Indexing for Subscription Covering in Publish-Subscribe Systems

Zhenhui Shen Srikanta Tirthapura Srinivas Aluru
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 currently exploited 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 well-known spatial indexing schemes based on k-d trees and space filling curves. Experimental results demonstrate that the proposed indexing schemes offer significant performance gains.

I. 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 in turn 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 [IBM].

If publish-subscribe systems are to scale to large networks, event routing must be performed in a distributed fashion. Many systems have been designed that demonstrate how to construct a distributed network of routers to accomplish this task [CRW01], [ASS+99], [MFB02]. The typical setup is as follows: Each router manages a set of local clients and is connected to a set of peer routers. Clients express interest in receiving events by registering their subscriptions at the attached router. Every router forwards all its registered subscriptions to its neighboring 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, multiple 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 as all notifications matching s_1 are already being received. This reduces the number of subscriptions forwarded, without losing interesting notifications. Repeating this process at every intermediate router to which a subscription is forwarded can potentially reduce the number of forwarded subscriptions significantly. The advantages are twofold: (1) The number of subscription and unsubscription control messages is reduced, and (2) The sizes of routing tables decrease. 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. We address this problem by proposing suitable indexing schemes.

A. Indexing for Covering Detection

In this paper, We propose a way to index a large database of subscriptions to quickly detect covering relationships. The index is dynamic, i.e., it supports

addition and deletion of subscriptions, and provides two operations.

Subscribe: When a new subscription s arrives, the indexing scheme is used to find if there is a current subscription covering s. If s is covered, it is not forwarded. Our 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 subscriptions. These should be detected, and forwarded to other routers. Otherwise, the clients might miss interested notifications in the future. To facilitate this task, we need a separate data structure to maintain the currently detected covering relations among subscriptions. We name this data structure the *relation graph*.

Our solution is 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 or deleted from the database.

Relation Graph: 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 for the graph, where 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 our knowledge, such an analysis has not been carried out so far. These theoretical findings are confirmed by experiments.

Indexing Numeric Subscriptions: The design of the lower layer (the index) 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 themselves are rectangular constraints on these numeric fields. Numbers are the most basic data type. In many applications, it is natural to describe data attributes by using numbers (examples are demographic census, stock prices). We believe that a good solution to the case of numeric attributes is significant in practice and also provides insight into the general problem.

However, a general solution to this problem is still far away, since techniques for indexing the two main types of data, numbers and strings, are very different. A full solution needs to find indexing schemes for each of these data types, and more importantly, integrate them into a single index. From this perspective, we view our indexing of numeric subscriptions as an important first step.

We use techniques from spatial data structures to design indexes for numeric subscriptions. Each subscription is a hyper-rectangular region in a high-dimensional space. 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 (hyper-)rectangle containment queries on a database of rectangles.

We use the well known point-dominance formulation to transform rectangle containment queries into range searching and investigate two promising data structures for range searching.

- The first index is based on the k-d tree, which is a multidimensional search tree that is formed by the recursive decomposition of the d-dimensional search space using $(d-1)$-dimensional hyperplanes.
- The second index is based on space filling curves, which are proximity preserving mappings of points from d dimensions to one dimension.

We implemented both the indexes, and compared their relative performance. While the k-d tree based index outperforms the space filling curve based index, the run time of both the indexes is an order of magnitude better than the run time in the absence of indexing.

B. Our Contributions

We make the following 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 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 from computational geometry, *k-d trees* and *space filling curves*.
- Experimental validation of our analytical results and efficiency of the proposed indexing techniques.

C. Related Work

The use of covering to reduce the number of forwarded subscriptions was first proposed Carzaniga *et al.* [CRW01]. They suggest using a strict partially ordered set (poset) to organize covering relationships among subscriptions. In Section III we will further discuss the strict poset and compare this with other

models of relation graph that we propose. The above solution is applicable to a general class of subscriptions. We further explore the case when subscriptions are multidimensional ranges, and provide efficient indexing schemes using spatial data structures.

Mühl *et al.* published a series of papers [MF01], [MFB02], [Müh01] on the subscription covering problem. They give a formal definition of the problem and discussed the covering and merging problems under various subscription models. However, their solution implicitly involves computation of set intersections, which is expensive.

The event-matching problem is a dual problem to the subscription covering problem. In event-matching, it is required to compare events against a database of subscriptions to find matching subscriptions. There has been much work on event-matching, including [CW03], [ASS+99], [FJL+01], [CCC+01].

We discuss related work on spatial data structures for indexing in Section IV.

The rest of the paper is organized as follows. In Section II we give an overview of our solution. In Section III, we discuss various alternatives for relation graphs, and analyze the tradeoffs. In Section IV, we introduce two indexes and the corresponding algorithms. We present our experimental results in Section V.

II. The Covering Data Structure

The covering data structure is organized in two layers. The upper layer is the relation graph, and the lower layer is the indexing data structure.

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

The underlying index provides one operation to the relation graph. If S is the current set of subscriptions, and s is a new subscription, the index provides a function $cov_k(S, s)$ which returns up to k subscriptions in S covering s. We will not require the index to return us *all* the subscriptions covering s, since this might be an inherently expensive operation. The index is also dynamic, and supports fast addition and deletion of subscriptions.

III. The Relation Graph

We now describe the relation graph in greater detail. Since the structure of the relation graph significantly affects the system performance, 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.

A. Relation Graph G^*

Let S denote the set of current subscriptions at the router. The 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 to be forwarded (unless the covering subscriptions arrived after the covered subscription was forwarded). Figure 1 contains an example of a G^* relation graph.

The problem with G^* is that it might have too many edges and might be very expensive to maintain as subscriptions are added and deleted. For example, if a new subscription s arrives which covers all other subscriptions, then edges have to be created from s to each existing subscription, and this would take time $O(|S|)$ (this is in addition to the time taken by the index to detect all these covering relationships, which itself is expensive). It might happen that s is unsubscribed immediately, so that all this work is wasted. As the above example illustrates, G^* can often be an overkill.

We look for alternatives to G^* which are simpler to maintain in the presence of changes to S. 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, since it will have a non-zero in-degree in G^*, and hence, a non-zero in-degree in G.

B. 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 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

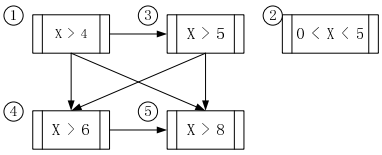

Fig. 1. An example of a G^* containing 5 subscriptions. Subscriptions are numbered according to the arrival order. The remaining relation graphs use the same set of subscriptions and arrival order.

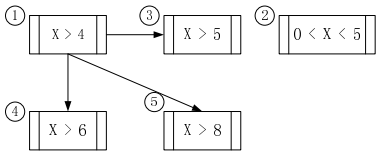

Fig. 2. The G_1 relation graph

the subscriptions which cover s (note that this set may be empty). We will refer to the algorithm for solving the covering problem using G_k as "the G_k algorithm".

C. 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.

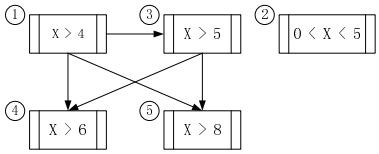

Fig. 3. The G_2 relation graph

D. Cost of Maintaining G_k

For which value of k does G_k give us the most efficient algorithm? Assuming that k is a small integer, the cost of handling a subscription is dominated by the cost of $cov_k()$. Since it is always more expensive to find more covering points, the cost of $cov_k()$ strictly increases with k. Thus, the cost of adding new subscriptions to G_k clearly increases with k.

However, the cost of handling an unsubscribe might decrease. If k increases, it is possible that an unsubscribe will cost lesser, for the following reason. If k increases, the average in-degree of a node in G_k increases, making it less likely that a subscription loses all its in-neighbors once another subscription is deleted from G_k. Note that the number of out-neighbors of a node in G_k also increases with k, so that the out-degree of the subscription being deleted is also higher, on average.

The question is whether the increase in the cost of handling subscribes can be offset by the decreased cost of handling unsubscribes. We now present evidence that choosing $k = 1$ gives us a total cost which is close to the optimal, on the average. While this does not mean that $k = 1$ is always the best choice, it implies that it is usually a good choice. This is also supported by our experimental results, which we present in Section V.

1) Analysis of G_1: We consider a probabilistic model of the pattern of unsubscribes for our analysis. 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 unsubscribes to a *randomly chosen* subscription that is currently subscribed for. Let n_s denote the number of subscribe operations in Q and n_u the number of unsubscribe operations, so that $n_s + n_u = n$. We do not make any assumptions on the formats of the subscriptions, hence our analysis is general.

The major cost in the algorithms for handling subscribes and unsubscribes 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.

Let q_s and q_u be the number of calls to $cov_k()$ made due to the subscribe and unsubscribe operations respectively.

$$cost_k(Q) = q_s + q_u \tag{1}$$

Since only one call to $cov_k()$ is made by any subscribe operation,

$$q_s = n_s \tag{2}$$

Lemma 1: If the relation graph used is G_1, then

$$E[q_u] \leq n_u$$

Proof: We label the unsubscribe operations $1, 2, \ldots n_u$. For $i = 1, 2, \ldots n_u$, let random variable X_i denote the number of calls to $cov_1()$ due to the ith unsubscription.

$$q_u = \sum_{i=1}^{i=n_u} X_i \tag{3}$$

By linearity of expectation, we have

$$E[q_u] = \sum_{i=1}^{i=n_u} E[X_i] \tag{4}$$

Suppose the ith unsubscribe operation was for subscription s. Then, X_i is equal to the out degree of s in G_1 before the deletion of s. Since s is chosen randomly from the current set of subscriptions in G_1, $E[X_i]$ is equal to the expected out-degree of a vertex in G_1.

In any directed graph, the sum of the out-degrees of all the nodes equals the sum of the in-degrees. In graph G_1, the in-degree of every node is no more than 1, so that the sum of in-degrees is no more than $|S|$. Thus, the expected out-degree of a random node in G_1 is no more than 1. We have $E[X_i] \leq 1$. From Equation 4, the lemma follows. ∎

Theorem 1: For every positive integer k,

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

Proof: From Equation 1, we know $cost_1(Q) = q_s + q_u$. From Lemma 1 and Equation 2, $E[cost_1(Q)] = E[q_s] + E[q_u] \leq n_s + n_u$. Since the number of unsubscribe operations can never be greater than the number of subscribe operations, $n_u \leq n_s$. Thus, we have

$$E[cost_1(Q)] \leq 2n_s$$

However, for any k, the algorithm G_k has to make at least n_s calls to $cov_k()$, even in the unlikely scenario that further calls to $cov_k()$ are never made for unsubscribe operations. Suppose q^* denotes the minimum number of queries made by any G_k, $k \geq 1$.

$$q^* \geq n_s$$

Thus, we have $E[q] \leq 2q^*$, which proves the theorem. ∎

The above theorem tells us that the expected number of calls to $cov_k()$ made by the G_1 algorithm is always within a factor of two from the optimal number. In addition, we know that the cost of $cov_k()$ increases with k, so that the covering queries made by G_1 are cheaper than queries made by any other G_k. This presents strong evidence that the G_1 algorithm is one of the best choices among all $G_k, k \geq 1$.

Strict Poset Relation Graph: A variant of G^* is the *strict poset* (see Figure 4), 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. This form of the relation graph was suggested for use in [CRW01]. However, it can be seen that the strict poset is also very expensive to maintain in the presence of addition and deletion of subscriptions. Since the strict poset implicitly contains every covering relationship present between subscriptions, the example described above for G^* also applies to the strict poset.

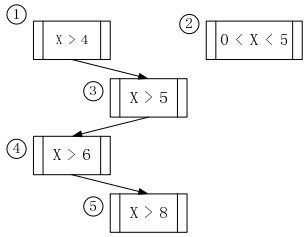

Fig. 4. A *strict poset* relation graph

IV. Indexing Multidimensional Ranges

We now consider the case when the notifications consist of numerical attributes, and the subscriptions are range constraints on the different attributes. We show how to build an index for such subscriptions to be able to efficiently answer covering queries.

A notification is a point in d-dimensional space, and a subscription is a rectangle in the same d-dimensional space. A rectangle in d-dimensional space $([\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_d, r_d])$ is the set of all points x such that for $i = 1 \ldots d$, the ith coordinate of x lies in the range $[\ell_i, r_i]$.

For example, a notification $(\text{region} = 1, \text{temperature} = 59, \text{humidity} = 87)$ can be represented as a point in 3-dimensional space $(1, 59, 87)$. The subscription $(\text{region} = 1, 0 \leq \text{temperature} \leq 30, 90 \leq \text{humidity} \leq 100)$ would be the following rectangle in 3-dimensional space: $([1, 1], [0, 30], [90, 100])$.

Definition 2: Rectangle r is said to cover rectangle r' in d-dimensional space, denoted by $r \supseteq r'$, iff every point within r' is also within r.

Since our subscriptions are rectangles, the subscription covering problem can be restated as the following geometric problem.

Rectangle Containment Problem: Process a database R of d-dimensional rectangles to support the following operations. Parameter k is a small natural number.

- Given a query rectangle r, report k elements of the set $\{s \in R \mid s \supseteq r\}$. If the set has less than k elements, report all of them.

- Add a new element r into S
- Delete an element r from S.

A. Reduction to Range Searching

As observed in [PS85] (Chapter 8), the above rectangle containment problem can be reduced to the point dominance problem, which itself is a special case of the range-searching problem.

Definition 3: Given two d-dimensional points, $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$, x dominates y iff for each $i = 1, \ldots, d$, it is true that $x_i \geq y_i$.

Given a d-dimensional rectangle $s = ([\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_d, r_d])$, we transform it into a $2d$-dimensional point $p(s) = (-\ell_1, r_1, -\ell_2, r_2, \ldots, -\ell_d, r_d)$. It is easy to verify the following fact.

Fact 1: If s and r are two rectangles in d-dimensions, then $s \supseteq r$ iff $p(s)$ dominates $p(r)$.

With this transformation, we can reduce the rectangle containment problem to the point dominance problem, which itself is a special case of the range-searching problem. We now focus on this problem.

Process a set of $2d$-dimensional points P to answer the following query. Given a point $x = (x_1, x_2, \ldots, x_{2d})$, report k points in the range $([x_1, \infty], [x_2 \infty], \ldots, [x_{2d}, \infty])$. If there are less than k points in the range, then report all of them. Note that in practice, ∞ can be replaced by finite boundaries meaningful to the application at hand.

B. Multidimensional Range Searching

We present an overview of the theoretical and practical work on multidimensional range-searching. We then focus on two techniques for building efficient indexes for multidimensional range searching, the k-d tree data structure and the space-filling curve.

Multidimensional range searching has been widely explored during last two decades. From a theoretical point of view, range searching is now almost completely solved, in that there exist very good lower bounds and almost matching upper bounds. Fredman [Fre81] showed that in a semigroup arithmetic model [AE99] in which the points can be inserted and deleted dynamically, a mixed sequence of n insertions, deletions and queries requires $\Omega(n \log^d n)$ time in d-dimensions for any range-searching algorithm. Chazelle [Cha90b], [Cha90a] shows that any data structure on a pointer machine that answers a d-dimensional range searching query in $O(\text{polylog}(n) + k)$ time must have size $\Omega(n(\log n / \log\log n)^{(d-1)})$ (where k is the size of the output). These lower bounds reveal the inherent complexity of range searching in a high dimensional space. There exist algorithms for range-searching based on *range trees* [Ben80] which achieve a query time of $O(\log^{d-1} n + k)$ where k is the size of the output using space $O(n \log^{d-1} n)$. However, due to the illustrated polylogarithmic overhead in either space or time, these theoretically worst-case efficient algorithms are not used in practice.

Next we examine some practical data structures. Even though these do not guarantee worst case bounds, they offer better results in typical cases. Since good sorting and searching algorithms are known for one dimensional space, one possibility is to reduce multidimensional range-searching problem to a one-dimensional search. The space-filling curve [OM84] provides such a mapping, and can be used to build an index. Another approach is to recursively partition the space, and to use the tree induced by this partition to organize the data. Range searching is accomplished through traversing the nodes of this tree. The k-d tree [Ben75] and the quadtree are based on this idea. Other range-searching data structures based on tree-like partitioning of the space include the R-tree [Gut84] and its variants, but these have been specifically designed for data structures in secondary storage. Our data structures have to be stored in main memory since the latency of the disk access is too high when compared to the router speed.

We decided to implement the k-d tree and the space-filling curves for the following reasons.

- Both of them are designed for range-searching in main memory, and have been used successfully in other applications.
- They have low update costs, which allows the router to cope with frequent subscribes and unsubscribes.
- They are both simple to understand and easy to implement, and have a linear space requirement.

C. k-d tree

Let n denote the number of points. We number the dimensions $1, 2, \ldots, d$. If we find the median coordinate of all the points along dimension 1, we can partition the points into two approximately equal sized sets - one set containing all the points whose coordinates along dimension 1 are less than or equal to this median and a second set containing all the points whose coordinates along dimension 1 are greater than the median. Each of these two sets is again partitioned using the same process except that the median of each set along dimension 2 is used. The partitioning is continued using dimensions $1, 2, \ldots, d$ in that order until each resulting set contains one point. If all the dimensions are exhausted before completely partitioning the data, we "wrap around" and reuse the dimensions again starting with 1. An example of such a partition for a two-dimensional data set containing 8 points is shown in Figure 1.

The process of organizing spatial data in the above manner is naturally represented by a binary tree. The

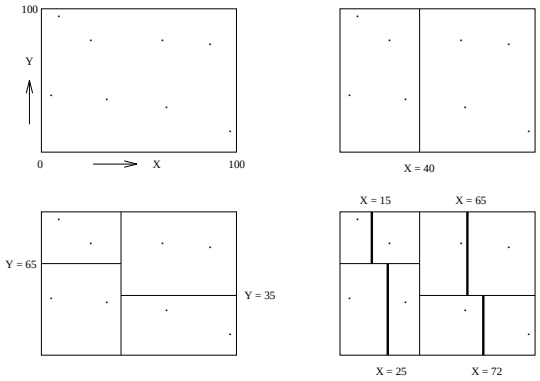

Fig. 5. Recursive partitioning of a two-dimensional data set.

root of the tree corresponds to the set containing all the n points. Each internal node corresponds to a set of points and the two subsets obtained by partitioning this set are represented by its children. The same dimension is used for partitioning the set of each internal node at the same level of the binary tree. For all nodes at level i of the tree (defining the root to be at level 0), dimension $(i \ mod \ d) + 1$ is used. The resulting tree is called a multidimensional binary search tree (abbreviated k-d tree), first introduced by Bentley [Ben75].

Our implementation is based on the adaptive k-d tree proposed by Bentley and Friedman [BF79]. All data points are stored at the leaves, and internal nodes just store the partitioning discriminator. Further, each leaf is a "bucket" holding many data points. There are two parameters associated with a bucket: the overflow limit and the underflow limit. During the evolution of the tree if a bucket's size exceeds the overflow limit, the bucket is split and new children are formed. If the bucket's size drops below the underflow limit, we merge the bucket with its sibling, provided that the sibling has not yet been split and the resulting capacity of the merged bucket is below the overflow limit.

The general idea behind range searching using k-d trees is to identify internal nodes that correspond to regions overlapping the query region. If the region represented by an internal node is completely contained in the query region, all points in its subtree are in the query region. If the overlap is proper, resolution is necessary by traversing lower levels. We modify this standard algorithm to always start the search at the rightmost leaf. The rationale is that the rightmost leaf corresponds to the top right region of the universe, and points lying in that region have a higher chance of dominating the query points. If enough covering points are not found, we backtrack to higher levels and consider their subtrees. Due to space constraints, further details are omitted.

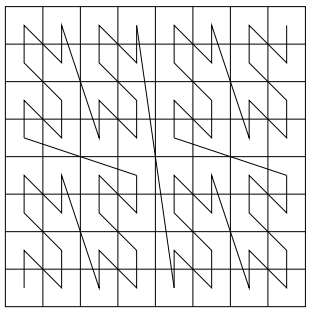

Fig. 6. Z-curve for 8×8 resolution.

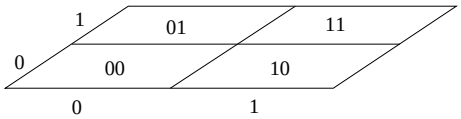

Fig. 7. Generating keys from coordinates of cells.

D. Space Filling Curve

Space filling curves are proximity preserving mappings from a uniform multidimensional space decomposition to one dimension. The path implied in the multidimensional space by the linear ordering forms a non-intersecting curve. For our work, we use the Morton ordering, also known as the Z-space filling curve [Mor66]. The Z-curve for a 8×8 two dimensional decomposition is shown in Figure 6.

Space-filling curves can be used to index multidimensional point sets and their use in indexing databases and supporting range queries has been first explored two decades ago [OM84]. Our range search algorithm is designed to specifically target the special class of range queries we encounter. We inductively define a cell as follows: A d-dimensional cube containing all the points is chosen as the root cell. For each cell, the 2^d equal-sized cubic subregions obtained by bisecting along each dimension of it are also cells. For convenience of presentation, assume a resolution that is sufficient to separate the points into different cells (we can and do relax this requirement in our implementation). It is well known that the position of a cell along the Z-curve can be obtained by treating the coordinates of the cell with respect to the cell space as binary numbers, and interleaving the bits (see Figure 7).

For example, the position of a cell with coordinates $(3, 5) = (011, 101)$ in a 8×8 resolution cell space is $011011 = 27$. This is called the key of the cell. To simultaneously use the same numbering system for cells at different resolutions, we prefix each interleaved bit string by a 1 bit and use this as the key. This allows us to distinguish between, for example, cell "00" at 2×2 resolution from cell "0000" at 4×4 resolution as the

corresponding keys will now be "100" and "10000". This notation allows us to seamlessly use integer keys for describing points and cells at various resolutions. This is advantageous as efficient bit manipulations can be used to carry out many operations required by the range search algorithm.

The main idea behind our range search algorithm is as follows: All points that lie in a multidimensional cell will be present as a consecutive segment of the curve. This segment can be quickly identified using a binary search for its extremal points in the Z-curve order. Given a range query, one can decompose the range into a disjoint union of maximal cells, locate the corresponding segments of the space-filling curve, and extract points within these segments. We designed a range search algorithm that exploits the fact that (1) each of our ranges extends to the rightmost boundary, and (2) only a few points within the range search region are sought. Hence, range search is biased in favor of larger cells contained within the query region and special heuristics are designed to examine certain extremal points which are likely contained in the query region. In addition, the one dimensional sorted order has to be maintained in the presence of insertions and deletions. We used the AVL tree data structure for this purpose. The full details of our algorithm are omitted for brevity.

V. Experimental Results

We implemented the covering data structure and indexing schemes using both the k-d trees and space filling curves. 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 the space-filling curve index, and the k-d tree index.

A. Experimental Settings

- A subscription consists of only numeric attributes. For instance, a subscription about used textbooks might be $S = (\text{price} \in [0, 50], \text{publication date} \in [1999, 2004], \text{shipping time} \in [2, 6])$.
- For simplicity, we require all attributes fall in the range $[0, 2^d - 1]$ for $d = 1$. This will help simplify the recursive partitioning of the space the space-filling curve. If an attribute takes real values in a different range, we can always scale the range appropriately.
- The dimension of our points range from 6 to 16 (note that this corresponds to subscriptions having between 3 and 8 attributes).
- The input to the covering data structure is a sequence of interleaved subscribes and unsubscribes. Each subscribe is a randomly chosen rectangle from the multidimensional space. By a randomly chosen rectangle, we mean that each edge of the (hyper)rectangle is a randomly chosen segment in the range $[0, 2^d - 1]$. Each unsubscribe operation removes a rectangle randomly chosen from the currently existing rectangles.
  We generate a subscribe command with a probability of $(1 - p)$, and an unsubscribe command at a chance of p. Clearly, $p < 0.5$ since there can never be more unsubscribes than subscribes. We vary the value of p from 0.05 to 0.4.

The following parameters can be tuned for each experiment.

- Total number of incoming commands
- The percentage of unsubscribe commands, p. The above two parameters determine average number of subscriptions in the database, i.e. the system load.
- The dimension of the data, d. The covering queries get harder as d increases.

Performance Metric: Our main performance metric is the total time taken by the covering data structure to process all the subscribe and unsubscribe commands. In addition to the above, we collect two statistics to help us interpret the data.

- *Nodes Visited:* One thing common to both k-d trees and space filling curves is that we first try to narrow the universe down to a smaller subspace before doing a linear search. These subspaces are indexed either in an explicit way, as in the k-d tree or implicitly, as in the space filling curve. The cost of both space filling curves and the k-d trees increase with the number of such regions visited, which is measured in the statistic "nodes visited"..
- *Point Comparisons:* When the search path ends at a leaf node, we simply conduct a linear search through the list of all points stored at the leaf. The number of comparisons that are made here between many high-dimensional points is measured by the statistic "Point Comparisons". Note that a range-search may have to backtrack and might visit several leaf nodes before it can complete.

The total run time increases with both the above statistics.

B. Impact of the Reporting Mode

In Section III, we defined a family of relation graphs G_k. The parameter k, which we call the "reporting mode", determines the maximum in-degree of any node in the graph. Our experimental results show that G_1 consistently yields the best performance among all G_k. This coincides with our theoretical predictions.

We plot the program's runtime under two different experimental settings, one under a light load and the

other under a heavy load. The following parameters are common to both settings.

1. number of commands = $60,000$
2. dimension = 10 (five attributes)
3. the spatial index: space filling curve and k-d tree

The load of the system is controlled by the parameter p (the percentage of unsubscribe operations). As p increases, the average size of the index decreases, since deletions become more frequent. We choose $p = 5\%$ to generate a heavy load, and $p = 40\%$ to generate a light load.

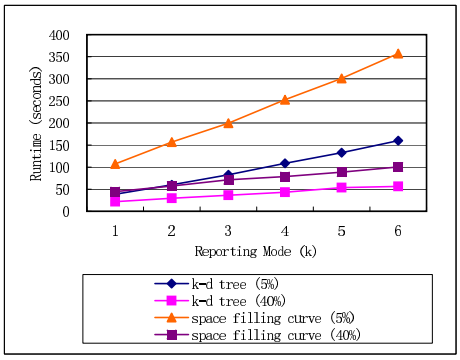

Fig. 8. Runtime vs Reporting mode

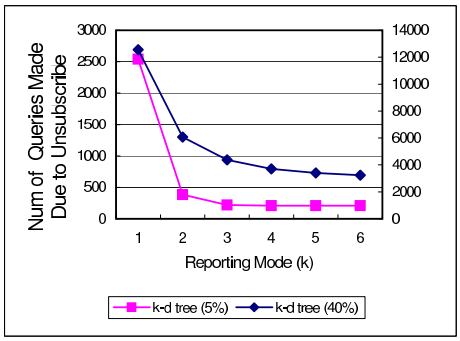

Fig. 9. (The number of queries made due to unsubscribe operations) vs Reporting Mode. The curve for $p = 5\%$ uses the left y-axis, and the curve for $p = 40\%$ uses the right y-axis. Note that the number of queries made due to subscribe operations does not change with the reporting mode.

Figure 8 shows that the runtime always increases with the reporting mode k, for both the k-d tree and the space filling curve. To further analyze the results, we measure the statistics mentioned above. We tabulate the data for the k-d tree in Tables I and II. Similar trends are observed for the space filling curves, though the actual numbers are different. The labels nv_s, nv_u, nv denote the number of *nodes visited* due to subscribes, unsubscribes, and the sum of the two respectively. Similarly pc_s, pc_u, pc denote the number of *point comparisons*

k	nv	nv_s	nv_u	pc	pc_s	pc_u
1	813	782	31	6147	5993	154
2	983	974	9	9967	9852	115
3	1140	1131	9	13111	12973	138
4	1275	1266	9	15816	15663	153
5	1400	1390	10	18274	18111	163
6	1508	1498	10	20407	20237	170

TABLE I

HEAVY LOAD: IMPACT OF REPORTING MODE ON THE STATISTICS FOR k-D TREE.

The above data is for 60,000 operations, with 10 dimensional points, under a light load ($p = 5\%$). Note: All numbers are multiplied by 1000.

k	nv	nv_s	nv_u	pc	pc_s	pc_u
1	556	415	141	6192	4684	1508
2	634	529	105	8952	7376	1576
3	714	615	99	11026	9365	1661
4	780	683	97	12661	10936	1725
5	839	740	99	14050	12260	1790
6	889	789	100	15233	13389	1844

TABLE II

LIGHT LOAD: SAME EXPERIMENT AS IN TABLE I, BUT WITH $p = 40\%$

due to subscribes, unsubscribes, and the sum of the two respectively.

We observe that for both the light load and the heavy load scenarios, both nv and pc increase with k. This immediately implies that the run time also increases with k. In going from $k = 1$ to $k = 2$, the numbers nv_u and pc_u decrease, which means that the unsubscribes are becoming cheaper to handle. However, the subscribes are getting more expensive since the cost of $cov_k()$ strictly increases with k. The decrease in unsubscribe cost cannot compensate for the increase in the subscribe cost, so that $k = 1$ gives the better total cost.

As we move from $k = 2$ to $k = 6$, the number of $cov_k()$ queries made by unsubscribe operations decreases slightly, as shown in Figure 9. However, the cost of each $cov_k()$ increases, so that the cost of unsubscribes increases on the whole, as shown in Tables I, II. Since the cost of subscribes always increases with k, the total cost increases.

C. 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 reference point, we also introduce the *naive* algorithm which uses no index. We present the results of our experiments in Figures 10, 11, and 12. The experimental settings are: $p = 15\%$, the total number of commands range from 10,000 to 100,000, and the dimension ranges from 6 to 16.

Our results indicate that while both the k-d trees and space-filling curves give substantial improvements in run times over the naive algorithm, k-d trees perform better

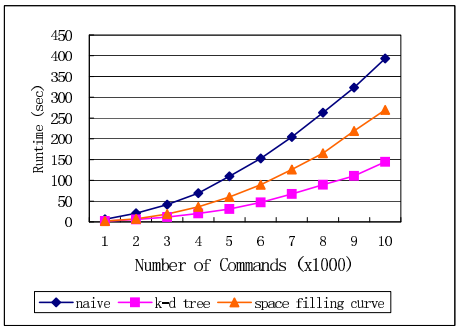

Fig. 10. Total runtime, dimension = 6

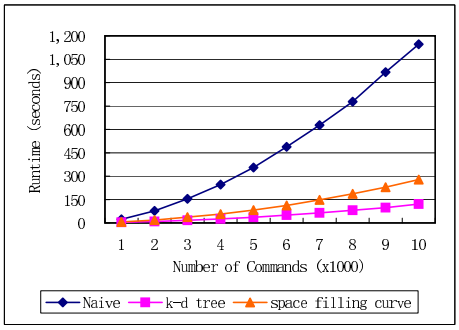

Fig. 11. Total runtime, dimension = 12

than space-filling curves. As the dimension increases, the gain due to indexing also increases. For example, in 16 dimensional space (where each subscription has constraints on 8 attributes), k-d trees run 10 times faster than the naive algorithm, and about 2 times faster than the space filling curve. The performance of the naive algorithm is better under lower dimensions, since there is a greater likelihood of finding a dominating point using a sequential scan of the data.

REFERENCES

[AE99] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 23:1–56, 1999.

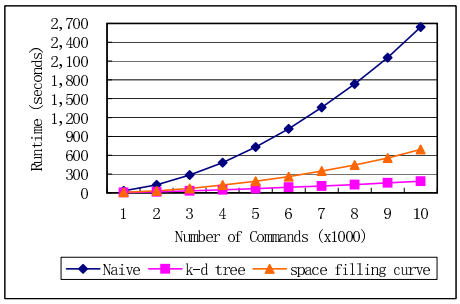

Fig. 12. Total runtime, dimension = 16

[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.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sep 1975.

[Ben80] J.L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[BF79] J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11:397–409, 1979.

[CCC+01] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *Proc. of the 19th Conference on Software Engineering*, Toronto, Canada, May 2001.

[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.

[CW03] A. Carzaniga and A.L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, Karlsruhe,Germany, August 2003.

[FJL+01] 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. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):115–126, 2001.

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

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, Jun 1998.

[Gut84] A. Guttman. A dynamic index structure for spatial searching. In *Proceeding of ACM SIGMOD Conference of Management of Data*, pages 47–57, Jun 1984.

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

[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.

[Mor66] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing, 1966. IBM,Ottawa,Canada.

[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.

[OM84] J.A. Orenstein and T.H. Merrett. A class of data structures for associative searching. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 181–190, 1984.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.