of pub/sub system from deterministic into probabilistic. While it may be acceptable in some systems, having false negatives in publication dissemination may not be tolerable in pub/sub systems that are used for dissemination of vital information such as financial information and stock market data.

Main idea: In this paper we propose a novel approach for exact subscription subsumption evaluation in d-dimensional content space. Our proposed approach is based on the following observation: *Verifying whether a set of existing subscriptions (covered region) subsumes a new subscription is exactly the same as verifying whether the new subscription intersects with the uncovered region (i.e., portion of the domain which is not covered by any existing subscription)* . We refer to the uncovered portion of the domain as the *negative space*. Then, one only needs to forward the subscriptions that overlap with the negative space to other nodes in the network. In a d-dimensional content space where subscriptions are d-dimensional rectangles, we can always represent the negative space using a set of *non-overlapping* d-dimensional rectangles. The main drawback of

[1] Note that while subsumption problem in general case is co-NP complete, if the subscriptions are d-dimensional rectangles as we show in this paper the problem can be solved in $O(n^d)$ where n is the number of existing subscriptions.

considering an exact representation of the negative space is that it might require maintaining a large number of rectangles. One can show that in the worst case this may lead to $O(n^d)$ rectangles where n is the number of active subscriptions. Besides the storage complexity, this can lead to poor performance during subsumption checking and creation of new negative rectangles as we will show later in the paper. To alleviate this problem, we propose an approximate subsumption evaluation technique that enhances performance significantly while adding a small overhead in terms of not determining some of subsumed subscription. This leads to a minor increment in subscription forwarding traffic without compromising the correctness of the pub/sub functionality, as illustrated by our experiments. The approximate approach provides knobs to control the accuracy of subsumption checking by adjusting the required space and time.

The remainder of the paper is organized as follows. In the next section we formalize the subscription subsumption problem. In Section 3 we present our approach for subsumption evaluation and subscription and unsubscription forwarding algorithms. Section 4 describes approximate subsumption evaluation along with the corresponding subscription and unsubscription algorithms. We evaluate the proposed approach in Section 5 followed by an overview of the related work in Section 6. Finally, we draw our conclusion in Section 7.

2 Problem Formulation

The architecture of pub/sub system consists of a set of broker nodes interconnected through transport-level links which form an *acyclic* overlay network. Each client is connected to one of these nodes that act as the proxies of the clients on this network. When a client issues a subscription to the node it is connected to, it in turn forwards the subscription (if need be) to its neighbors in the network. Forward propagation is carried out till every node in the network receives the subscription. When a client publishes an event, it sends the event to its designated node (broker) that forwards the content through the overlay network to the nodes that have a matching subscription. Finally, the node delivers the content to the actual client that subscribed to it. Figure 1 depicts a sample broker overlay network with 11 broker nodes and shows the clients connected to one of these nodes.

Subscription & publication routing: Subscriptions are broadcast to all nodes in the overlay network. Each node stores subscriptions in its subscription-table along with the information about which neighbor requested for which subscription. Upon receiving a publication from a neighbor or one of its clients, a broker matches it against the subscriptions in its table and forwards the publication to a neighbor if and only if it has received a matching subscription from that neighbor. Since the content matching operation is performed at every node along the path from publisher broker to subscribers, matching time has a significant effect on the speed of publication dissemination. Several efficient matching techniques have been proposed in the literature to reduce matching time [8, 9].

Redundancy minimization using pair-wise covering information To prevent unnecessary dissemination of subscriptions and reduce the size of sub-

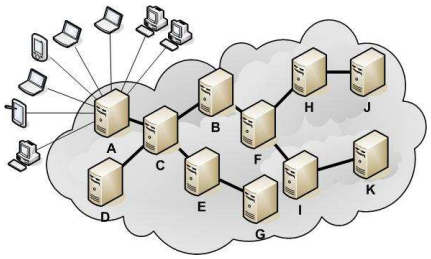

Fig. 1. A sample broker overlay network.

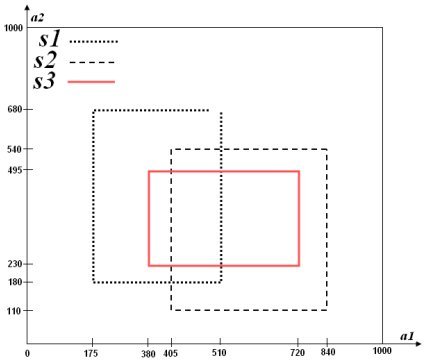

Fig. 2. s_3 is subsumed by s_1 and s_2.

scription tables, pub/sub systems use subscription covering and subsumption techniques. A subscription s_1 covers subscription s_2 if and only if all publications matching s_2 also match s_1. When a node N receives a new subscription from one of its neighbors, say subscription s_2 from $N^{'}$, such that it is covered by some previous subscription s_1 also forwarded by $N^{'}$ to N, then node N can simply drop s_2 without forwarding it to its other neighbors[2]. Since the covered subscriptions are not disseminated to all brokers, it results in lower network traffic and compact subscription tables. Note that N stores s_2 in its *passive subscription list* since it may need to forward s_2 should its covering subscriptions s_1 (and potentially others) be cancelled by an unsubscription request. If s_1 were to be unsubscribed, it is removed from N's (active) subscription table and the subscription(s) that were covered by s_1, such as s_2 in the passive subscription list are moved to its subscription table and forwarded to the neighbors along with the unsubscription request for s_1.

Optimal redundancy minimization using subsumption information: While the above discussion illustrates a simplistic approach to reducing subscription traffic, we address this problem in its most general form. In the general case, as long as the union of an existing set of subscriptions covers a new subscription entirely, the new subscription need not be forwarded. It is easy to see that pair-wise covering is a special case of the more general subsumption checking problem. Our goal in this paper is to develop an efficient approximation scheme for the subsumption checking problem. We define the problem formally below after describing the notations used in the rest of the paper.

2.1 Notation

C_d denotes the d-dimensional content space where each dimension represents an attribute. The set of independent (orthogonal) attributes along which subscrip-

[2] Note that if s_2 was forwarded by some other neighbor $N^{''}$ then, N would have to forward s_2 to $N^{'}$.

tions and publications are specified is denoted by the set $A = \{a_1, a_2, ..., a_d\}$. The domain of each attribute is also pre-specified and is assumed to be ordered[3]. One may visualize the content space as a d-dimensional rectangular region of space. We represent the lowest and highest values taken by an attribute a_i by l_i and u_i respectively. Each publication represents a point in the content space that is represented by a d-dimensional point $p = (v_1, v_2, ..., v_d)$ where v_i is the value of attribute a_i in the publication and $v_i \in [l_i, u_i]$. A subscription s is represented as a *conjunction* of d predicates where the i^{th} predicate represents an interval on the i^{th} attribute's domain. Each predicate in subscription s_j is represented as $[low_i^j, up_i^j]$ that indicates the boundaries of the subscription for i^{th} attribute[4]. Thus, a subscription corresponds to a *d-dimensional rectangle* in the content space. Subscription rectangles partition the content space into two parts, *positive space* and *negative space*.

Definition 1: For a given set of subscriptions, $S = \{s_1, s_2, .., s_n\}$, we define the *positive* space as the parts of the content space that are covered by at least one subscription rectangle. We represent the positive space as C_S^+ where $C_S^+ = \bigcup_{s_i \in S} s_i$. We also define the *negative* space as the portions of the content space that are not covered by any subscription rectangle. We represent the negative space as C_S^-. Of course, $C_S^+ \cup C_S^- = C_S$ and $C_S^+ \cap C_S^- = \varnothing$.

We say that publication p matches(satisfies) subscription s_j if and only if for each v_i in p, $v_i \in [low_i^j, up_i^j]$. Subscription s is subsumed by set of subscriptions $S = \{s_1, s_2, .., s_n\}$ if and only if for every publication p matching s, there is a $s_i \in S$ that matches p. We denote subsumption as $s \sqsubseteq S$.

2.2 Subscription subsumption problem

We define the subscription subsumption problem for pub/sub system as follows:

Definition 2: Given $S = \{s_1, s_2, .., s_n\}$ is the set of existing subscriptions, is a new subscription s subsumed by S?

The solution to the above problem is a "true" if and only if for every publication p matching s, there is a $s_i \in S$ that matches p. More succinctly, we denote this fact by "$s \sqsubseteq S$ iff $s \subset C_S^+$". A solution to the subscription subsumption problem returns a "true" or "false" when posed with an instance of the problem.

Figure 2 depicts the subscription subsumption concept in a 2-dimensional content space. There are three subscriptions in this example where $s_1 = \{[175, 510], [180, 680]\}$, $s_2 = \{[405, 840], [110, 540]\}$ and $s_3 = \{[380, 720], [230, 495]\}$. Neither subscription s_1 nor subscription s_2 completely cover subscription s_3. However, s_3 is fully covered by the union of s_1 and s_2.

[3] In case of nominal attributes, we assume some random order is assigned to the domain values. Alternatively, a partial order in the form of a taxonomy might also be utilized to assign the nominal values some numeric identifiers from an ordered domain.

[4] If no interval is specified along a dimension, the selectivity of the corresponding predicate is assumed to be equal to the whole domain of the attribute.

3 Exact Subscription Subsumption Checking

In this section we present the exact subsumption checking approach and describe the subscription and unsubscription routing algorithms. We also present a discussion on the complexity of the proposed algorithms.

As mentioned earlier, if $s \subset C_S^+$ then $s \sqsubseteq S$. However, since the positive space consists of the set of subscription rectangles which may be overlapping with each other subsumption evaluation can be more complicated. On the other hand, we can easily represent the negative space using a set of non-overlapping rectangles and more importantly this set can be maintained easily under updates (addition and deletion of subscriptions). Using negative rectangles we simply need to determine if a new subscription intersects with at least one of the negative rectangles. If this is the case, we can right away conclude that the new subscription is not subsumed by previous subscriptions.

Proposition 1: Subscription s is not subsumed by the set of existing subscriptions $S = \{s_1, s_2, .., s_n\}$ if and only if $s \cap C_S^- \neq \varnothing$.

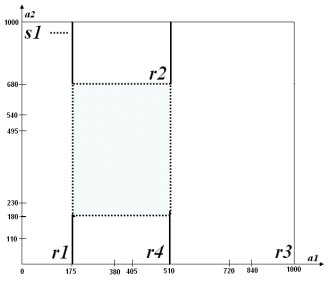

Fig. 3. Negative space partitioning

FUNCTION Subtract:
Input ← subscription s
Input ← A negative rectangle r
Input ← i: Subtraction dimension.
Output ← R_{new} set of new non-overlapping negative rectangles

1) $R_{new} = \varnothing$.
2) If $i \geq d$ then RETURN R_{new}
3) $NonCoverRange(i) = r(i) - s(i)$.
(0, 1 or 2 ranges in i^{th} dimension)
4) For each range $\delta \in NonCoverRange(i)$ {
5) Create new rectangle r_ρ where i^{th} dimension range is δ
6) and the rest of ranges are the same as r.
7) $R_{new} = R_{new} \cup \{r_\rho\}$.
8) }
9) $r_{Remaining}$: Remainig negative space where i^{th} dimension
10) range is $s(i)$ and the rest of ranges are the same as r.
11) $R_{new} = R_{new} \cup Subtract(s, r_{Remaining}, i + 1)$.
12) RETURN R_{new}

Fig. 4. Rectangle subtraction function for d-dimensional space.

Initially, when there is no subscription in the system the whole d-dimensional domain denotes the negative space, $C_S^- = C_d$ and the positive space $C_S^+ = \varnothing$. When the first subscription, s_1 is received we will have $C_S^- = C_S - \{s_1\}$ and $C_S^+ = \{s_1\}$ which means that the subscription is added to the positive space (and forwarded to the neighboring nodes) and at the same time, is subtracted from the negative space. The remainder of the negative space is not necessarily a complete rectangle, therefore, it needs to be partitioned into a set of smaller

non-overlapping rectangles. Figure 3 depicts one of the several possible ways of partitioning the negative space after adding the first subscription. We will refer to this as a *rectangle splitting* operation.

Figure 4 depicts the rectangle splitting function which returns the set of non-overlapping negative rectangles after subtraction of a subscription from a negative rectangle. The function receives the subscription and the intersecting negative rectangle along with the dimension number, (i), that the splitting should be done along with. The initial call of the function must pass dimension number zero $(i = 0)$. For the given splitting dimension number i, the function subtracts the subscription range from the given negative rectangle range in the i^{th} dimension and returns the remaining ranges as a set of non-overlapping ranges in i^{th} dimension (Line 3). Depending on the intersection form, the subtraction of the subscription range from the negative rectangle's range can split the rectangle's range into one, two or three sub ranges where only one of them intersect with the subscription's range in the i^{th} dimension. The sub ranges that are not intersecting with the subscription's range generate new non-overlapping negative rectangles with the other ranges of the negative rectagle in the other dimensions (Lines 4-8). The intersecting section of the rectangle range along with the ranges in other dimensions represent the remaing part of the negative rectangle. The function then recursively splits the remaining negative rectangle along with the other dimensions (Line 11).

In the example in figure 3, the split leads to 4 smaller rectangles, that is 3 more rectangles than before the split. Figure 5 also depicts the partitioned negative space after adding s_1, s_2 and s_3. The general case of d dimensions is captured in the proposition below.

Proposition 2: In a d-dimensional content space, a rectangle splitting operation can lead to at most $2d$ - 1 new negative rectangles.

Proof: After subtracting the intersecting region of a subscription from a negative rectangle, the remaining negative region can always be split into γ smaller rectangles, where γ is the number of surfaces of the subscription that completely or partially intersect with the negative rectangle. Since there are 2d faces of a d-dimensional rectangle, we have at most 2d-1 new rectangles generated in this process (counting one of them as the old rectangle). □

Observation: After n subscriptions have been added, let the negative space be represented as an union of some m non-overlapping rectangles, then after the $(n+1)^{th}$ subscription, the new (reduced) negative space can still be represented as an union of non-overlapping rectangles by carrying out at most $O(m)$ rectangle splitting operations.

On an average, the number of splitting operations is much smaller than $O(m)$. We represent the set of non-overlapping negative rectangles as $R(C_S^-) = \{r_1^-, r_2^-, .., r_m^-\}$. As stated in the proposition 2 above, this set can be constructed and maintained incrementally. We illustrate the subsumption checking procedure using a sequence of 3 subscriptions in Figure 5.

As it can be seen, the negative space is partitioned into four new non-overlapping rectangles. Assume we use the partitioning method depicted in fig-

ure 3 and we add subscription s_2 which is depicted in figure 2. In this case since subscription s_2 intersects with negative rectangles r_3 and r_4 the subscription is not subsumed and we need to add it into the active subscription list. We also need to subtract the subscription from the intersecting rectangles and represent the remaining parts of each negative rectangle after subtraction as non-overlapping rectangles. The new partitioning of the negative space after adding subscription s_2 is depicted in figure 5. The negative space now consists of seven non-overlapping rectangles. Finally, when we add subscription s_3, since it does not intersect between the subscription and any of the negative rectangles, we conclude that the subscription s_3 is subsumed.

Next, we formally describe the subscription and unsubscription forwarding algorithms.

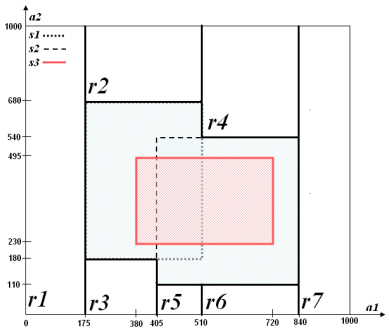

Fig. 5. Negative space partitioning after adding subscriptions s_1, s_2 and s_3.

Input ← subscription s
Input ← set of negative rectangles $R = \{r_1, r_2, .., r_n\}$
Input ← set of active subscriptions, S_A
Input ← set of passive subscriptions, S_P

1) Find $R_{intersect}$: the set of intersecting rectangles in R.
2) If $R_{intersect} = \varnothing$ then { //s is subsumed and is not forwarded.
3) $S_P = S_P \cup \{s\}$.
4) RETURN.
5) }
6) Otherwise { // s is **NOT** subsumed and is forwarded.
7) $S_A = S_A \cup \{s\}$.
8) For every $r_i \in R_{intersect}$ do {
9) $R = R - \{r_i\}$.
10) $R_i^{Remaining} = r_i - s$.
11) Partition $R_i^{Remaining}$ into non-overlapping rectangles.
12) and add them to (R_{r_i}) set.
13) $R = R \cup R_{r_i}$.
14) }
15) Forward s.
16) }

Fig. 6. Subscription forwarding Algorithm with exact subsumption cheching.

3.1 Subscription forwarding algorithm

When a new subscription is issued, we need to quickly determine if it intersects with any negative rectangle and if so, we need to identify all the rectangles that may potentially undergo splitting. Any popular multidimensional indexing structure such as R-Tree or KD-Tree [13, 14] can be used to speed up access to the rectangles. The data structure used by the algorithm maintains the following information: (i) The set of negative rectangles; (ii) The list of active subscriptions consisting of the subscriptions that have been forwarded; (iii) The list of passive subscriptions that contains the subscriptions that are subsumed

and therefore, have not been forwarded. Figure 6 represents the subscription forwarding algorithm with exact subsumption checking.

The algorithm starts with finding all the negative rectangles that intersect with the subscription. If the set of intersecting negative rectangles is empty, the subscription is subsumed and there is no need to forward it. In this case the subscription is added to the list of passive subscriptions (Lines 2-5). On the other hand, if the set of intersecting negative rectangles is not empty, this implies that the subscription is not subsumed and it must be forwarded. In this case, the algorithm first adds the subscription into the list of active subscriptions. For each of the negative rectangles in the intersecting set, it removes the negative rectangle from the data structure (Line 9) and carries out the rectangle splitting operation after determining the intersection area (Line 10-12). Then, the newly created rectangles are added to the set representing the cover of the negative space. Finally, the new subscription is forwarded to neighbor brokers except the one that it was received from.

The space and time complexity of the subscription forwarding with exact subsumption checking depends on the number of the non-overlapping negative rectangles. Assuming the number of such rectangles is $m(i)$ after i subscriptions have been forwarded, we require $O(m(i))$ space to represent these rectangles. In the worst-case, the time complexity of the $(i+1)^{th}$ subscription forwarding step is $O(d.m(i))$ where d is the dimension of the content space. This is so, because the $(i+1)^{th}$ subscription may intersect with $O(m(i))$ negative rectangles resulting in $O(d(m(i))$ splitting related operations. To represent the complexity of the algorithm based on the number of subscriptions we present an upper bound for m. We start our analysis with the following proposition about the number of partitions in one dimensional space.

Proposition 3: In a one dimensional domain n ranges result in at most $2n+1$ non-overlapping ranges.

Proof: Proof is based on induction. Each range has two bounding points, start and end. Initially there is only one range which covers all the domain. After adding the first range, the domain will contain at most three ranges (2*1+1). Assuming the maximum number of partitions resulting from n ranges is $2n+1$, we add the $(n+1)^{th}$ range. The boundaries of the new range will intersect with at most two of the existing non-overlapping ranges which results in partitioning each of these two ranges into two smaller ranges. Therefore, two new ranges is added to the number of partitions and the number of non-overlapping ranges that the domain is partitioned into will be $2n+1+2 = 2(n+1)+1$ □.

Using the above proposition, we can provide an upper bound for the number of rectangles (positive and negative) that can be generated by n rectangles.

Theorem 1: Given a set of n rectangles in d-dimensional space, an upper bound to the number of non-overlapping rectangles that can partition the space based on these rectangles is $O(n^d)$.

Proof: Each of the rectangles has a range in each dimension. Therefore, the domain of each dimension is partitioned with n ranges. The maximum possible number of non-overlapping ranges resulting from this partitioning is $2n+1$ in

each dimension (according to the Proposition 3). Using the partitioning ranges in each dimension, the d-dimensional space can be partitioned at most into $(2n+1)^d$ non-overlapping rectangles. Therefore, the maximum number of non-overlapping rectangles partitioning the space is $O(n^d)\square$.

Based on the above theorem, the following theorem provides an upper bound for the number of negative rectangles when there are n subscriptions.

Theorem 2: Given a set of n rectangles in d-dimensional space, an upper bound for the number of non-overlapping rectangles that partition the negative space is $O(n^d)$.

Proof: According to the Theorem 1 we know that the upper bound for total number of rectangles partitioning all the space resulted from n rectangles is $O(n^d)$. Each of the given positive rectangles at least include one of the partitioning rectangles. Therefore, the number of remaining rectangles for the negative space is at least $O(n^d - n)$ which is $O(n^d)$ $\square$.

Since, the number of negative rectangles can grow really fast ($O(n^d)$ for n subscriptions), the time and space complexity of the algorithm can become prohibitive. This motivates our approximation algorithm which checks this growth and which will be discussed in Section 4.

3.2 Subscription cancellation algorithm

If a subscriber wants to cancel a subscription, it makes a "unsubscription" request that is forwarded along the path that the corresponding subscription was forwarded earlier. When such a request arrives at a broker, it needs to check which subscriptions in the passive list might now get uncovered due to removal of this subscription and ensure that these queries are forwarded. Figure 7 shows the algorithm.

To cancel a subscription s the unsubscription algorithm first checks if s is in the passive set of subscriptions. If it is, then the subscription is subsumed by previously forwarded queries (active subscriptions) and he only needs to remove it from the list of passive subscriptions (Lines 1-4). Otherwise, it first removes the subscription from the list of active subscriptions (Line 5). Then, it finds $S_A^{intersect}$ which is the set of all active subscriptions that intersect with s. Unsubscribing s can result in some uncovered space that needs to be added to the negative cover. Obviously, the regions within s that are covered by other active subscriptions should not be added to the negative space. To compute these regions, the algorithm iterates over the set of intersecting active rectangles in $S_A^{intersect}$ and subtracts these regions from the s (Lines 8-19). Finally, the new negative space (if there is one) is added to the set of negative rectangles in line 20. Now, the algorithm needs to take care of the set of affected passive subscriptions which is done in lines 21-26. Here, all the passive subscriptions that intersect with s are detected and removed from the passive subscription list. Then, each of these subscriptions are evaluated for subsumption against the new set of negative rectangles. The subscriptions that are not subsumed anymore are then forwarded to neighbors along with the unsubscription request.

Input ← subscription s that must be cancelled
Input ← set of negative non-overlapping hyper rectangles $R = \{r_1, r_2, .., r_n\}$
Input ← set of active subscriptions, S_A
Input ← set of passive subscriptions, S_P

1) If $s \in S_P$ {
2) $S_P = S_P - \{s\}$.
3) RETURN.
4) }
5) $S_A = S_A - \{s\}$
6) Find $S_A^{intersect}$: the set of active subscriptions in S_A that intersect with s.
7) Set $R_{newNegative} = \{s\}$ as the set of new negative rectangles.
8) For every $s_i \in S_A^{intersect}$ do {
9) $S_A^{intersect} = S_A^{intersect} - \{s_i\}$.
10) $R_{newNegative}^{s_i} = \varnothing$.
11) For every $r_j \in R_{newNegative}$ do {
12) If $r_j \cap s_i \neq \varnothing$ {
13) $R_{newNegative} = R_{newNegative} - \{r_j\}$.
14) $R_{r_j} = r_j - s_i$ where R_{r_j} is in the form of non-overlapping rectangles.
15) $R_{newNegative}^{s_i} = R_{newNegative}^{s_i} \cup R_{r_j}$.
16) }
17) }
18) $R_{newNegative} = R_{newNegative} \cup R_{newNegative}^{s_i}$.
19) }
20) $R = R \cup R_{newNegative}$.
21) Find $S_P^{intersect}$: the set of passive subscriptions in S_P that intersect with s.
22) $S_{newActive} = \varnothing$.
23) For each $s_i \in S_P^{intersect}$ do {
24) $S_P = S_P - \{s_i\}$
25) Subscribe(s_i, R, S_A, S_P).
36) }

Fig. 7. Unsubscription algorithm for the exact case.

The unsubscription algorithm searches the negative rectangles, the active subscription and the passive subscription lists. Therefore, the time complexity of the unsubscription algorithm is $O(d.(m + |S_A| + |S_P|))$.

As we mentioned above, the number of negative rectangles can grow very quickly and make the subscription forwarding as well as unsubscription procedures very expensive. We now develop an approximation algorithms that remarkably enhances the efficiency at the cost of slightly increased traffic.

4 Approximate Subscription Subsumption Checking

In this section we introduce a heuristic that help maintain an acceptable upper bound on the number of negative rectangles. The proposed heuristic pays a small penalty in terms of falsely concluding some subscriptions as being "not subsumed" when in reality they are subsumed by the set of active subscriptions. However, such false decisions do not have any effect on the correct execution of the pub/sub system.

Subsumption checking heuristic: We restrict the number of new negative rectangles that are created after adding a new subscription to at most k, a

user specified constant. Assume $R_{intersect} = \{r_1, r_2, .., r_\alpha\}$ is the set of α negative rectangles that intersect with a new subscription s. If $\alpha > k$, we choose a $R^{Selected}_{intersect} \subseteq R_{intersect}$ such that the number of new negative rectangles created from subtracting s from rectangles in $R^{Selected}_{intersect}$ is at most k. The remaining rectangles in $R_{intersect}$ are not modified. This relaxation increases the chance of wrongly concluding that a latter query is not subsumed when it is in fact subsumed. Restricting the number of newly generated rectangles to k results in at most $O(k.n)$ negative rectangles after n active subscriptions, which is a significant improvement over the $O(n^d)$ worst-case bound.

Tuning the accuracy of the approximate approach: We can control the accuracy of subsumption detection by adjusting the value of k. By varying k one can explore the tradeoff space between the probability of false-positives in subscription dissemination and the number of negative rectangles generated.

The approximate subsumption checking can either assume a fixed value for k or a dynamic threshold which changes for each subscription based on the number of existing negative rectangles. The dynamic approach allows more fine tuning to balance the two competing factors. For instance, let the threshold be a function of the order of the subscription, say $\zeta(i) = k.i - m_{i-1}$ where $k.i$ is the maximum possible number of negative rectangles after i queries and m_{i-1} is the actual number after $i-1$ queries. This implies, if the number of negative rectangles in the system is less than the maximum expected number, for the new subscription s_i, we can add a larger number of rectangles to the system and therefore increase the accuracy of subsequent subsumption checks.

Top-k selection: Given the maximum number of new rectangles allowed (k or $\zeta(i)$), we need to select the best candidates for splitting. We propose a model based on *benefit/cost* for selecting these rectangles. We define the *benefit* of partitioning a negative rectangle with respect to a subscription as the "volume" of the intersecting region. We define the corresponding *cost* to be "the number of new negative rectangles that are created". The benefit is proportional to the increase in chance of determining subsumption while the cost is proportional to the space required for the new rectangles as well as the increase in computational cost to determine intersections for new subscriptions. Therefore, we choose the top-k negative rectangles with highest benefit to cost ratio are chosen for splitting. Figure 8 shows the approximate subsumption checking algorithm.

The approximate subsumption checking algorithm in addition to the standard inputs, also requires the set $CoveredBy_s$ to be specified. This is the set of rectangles in the content space that are covered only by subscription s. We also specify k which is the maximum number of new rectangles allowed per subscription (we only provide the algorithm for the constant k case. The version for variable ζ is similar). Similar to the exact algorithm, if the subscription does not intersect with the negative space it is subsumed (Lines 1-5). Otherwise it is added to the active subscription set. Then for each of the intersecting negative rectangles, the algorithm checks if the remaining rectangle after subtracting the subscription from it is a complete rectangle. If the remaining is a complete rectangle subtracting the subscription does not create a new negative rectan-

Input ← subscription s
Input ← $CoveredBy_s = \varnothing$: list of content space sections covered only by s
Input ← k: maximum number of new negative rectangle for s
Input ← set of negative non-overlapping hyper rectangles $R = \{r_1, r_2, .., r_n\}$
Input ← set of active subscriptions, S_A
Input ← set of passive subscriptions, S_P
1) Find $R_{intersect}$: the set of rectangles in R that intersect with s.
2) If $R_{intersect} = \varnothing$ then { // s is subsumed.
3) $S_P = S_P \cup \{s\}$.
4) RETURN .
5) }
6) Otherwise { //s is **NOT** subsumed.
7) $S_A = S_A \cup \{s\}$.
8) For every $r_i \in R_{intersect}$ do {
9) $r_i^{Remaining} = r_i - s$.
9) If $r_i^{Remaining}$ is a compelete rectangle {
10) $CoveredBy_s = CoveredBy_s \cup (r_i \cap s)$.
11) $R = R - \{r_i\}$.
12) $R = R \cup \{r_i^{Remaining}\}$.
13) }
14) Otherwise {
15) Compute the selection metric ($\frac{Benefit}{Cost}$) for r_i.
16) Add r_i to the $IntersectingSelectionList$.
17) }
18) }
19) newNegativeCount = 0
20) While (newNegativeCount ≤ k) {
21) r_i = the negative rectangle in $IntersectingSelectionList$ with the maximum selection metric value ($\frac{Benefit}{Cost}$).
22) $R = R - \{r_i\}$.
23) $r_i^{Remaining} = r_i - s$.
24) Partition $r_i^{Remaining}$ into non-overlapping rectangles.
25) and add them to (R_{r_i}) set.
26) newNegativeCount = newNegativeCount + $|R_{r_i}|$
27) $R = R \cup R_{r_i}$
28) $CoveredBy_s = CoveredBy_s \cup (r_i \cap s)$.
29) }
30) Forward s.
31) }

Fig. 8. subscription forwarding algorithm with approximate subsumption checking.

gle and the algorithm updates the negative rectangle (Lines 9-13). Otherwise, the selection metric value is computed for the intersecting negative rectangle and the rectangle is added to the $IntersectingSelectionList$ (Lines 14-17). The $IntersectingSelectionList$ contains the set of negative rectangles that intersect with subscription s (i.e., those that create extra negative rectangles after subtracting s from them). The algorithm then picks the negative rectangle with the highest benefit/cost ratio and removes it from the set of negative rectangles. Then, the algorithm subtracts the subscription from it and updates the set of negative rectangles. It also adds the intersecting section to the $CoveredBy_s$ list since this section is only covered by this subscription. The algorithm iterates till the number of extra negative rectangle reaches k (Line 19-29).

The unsubscription algorithm can be quite complicated depending on the level of accuracy. We propose an approximate unsubscription algorithm that may result in larger negative space. However, as mentioned before the larger negative

space only results in not detecting some of subsumption and does not affect the correctness of the pub/sub system. Also, assuming the rate of unsubscriptions is much lower that the rate of subscriptions such expansion of the negative space may be tolerated in many cases. Figure 8 shows the approximate unsubscription algorithm.

Input ← subscription s
Input ← $CoveredBy_s$: list of content space sections covered only by s
Input ← set of negative non-overlapping hyper rectangles $R = \{r_1, r_2, , r_n\}$
Input ← set of active subscriptions, S_A
Input ← set of passive subscriptions, S_P

1) If $s \in S_P$ {
2) $S_P = S_P - \{s\}$.
3) RETURN.
4) }
5) $S_A = S_A - \{s\}$
6) $R = R \cup CoveredBy_s$
7) Find $S_P^{intersect}$: the set of passive subscriptions in S_P that intersect with $CoveredBy_s$.
8) $S_{newActive} = \varnothing$.
9) For each $s_i \in S_P^{intersect}$ {
10) $S_P = S_P - \{s_i\}$
11) Boolean isActive = Subscribe(s_i, R, S_A, S_P).
12) If isActive=TRUE {
13) $S_{newActive} = S_{newActive} \cup \{s_i\}$.
14) }
15) $S_A = S_A \cup S_{newActive}$.
16) Forward all subscriptions in $S_{newActive}$ along with unsubscription message.
17) }

Fig. 9. Approximate unsubsumption agorithm.

Similar to the exact case the subsumed subscription is only removed from the passive subscription list (Lines 1-4). When a request for cancellation of an active subscription is received, the algorithm first removes the subscription from the active subscription list (Line 5). Then, the rectangles in $CoveredBy_s$ are added to the list of negative non-overlapping rectangles (line 6). The algorithm then detects all the passive subscriptions that intersect with the new negative rectangles in $CoveredBy_s$ (Line 7). For each passive subscription that is not subsumed anymore the algorithm forwards it along with the unsubscription request to its neighbors (Lines 9-17).

The approximate unsubscription may increase the negative space volume incrementally that may affect the subsumption evaluation accuracy for later subscriptions. In order to achieve more accurate subsumption checking, we can reconstruct the whole negative space using the set of subscriptions in the system. This can be done periodically in an offline manner, where one can recompute the negative space by making one pass over complete set of existing subscriptions. If the number of subscriptions is very large, then some pre-processing can be done to speed up this computation. We will not go into the details of such a step in

this paper. We simply note that such a re-computation process results in new S_A, S_P, R and $CoveredBy_s$ sets.

5 Experimental evaluation

In this section we evaluate the effectiveness of our proposed subscription subsumption checking approach using extensive simulations. As described in previous sections, the efficiency of our approach directly depends on the number of negative rectangles stored. We now describe the setup and the various experiments that we carried out.

Simulation setup: We perform our simulations using 10,000 subscriptions in 2, 3, 4 and 5 dimensional content space. The domain of each dimension is set as the range [0,1000] and the subscriptions are d-dimensional rectangles in this domain. We generate the subscriptions by fixing the lower end-point of the range along each dimension, say x and then selecting the size of the range randomly from the interval $[0, 1000 - x]$ which determines the upper end-point along that dimension. We sample the Zipfian distribution to pick the lower end-point of a range along each dimension.

For the approximate subsumption checking experiments, we fix the value of k to be 50. This implies that we limit the number of new negative rectangles that are added to the index for each new subscription to at most 50. Recall that, when the actual number of rectangles is more than k, we compute the best set of rectangles to split by computing the ratio of benefit to cost and choosing the ones that yield the highest values. For a negative rectangle and a subscription the benefit is measured as the volume of the intersecting region and the cost is the number of newly created negative rectangles as a result of the intersection.

Measuring advantage of subsumption checking: To compare the relative merit of subsumption checking against the pair-wise covering approach, we measure how many messages were prevented from being forwarded by employing each of these approaches. Figure 10 plots the number of redundant subscriptions detected by the subsumption checking algorithm and the pair-wise subscription covering approach. The results shown are for 2, 3 and 4 dimensional content space. As expected, the number of redundant subscriptions detected using the subsumption checking algorithm is always greater that the covering one. The graph shows another interesting fact, that the number of subsumed subscriptions is an inversely proportional to the dimension of the content space. This can be justified by observing that the probability of overlap reduces with increasing dimensionality, therefore reducing the probability of subsumption. The same trend is seen in the covering relation between subscriptions and increasing dimensionality of the space. However, even at higher dimensions, the number of queries subsumed by the union of 2 or more queries is substantially greater than number of pair-wise coverings. This result reveals the significance of exploiting subscription subsumption compared to using only pair-wise subscription covering. Figure 11 depicts the same results for the approximate subsumption checking.

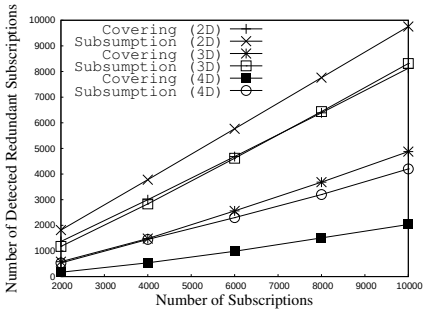

Fig. 10. Subscription Subsumption vs. Covering.

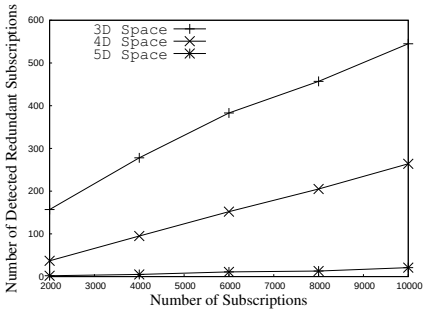

Fig. 11. Extra traffic reduction by approximate subsumption algorithm.

Negative rectangle creation rate: The number of negative rectangles directly affects the efficiency of our subsumption checking algorithm. Therefore, we measure how the number of negative rectangles vary with increasing number of subscriptions and as well as content space dimensionality. Table 1 shows the number of negative rectangles generated against number of subscriptions and number of dimensions for the exact subsumption checking algorithm. Generally, by adding more subscriptions, the number of negative rectangle grows. However, in 2-dimensional space the observed behavior is counter-intuitive, wherein the number of negative rectangles sharply grow for the first 100 subscriptions and then remains steady between 140 and 200. This behavior can be explained considering the number of covered and subsumed subscriptions for 2-dimensional scenario in our simulations. As depicted in figure 10 more than 95% of subscriptions are subsumed in this scenario and since these subscriptions do not generate new negative rectangles, the total number of negative rectangles in the 2-dimensional scenario remains significantly small. On the other hand, the number of negative rectangles significantly increases for 3 and 4 dimensional case. If we store the negative rectangles in broker's memory the algorithm may consume all of the available memory as it is shown in the table for 4-dimensional case with more than 4000 subscription and a machine with 1GB memory. The significant number of negative rectangles is a clear justification for using our approximate subsumption checking algorithm.

Table 2 represents the number of negative rectangles resulting from usage of the approximate algorithm with value of k set to 50. As we expected, the number of negative rectangles significantly drops for 4-dimensional space which results in less space requirement and faster subsumption evaluation. However, this comes at the cost of not detecting all the subsumed subscriptions.

Table 2 also depicts an unexpected trend for 2 and 3 dimensional space where using the approximate algorithm results in more negative rectangles. Since the approximate algorithm may not partition all the intersecting negative rectangles for a subscription, subsequent subscriptions that would have been subsumed will intersect with the negative space and generate more negative rectangles. This

explains the reason for having more negative rectangles in approximate case compared to the exact case. However, the number of negative rectangles remains below the threshold of $O(k.n)$.

	2000	**4000**	**6000**	**8000**	**10000**
2D	181	118	81	73	66
3D	15983	20154	19667	20756	22230
4D	364740	665000	–	–	–

Table 1. Number of negative rectangles for exact subsumption

	2000	**4000**	**6000**	**8000**	**10000**
2D	367	405	430	438	443
3D	27928	30221	31755	33071	34079
4D	13455	23414	32148	39969	47064

Table 2. Number of negative rectangles for approximate subsumption

Approximate algorithm: As mentioned earlier, the approximate algorithm substantially improves the space and time complexities of subsumption checking by restricting the number of new negative rectangles. We evaluate the advantage of using the approximate algorithm for two main decision factors. First, we investigate the effect of threshold k on the number of detected subsumptions and the number of new negative rectangles created. Then we evaluate the approximate algorithm based on the function that is used to select the negative rectangles for partitioning. The subsumed subscriptions detected in the following experiments are in addition to the covered subscriptions and can not be detected using traditional pair-wise covering techniques.

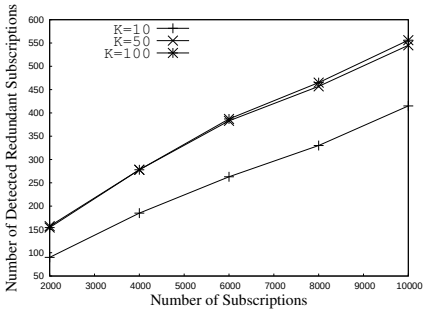

Fig. 12. Effect of k on subsumtion detection.

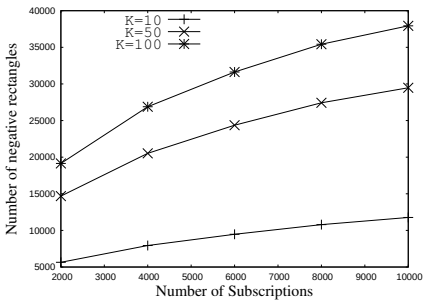

Fig. 13. Negative rectangles for different k values.

Effect of k: Figure 12 depicts the advantage of using the approximate algorithm for three different k values, 10, 50 and 100 in a 3-dimensional content space. As we expected, by increasing the value of k the number of detected subscription subsumptions also increases. Increasing the value of k from 10 to 50 results in considerable improvement in subsumption detection, however, as it can be seen in the graph there is no considerable improvement in increasing the value of k from 50 to 100. On the other hand, as shown in figure 13 the number of negative

rectangles considerably increases when we vary k from 50 to 100. Based on the results in figures 12 and 13 we can conclude that increasing k may not always result in significant improvement in subsumption detection, however, it results in increasing number of negative rectangles. Therefore, selecting proper value for k can improve the performance of the approximate algorithm significantly.

Other Selection Metric Value Function: Figures 14 and 15 show the results for three different negative rectangle selection functions. Recall, the default selection function is the ratio of the intersecting areas volume (benefit) to the number of new generated negative rectangles (cost). We consider two other functions where in the first one we only consider the benefit as the selection function and in the other one we consider the inverse of cost as the selection function. As it can be seen, using the ratio of benefit to cost results in better detection of subsumed subscriptions and fewer negative rectangles. The graphs also show that considering benefit alone performs slightly better than considering the cost. The reason is that despite considering only cost reduces the number of new negative rectangles for a new subscription but it may result in more intersections and therefore more negative rectangles for the future subscriptions. On the other hand, if we select negative rectangles based on the intersecting volume, we increase the probability of detecting subsumption for future subscriptions and therefore resulting in fewer negative rectangles.

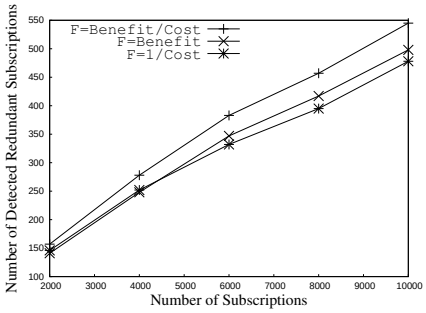

Fig. 14. Effect of negative rectangle selection function.

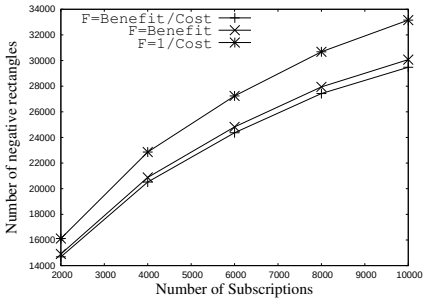

Fig. 15. Number of negative rectangles for different selection functions.

6 Related Work

Subscription covering concept in pub/sub systems was introduced in Siena event dissemination system [2]. Siena organizes subscriptions in a partially ordered set (poset) where the order is defined by covering relation. Siena only considers pairwise covering relation between subscriptions and does not exploit subsumption. REBECA is another pub/sub system that not only uses covering, but also considers subscription merging [6]. Subscription covering and merging algorithms in REBECA have linear execution time regarding to the number of subscriptions.

Li *et al.* propose a representation of subscriptions using modified binary decision diagrams (MBD) in PADRES pub/sub system [4]. They propose subscription covering, merging and content matching algorithms based on this representation. However, the MBD-based approach does not consider subscription subsumption relation among subscriptions.

Shen *et al.* propose a novel approach for approximate subscription covering detection using Space Filling Curves(SFC) [15]. In this approach a subscription $s = \{[low_1, up_1],...,[low_d, up_d]\}$ in d-dimensional space which is a d-dimensional rectangle is transformed into a point $p(s) = \{-low_1, up_1,...,-low_d, up_d\}$ in 2d-dimensional space. Considering each subscription as a point, the covering problem in d-dimensional space is converted into point dominance problem in 2d-dimensional space. The point dominance problem then is answered using space filling curve where the rectangle corresponding to a subscription is represented as set of one dimensional ranges using SFC and if the point of another subscription falls into these ranges it is covered by the subscription. However, the proposed approach can only detect covering and it is not clear how to extend it for subsumption.

To effectively detect covering subscriptions Triantafillou *et al.* propose an approach based on subscription summaries [7]. Attributes of each incoming subscription are independently merged into their corresponding summary structures. The summaries will ensure reduction in the network bandwidth required to propagate subscriptions and the storage overhead to maintain them.

Ouksel *et al.* present a Monte Carlo type probabilistic algorithm for the subsumption checking [3]. The algorithm has $O(k.m.d)$ time complexity where k is the number of subscriptions, m is the number of distinct attributes (dimensions) in subscriptions, and d is the number of tests performed to detect subsumption of a new subscription. This algorithm may result in false negatives in publication dissemination. In this algorithm it is possible that propagation of a subscription is stopped while it is not subsumed by the existing subscriptions. This may result in not delivering publications to some subscribers that may not be acceptable in applications like stock ticker.

7 Conclusion & Future Work

In this paper, we studied the problem of subsumption checking for queries in publish/subscribe (pub/sub) systems. Efficient query subsumption checking can greatly improve the performance of pub/sub systems by reducing subscription routing traffic between brokers. We developed a novel approach based on negative (uncovered) space representation which allows for fast subsumption checking. Specifically, we provided algorithms to maintain a cover of the negative space using a set of disjoint rectangles, in a dynamic environment where both subscription and unsubscriptions requests are made by clients. Further, since certain query workloads can lead to large number of covering (negative) rectangles that can adversely effect performance and storage, we developed a heuristic that checks the growth of the cover-size. Our approximate subsumption checking

algorithm reduces subscription forwarding traffic without affecting the correctness of execution. Finally, we carried out extensive simulated experiments to illustrate the advantage of our proposed approach.

As our ongoing and future work, we will be investigating some of other heuristics for efficiency of subsumption checking as well as carry out tests in a real system. We will look at the other strategies like rectangle merging to reduce the number of negative rectangles. An interesting direction is to extend this approach to other more complex query classes and different query workloads, with higher proportions of unsubscription request.

References

1. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui and Anne-Marie Kermarrec, *The many faces of publish/subscribe.* ,ACM Computing Surveys, Vol 35(2), June 2003.
2. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, Design and Evaluation of a Wide-Area Event Notification Service, *ACM Transactions on Computer Systems*, 19(3):332-383, Aug 2001.
3. Aris M. Ouksel, Oana Jurca, Ivana Podnar and Karl Aberer, Efficient Probabilistic Subsumption Checking for Content-Based Publish/Subscribe Systems, In proceedings of *Middleware 2006*, pp. 121-140.
4. Guoli Li, Shuang Hou and Hans-Arno Jacobsen, A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems Based on Modified Binary Decision Diagrams, In proceedings of *IEEE ICDCS 2005*, pp. 447-457.
5. Z. Shen and S. Tirthapura, Approximate Covering Detection among Content-Based Subscriptions Using Space Filling Curves, In proceedings of *IEEE ICDCS 2007.*
6. G. Mühl. Large-scale content-based publish/subscribe systems. *Ph.D Dissertation*, University of Darmstadt, September 2002.
7. P. Triantafillou, A. Economides, Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems , In proceedings of *ICDCS 04* , pp. 562-571.
8. F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, Filtering algorithms and implementation for very fast publish/subscribe systems, In proceedings of *ACM SIGMOD 2001*, pp. 115126.
9. A. Carzaniga and A. L. Wolf, Forwarding in a Content-Based Network, In proceedings of *ACM SIGCOMM 2003* pp. 163-174.
10. Srivastava, D., Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence.* 8 (1992), pp. 315343.
11. P. Costa, G. P. Picco, Semi-Probabilistic Content-Based Publish-Subscribe. In proceedings of *ICDCS 2005*, pp. 575-585.
12. S. Castelli, P. Costa, G. P. Picco, HyperCBR: Large-Scale Content-Based Routing in a Multidimensional Space, *IEEE INFOCOM 2008.*
13. A. Guttman, R-trees: a dynamic index structure for spatial searching. In proceedings of *ACM SIGMOD 1984*, pp. 47-57.
14. Preparata, Franco P., Shamos, Michael Ian, Computational Geometry: An Introduction. *Springer-Verlag*, 1985.
15. B. Moon, H. V. Jagadish, C. Faloutsos, J. H. Saltz, Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Trans. Knowl. Data Eng.* 13(1): pp.124-141, (2001).