Indexing for Subscription Covering in Publish-Subscribe Systems

Zhenhui Shen Srinivas Aluru Srikanta Tirthapura
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011

Abstract

Content based publish-subscribe systems are being increasingly used to deliver information in large distributed environments. Subscription covering is an effective way to reduce the complexity of content-based routing and avoid unnecessary proliferation of subscriptions throughout the system. Although covering detection has been implemented in current systems, their efficient implementation has not been systematically studied so far.

In this paper, we propose a general framework for covering detection and for maintaining covering relationships. We formalize the interaction between the above two tasks and show that a simple heuristic provides a 2-approximation algorithm for optimizing the number of covering queries on average. We also present efficient indexing schemes for covering queries resulting from range-based numeric subscriptions. We formulate this as a multidimensional range-search problem and explore the use of two well-known spatial indexes, k-d tree and space filling curve. Experimental results demonstrate that the proposed indexing schemes offer significant performance gains.

Keywords: publish subscribe, subscription covering, relation graph, multidimensional index

1 Introduction

Content based publish-subscribe systems are a flexible event-based middleware for distributed applications. In these systems, *notifications* from senders (publishers) are routed to receivers (subscribers) based on their content, rather than a fixed destination address. Receivers express their interests in receiving a certain class of notifications by submitting *subscriptions* to the system, which are predicates on the notification content. Content based publish-subscribe middleware is being used widely − for example, in stock market applications and in real-time delivery of events and statistics in sporting events [Cen].

If publish-subscribe systems are to scale to large networks, event routing must be performed in a distributed fashion. Existing systems, such as Siena [CRW01], Gryphon [ASS+99] and Rebeca [MFB02], have demonstrated how to construct a distributed network of routers to accomplish this task. The typical setup is as follows: Each router manages a set of local clients and is connected to a set of peer routers. Clients register their subscriptions at the local router. Every router forwards its local subscriptions to its adjacent routers. These subscriptions are further forwarded in the network until essentially every router knows the subscriptions registered at every other router. Whenever a notification is published by a client, the router forwards it to all neighboring routers from whom matching subscriptions were received. Thus, the forwarding of the subscriptions through the network of routers creates a reverse path for matching notifications to flow back to the interested subscriber.

A problem with the above approach is that every router needs global knowledge of all subscriptions, which results in large routing tables. This in turn leads to slower matching of notifications and greater latency for event delivery. This is a serious problem because timely delivery of events is important in most systems. To alleviate this, several groups [CRW01, MFB02] have proposed taking advantage of *covering relationships* among subscriptions to significantly reduce routing table sizes.

Definition 1 *Let $N(s)$ denote the set of all notifications that match subscription s. Let s_1 and s_2 be two arbitrary subscriptions. We say that s_1* covers *s_2, denoted by $s_1 \supseteq s_2$, iff $N(s_1) \supseteq N(s_2)$.*

Covering relationships can be exploited to increase system efficiency as follows. If a newly arrived subscription s_1 is covered by an existing subscription s_2, then s_1 is not forwarded since all notifications matching s_1 are already being received. This reduces the number of subscriptions forwarded in the system, without losing interesting notifications. Repeating this process at every intermediate router can result in a significant reduction in the number of forwarded subscriptions. The advantages are twofold: (1)

The number of subscribe and unsubscribe control messages is reduced, and (2) The sizes of routing tables decrease, leading to faster forwarding. While these advantages are well recognized and current solutions employ covering detection [CRW01, Müh01], there has been little work on developing *efficient* algorithms for covering detection in a large database of subscriptions. Current systems, such as Gryphon [ASS+99] and Siena [CRW01] can benefit from such improved indexing schemes. In this work, we present a general framework for detecting and maintaining covering relationships among subscriptions, and present specific solutions for indexing numeric data.

1.1 General Framework for Covering Detection

Indexing Subscriptions: We propose a way to index a large database of subscriptions to quickly detect covering relationships. The index is dynamic, so that subscriptions can be added and deleted from the database. The index provides two operations.

—*Subscribe* : When a new subscription s arrives, the index is used to detect if there is a current subscription covering s. If s is covered, it is not forwarded. The index can find covering subscriptions (if any) by examining only a fraction of the subscriptions in the database.

—*Unsubscribe* : When a previously issued subscription s is unsubscribed, it must be deleted from the database. As a result, some subscriptions which were previously covered by s may no longer be covered by any other subscription. Such subscriptions should be identified, and forwarded to other routers; otherwise, the clients might miss interesting notifications in the future.

To facilitate easy identification of such subscriptions, we need a separate data structure to maintain currently detected covering relations among subscriptions. We call this data structure the *relation graph*.

Our solution is thus structured in two layers:

- The upper layer is the relation graph, which stores the currently detected covering relationships among subscriptions.
- The lower layer is the actual index, which is used to detect new covering relationships when subscriptions are added to or deleted from the database.

Relation Graph: Relation graph exhibits the covering relationships among existing subscriptions. A redundant relation graph reveals more covering information, yet may be expensive to update. There are many possible organizations for the relation graph, and we investigate the tradeoffs among these. We present a formal analysis to show that a simple structure of the relation graph called the G_1 graph, where at most one covering subscription is detected (if any) for each new subscription, provides a 2-approximation algorithm for optimizing the average total number of queries to the underlying index. To the best of our knowledge, this is the first study which has investigated these tradeoffs among the possible structures of the relation graph. Our theoretical findings are corroborated by our experiments, which show that the G_1 relation graph outperforms other relation graphs over a range of system parameters.

1.2 Indexing Numeric Subscriptions

The index design is specific to the format of subscriptions, and the expressiveness of the query language. In this paper, we study an important special case where the different fields of the notifications are numeric values, and the subscriptions are range constraints on these fields. Numbers are the most fundamental data type. We believe that a good solution to this case is not only significant but also provides insight into the general problem of covering detection.

In Section 3, we show that covering detection for numeric subscriptions can be reduced to a multi-dimensional range-search problem. We then customize two promising spatial data structures, k-d tree and space filling curve, to index a large database of subscriptions.

Experiments: We have implemented covering data structures based on both indexes, and using different organizations of the relation graph. We present detailed experimental results in Section 3. These experiments clearly show that indexing can result in a significant speedup in the detection and maintenance of covering relationships in publish-subscribe systems.

1.3 Our Contributions

- A general framework for building a dynamic index for detecting, maintaining and exploiting subscription covering in publish-subscribe systems.
- A proof that a simple subscription relation graph that tracks a single covering subscription for each subscription is sufficient to derive an algorithm whose average total covering detection needs are within a factor of two of optimal.
- Solutions for numeric subscriptions based on two classical spatial data structures, *k-d trees* and *space filling curves*.
- Experimental validation of our analytical results and efficiency of the proposed indexing techniques.

1.4 Related Work

The use of subscription covering was first proposed by Carzaniga *et al.* [CRW01] to reduce the number of forwarded subscriptions. They suggest using a strict partially ordered set (poset) as the structure of a relation graph. In Section 2 we further discuss the strict poset, argue that it is expensive to maintain, and propose better alternatives.

Mühl *et al.* [MF01, MFB02, Müh01] discuss the covering and merging problems under various subscription models and formalize the problem itself. But the efficiency of indexing schemes is not explored.

Guoli Li *et al.* [LHJ05] present a work on a unified approach to routing, covering and merging in publish subscribe systems. They decompose a new subscription into attributes. The algorithm first filters out covering subscriptions for each attribute. Next an intersection of those subscriptions yields a group of subscriptions that cover the new one on all the attributes. However set intersection is a costly operation, which we think limits the efficiency of this approach.

The rest of the paper is organized as follows. In Section 2, we discuss various alternatives for relation graphs, and analyze the tradeoffs. In Section 3, we introduce two spatial indexes and the corresponding algorithms. We also present our experimental results in Section 3.

2 The Relation Graph

The relation graph maintains currently detected covering relationships among subscriptions. The relation graph is updated whenever a new subscription is inserted, or an old one is deleted.

The structure of the relation graph significantly affects the system performance. In this section, we discuss various alternatives and analyze the tradeoffs. We analytically show that a simple structure for the relation graph, where up to one covering subscription is detected for each subscription, is near optimal in the average case.

2.1 Relation Graph G^*

The relation graph $G^* = (S, E^*)$ is defined as follows. The nodes of G^* are the set of subscriptions, S. There is an edge in G^* directed from $s_1 \in S$ to $s_2 \in S$ iff $s_1 \supseteq s_2$. Clearly, we only need to forward the subscriptions in S which have no incoming edges in G^*. The rest are covered by at least one other subscription and do not need forwarding. Figure 1 shows an example of a G^* relation graph.

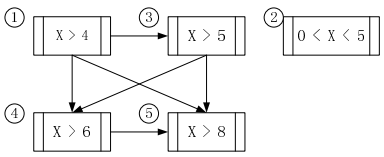

Figure 1: An example of a G^* containing 5 subscriptions. Subscriptions are numbered according to the arrival order.

The problem with G^* is that it is too redundant to be easily maintained. For example, if a new subscription s arrives, which is covered by all existing subscriptions, then edges have to be created from each existing subscription into s, and this would take at least $O(|S|)$ time. It might happen that s is unsubscribed immediately, so that all the work is wasted. As the above example illustrates, G^* can often be an overkill.

A variant of G^* is the *strict partially ordered set*, where redundant edges are removed from G^*, but every directed edge in G^* is implicitly contained in the strict poset through a directed path between the vertices. It was suggested for use in Siena [CRW01]. Since the poset contains the same amount of information as G^* does, it may also be expensive to maintain.

Thus, we look for other alternatives to G^* which are simpler to maintain in the presence of changes to S. Formally, a simpler relation graph is a directed graph G with vertex set S (the same as G^*), and edge set $E \subset E^*$. The edge set E must satisfy the following key property.

Property 1 *If $s \in S$ has a non-zero in-degree in G^*, then it should have a non-zero in-degree in G.*

This way, a new subscription which is covered by an existing one will never be forwarded.

2.2 Relation Graph G_k

We now define a family of relation graphs G_k for $k \geq 1$, which are much simpler to maintain than G^*. Let parameter $k > 0$ be a small natural number. The vertex set of G_k is S and the edge set $E(G_k) \subseteq E^*$ has the following property. For subscription s, let $C(s)$ denote $\{t \in S,\ t \neq s \mid t \supseteq s\}$, i.e., the set of all subscriptions covering s; note that $C(s)$ might be empty. If $|C(s)| \geq k$, then in G_k, there are incoming edges into s from exactly k other subscriptions which cover it. If $|C(s)| < k$, then there are incoming edges into s from all subscriptions in $C(s)$. Clearly, for all $k > 0$, G_k satisfies Property 1.

The algorithm to maintain G_k in the presence of additions and deletions to S follows. We assume

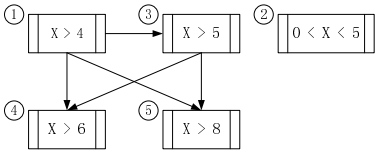

Figure 2: An instance of a G_2 relation graph.

that the underlying indexing layer gives us the function $cov_\ell(S, s)$, defined as follows. If s is covered by ℓ or more subscriptions in S, the function returns ℓ subscriptions which cover s. Otherwise, the function returns all the subscriptions which cover s (note that this set may be empty).

2.3 The G_k algorithm

Event: New subscription s arrives.

1. Create a new node for s in G_k.
2. $C \leftarrow cov_k(S, s)$.
3. In G_k, add directed edges from every subscription in C to s
4. If $|C| = 0$, then forward s to other routers, else do not forward s to other routers.

Event: Unsubscription for s arrives.

1. Delete s from S, and from G_k
2. Let T denote the out-neighbors of s in G_k which are no longer covered by any subscription after s is deleted (i.e. which have in-degree of zero).
3. For each $t \in T$,
   (a) $C_t \leftarrow cov_k(S, t)$.
   (b) Add directed edges from every node in C_t into t.
   (c) If $|C_t| = 0$, now forward t to neighboring routers (since t is no longer covered by any subscription).
4. If the subscription to s was forwarded to other routers, then forward the unsubscription also.

2.4 Cost of Maintaining G_k

For which value of k does G_k give us the most efficient algorithm? We conducted the following probabilistic analysis on k. Consider a sequence $Q = o_1, o_2, \ldots o_n$ of n subscribe and unsubscribe operations. The subscribe operations may be for arbitrary subscriptions, and each unsubscribe operation removes a *randomly chosen* existing subscription.

The major cost in the G_k algorithm is the time spent in the routine $cov_k()$. Hence, our metric for the cost of handling Q, denoted by $cost_k(Q)$, is defined as the number of times a call is made to $cov_k()$. Note that $cost_k(Q)$ is a random variable since the unsubscriptions in Q are chosen randomly from the set of existing subscriptions. We obtained the following theorem. In the theorem, the notation $E[cost_k(Q)]$ denotes the expected value of that cost. As we lack space here, we leave out the proof part and present it in our technical report [SAT05].

Theorem 1 *For every positive integer k,*

$$E[cost_1(Q)] \leq 2 \cdot cost_k(Q)$$

3 Multidimensional Indexing

With regard to numeric subscriptions, we model each of them as a hyper-rectangular region in a space of appropriate dimensionality. For two subscriptions s_1, s_2, if $s_1 \supseteq s_2$, the rectangle corresponding to s_1 completely contains the rectangle corresponding to s_2. Thus, building an index for these subscriptions involves the design of a data structure which can efficiently answer rectangle containment queries on a database of rectangles.

We transform the problem of rectangle containment to multidimensional range search problem and adapt two classical spatial data structures, *k-d trees* and *space filling curve*, to index these rectangles. The two structures are tailored to better fit the specific problem. Due to the space constraint, we move the details to our technical report [SAT05].

We also run experiments to evaluate our overall design. The objectives of our experiments are two-fold: First, we want to determine the appropriate form of the relation graph. Second, we want to evaluate and compare the performance of spatial indexes.

3.1 Experimental Settings

1. A subscription consists of numeric attributes. For instance, a subscription about used textbooks might be $S = (\text{price} \in [0, 50], \text{year} \in [1999, 2004])$.
2. For simplicity, we require all attribute values are intervals within the range $[0, 2^m - 1]$ for a constant m. If an attribute takes real values in a different range, we can always scale the range appropriately.
3. The dimension of the space varies from 6 to 16 (note that this corresponds to subscriptions having between 3 and 8 attributes).
4. The input is a sequence of mixed operations; each operation is either a subscribe or an unsubscribe. Each subscribe adds a rectangle chosen uniformly at random from the multidimensional space. Each unsub-

scribe removes a rectangle randomly chosen from the currently existing rectangles.

5. We conducted experiments with various input sizes, ranging from $10,000$ to $100,000$ operations. Each operation is a subscribe with probability $(1-p)$, and an unsubscribe with probability p. Clearly, $p < 0.5$ since there can never be more unsubscribes than subscribes. We vary the value of p from 0 to 0.4 with each increment being 0.05

In summary, the following parameters can be tuned for each experiment.

- Input size, n
- Percentage of unsubscribes, p
- Dimension of points, d
- Reporting mode, k

Performance Metric: Our performance metric is the total time taken by the covering data structure to process all the subscribe and unsubscribe commands.

3.2 The Ideal Structure of Relation Graph

In Section 2, we defined a family of relation graphs G_k. The subscript k, which we call the "reporting mode", determines the maximum in-degree of any node in the graph. *Our experiments show that G_1 consistently yields the best performance among all G_k, no matter which index is used.* This is consistent with our theoretical predictions.

Due to space constraints, we only show results obtained by using the k-d tree index. Using the space-filling curve yielded similar results. We choose one scenario, where $n = 80,000$ and $d = 10$.

In this experiment, we change p from 0 to 0.4 at an increment of 0.05. We show the results for $k = 1, 2$ and 4 in Figure 3.

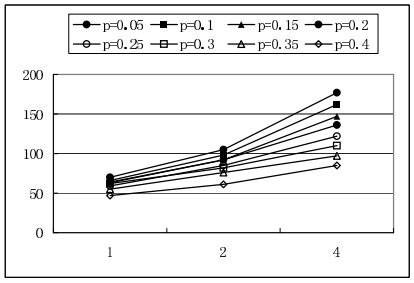

Figure 3: Total Runtime (seconds) vs. Reporting Mode.$n = 80,000$, $d = 10$.

From Figure 3, it is clear that G_1 consistently yields the shortest runtime. It is understood that the cost of subscribe operations increases with k, but the experiments also revealed that the cost of unsubscribe operations also increased with k, although the actual number of nodes which are uncovered due to unsubscribes falls drastically with k.

The above observation can be explained as follows. (1) A large k increases the average node's degree in the relation graph. When a node is deleted, we have to examine more children,and this contributes to the complexity of unsubscribes. (2) Any node that lost all its parents due to an unsubscribe needs to be re-inserted into the relation graph. A large k requires us to find more parent nodes during the re-insertion, and makes each re-insertion more expensive.

3.3 Comparison of the Two Indexes

We now compare the performance of the two spatial indexes, the k-d tree and the space filling curve. To provide a reference point, we also introduce the *naive* algorithm which uses no index. The aim of the comparison is to evaluate the scalability of our spatial indexes with respect to the input size and the dimension of the space. Our results indicate that indexing gives an order of magnitude improvement in terms of runtime over the naive algorithm.

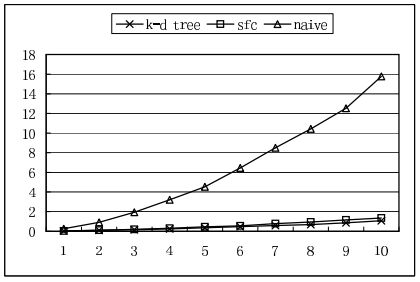

Figure 4: Total Runtime (seconds) vs. Input Size. $d = 12$, $p = 0.05$, $k = 1$ (heavy load). x-unit is 10^4. y-unit is 10^2.

We select two contrasting scenarios to illustrate the performance gain. The results are plotted in Figure 4 and 5, respectively.

- The first is a *heavy load* scenario, where $d = 12$, $p = 0.05$, and $k = 1$. A heavy load scenario is characterized by a high dimension and low unsubscribe ratio.
- The second is a *light load* scenario, where $d = 4$, $p = 0.40$, and $k = 1$. A light load scenario features a low dimension, but high unsubscribe ratio.

It is obvious that as both the dimension and input size increase, the gain due to indexing becomes

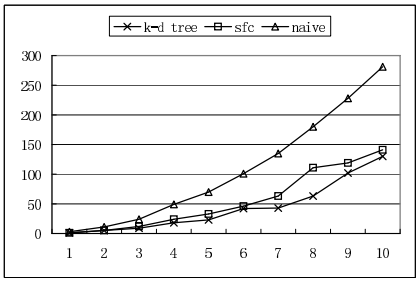

Figure 5: Total Runtime (seconds) vs. Input Size. $d = 4$, $p = 0.40$, $k = 1$ (light load). x-unit is 10^4.

more significant. Even under the light load scenario the spatial indexes run approximately twice as fast as the naive approach. In general we expect that the naive approach would perform better in lower dimensional spaces, since there is a greater likelihood of finding a dominating point through a sequential scan of the data.

4 Conclusion

In this paper, we present a work on exploiting subscription covering to scale the publish-subscribe systems. We set up a general two-layered framework. We provide the first systematic study on the structure of a relation graph and its interaction with the underlying index. We propose a near-optimal design of the graph. Furthermore the graph organization does not depend on a subscription's format and language. Therefore it can be used as a basic building block for detecting and maintaining covering relationship among subscriptions. We also design efficient indexes to detect subscription covering. We achieve good results for numeric subscriptions. The future work will be to design multidimensional indexes for subscriptions having string attributes.

References

[ASS+99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 1999.

[Cen] IBM T.J. Watson Research Center. Gryphon: Publish/subscribe over public networks. White paper.

[Cha90a] B. Chazelle. Lower bounds for orthogonal range searching:ii.the arithmetic model. *Journal of the ACM*, 37:439–463, Jul 1990.

[Cha90b] B. Chazelle. Lower bounds for orthogonal range searching:i.the reporting case. *Journal of the ACM*, 37:200–212, Apr 1990.

[CRW01] 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, 2001.

[Fre81] M.L. Fredman. A lower bound on the complexity of orthogonal range queries. *Journal of the ACM*, 28:696–705, Oct 1981.

[LHJ05] 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 *International Conference on Distributed Computing Systems(ICDCS'05)*, 2005.

[MF01] G. Mühl and L. Fiege. Supporting covering and merging in content-based publish/subscribe systems: Beyond name/value pairs. *IEEE Distributed Systems Online (DSOnline)*, 2(7), 2001.

[MFB02] G. Mühl, L. Fiege, and A.P. Buchmann. Filter similarities in content-based publish/subscribe systems. In *International Conference on Architecture of Computing Systems (ARCS)*, volume 2299 of *Lecture Notes in Computer Science*, pages 224–238, 2002.

[Müh01] G. Mühl. Generic constraints for content-based publish/subscribe systems. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, volume 2172 of *Lecture Notes in Computer Science*, pages 211–225, Sep 2001.

[SAT05] Zhenhui Shen, Srinivas Aluru, and Srikanta Tirthapura. Indexing for subscription covering in publish-subscribe systems. Technical Report TR-2005-07-2, 2005. http://archives.ece.iastate.edu/archive/00000163/.