Approximate Covering Detection among Content-Based Subscriptions Using Space Filling Curves

Zhenhui Shen Srikanta Tirthapura *
Department of Electrical and Computer Engineering, Iowa State University
2215 Coover Hall, Ames, IA, 50010, USA
{zhshen,snt}@iastate.edu

Abstract

We consider a problem that arises during the propagation of subscriptions in a content-based publish-subscribe system. Subscription covering is a promising optimization that reduces the number of subscriptions propagated, and hence, the size of routing tables in a content-based publish-subscribe system. However, detecting covering relationships among subscriptions can be an expensive computational task that potentially reduces the utility of covering as an optimization.

We introduce an alternate approach approximate subscription covering*, which provide much of the benefits of subscription covering at a fraction of its cost. By forgoing an exhaustive search for covering subscriptions in favor of an approximate search, it is shown that the time complexity of covering detection can be dramatically reduced. The trade off between efficiency of covering detection and the approximation error is demonstrated through the analysis of indexes for multi-attribute subscriptions based on space filling curves.*

1. Introduction

Content-based publish-subscribe systems offer an expressive middleware to distributed applications. These systems route messages from senders (publishers) to receivers (subscribers) based on the message content, rather than on a fixed destination address, as in conventional IP routing. In order to express their personal interests, subscribers provide *subscriptions* to the system, which are predicates on the message content. A subscriber is delivered only those messages which match its subscriptions. For example, the event $[\text{stock} = \text{IBM}, \text{volume} = 1000, \text{current} = 88]$ must be delivered to a client which has issued the subscription $[\text{stock} = \text{IBM}, \text{volume} > 500, \text{current} < 95]$. A content-based routing middleware greatly simplifies the design of distributed applications. Since a centralized implementation of publish-subscribe may not scale to large networks, many existing systems like Gryphon [26], Siena [1], JEDI [4] and REBECA [14] use a distributed network of routers to implement publish-subscribe.

Subscription propagation is a key component in any distributed publish-subscribe system. A subscription registered at one router has to be propagated within the network so that events published at remote routers can be delivered back. One approach to propagating subscriptions is to flood each subscription to all routers in the network. However, such a naive approach can greatly increase network traffic as well as the size of routing tables, making event forwarding less efficient. A more efficient approach (that has been proposed by multiple groups [11, 1, 22, 19]) is to take advantage of *covering* relationships among subscriptions to reduce the number of subscriptions propagated in the system. For a subscription s, let $N(s)$ denote the set of all messages that match s. Let s_1 and s_2 be two subscriptions. We say that s_1 *covers* s_2, iff $N(s_1) \supseteq N(s_2)$. If a newly arrived subscription s_2 is covered by an existing subscription s_1, then there is no need to forward s_2, since all messages that are matching s_2 are already being received. Repeating this process at every router in the network can lead to a significant reduction in the size of the routing tables at the nodes, and eventually lead to shorter delivery latencies. Subscription covering has been implemented in existing systems, including [1, 4, 14].

However, identifying covering relationships among subscriptions is itself a hard combinatorial problem, for which no solutions are known that are time-efficient in the worst case. Consider a system where each message is composed of multiple numeric attributes, and each subscription is a conjunction of range constraints over the attributes. Suppose a router has already received a set of subscriptions, S. Given an arriving subscription s, the problem of deciding whether or not there is a subscription in S that covers s can

*Supported in part by NSF grant CNS 0520102 and by ICUBE, Iowa State University

be shown to be equivalent to the *point dominance* problem in high-dimensional space. Point dominance is well studied in computational geometry and strong lower bounds are known for its time and space complexity [2, 3], showing that there does not exist a worst-case time efficient solution. This is called the *curse of dimensionality* – the cost of indexing geometric objects in high dimensions often increases exponentially with the number of dimensions. While there exist worst-case efficient data structures for point dominance in one and two dimensional spaces, the same is not true for higher dimensions.

We observe that in the publish-subscribe application, there is no need to search for covering subscriptions exhaustively every time. Covering is only an optimization; the system will continue to work correctly if covering relationships go undetected. However, if routers continue to forward subscriptions that are covered, the system could soon degrade in performance. Thus, we have two extremes, neither of them very desirable – one is to ignore covering completely and the other is to follow it exactly all the time. In this paper we propose a middle ground, *approximate* covering detection. When a new subscription arrives, the set of existing subscriptions at a router is partially searched for a covering subscription, and if none is found, then the new subscription is forwarded. We precisely quantify the fraction of the space that is searched for covering subscriptions. Our main result is that this middle ground is quite attractive. *It is possible to search most of the solution space consisting of covering subscriptions at a fraction of the cost it takes to exhaustively search for covering subscriptions.*

We design indexes for approximate covering based on *space-filling curves*. A space filling curve (referred to as "SFC" henceforth) is a proximity preserving mapping from a high dimensional space to a single dimension. SFCs are one of the most popular techniques for indexing high-dimensional data, and have been widely used for tasks such as nearest neighbor search and range queries in high-dimensional spaces [7], data partitioning in parallel computers [10], and in scientific computing [23, 15]. However, as explained above, SFCs (or any other spatial data structure, for that matter) do not provide worst-case time efficient solutions to exact point dominance in high dimension, and hence to subscription covering. We show however, that they can be used to answer approximate point dominance queries much more efficiently.

1.1. Subscription Covering through Point Dominance

We consider a publish-subscribe system where each message has β numerical attributes, and each subscription is a conjunction of range constraints, with one constraint per attribute. A message can be treated as a point in β dimensional space, and a subscription can be treated as a β dimensional rectangle that matches all messages whose corresponding points lie inside the rectangle.

Let S denote the set of subscriptions that have already arrived at the router. Given an incoming subscription s, the problem of finding whether or not it is covered by an existing subscription in S is equivalent to the problem of finding an existing rectangle that encloses the incoming rectangle. We apply the following well-known transformation (Edelsbrunner and Overmars[5]) to convert this into an equivalent *point dominance* problem in 2β-dimensional space. A β-dimensional rectangle (subscription) $s = ([\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_\beta, r_\beta])$ is transformed into a 2β-dimensional point $p(s) = (-\ell_1, r_1, -\ell_2, r_2, \ldots, -\ell_\beta, r_\beta)$. The following fact can be easily verified : *for two β-dimensional subscriptions s_1 and s_2, s_1 covers s_2 iff every coordinate of $p(s_1)$ is no less than the corresponding coordinate of $p(s_2)$*. Note that the transformation goes both ways; it is shown [5] that 2β-dimensional point dominance can be reduced to the β-dimensional rectangle enclosure (or subscription covering) problem. Henceforth, we consider the following point dominance formulation of subscription covering.

Problem 1 (Point Dominance) *Index a set P of points in d dimensional space to answer the following query efficiently. Given a d dimensional point $x = (x_1, x_2, \ldots, x_d)$, report any point in P that lies in the region $([x_1, \infty], [x_2 \infty], \ldots, [x_d, \infty])$. If there are no points in the region, then report "empty".*

Note that we use ∞ to denote the maximum value that can be taken by a coordinate along a dimension. This maximum may be different along different dimensions. Our formulation of approximate subscription covering is through the following relaxed version of point dominance, called *ϵ-approximate point dominance*, for a user specified ϵ.

Problem 2 (ϵ-Approximate Point Dominance) *Index a set P of points in d dimensional space to answer the following query efficiently. Given a user defined parameter $0 < \epsilon < 1$, and a d dimensional query point $x = (x_1, x_2, \ldots, x_d)$, search a subset of the region $([x_1, \infty], [x_2, \infty], \ldots, [x_d, \infty])$ whose volume is at least $(1 - \epsilon)$ of the volume of the entire region. If any point was found in the search, return it, and return "empty" otherwise.*

For example, a 0.05-approximate point dominance query searches 95% of the volume of the region that contains points corresponding to covering subscriptions. The only time when it fails is the case when the query subscription is covered, but all covering subscriptions lie in the remaining 5% of the region that has not been searched. If subscriptions

are well distributed over the universe, then an approximate point dominance search can be expected to find most existing covering relations between subscriptions.

For a point dominance query, let b_{max} and b_{min} denote the number of bits required to represent the longest and the shortest sides respectively, of the query rectangle. The aspect ratio α of the query rectangle is defined as $\alpha = b_{max} - b_{min}$[1]. Informally, the aspect ratio is small (close to zero) when the sides of the query region are approximately the same length and large when they are of significantly different lengths.

We consider indexes for approximate point dominance based on the Z space filling curve. The Z curve has been used in a variety of indexing applications, including commercial data products such as Oracle [16, 8]. Other popularly used SFCs are the Hilbert curve [9] and the Gray code curve [6]. It has been observed [12] that the performance of the Z and Hilbert curves for many indexing applications are within a constant fraction of each other.

1.2. Our Contributions

We introduce the notion of *approximate covering* to optimize subscription propagation in content-based publish-subscribe systems. Using the point dominance formulation of subscription covering, we show the following. *For point dominance queries where the aspect ratio of the query region is small, approximate point dominance is much cheaper than exhaustive point dominance.* More precisely,

1. The worst case time complexity of an ϵ-approximate point dominance query in d dimensions using the Z SFC is $O\left[\log \frac{d}{\epsilon} \cdot \left(2^{\alpha+1}\frac{d}{\epsilon}\right)^{d-1}\right]$
2. In contrast, the worst case time complexity of an exhaustive point dominance query using the Z SFC is $\Omega\left[\left(2^{\alpha-1}\ell\right)^{d-1}\right]$ where ℓ is the length of the shortest side of the query rectangle.
3. We present a simple algorithm for approximate covering detection based on the Z space filling curve.

Somewhat surprisingly, this shows that for a point dominance query with a small aspect ratio, the complexity of an ϵ-approximate query is *independent of the side lengths of the query region*, while the complexity of an exhaustive point dominance query increases as the $(d-1)$th power of the smallest side length of the query region. This implies that an ϵ-approximate query (for a constant ϵ) is much cheaper than an exhaustive query. Further, we can expect that the benefits of approximate covering over exhaustive covering will be more pronounced as the query region gets larger. For query subscriptions with a small aspect ratio, approximate covering can yield most of the benefits of exhaustive covering at a small fraction of the cost, thus making a strong case for using approximate covering in optimizing content-based routing.

If however, the aspect ratio of the query rectangle was large, then the term 2^{α} will dominate the above expressions, and though approximate covering is still cheaper than exhaustive covering, the benefits will not be as much as the case of small aspect ratio. An extreme case in two dimensions is a $M \times 1$ rectangle, which is not efficiently handled by most popular SFCs.

1.3. Related Work

From a worst-case time complexity perspective, the current best solution to point-dominance problem (see [18][Ch. 8], [24, 25]) over a set of n of points in d dimensions has a query time of $O(\log^{d-1} n)$, insertion and deletion times of $O(\log^{d} n)$. A serious limitation of this solution is the space complexity, which is $O(n \log^{d} n)$, making them impractical for use in a pub-sub system. For example, with 10^4 subscriptions each with 4 attributes, the space requirement is easily outside the capacity of the main memory.

Existing solutions to the problem of subscription covering [11, 22] do not provide any formal analysis of the performance. In a recent work, Ouksel *et. al.*[17] consider a relaxed notion of subscription covering and give a probabilistic algorithm for covering detection. The complexity is $O(nm)$, where n is the number of subscriptions and and m is the number of attributes. To our knowledge, ours is the first algorithm for exact or approximate covering with a time complexity that is sublinear in the number of subscriptions being indexed.

Though there have been numerous applications of SFCs for indexing multidimensional data and corresponding experimental analysis, there has been relatively little work on a formal analysis of their performance. Moon *et al.*[12] present an analysis of the clustering properties of the Hilbert SFC. They show that given a query region which is a high dimensional rectangle, the average number of clusters of points inside the rectangle is proportional to the surface area of the query rectangle. Their analysis considers exhaustive search while we consider approximate search. Tirthapura, Seal and Aluru [21] show a formal analysis of the performance of space filling curves for parallel domain decomposition.

Roadmap. The rest of this paper is organized as follows. In Section 2 we review space filling curves. In Section 3, we first present some intuition as to why approximate point

[1] In two dimensional space, the aspect ratio of a rectangle is traditionally defined as the ratio of the longer to the shorter side. Our definition of aspect ratio is approximately the logarithm (to base 2) of the traditional definition. This definition leads to a convenient statement of our results.

dominance may be cheaper than exhaustive point dominance, and then present the upper bound on the cost of approximate point dominance. This is followed by an analysis of a lower bound on the cost of exhaustive point dominance in Section 4. We then sketch our algorithm for approximate point dominance in Section 5. Section 6 concludes our work.

2. Space Filling Curves

We consider a d-dimensional universe $2^k \times 2^k \cdots \times 2^k$. Note that the number of dimensions d is twice the number of attributes in a subscription. Each element of this universe $p = (x_1, x_2, \ldots, x_d)$, where for all $i \in [1, d]$, $x_i \in [0, 2^k - 1]$, is called a *cell*. The SFC imposes a linear order on all 2^{kd} cells. Henceforth we use the term *cube* to refer to a cube in d dimensions and *rectangle* to refer to a rectangle in d dimensions.

Most SFCs used for indexing including the Z curve [13] and the Hilbert curve [9], utilize a recursive partitioning of the universe. The universe is first divided into 2^d cubes, each of side length 2^{k-1}, by bisecting along every dimension. Each resulting cube is recursively divided $k-1$ times until we are left with unit cubes. We use the term *standard cube* to refer to each intermediate cube resulting from this process. When the cube is recursively decomposed $\ell \leq k$ times, there are $2^{d\ell}$ standard cubes each containing $2^{d(k-\ell)}$ cells. Each such cube is referred to as a *standard cube at level* ℓ. Standard cubes at level k are the individual cells. The following useful property is proved formally in our technical report [20].

Lemma 2.1 *Let C and D be two standard cubes which are not equal to each other. Then either C contains D or D contains C or C and D are disjoint from each other.*

Each standard cube at level $\ell \leq k$ is assigned a unique $d \cdot \ell$ bit number called its *key*, which defines its position in the total order. Different SFCs differ in the assignment of keys to different standard cubes at the same level. The input points are sorted according to the keys of the cells containing them, and stored in a one-dimensional data structure called the *SFC array* (note that the SFC array could be implemented using any dynamic unidimensional data structure such as a binary tree or a skip list).

A *run* is defined as a set of cells that are consecutively ordered by the SFC. For example, in Figure 1, for the rectangular region, there are three runs in the Z curve, and two runs in the Hilbert curve. All points belonging to a run appear as a contiguous segment in the SFC array. Accessing a run in the SFC requires two binary searches on the keys corresponding to the first and last cells in the run. Hence, an operation on a run, such as examining if a run is empty or not, is very efficient, whether the run is small or large.

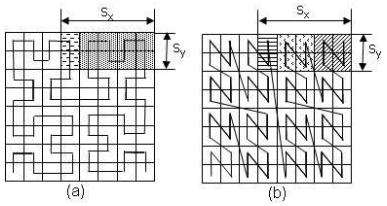

Figure 1. For the same $S_x \times S_y$ rectangle, there are (a)Two runs for the Hilbert SFC and (b)Three runs for the Z SFC

Hence, the performance of an SFC based query over a region depends on the minimum number of runs the region can be decomposed into. The following fact is true for the Z SFC, and for all SFCs that use a recursive partitioning of the universe. Informally, once an SFC enters a standard cube, it will leave the standard cube only after visiting every cell inside it.

Fact 2.1 *A standard d-cube is a single run.*

3. Upper Bound for Approximate Point Dominance

In this section we derive an upper bound for the cost of an approximate point dominance query.

3.1. Intuition

The performance of the space filling curve on a point dominance query, whether exhaustive or approximate, depends on how many runs the query region can be partitioned into. Accessing different runs costs the same, but the volume covered by different runs can be vastly different. As a result, the regions that are considered by the approximate and exhaustive point dominance queries may differ only slightly in terms of volume, but widely in terms of the number of runs required to cover them, and hence in the cost of processing.

For example, consider a universe indexed by the Z curve. Figure 2 shows two query regions, each corresponding to a different point dominance query. The first query region is a square of size 256×256, and the second query region is of size 257×257. For the first query, there is a single run that exactly equals the query region, and the cost of answering this query is very small. On the other hand, the number of runs required to exhaustively cover the second query region is 385, since we are forced to cover the periphery of the region using very short runs. In fact, it is known [12] that the cost of exhaustively covering a d-dimensional rectangle is proportional to the surface area (the perimeter, in two dimensions) of the rectangle. However, for the second query

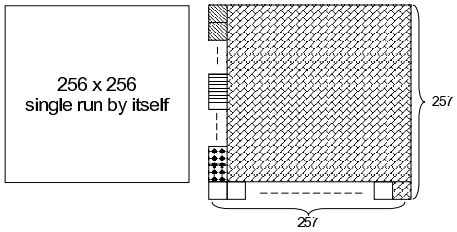

Figure 2. Two example point dominance queries for the Z curve. Standard cubes belonging to the same run are drawn using identical patterns.

region, one of the runs covers more than 99% of the query region, while all the other runs together cover only 1% of the query region. If we only wanted a 0.01-approximate point dominance search, we would be done if we only examined the largest run, and ignored the rest. Thus an approximate point dominance search would be much faster than an exhaustive in this case.

The algorithm for approximate point dominance will select a subspace of the query region such that the volume of the subspace is at least $(1 - \epsilon)$ fraction of the volume of the query region, but the number of runs required to cover it is much smaller. We now describe the way in which this subspace is selected.

Given a point $p = (x_1, x_2, \ldots, x_d)$, exhaustive point dominance searches for a point in the rectangle $([x_1, 2^k - 1], [x_2, 2^k - 1], \ldots, [x_d, 2^k - 1])$. We refer to such a rectangle as an *extremal rectangle*, since one of its vertices is the point $(2^k - 1, 2^k - 1, \ldots, 2^k - 1)$. Note that an extremal rectangle can be completely specified through specifying its lengths along each dimension, since one of its vertices is fixed. Let $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$ be a vector where for each $i = 1, \ldots, d$, $1 \leq \ell_i \leq 2^k$. We use $R(\ell)$ to denote an extremal rectangle, whose side lengths along dimensions $1, 2, \ldots, d$ are $\ell_1, \ell_2, \ldots, \ell_d$ respectively.

For a positive integer x, let $b(x)$ denote the number of bits in the binary representation of x, where the most significant bit is 1. For example, $b(9) = 4$. For positive integer x and $m < b(x)$, let $t(x, m)$ denote the integer formed by retaining the m most significant bits in $b(x)$ and setting the rest to zero. When the input is a vector, the operator $t()$ will be applied to each element in the vector. For example, if $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$, then $t(\ell, m) = (t(\ell_1, m), t(\ell_2, m), \cdots, t(\ell_d, m))$.

Given an initial query region $R(\ell)$, the approximate point dominance query considers a smaller extremal rectangle $R(t(\ell, m))$ that is completely contained within $R(\ell)$. For ease of notation, we use $R^m(\ell)$ to represent $R(t(\ell, m))$. The parameter m is chosen as a function of ϵ (the user desired coverage) such that the chosen rectangle covers at least $(1 - \epsilon)$ fraction of the volume of $R(\ell)$.

3.2. Upper Bound

We now formally prove an upper bound on the cost of approximate point dominance for the Z SFC. The main result in this section is Theorem 3.1. Due to space constraints, the proofs are omitted, and can be found in our technical report [20].

Definition 3.1 *For rectangle T, $cubes(T)$ is defined as the minimum number of standard cubes into which T can be partitioned, and $runs(T)$ is defined as the minimum number of runs whose union is equal to all cells of T in the SFC array.*

The following lemma follows since each cube is a single run.

Lemma 3.1 *For any rectangle T, $cubes(T) \geq runs(T)$.*

The worst-case cost of approximate point dominance is equal to $runs(R^m(\ell))$. According to Lemma 3.1, this is bounded by $cubes(R^m(\ell))$. Our strategy for the proof is as follows. Lemma 3.3 shows that a greedy algorithm can partition an arbitrary region into a minimum number of standard cubes. Lemma 3.4 classifies the obtained standard cubes according to their sizes, and lets us compute the total number of cubes of each size. Finally, these are combined in Lemma 3.6 to yield an upper bound on the total number of runs that make up any query region. Note that our proofs are fairly general so that they remain valid for other SFCs, such as the Hilbert curve, which is based on a recursive partitioning of the space.

As the first step, we decide what value of m such that $R^m(\ell)$ has the required space coverage. The following lemma shows that if we truncated each side length of $R(\ell)$ down to its $\log_2 \frac{2d}{\epsilon}$ most significant bits, then the volume of the resulting rectangle is within $(1 - \epsilon)$ of the volume of $R(\ell)$.

Lemma 3.2 *Let $0 < \epsilon < 1$. If $m \geq \log_2 \frac{2d}{\epsilon}$, then $\frac{vol(R^m(\ell))}{vol(R(\ell))} \geq 1 - \epsilon$*

Next we consider the partitioning of a given extremal rectangle into standard cubes. It is always trivial to partition any rectangle into individual cells (note that each cell is a standard cube). We now describe a greedy algorithm that outputs the partition of any region (which is not necessarily a rectangle) into a *minimum* number of standard cubes. The greedy algorithm works as follows. *If the region R is empty, then the algorithm outputs nothing, and exits. Otherwise, the largest standard cube that fits within R, say C, is chosen and output. The algorithm is then recursively applied on input $R - C$.*

Lemma 3.3 *The greedy algorithm, when applied to region R, produces an optimal partition of R into a minimum number of standard cubes.*

Let D represent the set of standard cubes resulting from a greedy decomposition of $R(\ell)$. Let D_i represent a subset of D consisting of standard cubes of side length 2^i. For integer x, let x_j denote the jth bit in x. Let $S_i(x) = \sum_{j=i}^{j=b(x)-1} x_j 2^j$, which is the result of choosing only the most significant bits in x starting from x_i onwards. When the input is a vector, the operator $S_i()$ will be applied to each element in the vector. Thus $S_i(\ell) = (S_i(\ell_1), S_i(\ell_2), \cdots, S_i(\ell_d))$.

Let $\ell_{i,j}$ be the jth bit of ℓ_i. For $j \in [1, d]$, indicator variable O_j is defined as follows: $O_j = 0$, if $\ell_{i,j} = 0$ for all $i \in [1, d]$; $O_j = 1$, if $\ell_{i,j} = 1$ for some $i \in [1, d]$. Assume w.l.o.g. $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_d$. The following lemma characterizes the type and location of the standard cubes resulting from an optimal partition of $R(\ell)$.

Lemma 3.4 *For $b(\ell_1) \leq i \leq b(\ell_d) - 1$, D_i is empty. For $0 \leq i \leq b(\ell_1) - 1$*
(1) D_i is non-empty if and only if $O_i = 1$.
(2) The region occupied by $\cup_{j=i}^{j=b(\ell_1)-1} D_j$ is the extremal rectangle $R(S_i(\ell))$.

Lemma 3.4 can be used to totalize the minimum number of standard cubes in $R^m(\ell)$ as stated in Lemma 3.6.

Lemma 3.5 *$cubes(R^m(\ell))$ is maximized iff (1) $\ell_{j,x} = 1$ for $x \in [b(\ell_j) - m, b(\ell_j) - 1]$ and $j \in [1, d]$ and (2) $b(\ell_j) = b(\ell_d)$ for $1 < j < d$.*

Lemma 3.6 $cubes(R^m(\ell)) < m \cdot [2^\alpha(2^m - 1)]^{d-1}$

Recall that $\alpha = b(\ell_d) - b(\ell_1)$ is the aspect ratio of the rectangle, and $0 < \epsilon < 1$ is a user parameter indicating the desired coverage of the approximate query.

Theorem 3.1 *For any SFC that is based on a recursive partitioning of the universe (such as the Z curve and the Hilbert curve), the cost of an ϵ-approximate point dominance query is $O(\log \frac{d}{\epsilon} \cdot \left(2^{\alpha+1}\frac{d}{\epsilon}\right)^{d-1})$*

4. Lower Bound for Exhaustive Point Dominance

In this section, we prove a lower bound on the worst-case cost of exhaustive point dominance using Z-curve (Theorem 4.1). The worst-case cost is equal to $runs(R(\ell))$. However, estimating the size of $runs(R(\ell))$ is much harder than estimating the size of $cubes(R(\ell))$, because there is no simple characterization for runs like the "greedy" characterization for cubes.

Our strategy for proving a lower bound is as follows. We construct an extremal rectangle $R(\ell)$ such that for a large subset of the standard cubes resulting from a greedy (optimal) partition of $R(\ell)$, no two cubes in the subset can belong to the same run in the Z curve. Thus $runs(R(\ell))$ is no less than the size of this subset.

Let γ be an integer in $(0, k - \alpha]$. Given a value of the aspect ratio α, we consider the following extremal rectangle $R(\ell)$: (1) $\ell_d = 2^\gamma - 1$ and (2) $b(\ell_i) = \gamma + \alpha$ for $i \in [1, d)$. We examine a smaller rectangle R_0 contained inside $R(\ell)$. R_0 is constructed by selecting the least significant bit from ℓ_d and the most significant bit from ℓ_i for $i \in [1, d)$. Thus, the side length of R_0 along dimension d is 1, and the side length of R_0 along all other dimensions is $2^{b(\ell_1)-1}$. When the greedy partition is applied, R_0 is filled with standard cubes with a side length of 1, since its shortest side has length 1.

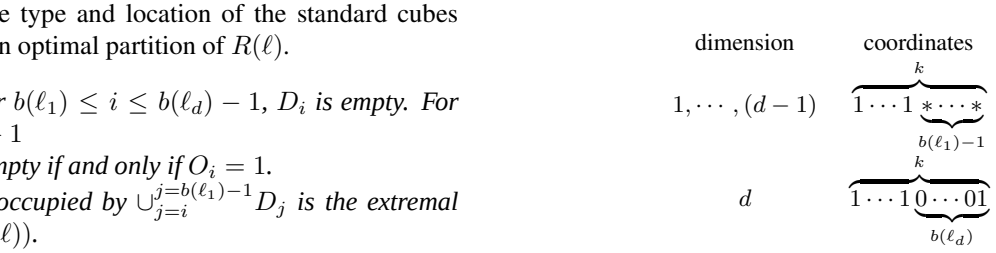

Figure 3. Coordinates of a cell in R_0. Symbol $*$ can be either 1 or 0. The size of the universe along each dimension is 2^k.

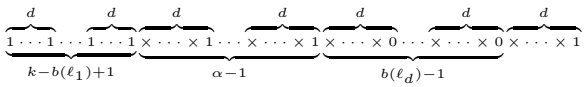

Figure 4. Key of a cell in R_0

The position of each standard cube can be specified by its coordinates. For example, square "a" in Figure 5(c) has coordinates $(010, 011)$. The Z-curve computes the key of a standard cube by interleaving the bits representing its coordinates, starting from dimension 1, and then proceeding to higher dimensions. For instance, the key of square "a" is $(001101)_2 = 13$. Based on the way we construct R_0, we claim that the coordinates of each standard cube in R_0 exhibits the pattern shown in Figure 3. It then follows that the key of a cube in R_0 must satisfy the pattern shown in Figure 4. Lemma 4.1 uses this derived pattern to prove that no two standard cubes in R_0 belong to the same run on the Z-curve.

Lemma 4.1 *No two standard cubes in R_0 belong to the same run on the Z space filling curve.*

Theorem 4.1 *For any integer $0 \leq \alpha < k$, there exists an extremal rectangle $R(\ell)$ whose aspect ratio is α and the cost of an exhaustive search of $R(\ell)$ using the Z space filling curve is $\Omega\left[\left(2^{\alpha-1} \cdot \ell_d\right)^{d-1}\right]$*

The proofs can be found in our technical report [20].

5. Algorithm for Approximate Point Dominance

In this section, we sketch an algorithm for approximate point dominance based on the Z SFC. The only data structure to maintain is the SFC array, which sorts input points according to their positions on the Z curve. It is easy to maintain this sorted order, while allowing frequent additions and deletions of points, by using a dynamic ordered data structure such as a balanced binary tree.

Given a point dominance query, our algorithm follows the greedy approach to partition the query region into a minimum number of standard cubes. It then searches these cubes in the SFC array for covering subscriptions, in the descending order of their volumes. Meanwhile it keeps track of the ratio of the volume searched to the volume of the query region. The search terminates when either a covering subscription is found or this ratio exceeds $1 - \epsilon$.

The major operation in the above algorithm is to compute the keys (defined in Section 2) of the standard cubes produced by the greedy decomposition – these keys are required by the search in the SFC array.

Let $R(\ell)$ denote the extremal rectangle to be searched for a point dominance query, where $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$. Recall that D_i (defined in Section 3) is the set of all the standard cubes resulting from the greedy decomposition, whose side length is 2^i. It is sufficient to demonstrate how to compute the keys of all standard cubes within a particular D_i. We use the following two-stage algorithm. In the first stage, we decompose the space occupied by D_i into disjoint rectangles with the following two properties: (1) The side length of the rectangle along every dimension must be a multiple of 2^i so that the entire rectangle is a union of standard cubes in D_i. (2) There exists at least one dimension along which the side length of the rectangle is exactly 2^i. In the second stage, we identify the standard cubes within each such rectangle and compute their keys.

Consider such a rectangle r in D_j that satisfies the above two properties. To decide r's side length on dimension $j (j = 1 \ldots d)$, we only need to consider the non-zero bits in ℓ_j, whose position in ℓ_j is at least i (the position of a bit b in a binary number is defined as the number of bits to the right of b in the binary number – note that the least significant bit is at position 0). Let $P = (P_1, P_2, \ldots, P_d)$ be a vector defined as follows. For $j = 1 \ldots d$, P_j is the position of a non-zero bit in ℓ_j such that $P_j \geq i$. It can be verified that

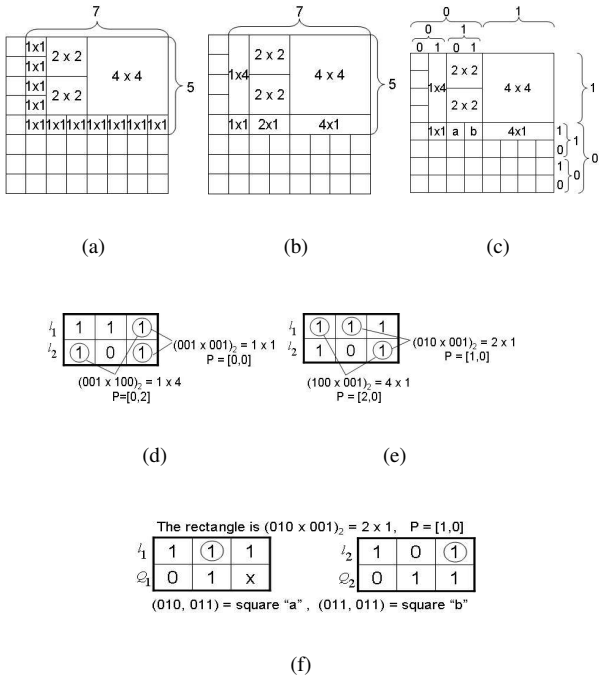

Figure 5. Computation of the keys of standard cubes resulting from a greedy decomposition. Symbol $\times$ can be either 1 or 0

each unique instance of P corresponds to a different rectangle lying in the space occupied by D_i, which satisfies the properties listed in the previous paragraph. For example, all the 1×1 squares in Figure 5(a) constitute the space occupied by D_0. We can divide it into four rectangles as shown in Figure 5(b). The corresponding instances of P are shown in Figures 5(d) and 5(e).

Consider the rectangle represented by a vector P. The next step is to compute the keys of standard cubes contained within P. We create another vector $Q = (Q_1, Q_2, \ldots, Q_d)$ to store the coordinates of a standard cube within P, where Q_j is the coordinate of the standard cube along dimension j. We know that standard cubes in D_i result from $(k - i)$ rounds of recursive partitioning of the universe. Thus we can use $(k - i)$ bits to represent Q_j, for $j = 1 \ldots d$. Let $Q_{j,y}$ be the bit of Q_j at position y. We can compute Q_j from P_j and ℓ_j in the following way:

$$\begin{array}{ll} Q_{j,y-i} = \neg \ell_{x,y}, & \text{for } y \in (P_x, k-1] \\ Q_{j,y-i} = \ell_{x,y}, & \text{for } y = P_x \\ Q_{j,y-i} = \text{either 0 or 1}, & \text{for } y \in [i, P_x) \end{array} \quad (1)$$

For example, the rectangle "2×1" in Figure 5(b) consists of two standard squares "a" and "b" as shown in Figure 5(c). We get the rectangle by selecting the first bit from ℓ_1 and the

rightmost bit from ℓ_2. Figure 5(f) shows how Equation (1) can be applied to compute the coordinates of "a" and "b". Once the coordinates of a cube are available, we can get its key by interleaving the bits, and the standard squares can be searched in the SFC array. For a formal description of the algorithm, we refer the reader to the full version [20].

6 Conclusion

Detection of covering among content-based subscriptions is a hard combinatorial problem. Previous work on this problem has focused on *exact* algorithms for covering detection. However we point out that subscription covering is just an optimization, which does not need to be strictly followed all the time. Based on this observation, we introduce the notion of *approximate* covering to retain most benefits of exact covering at a fraction of its cost. Our approximate solution is based on space filling curves and supports numeric subscriptions. We formally analyze the algorithm's complexity. Our analysis shows that when the aspect ratio of the query region is small, approximate covering is much cheaper than exact covering. Furthermore we present a simple implementation of the algorithm so that our results may directly benefit current implementations of content-based publish/subscribe system.

References

[1] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proc. IEEE INFOCOM 2004*, 2004.

[2] B. Chazelle. Lower Bounds for Orthogonal Range Searching: I. The Reporting Case. *Journal of the ACM*, 37:200–212, Apr 1990.

[3] B. Chazelle. Lower Bounds for Orthogonal Range Searching: II. The Arithmetic Model. *Journal of the ACM*, 37:439–463, Jul 1990.

[4] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[5] H. Edelsbrunner and M. H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14(3):124–127, 1982.

[6] C. Faloutsos. Multiattribute hashing using gray codes. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 227–238, 1986.

[7] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, 1988.

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

[9] D. Hilbert. Uber die stegie abbildung einer linie auf flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.

[10] V. J. Leung, E. M. Arkin, M. A. Bender, D. P. Bunde, J. Johnston, A. Lal, J. S. B. Mitchell, C. A. Phillips, and S. S. Seiden. Processor allocation on cplant: Achieving general processor locality using one-dimensional allocation strategies. In *Proc. IEEE International Conference on Cluster Computing (CLUSTER)*, pages 296–304, 2002.

[11] G. Li, S. Hou, and H. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 447–457, 2005.

[12] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

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

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

[15] A. Nakano, R. K. Kalia, and P. Vashishta. Scalable molecular dynamics visualization, and data management algorithms for materials simulations. *IEEE Computing in Science and Engineering*, 1(5):39–47, 1999.

[16] Oracle spatial. `http://www.orafaq.com/faq/spatial`, 2001.

[17] A. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient probabilistic subsumption checking for content-based publish/subscribe systems. In *Proc. ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, pages 121–140, 2006.

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

[19] Z. Shen, S. Aluru, and S. Tirthapura. Indexing for subscription covering in publish-subscribe systems. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 328–333, 2005.

[20] Z. Shen and S. Tirthapura. ISU Technical Report TR-2007-03-1. `http://archives.ece.iastate.edu/archive/00000306/`.

[21] S. Tirthapura, S. Seal, and S. Aluru. A formal analysis of space filling curves for parallel domain decomposition. In *Proc. IEEE International Conference on Parallel Processing*, pages 505–512, 2006.

[22] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 562–571, 2004.

[23] M. Warren and J. Salmon. A parallel hashed-octtree N-body algorithm. In *Proc. Supercomputing*, pages 12–21, 1993.

[24] D. E. Willard. New trie data structures which support very fast search operations. *Journal of Computer and System Sciences*, 28(3):379–394, 1984.

[25] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, 1985.

[26] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. In *Proc. ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, pages 274–293, 2004.