New Algorithms for Content-Based Publication-Subscription Systems

Anton Riabov
Columbia University
IEOR Department
500 West 120 Street, New York, NY 10027
avr11@columbia.edu

Zhen Liu, Joel L. Wolf, Philip S. Yu and Li Zhang
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{zhenl,jlwolf,psyu,zhangli}@us.ibm.com

Abstract

This paper introduces new algorithms specifically designed for content-based publication-subscription systems. These algorithms can be used to determine multicast groups with as much commonality as possible, based on the totality of subscribers' interests. The algorithms are based on concepts borrowed from the literature on spatial databases and clustering. These algorithms perform well in the context of highly heterogeneous subscriptions, and they also scale well. Based on concepts borrowed from the spatial database literature, we develop an algorithm to match publications to subscribers in real-time. We also investigate the benefits of dynamically determining whether to unicast, multicast or broadcast information about the events over the network to the matched subscribers. We call this the distribution method problem. Some of these same concepts can be applied to match publications to subscribers in real-time, and also to determine dynamically whether to unicast, multicast or broadcast information about the events over the network to the matched subscribers. We demonstrate the quality of our algorithms via a number of realistic simulation experiments.

1. Introduction

Publication-Subscription systems (*pub-sub* for short) provide information on specific real-time *events* from *publishers* to interested *subscribers*. The subscribers express their interest in the form of multiple *subscriptions*. The publishers and subscribers may be located at arbitrary nodes in a distributed network. There are two basic types of pub-sub systems, with differing degrees of sophistication.

Content-based pub-sub systems are now becoming quite popular. See, for example, the pioneering work of the Gryphon project [3, 5, 13]. Other systems can be found in NEONet [11] and READY [7]. Such systems differ from the more mature *subject-based* pub-sub systems in their generality, usability and flexibility. Borrowing extensively from the classic stock market example used by Gryphon, a subject-based pub-sub system would allow subscriptions based on subject matter. A subscriber might request all events related to IBM, for instance. Such a system is powerful but relatively inflexible. The subscriber might receive many publications involving IBM stock market events of little interest. On the other hand, a content-based pub-sub system would allow subscriptions which were based on the conjunction of potentially multiple predicates related to different attributes. The motivating Gryphon subscription example was based on three distinct attributes: The first was an equality predicate based on a character string referring to the stock name: *name*=IBM. The second was an inequality predicate based on a two decimal numerical attribute referring to the stock price. We will modify this slightly to consider two predicates, together forming a range: $75.00 < price \leq 80.00$. (This is perhaps a more likely scenario.) The third was an inequality predicate based on an integer attribute referring to the volume: $volume \geq 1000$. A subscriber with this subscription would receive information about all IBM trades with a price between 75 and 80, with a volume of at least 1000 shares. The capability of allowing a subscriber to chose just a subset of all the possible publications pertaining to IBM clearly results in a more valuable service.

So content-based pub-sub is more general and much more personalizable than subject-based pub-sub. But the extra function comes at a price. It is technically challenging for a content-based pub-sub system to efficiently distribute the many publication events to the interested subscribers over the network. And it is technically challenging to do so in a manner which scales with the dimensionality of the underlying event space, the number of publishers and the number of subscriptions.

As identified in the series of Gryphon papers, there are two key dynamic problems that a content-based pub-sub system needs to solve:

1. One must *match* a given real-time event quickly to determine the set of relevant subscribers. This is the so-called *matching* problem.

2. One must unicast, multicast and/or broadcast information about the event over the network efficiently to the matched subscribers (or possibly to a superset of those subscribers, if that is more algorithmically effective, to be filtered out as necessary). We shall call this the *distribution method* problem.

Both of these problems must be solved in real time as new events are published. However, it is clear that there is a related, static "preprocessing" problem that should be solved in order to enable the real-time algorithms to function efficiently. Specifically we must precompute a set of high quality multicast groups based on the totality of subscribers' interests with as much commonality as possible. We shall call this the *subscription clustering* problem.

The Gryphon papers do tackle the dynamic problems above in elegant ways. In [3] and [5], for instance, the authors consider the matching problem. The interested reader is referred to [6] for a detailed description of the matching algorithm, a performance analysis and a comparison with existing matching algorithms. In [5] and [13] the authors consider multicasting techniques. It seems fair to say, however, that the authors base their algorithms primarily on subscriptions in which each dimension is based on either equality or wild-card predicates. (A *wild-card* (*) indicates that the subscriber does not care about the value in that dimension.) While their algorithms will certainly work in full generality, we believe that they are optimized for their motivating predicate types.

In this paper we focus instead on more general instances of the possible predicates. That is, we consider each predicate to be *range-based*, composed of *intervals* in the underlying domain of the predicate. (Because computers can handle only inherently finite and discrete attribute values, one can assume without loss of generality that all possible intervals are actually open on the left and closed on the right. For example, the previous volume predicate can be expressed as $999 < volume \leq maxvolume$. This assumption allows the intervals to 'fit together' more cleanly. By decomposing a subscription with multiple such ranges into multiple subscriptions consisting of single ranges we can see that it is sufficient to only consider intervals, albeit at a cost of more subscriptions. And even attributes such as name, not typically thought of as numerical, can be indexed and therefore linearized in some fashion.)

This perspective allows us to think of a subscription as a half-open, half-closed aligned rectangle in space, each dimension corresponding to a different attribute. (The term *aligned* means that the rectangle is parallel to the various axes.) A published event becomes a point in the underlying space. This mindset enables us to borrow from the spatial database literature in solving the problems. (We also borrow from the clustering literature.)

In a previous paper [15] we proposed a new approach for subscription clustering. This approach imposes a grid structure in each attribute dimension and decomposes the subscriptions by intersecting them with the resulting aligned rectangles. Cells of the grid can then be clustered using known clustering algorithms, making use of feature vectors and distance functions that we proposed.

In the current paper we shall take the preprocessing stage clustering algorithms as given, and focus instead on the dynamic aspects of the pub/sub problem. This includes both matching and distribution method algorithms. For the matching problem we will employ algorithms borrowed from the spatial database literature which are designed to handle so-called *point* queries. Such algorithms, based on constructs such as *S-trees* [2] and *Hilbert-packed R-trees* [8] can efficiently compute or bound the number of subscribers who will be interested in a particular message. (In the query the subscriptions correspond to the aligned rectangles and the message corresponds to the point.) Then we will examine the overall performance gains made possible via a simple but powerful distribution method scheme. In other words, we assume that the clustering algorithms have been used to form multicast groups. When a publication event happens we use the matching algorithm to compute the number of interested subscribers. Then we make on-line decisions whether we do multicast or unicast. We base our decisions on the number (or the ratio of the number to the group size) of subscriptions relevant to each publication event. Remember that it is not always beneficial to do multicast for all publications. This scheme tells us when it is a good idea.

Our results are relevant to two flavors of multicasting, network supported and application level. See [4] for a description of the various tradeoffs. See [14] for a description of application level multicasting.

We present a brief overview of related work. In [16] the authors tackle the problem of finding efficient multicast groups in a very general setting. They concentrate, as we do, on the case where interests are highly heterogeneous. Their goal is also similar, namely to maximize preference overlaps. They make use of clustering techniques and provide both initial and incremental algorithms. The former is useful for devising the first set of multicast groups, and the latter is used to retain high quality in the presence of ongoing and inevitable changes. Consideration of receiver interest is also studied in [10] and [12]. An integer programming formulation for the clustering problem is proposed in [1].

The remainder of this paper is organized as follows: In Section 2 we introduce the required notation. Section 3 describes a possible matching algorithm. (We focus on an *S-tree* implementation here. Comparisons of this with other approaches will appear in a subsequent paper.) Section 4 describes the on-line distribution method scheme we are proposing. In Section 5 we describe simulation experiments which show the performance of the on-line scheme, and present data study to justify the choice of parameters for our simulations. Section 6 contains conclusions and areas for future work. The Appendix contains a brief description for the various clustering algorithms used in this paper.

2. Problem notation

In this section we define some key parameters we will use in our problem descriptions.

- Let Ω denote the event space. Each event being published within the system can be uniquely described with a single value ω such that $\omega \in \Omega$.
- Let N denote the number of dimensions (or attributes) in Ω, so that $\Omega \subseteq \mathbf{R}^N$.
- Let the probability density function publications $p(x)$ defined for each set $x \subseteq \Omega$ (more precisely, $p(x)$ the integral of the p.d.f. over the set x).
- Define the underlying network topology as an undirected graph $G = (V, E)$.
- Define the communication costs to be $c_e \geq 0$ for each edge $e \in E$.
- Let the set of publisher nodes be $V_P \subseteq V$.
- Define the set of subscriber nodes as $V_S \subseteq V$.
- Assume that each subscriber $v_i \in V_S, i = 1..N_S$ has a set of subscription preference rectangles $\mathcal{I}_i = \{b_{ij}\}_{j=1}^{r_i}$ associated with it. Each $b_{ij} \subseteq \Omega$ is an aligned rectangle in space Ω.
- Denote $\mathcal{I} := \bigcup_{v \in V_S} \mathcal{I}_v$ – the set including all subscription rectangles. Denote $k := |\mathcal{I}|$.

The sizes of the problems is defined by values of k and N, and we are interested in algorithms that scale well with respect to these values.

3. Matching problem scheme

In the spatial database literature data objects (which are arbitrary subsets of $\mathbf{R}^N$) are typically approximated by their so-called *bounding boxes* – the smallest aligned rectangle which contains them. A so-called *region* query is a request for all data objects which intersect with a given aligned query rectangle. (This query rectangle is itself typically the bounding box for an arbitrarily shaped query data object.) A *point* query, relevant in the current paper, is a request for all data objects which intersect with a given point. Obviously this is a special case of a region query. To solve the matching problem we need to quickly identify all subscriptions in $\mathcal{I}$ which contain a publication event ω, and then compute the set of subscribers associated with these rectangles. Note that in this point query the subscriptions correspond precisely to their bounding boxes, so that the query yields an exact rather than approximate result.

Trees are indexes which are very efficient at answering region and point queries. In this paper we shall focus on an efficient data structure called the *S-tree* [2]. The structure of the leaf and internal nodes in S-trees are identical to the internal record structure of R-trees. In our application, each record in a leaf node contains entries of the form $(I, \textit{subscription-identifier})$, where I, an N-dimensional rectangle, is the corresponding subscription. Similarly, each internal node contains entries of the form $(I, \textit{pointer})$, where *pointer* contains the address of a lower node in the S-tree, and I is the minimum bounding rectangle of the set all subscriptions in the leaf descendants of that node. Each node has a branch factor of at most M.

To answer a region (or point) query one simply searches downward from the root of the tree, pruning off all nodes (and thus their children) for which the intersection of the query rectangle (or point) and the minimum bounding rectangle is empty. Obviously the choice of tree packing influences the number of node pages which need to be examined. We describe this packing now.

Unlike an R-tree, an S-tree will not necessarily be height balanced. In fact, associated with each S-tree is a parameter p which referred to as the *skew factor* of the tree. This factor p lies in the range $(0, 1/2]$, and will be a good indicator of how well-balanced the tree is required to be. A low value of p results in potentially greater imbalance but greater design flexibility. A skew factor near $1/2$ implies that the tree will be quite well-balanced, but may not have good performance characteristics. Typically p is chosen to be about 0.3.

The *branch factor* M of a node is defined as the number of children of that node. In other words, this is the fanout. We choose M based on page size: A node should fit on a page. M is typically chosen to be about 40, but varies based on the number of dimensions. The *leaf number* N_A of a node A is defined as the total number of data objects in the leaf descendents of that node. Denote the volume of an aligned rectangle I by $\mathcal{V}(I)$.

The process of building an S-tree proceeds in two stages:

1. *Binarization:* In this stage we construct a binary tree such that the entries in the leaf nodes correspond to the spatial data objects. Thus all non-leaf nodes have branch factors of 2. Let A be an internal node of the binary tree with children B_1 and B_2. We ensure that the leaf numbers N_{B_1} and N_{B_2} of each of these children are each at least $p \cdot N_A$. Among all partitions examined which satisfy this condition we choose the one which minimizes the sum $\mathcal{V}(I_{B_1}) + \mathcal{V}(I_{B_2})$ of the volumes of the minimum bounding rectangles.
2. *Compression:* In this stage we convert the binary tree into a tree for which all but the leaf and penultimate nodes have branch factors of M. We achieve this by iteratively compressing pairs of non-leaf parent and child

nodes in the tree into single nodes. The leaf nodes are not affected by this operation.

Here is a brief overview of each stage.

3.1. Binarization

The binary tree is constructed in a top-down fashion. (By comparison, the packing method for Hilbert-packed R-trees can be classified as bottom-up instead.) Thus we start with the full set of data objects and partition them into two sets which satisfy the properties described above. Then we partition each of these two sets of data objects in turn using the same method, and so on recursively until we arrive at the individual leaf nodes.

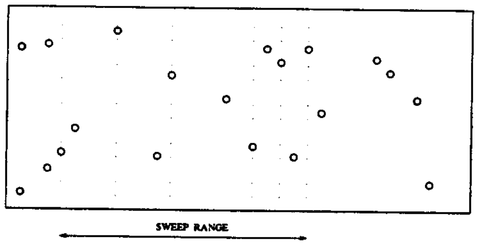

Figure 1. Sweep operation for creation of binary tree.

It is therefore sufficient to describe the binarization process for an arbitrary node A. If $N_A \leq M$ we make A a leaf node and stop. Otherwise, we consider the minimum bounding rectangle I_A, and choose a dimension n for which this rectangle is longest. (Choosing this dimension will tend to cause relatively square minimum bounding rectangles.) Then we represent each of the N_A data objects by their geometrical center, and order them by increasing values of their nth coordinate. Candidate partitions (B_1, B_2) considered will have the property that B_1 contains the first q data objects according to this order, B_2 contains the remaining $N_A - q$ data objects, and $p \cdot N_A \leq q \leq (1-p) \cdot N_A$. *Sweeping* over these $(1-2p) \cdot N_A$ choices in increments of M, we choose the one for which $\mathcal{V}(I_{B_1}) + \mathcal{V}(I_{B_2})$ is minimized. Ties can be adjudicated in favor of rectangle pairs whose total perimeter is minimized. Clearly the rectangles I_{B_1} and I_{B_2} can be computed incrementally as the sweep progresses. Figure 1 shows a sample sweep operation. The bounding rectangle is longest in the x-dimension, and we are assuming that $p = 1/4$ and $M = 2$.

3.2. Compression

We first discuss what it means to collapse two nodes into one. First of all, two nodes A and B can be collapsed into one node when they satisfy a parent-child relationship. Suppose, for example, that node A is the parent of node B. The new node AB has the same ancestor as node A. The children of node AB are the union of the children of node A (excluding B) and the children of node B. This situation is illustrated in Figure 2. Note that the child node B has a branch factor of 2, and therefore the branch factor of the new node AB is 1 more than the branch factor of the original node A.

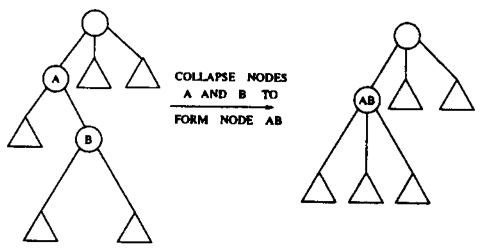

Figure 2. Collapsing two nodes into one.

First we compress bottom up one level, as follows: Identify each node A arising from the binarization stage with the property that the number of leaf descendents of A is less than or equal to M, but the number of leaf descendents of the parent of A is greater than M. These will become the penultimate nodes of the final S-tree, so collapse all non-leaf nodes underneath each such A. (A node is said to be *penultimate* if all of its children are leaf nodes.)

The remaining compression steps always collapse a parent node with a child node having a branch factor of 2, so that the branch factors of the resulting new node will always increase by 1 at a time. In this way we can be certain that collapsing will never result in a node with a branch factor exceeding M. The idea is to pick a node, collapse it with one of its children to form a new node, and keep iterating the process until either the branch factor of that node is exactly M, or all children of that node are leaf nodes. Of course, given the parent node we need to choose the child to collapse it with. The rule is simple: We always choose the child with the highest leaf number. The motivation behind this rule is that it restricts the amount of skewness in the tree. Notice also that this rule automatically guarantees that the chosen child will *not* be a leaf node, because all children which are leaf nodes will have leaf numbers less than or equal to M, while all children which are not will have leaf numbers greater than M. It remains to choose the order in which we choose the parent nodes, and this is top-down. To accomplish this, we start with an ordered list of the non-leaf nodes obtained via breadth-first search starting at the root, removing collapsed parent nodes but replacing them with the newer nodes unless their branch factors are full. The process continues until this list of potential parents is empty. At its conclusion, only the penultimate and leaf nodes can have branch factors of less than M.

4. Distribution method scheme

In this section we shall first briefly outline our algorithms for solving the subscription clustering problem. Then we describe the new distribution method scheme itself. This scheme uses the matching problem algorithm in order to make on-line decisions whether to do multicast or unicast.

In our recent paper [15] we proposed a family of algorithms for clustering subscription interest. The proposed algorithms partition the event space Ω into $(n+1)$ non-overlapping subsets for a fixed number n of multicast groups n. Thus there are n subsets $S_1, .., S_n$ created by the algorithm, plus a remaining catchall subset which includes the rest of Ω: $S_0 = \Omega \backslash \bigcup_{q=1}^{n} S_q$. For each set S_q with $1 \leq q \leq n$ a multicast group is formed from all subscribers who have overlapping subscriptions. In other words, we choose a multicast group $M_q = \{i \in V_S : \exists j : b_{ij} \cap S_q \neq \emptyset\}$.

(Briefly, these algorithms were borrowed from the data clustering literature, combining similar cells of a regular grid in Ω. Similarity of cells is defined based on membership vectors and a publication density function $p(.)$. Then the algorithms can be applied to form locally optimal solutions. In [15] we compared performance results for the so-called *K-means, Forgy K-means, minimum spanning tree and pair-wise grouping* algorithms in this framework.)

Given the set of multicast groups and a new publication vector ω the distribution method algorithm proceeds as follows. If $\omega \in S_0$, then the publication is delivered using unicast messages. Otherwise there exists unique value of q, $1 \leq q \leq n$, such that $\omega \in S_q$.

It is clear that all subscribers interested in receiving message ω are in the group S_q, but not all members of this group may be interested in receiving this message. So the *S-tree* matching algorithm is used to build a list s of interested subscribers for a given message ω. If this list is empty, the publication will be not sent. If it is not empty, the size of this list can be used to decide whether the publication should be delivered using unicast messages or using a multicast message to the group S_q. The algorithm simply decides to send unicast messages if the proportion of interested subscribers is below a certain fixed level t, i.e. if $|s|/|S_q| < t$. (We will evaluate the influence of the threshold parameter on the average message cost in the numerical experiments section.)

5. Experiments

We evaluated the performance of these new matching and distribution method schemes via simulation experiments, and we now describe our methodology. First we describe the network topology. We adopted the GT-ITM package [17] from Georgia Institute of Technology to generate a network with six hundred nodes according to a hierarchical scheme. We borrow their terminology here. First, three *transit blocks* were generated, with an average of five *transit nodes* in each block. Each transit node was connected to two *stubs* on average, each stub having an average of twenty nodes. We will present our empirical studies based on this network (Figure 3).

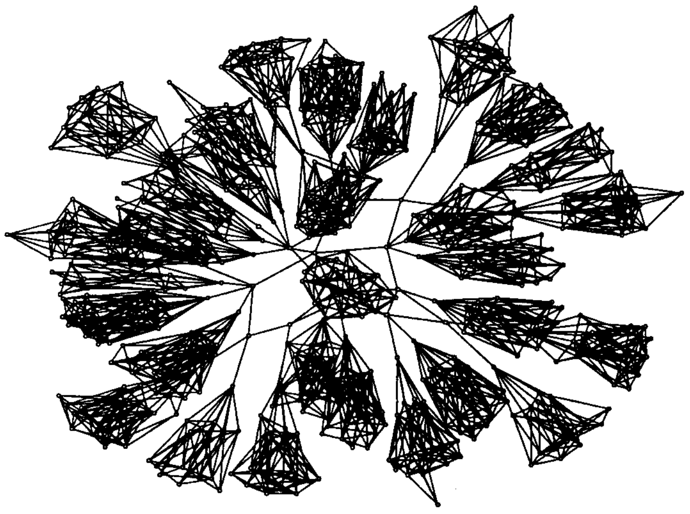

Figure 3. The Generated Network Topology

For a given network topology, we generated subscriptions for each node. We first generated one thousand subscriptions with a $\{40\%, 30\%, 30\%\}$ breakdown for the three transit blocks. Within each transit block we used a Zipf-like distribution [9] for the number of subscriptions between all the connected stubs. Subscriptions were distributed according to another (common) Zipf-like distribution within each stub.

The generated interval subscriptions were of the form $\{bst, name, quote, volume\}$. The first field, *bst*, which stands for *buy*, *sell* and *transaction*, took value B, S and T with probabilities 0.4, 0.4, and 0.2, respectively. The center of the interval for the *name* field followed a normal distribution, with mean centered around the points specific to transit block number (3, 10 and 17), and standard deviation of 4. The length of this interval also followed a Zipf-like distribution. The intervals for the *quote* and *volume* fields were generated according to the same parametric distribution with different parameters. This parametric distribution took values as follows:

$*$	with probability q_0,
$[n, +\infty)$	with probability q_1 and $n \sim N(\mu_1, \sigma_1)$,
$(-\infty, n]$	with probability q_2 and $n \sim N(\mu_2, \sigma_2)$,
$[n_1, n_2]$	otherwise, the center of the interval $\sim N(\mu_3, \sigma_3)$ and interval length following a Pareto (c, α) distribution.

The parameters were assigned according to the following table:

	q_0	q_1	q_2	μ_1, σ_1	μ_2, σ_2	μ_3, σ_3	c, α
price	0.15	0.1	0.1	9, 1	9, 1	9, 2	4, 1
volume	0.35	0.1	0.1	9, 1	9, 1	9, 2	4, 1

The generation of the subscriptions was intended to mimic the real life scenario that people's interests in stocks are centered around the current prices (Figure 4 (a)), the popularity of the information for different stocks has a Zipf-like distribution (Figure 4 (b)), and the popularity of the participants also has a Zipf-like distribution.

The publications, points in the subscription space, were generated according to a mixture of multivariate normal distributions. The different peaks in the multivariate normal distributions represent multiple hot spots where events are published more frequently. We studied three scenarios, mixtures of one, four and nine multivariate normal distributions, respectively. Writing $N(a, b)$ for a distribution with mean a and standard deviation b, the single mode multivariate normal distribution were $N(1, 1)$, $N(10, 6)$, $N(9, 2)$ and $N(9, 6)$ for each of the four dimensions. The four (two by two) mode distribution was constructed by sampling independent mixtures of multivariate normal distributions in each dimension. In the first and fourth dimensions we used $N(1, 1)$ and $N(9, 6)$, respectively. The second dimension was a normal random variable $N(12, 3)$ with probability 0.5 and $N(6, 2)$ with probability 0.5. The third dimension was a normal random variable $N(4, 2)$ with probability 0.5 and $N(16, 2)$ with probability 0.5. For the nine (three by three) mode distribution, the parameters for the first and fourth dimensions remained the same. The third dimension was $N(4, 3)$ with probability 0.3, $N(11, 3)$ with probability 0.4 and $N(18, 3)$ with probability 0.3. The fourth dimension was $N(4, 3)$ with probability 0.3, $N(9, 3)$ with probability 0.4 and $N(16, 3)$ with probability 0.3.

It should be noted that this experimental framework is flexible enough to accommodate other probability distributions for the subscriptions and publications.

In the following study, we constructed 1000 subscriptions for the network with 600 nodes generated from the GT-ITM package.

5.1. Data analysis

In order to obtain a better understanding of the publication and subscription environments in real applications, we analyzed the stock trading data from the New York Stock Exchange. This data was generated on September 24, 1999. We studied the price distributions of all the trades by normalizing the price of a stock by its opening price. The overall price distributions is shown in Figure 4(a). This distribution can be approximated reasonably closely by a normal distribution. We further collected information on the number of trades for each stock and sorted them in decreasing order. Figure 4(b) plots this frequency information against the popularity index, and is approximately a Zipf-like distribution. Figure 4 (c) shows the distribution for the amount of money for the trades, which can also be approximated by a Zipf-like distribution. Figure 5 presents the plots of the normalized price and trade amount distributions for the three most frequently traded stocks. We observe that the price distributions do exhibit bell shapes centering around the averages, which are approximately normal distributions. The amount of money for each trade appears to follow a Pareto distribution, except for several irregular spikes. This stock trading information provides the guidelines and support for our choice of distributions in the experimental studies.

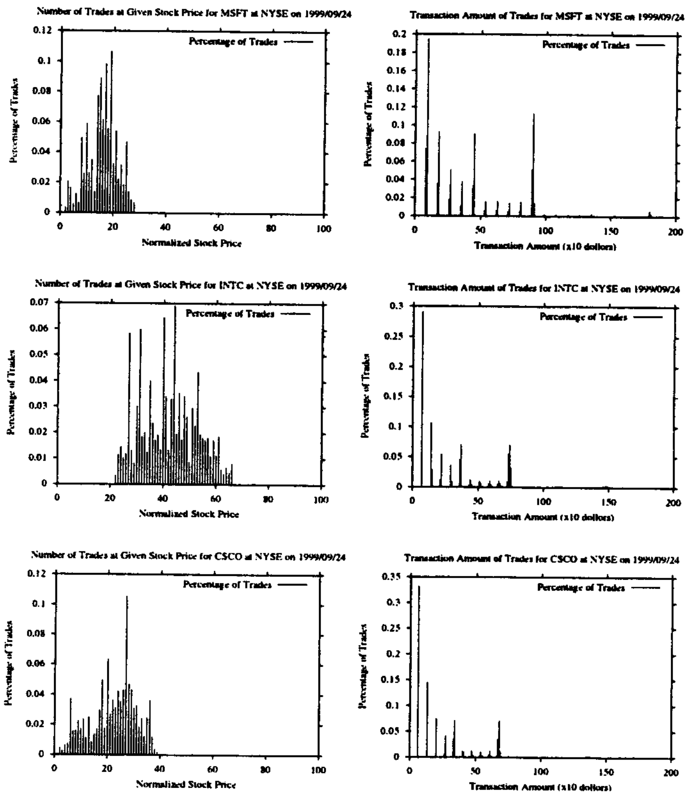

Figure 5. Most frequently traded stocks.

5.2. Experimental results

We performed experiments on the generated testbed to evaluate the performance of the different schemes for forming multicast groups under two multicast frameworks. One approach for implementing the multicast mechanism is network supported multicast. This requires the network routers to have multicast capabilities, to be able to recognize multicast groups and forward the information to the proper members of the group. There are two types of network supported multicast algorithms currently used in routers: *dense mode* and *sparse mode* multicast. The implementations differ in the amount of state information and in the structure of the routing tree. We assume the *dense mode* multicast where the routing tree is a shortest path tree rooted at the publisher. The amount of state information is proportional to both the number of publishers and the number of groups.

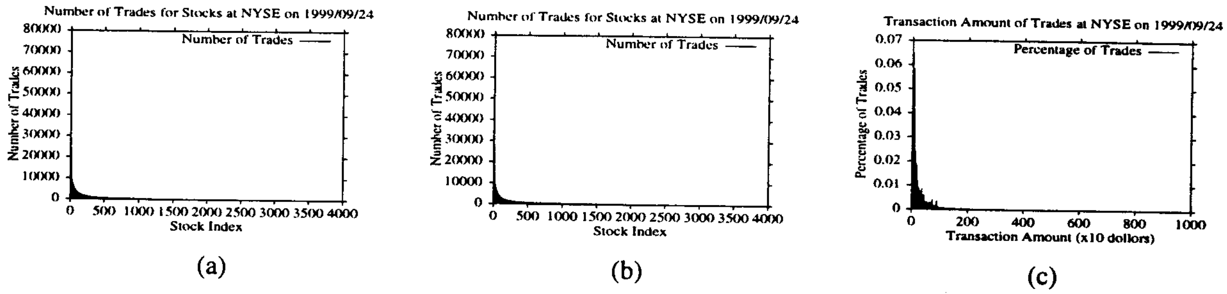

Figure 4. The normalized price, frequency and amount distribution of stock trades

In order to measure algorithm performance, the cost of communication was computed by summing up edge costs (generated by the GT-ITM package) on the links on which communication took place. We normalized the costs to make comparisons easier. Thus the vertical axis in most plots shows "improvement percentage" over unicast. In other words, 0% communication cost improvement is achieved by using unicast to deliver each message. 100% cost improvement corresponds to the cost of delivering each message to a multicast group specially formed only of clients interested in this particular message, which is the best possible, and in the worst case requires as many as $O(k^N)$ multicast groups. The goal of clustering algorithms is to get as close to this performance bound as possible, using more no more than a fixed (and small) number of groups.

Figure 6 illustrates the advantage of dynamically deciding whether to use multicast or unicasts. For 11 and 61 multicast groups the figures show how the average cost of delivering a message changes with the threshold value. The algorithm chooses to send a message via unicasts if the ratio of interested subscribers to the total number of subscribers in the corresponding group is below the threshold value. Note that setting the threshold at 0% gives the same results as not using a dynamic distribution method at all. We have evaluated Forgy k-means, pairwise grouping and minimum spanning tree clustering algorithms in our grid-based subscription clustering framework. The Forgy k-means algorithm performs the best in most of the experiments. Although the performance of the different clustering algorithms varies, setting the distribution method threshold at approximately 15% of the interested subscribers seems to give the consistently best improvements. Significant improvement for the Forgy algorithm with the 15% threshold is observed for realistic environments with 9 modes and 11 multicast groups.

It was shown in [15] that Forgy k-means in most cases produces better solutions than all other clustering algorithms considered for the static multicast distribution method. In addition, it was shown that in practice the Forgy algorithm has the shortest running time on a fixed set of input data. Our experiments described in this section show that determining the distribution method dynamically can further improve the solutions produced by the Forgy algorithm.

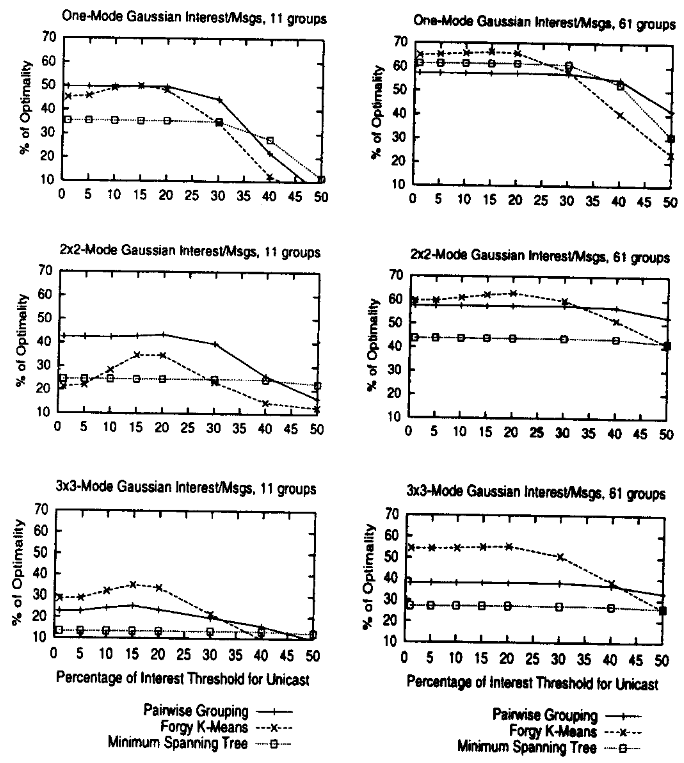

Figure 6. Effects of switching to unicast messages based on the proportion of interested clients (11 and 61 groups).

6. Conclusions and future work

In this paper we have introduced a new algorithm for efficiently handling the distribution of publication events to subscribers in content-based pub-sub systems. In particular, we have devised the so-called *distribution method* problem and analyzed the associated performance improvements.

It would be nice to have some theoretical and practical measures which could help determine how efficient a multicast group has to be in order to actually employ it. A highly inefficient publication multicast event for which most subscribers would have to filter out the results should clearly be handled by multiple unicasts. The question is where to draw the line on this. We leave this for future work.

References

[1] M. Adler, Z. Ge, J. Kurose, D. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. *Proceedings of the 9th IEEE International Conference on Network Protocols*, 2001.

[2] C. Aggarwal, J. Wolf, P. Yu, and M. Epelman. Using unbalanced trees for indexing multidimensional objects. *Knowledge and Information Systems*, 1:309–336, 1999.

[3] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, Atlanta, May 1999.

[4] K. Almeroth. The evolution of multicast: From the mbone to inter-domain multicast to internet2 deployment. *IEEE Network*, January/February 2000.

[5] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. *Proceedings of the 19th International Conference on Distributed Computing Systems*, May 1999.

[6] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. *Technical report, INRIA*, 2000. http://wwwcaravel. inria.fr/pereira/matching.ps.

[7] R. E. Gruber, B. Krishnamurthy, and Panagos. The architecture of the ready event notification service. *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[8] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the 20th VLDB conference*, Santiago, Chile, 1994.

[9] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

[10] B. Levine, J. Crowcroft, C. Diot, J. Garcia-Luna-Aceves, and J. Kurose. Consideration of receiver interest for ip multicast delivery. In *Proceedings of Infocom 2000*, Tel Aviv, March 2000.

[11] NEONet. New era of networks inc, neonet. http://www.neonsoft.com/products/NEONet.html.

[12] M. Oliveira, J. Crowcroft, and C. Diot. Router level filtering for receiver interest delivery. In *Proceedings of the NGC 2000 on Networked Group Communications*, Stanford, November 2000.

[13] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, New York, April 2000.

[14] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. *Symposium on Internet Technologies*, March 2001.

[15] A. Riabov, Z. Liu, J. Wolf, P. Yu, and L. Zhang. On the clustering of rectangular preference sets in content-based pub-sub systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems*, Vienna, July 2002.

[16] T. Wong, R. Katz, and S. McCanne. An evaluation of preference clustering in large-scale multicast applications. In *Proceedings of IEEE Infocom 2000*, Tel Aviv, March 2000.

[17] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom 1996*, San Francisco, 1996.

A. Clustering algorithms

In this appendix we shall briefly describe the clustering algorithms that we used for our experiments. We illustrate our general framework with the detailed description of the K-means algorithm. More details and performance results for these clustering algorithms can be found in our previous work [15].

A.1. General presentations of the clustering algorithms

In Figure 6 we demonstrate the effect of switching to unicast on the groups formed by subscription clustering algorithms, which we call *Forgy k-means, pairwise grouping* and *minimum spanning tree*. Here we continue to use these brief names for subscription clustering algorithms initially introduced in [15].

All three algorithms partition the event space Ω into $(n+1)$ non-overlapping subsets for an allowed number of multicast groups n. In other words, n subsets $S_1, .., S_n$ are formed by the algorithm, and the remaining subset includes the rest of Ω: $S_0 = \Omega \setminus \bigcup_{q=1..n} S_q$. For each set $S_q, q = 1..n$, a multicast group is formed consisting of all subscribers who have subscriptions overlapping this subset. In other words, the multicast group M_q is:

$$M_q = \{v_i \in V_S : \exists j : b_{ij} \cap S_q \neq \emptyset\},\ q = 1..n.$$

Given this set of multicast groups and a new publication vector ω the matching algorithm proceeds as follows. If $\omega \in S_0$, then the publication is delivered using unicast messages. Otherwise there exists unique value of q, $1 \leq q \leq n$, such that $\omega \in S_q$.

It is clear that all subscribers interested in receiving message ω are in the group S_q, but not all members of this group may be interested in receiving this message. The S-tree algorithm is used to build a list s of interested subscribers for a given message ω. If this list is empty, the publication will be not sent. If it is not empty, the size of this list can be used to decide whether the publication should be delivered using unicast messages or using a multicast message to the group S_q. The algorithm can decide to send unicast messages if the proportion of interested subscribers is below a certain fixed level t, i.e. if $|s|/|S_q| < t$.

A.2. Forgy k-means cell clustering algorithm

This algorithm applies a standard k-means clustering algorithm to cells of a regular grid. A special distance function is used to minimize the expected number of wasted messages during multicasts. The k-means clustering algorithm is proven to converge to a local optimum, although we cannot give a polynomial bound on the number of iterations it will require. We therefore artificially limit the maximum number of improvement iterations the algorithm can make, as well as the number of cells that it works with. (This constant T was set to 200 in our experiments.)

The use of k-means algorithm has been considered in literature for point preferences [13, 16]. In our previous paper [15] we extended this approach for more general rectangular interest sets. We also have shown that in many cases k-means produces the best solution among the algorithms, while also having the smallest running time.

Given that the algorithm is allowed to form at most n groups, preprocessing is done according to the following steps:

```
Step 0. Form a grid $G = \{g_x\}$ in the event space $\Omega$.
          For each cell $g_x \in G$ form set of subscribers $l(g_x)$.
          Let $h$ be the list of $T$ cells $g_x$ (for fixed $T > n$)
             having the largest values of $p_p(g_x)n(g_x)$
             among the cells in $G$.
Step 1. Initialize $n$ groups in the set of groups $\mathcal{S}$
             by creating $n$ groups, each containing one element
             for the first $n$ elements of $h$.
          Assign the rest of the elements of $h$ to
             closest clusters already in $\mathcal{S}$.
Step 2. For each element of $h$:
             if it is not the only element in its cluster
                remove it from its cluster and place into
                the closest cluster from $\mathcal{S}$.
          Compute $l(\cdot)$ for each of $n$ clusters as union of $l(\cdot)$
          sets of participating cells, and update EW (see below).
Step 3. Repeat Step 2 until cluster membership stabilizes
          or the maximum number of iterations is reached.
Step 4. $\mathcal{S}$ contains the required set of subsets of $\Omega$.
```

The grid G is formed by choosing at most C adjacent non-overlapping intervals of equal length in each dimension, so that the resulting grid covers all interest rectangles b_{ij}. Clearly the grid has at most C^N cells.

The list of subscribers $l(g_x)$ interested in the set $g_x \subseteq \Omega$ is defined as the set of subscribers who have non-empty interest intersections with the set g_x:

$$l(g_x) := \{i : \exists j : b_{ij} \cap g_x \neq \emptyset\}.$$

In steps 1 and 2 of the clustering algorithm we need to compute distances between a cell and a group of cells. This distance is defined as the amount of increase in the expected number of wasted messages after the cell is added to the group.

The value EW of the expected waste can be defined recursively. EW of a single cell is 0. When a cell g_x is added to a group of cells $\mathcal{G}$, the expected waste can be calculated as $EW_{new} :=$

$$\frac{EW_{old}p_p(\mathcal{G})[1 + |l(g_x) \setminus l(\mathcal{G})|] + p_p(g_x)|l(\mathcal{G}) \setminus l(g_x)|}{p_p(g_x) + p_p(\mathcal{G})}.$$

A similar distance function was proposed in [13] for point preferences.

A.3. Pairwise grouping and minimum spanning tree

Our k-means algorithm applies a standard partitioning clustering algorithm to the set of grid cells using a customized distance function. The other two clustering algorithms we consider employ the following different but standard techniques to the same problem.

The *pairwise grouping* algorithm was studied in [13] for the point preferences case, and we analyze its performance in a more general rectangular preferences setting in [15]. Our experiments suggest that although this algorithm can achieve better performance than k-means, its running time characteristics are significantly worse.

Pairwise grouping starts with the set of the T highest-weight cells, similar to k-means, and replaces a pair of closest cells with their combination. This step is repeated until only the required number n of combinations remain.

The *minimum spanning tree* algorithm did not perform as well as the others in our experiments, but its running time characteristics are much better than those of pairwise grouping. In fact, the minimum spanning tree algorithm is a simplified version of pairwise grouping. Instead of recalculating distances each time a pair is grouped, the algorithm computes all distances once.

Using a graph representation of the problem, where nodes correspond to the T cells, and distances between nodes are computed according to the expected weight function, the algorithm introduces edges connecting the nodes one by one in order of increasing distance, starting with the shortest, until exactly n connected components are formed.