Towards Approximate Event Processing in a Large-Scale Content-Based Network

Yaxiong Zhao and Jie Wu
Department of Computer and Information Sciences
Temple University
{yaxiong.zhao, jiewu}@temple.edu

***Abstract*—Event matching is a critical component of large-scale content-based publish/subscribe systems. However, most existing methods suffer from a dramatic performance degradation when the system scales up. In this paper, we present *TAMA* (TAble MAtch), a highly efficient content-based event matching and forwarding engine. We consider range-based attribute constraints that are widely used in real-world applications. TAMA employs *approximate matching* in order to provide fast event matching against an enormous amount of subscriptions. TAMA uses a hierarchical indexing table to store subscriptions. Event matching in TAMA becomes query to this table, which is substantially faster than traditional methods. Additionally, the false positive of matching events in TAMA can be adjusted by tuning the size of the matching table, which makes TAMA favorable in practice. We implement TAMA as a forwarding component in *Siena* and conduct extensive experiments with realistic settings. The results demonstrate that TAMA has a significantly faster event matching speed compared to existing methods, and only incurs a small fraction of false positive.**

***Index Terms*—Content-based publish/subscribe, approximate event matching, Boolean expression, attribute constraint.**

I. Introduction

Content-based publish/subscribe networks (CBNs) have extensive use in systems monitoring [1], Web advertisement [12], data query in wireless sensor networks [19] and information filtering [27]. A highly-efficient *event matching and forwarding* engine is a critical component of CBNs. Since the subscriptions of all clients are broadcasted in the network, brokers need to process an enormous amount of subscriptions in a large-scale CBN. It becomes extremely challenging for brokers to sustain a fast event processing speed under such stress. To deal with this problem, a lot of matching and forwarding algorithms are proposed [3], [5], [6], [8], [11], [12], [26], [27], [28]. However, all these proposals use *exact matching*, which requires that for each event, exactly all matched subscriptions must be found. In the worst case, all subscriptions might need to be examined, which results in a processing time linear to the number of subscriptions. Many techniques are proposed to accelerate this process by shortcutting the matching to save unnecessary computation, as shown in [8], [11]. However, it is unlikely to break the complexity barrier enforced by the nature of the algorithms, which limits the system scalability when the network grows larger.

On the other hand, introducing a small amount of error in the event matching brings about new algorithms that may overcome the complexity barrier. This is what we proposed to do in TAMA. Although the idea of *approximate event matching* is not new [26], our contribution is a novel data structure that utilizes the idea of *hierarchical discretization*, and provides controllable false positives in event matching.

In this paper, we consider only *range constraints* (constraints for short). A range constraint is expressed as a 4-tuple of $\{attribute\_name, lower, upper, type\}$, which is equivalent to the Boolean expression of $\{attribute\_name \in [lower, upper]\}$ or $\{attribute\_name \geq lower\} \wedge \{attribute\ name \leq upper\}$. Single-sided constraints, like $\{attribute\_name, operator, val, type\}$, are implicitly converted to range constraints by including a maximum or minimum attribute value.

In order to achieve fast matching, TAMA uses a discrete data structure to index range constraints, which is similar in spirit to MICS [13]. A straightforward indexing method is to separate the whole attribute space into indexed sub-cells. Each attribute constraint can then be composited by multiple such cells, and is encoded as a collection of the corresponding cell IDs. It is obvious that the number of cell IDs is linear to the width of the interval of each constraint, which results in a prohibitive memory consumption when a low false positive is required. In order to overcome this limitation, TAMA employs a *hierarchical indexing* approach, which helps it achieve a controllable false positive using a limited amount of memory space. The number of cell IDs generated by this process is $O(log_2\frac{1}{\sigma})$, where σ is the required false positive rate. TAMA stores a hierarchical associative map $\langle cellID, subID\rangle$. This structure is obtained for each attribute that appears in the system. Ultimately, TAMA builds a three-layer index table. The matching of events becomes a table lookup and a checking using the counting algorithm [4], [27]. Actually, TAMA stands for TAble MAtch, which is named after its matching operation.

TAMA has the following properties:

- Storage consumption grows in $O(|C| \times log_2\frac{1}{\sigma})$, where C is the set of all constraints and $|C|$ is its cardinality. Note that this does not mean that TAMA's storage is $log_2\frac{1}{\sigma}$ times the raw storage;
- Controllable false positive: false positive rate reduces by half when using $O(|C|)$ more memory;
- Matching time of an event grows linearly with the number of matched constraints.

These properties are proven to be able to provide a faster event processing speed in large scale systems with acceptable

memory consumption, which is demonstrated in our analysis and experiment results. Our contributions in this paper are as follows:

- We present TAMA, an approximate matching and forwarding engine for large-scale CBNs;
- We analyze TAMA's performance in terms of event matching speed and memory consumption of the indexing table, and verify them through extensive experiments;
- We implement TAMA's matching algorithm as a forwarding component of $Siena$ [6] and design a prototype system.

The rest of the paper is organized as follows: Section II presents our content matching model and the overview of TAMA; Section III presents the design and implementation of TAMA; Section IV describes the implementation of the prototype system; Section V analyzes experiment results; Section VI discusses relevant previous work; Section VII concludes this paper.

II. Model and overview

Following the naming convention of $Siena$ [8], an *primitive attribute constraint*, or *primitive constraint*, is a Boolean expression expressed by a 4-tuple of $\{name,\ operator,\ value,\ type\}$. The operator can be any of $\{<, \leq, =, \neq, \geq, >\}$. The value types are {*integer, double, string*}. We consider *range constraints*, which is a satisfiable conjunction of two primitive constraints and is expressed as a 4-tuple $\{name,\ lower,\ upper,\ type\}$. It has the semantic of $\{name \geq lower\} \wedge \{name \leq upper\}$. For example, the subscription $\{temperature,\ 30,\ 100, double\}$ will be matched by events of which the temperature is between $[30, 100]$.

Range constraints are assumed to be in inclusive form and are normalized to $[0, 1]$. All ranges of floating-point numbers are implicitly treated as inclusive intervals. Due to the nature of continuous distributions, this will not cause problems. For integer values, an open interval is transformed to an inclusive one by reducing its end points' values. For example, $(1, 100)$ becomes $[2, 99]$. Single sided constraints are converted into a range constraint by including a lower or upper bound value. For example, $\{temperature,\ <,\ 100\}$ becomes $\{temperature,$ -273.15, $100\}$, where -273.15$°C$ is the lowest temperature (absolute zero temperature) allowed in the system. An equality constraint is a primitive constraint with operator $=$. They are defined only for $integer$ and $string$ values. *Equality constraints* cannot be converted to range constraints.

A *subscription* is a conjunction of multiple range constraints and equality constraints. It is also called *filter* in the context of content-based routing. Each subscription has a unique **subID**. A *predicate* is a disjunction of multiple subscriptions. Each *interface* is associated with a predicate composed of subscriptions that are received from this interface. An *event*/*message* has *attribute assignments* that specify the values of all its defined attributes. An event matches a constraint only if it has an attribute assignment that satisfies the constraint. An event matches a subscription if its assignments satisfy all the constraints of the subscription. An event are forwarded through the interface of which at least one subscription is matched by it. The responsibility of the event matching and forwarding engine is to find such interfaces for any incoming event.

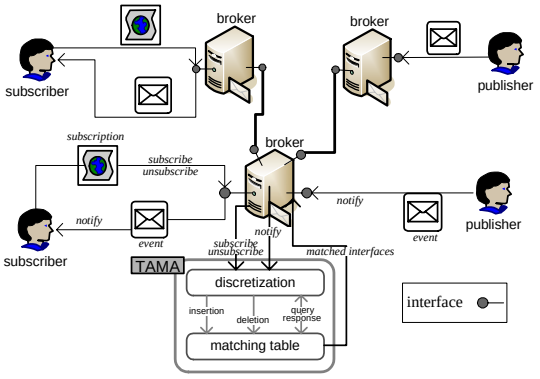

Fig. 1. A distributed pub/sub system where TAMA is working on all brokers.

We give an overview of TAMA in Fig. 1. The figure shows a canonical distributed content-based publish/subscribe (pub/sub for short) system. TAMA runs on each broker. Clients can *subscribe* and *unsubscribe* their subscriptions to TAMA. Before a subscription is being stored into the matching table, all its constraints are transformed into discrete indexes by the *discretization* module. TAMA employs a *hierarchical discretization* to index subscriptions (will be discussed in the next section). The benefit of this indexing structure is that it provides a controllable false positive for event matching with limited memory consumption. An event's attribute assignments are transformed in a similar way also by the discretization module. The resultant indexes are then used as keys to retrieve matched subscriptions from the **match_table**. Finally, the correct interfaces for each incoming event are returned.

III. Design and implementation of TAMA

In this section, we first describe TAMA's approximate matching table. We then discuss how to do event matching using this data structure. Finally, we present several optimizations used in the implementation of TAMA.

A. Approximate matching table

The basic idea of TAMA's matching table is to organize (range) constraints into an indexing structure, so that matched constraints can be efficiently retrieved for each event. The counting algorithm [4], [27] is then used to get the matched subscriptions. A straightforward method of indexing a range constraint is to separate the content space into *cells* and use the IDs of the cells intersected with the constraint to represent it. This method is called *discretization*, as shown in Fig. 2. A *map* of cell IDs and **subID**s is then built. This operation results in $O(W_{sub})$ cells and a maximum false positive of $\frac{W_{cell}}{W'_{sub}}$, where W_{sub}, W'_{sub}, and W_{cell} are the widths of the input range, the resultant ranges after discretization and the cell, respectively. An obvious problem of this approach is that reducing false positive requires a substantial increase of the storage memory. For example, in the fine-grained discretization in the bottom

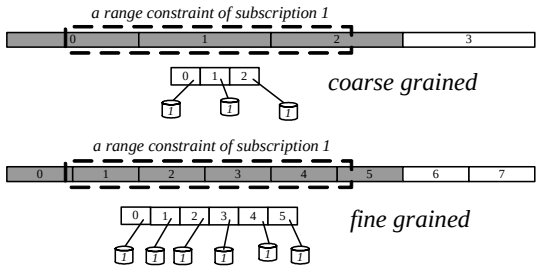

Fig. 2. A straightforward indexing method, which results in irreconcilable conflict between the memory consumption and the false positive.

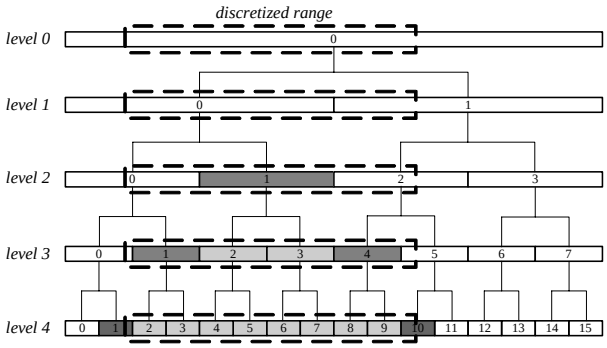

Fig. 3. Discretization of a range on 5 levels with different granularities. The heavily-shaded cells are obtained at each level; and the lightly-shaded ones are the already covered cells of the last level.

of Fig. 2, 6 cell IDs are obtained, which is twice of that in the coarse-grained case. Generally, to obtain a false positive that is ϕ times that of the original discretization, $\frac{1}{\phi}$ times the original memory is needed. This makes it expensive to achieve a low false positive using this approach.

In order to resolve this problem, we propose *hierarchical discretization*. Fig. 3 illustrates the idea. First, the entire content space of each attribute is separated multiple times with different *granularities*, which is represented by the concept called *level*. Specifically, at the i-th level, the attribute space is separated into 2^i cells, and each cell at the i-th level will be separated into two cells at the $(i+1)$-th level. This can be seen from Fig. 3, where a 5-level discretization is applied.

The algorithm is depicted in Algorithm 1. The illustration of the algorithm is also given in Fig. 3. We first find the level at which at least one cell is contained in the input range ($lines\ 4{-}9$), which is level 2 in Fig. 3. This level is denoted by $start$. The remaining portions of the input range, represented by $[l, left]$ and $[right, r]$, are then discretized at higher levels ($lines\ 10{-}18$), which is level 3 in Fig. 3. Here, $left$ and $right$ are the left and right ends of the range that are produced thus far. At the maximum level, level 4 in Fig. 3, additional ranges are included to finish the discretization with a false positive.

Since the resultant ranges contain parts of the content space that are not in the original range, Algorithm 1 causes false positives in event matching. We define the *effective discretization level* to be $(m - start + 1)$, which is the number of levels spanned from the starting level through the maximum level. An important property of Algorithm 1 is that at most two cells are produced at each level. This directly follows from the discretization process shown in Fig. 3. Based on this property, we obtain a theorem about the false positive rate (FPR):

Theorem 1: The FPR of hierarchical discretization is reduced at least by half by using one more level of discretization.

Proof: Since the maximum range width is at least the width of the starting level: W_{cell}^{start}, and the falsely included range is at most twice the width of the cell at the maximum level m: W_{cell}^{m}, the false positive is at most $\frac{2 \times W_{cell}^{m}}{W_{cell}^{start} + 2 \times W_{cell}^{m}}$. Assuming uniform separation of the content space at all levels, the false positive is at most $\frac{1}{2^{m-start-1}+1}$. When $(m - start - 1)$ is large, the FPR becomes $\frac{1}{2^{m-start-1}}$. Therefore, increasing one level in discretization will increase m by 1, which will reduce the false positive by half. The theorem is proved. Note also that the effective discretization level, which is $(m - start + 1)$, determines the false positive of discretization, which is found to be $\frac{1}{2^{m-start-1}+1}$. ■

We give the illustration of TAMA's matching table in Fig. 4. Using Algorithm 1, we can insert a subscription into the *approximate matching table*. The difference from the straightforward approach is that an additional layer, discretization level, is included in the table. TAMA uses an attribute-wise indexing structure. All attributes are organized into a linear table. For each attribute constraint, a set of cell IDs are obtained after discretization. These IDs become indexes of the 3-layer matching table $\{attribute\ name \rightarrow level\ number \rightarrow cell\ ID \rightarrow sub\ ID\}$. A 3-tuple of $\{attribute\ name, level\ number, cell\ ID\}$ is obtained for each constraint. The **subID** of the subscription, along with the tuple of $\langle attribute\ name, level\ number, cell\ ID\rangle$, are inserted into the matching table. There is another table called **attr_count_table** that stores the number of constraints for each subscription, which is used in the matching algorithm to determine if a subscription is matched.

1) Memory consumption: We assume that the effective discretization level of all constraints is l. As explained above, at most 2 cell IDs need to be stored at each level. Each cell ID associates with one subscription ID. We use 4-byte integers to represent cell and subscription IDs, so each attribute constraint needs at most $(8 \times l)$ bytes to store the resultant IDs. In experiments, we put a maximum of 10 million of constraints

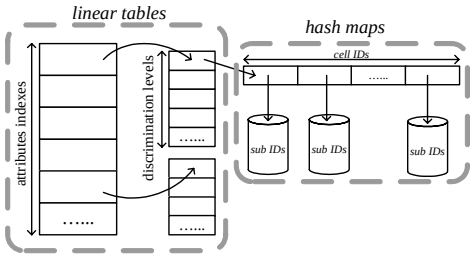

Fig. 4. The structure of the matching table. The first two layers are linear tables; the last layer is a hash map, which is fast for indexing sparse key-value pairs.

Algorithm 1 Range discretization

```
1: input: m := maximum level; [l, r] := input range;
2: output: map⟨int, set⟨int⟩⟩ ID;
3: start = left = right = 0;
4: for level := 0 : m do
5:    if there is a set of cells c contained by [l, r] then
6:       ID[level].union(c);
7:       start = level;
8:    end if
9: end for
10: left = ID[start].min().left();
11: right = ID[start].max().right();
12: for level := start + 1 : m − 1 do
13:    if there is a set of cells c contained by [l, left] and
       [right, r] then
14:       ID[level].union(c);
15:       left = ID[level].min().left();
16:       right = ID[level].max().right();
17:    end if
18: end for
19: Find cells c that contains [l, left] and [right, r]
20: ID[m].union(c)
```

Algorithm 2 Event matching

```
1: input: event e
2: output: matched subIDs ID
3: matchConsCount := {,}
4: min := match_table.minLevel()
5: max := match_table.maxLevel()
6: for all attr ∈ e.attributes() do
7:    for level := min : max do
8:       cellID = attr.discretize(level)
9:       for all id ∈ matchIds do
10:         ++ matchConsCount[id]
11:      end for
12:   end for
13: end for
14: for all ⟨id, count⟩ ∈ matchConsCount do
15:   if attr_count_table[id] == count then
16:      ID.union(id)
17:   end if
18: end for
```

in the table and use $l = 10$; the total memory assumption is $1.49G$ bytes in the worst case. This is acceptable considering the current trend of hardware development. Additionally, the actual consumption in real-world applications is likely to be smaller, since it is likely that no cell IDs are produced at certain levels for a constraint, and the same cell IDs are produced for multiple constraints.

2) Insertion and deletion of subscriptions: The insertion of a subscription into this table is straightforward: discretize each constraint and insert its **subID** into the corresponding entry; update the **attr_count_table**. The deletion follows a similar process: If the subscription's **subID** is given, each constraint is first discretized, and the **subID** is removed from all the corresponding table entries. Otherwise, a lookup is performed at all levels to obtain a number of sets of **subID**s. The intersection of all sets gives the **subID** of the subscriptions that has the same discretization results. The raw subscription is then checked to determine the exact subscription that needs to be removed. In the case of multiple identical subscriptions are obtained, TAMA only removes the one associated with the interface from which the removed subscription is received. This guarantees the correctness of the routing. After all these are done, **attr_count_table** deletes the record of the removed subscription as before.

B. Matching algorithm

When an event is fed into TAMA, its attribute assignments are first discretized into cells. Suppose an attribute assignment is in the form of $[attribute = value]$, it is represented by cells containing it at different levels, which uses the same cell separations as the discretization of range constraints. For example, $temperature = 0.7$ (normalized value) is 0 at the 0-th level and 1 at the 1-st level in Fig. 3. Algorithm 2 is then executed to do the matching. The cell IDs are used to look up the matching table to get a set of **subID**s, which have constraints that are satisfied by the assignment. Since cells are contained by subscriptions recorded in each entry of the matching table and the assignment is contained in cells, the assignment must also be contained by the range representing the constraints. In other words, the event satisfies the constraints of the returned **subID**s defined on the attribute name.

This process is done for all attributes at all levels of the **match_table** ($lines\ 6-13$). The above lookup returns a set of **subID**s. The appearance count of each **subID** indicates the number of satisfied constraints of the subscription. The **attr_count_table** is then checked: a subscription is matched by the event if its appearance count is equal to the associated value in the table ($lines\ 14-17$). We formally state its correctness in the following theorem:

Theorem 2: Algorithm 2 returns the IDs of the subscriptions that are matched by the input event with a bounded false positive.

Proof: We first prove that all subscriptions that are matched by the input event are returned by Algorithm 2. An equivalent argument is that the appearance count of the ID of a matched subscription will be identical to the corresponding value in the **attr_count_table**. To show this, we first note that if a subscription has a constraint matched by the event, its ID will be found. We prove this argument by contradiction. Suppose the ID of the matched subscription is not returned, the attribute assignment's cells do not intersect with any one of the constraints at any level. Therefore, this assignment is not inside the constraint, which is contradictory to the assumption. The argument is proved.

Since the **subID**s of different combinations of $attribute\ name$, $level\ number$ and $cell\ ID$ are returned, we

need to prove, that at different levels, the returned **subID**s of any attribute assignment of the input event are non-duplicated. Assume without loss of generality that there are two different levels ($L_0 < L_1$) at which the event's cell IDs C_{L_0} and C_{L_1} return the same **subID** i, then those two cells are produced from subscription i's discretization process. According to the hierarchical separation, C_{L_1} must be contained by C_{L_0}. However, the discretization process does not allow such a situation, since if a cell is chosen at level L_0, all its contained cells at higher levels will not be chosen. This makes the assumption invalid, so the argument is proved. Combining these two arguments, we prove that all matched subscriptions' **subID**s are returned by Algorithm 2.

The bounded false positive follows directly from the analysis in the proof of Theorem 1. Consider a subscription comprising of N_A attribute constraints, for each attribute $A_i \in \{A_0, A_1, ..., A_{N_A}\}$, its false positive rate is σ_i. Since the subscription is a conjunction of constraints, the FPR of subscription matching is $1 - \prod_{i\in[0,N_A]}(1-\sigma_i) \approx \sum_{i\in[0,N_A]}\sigma_i$. We usually choose a fixed σ for all attributes, so it becomes $(N_A \times \sigma)$, which is clearly bounded. ■

A theorem similar to Theorem 1 is given below, which describe the trade-off between memory consumption and the matching FPR:

Theorem 3: The matching FPR of Algorithm 2 is reduced by at least half by increasing one level of discretization.

Proof: As stated in Theorem 1, the matching FPR for each attribute reduces by half when increasing one level of discretization. According to the analysis in the proof of Theorem 2, the FPR of matching is $\sum_i \sigma_i$, which is the sum of all FPRs of all attributes. If the FPR of all attributes can be reduced by half, the overall matching FPR can be reduced by half too. The theorem is proved. ■

1) Handling equality constraints: The value of an equality constraint, along with the ID of the subscription that contains it, are stored into a hash map. Algorithm 2 needs to look up this map and the **match_table** to find the correct **subID**s before applying the counting algorithm before line 14. Note that there is no false positive for the matching of equality constraints.

2) The false positive of interface matching: Since each interface is associated with multiple filters, the probability of an interface is falsely matched by an event is much smaller if multiple subscriptions are matched. This can be verified as follows: suppose an interface inf is associated with multiple filters, of which $filter_i$ has a FPR of fpr_i. If a set of filters $filter_{matched}$ are matched by an event, the probability that this interface is falsely matched is the probability that all filters are falsely matched, which is described as follows:

$$FPR = \prod_{i\in filters_{matched}} fpr_i \tag{1}$$

Since the filters' FPRs are generally very small, the interface FPR can be greatly reduced, which results in desirable routing performance. We propose *dynamic matching* to exploit this property in Section III-E.

3) Complexity: TAMA's match algorithm has a very appealing complexity. It works in two phases: *querying phase* and *updating phase*. In the querying phase ($lines\ 7-14$), all partially-matched subscriptions are returned. The truly-matched subscriptions are found in the updating phase ($lines\ 15-17$). In the querying phase, all attributes and discretization levels are checked to obtain subscriptions' IDs. Using *hash map* results in a constant time complexity in retrieval (assuming a perfect hash function is available). The overall time complexity will be $O(N_A N_L)$, where N_A is the number of the attributes of the event and N_L is the number of levels that have been checked in **match_table**.

In the updating phase, the algorithm updates the number of matched constraints for all partially-matched subscriptions. This operation has a time complexity linear to the number of partially-matched subscriptions, which is at most the number of matched constraints $O(N_{cons})$. So, the overall time complexity of this matching algorithm is $O(N_A N_L + N_{cons})$. For algorithms based on the counting algorithm [4], [27], the updating phase is inescapable. TAMA achieves a fast matching speed due to its fast querying phase.

Algorithm 3 Optimal separation

```
1: input: m := maximum level; cells[0] = {[0,1]};
2: output: cells := cell separations for level 0 through m;
3: for level := 1 : m do
4:    for all cell ∈ cells[level − 1] do
5:       median := median(cell_begin, cell_end);
6:       separate cell into:
         {[cell_begin, median], [median, cell_end]};
7:       Insert these two cells into cells[level];
8:    end for
9: end for
```

C. Optimal discretization under non-uniform distributions of event attribute values

The above analysis assumes uniform distribution of events' attribute values, so uniform separation is used at all levels of the hierarchical discretization. It is obvious that this will work poorly for skewed distributions of attribute values, which are normal for real-world application [4]. Therefore, non-uniform separation is needed. Given a latent distribution of the value of an attribute, which is described by a distribution function $D(x)$, we propose an optimal binary separation process in Algorithm 3.

We need a helper function $median(x, y)$, which gives the median of the distribution function on the interval of $[x, y]$, which is defined as $median(x, y) = D^{-1}(\frac{D(x)+D(y)}{2})$. Using this method, TAMA can guarantee Theorem 3 under non-uniform event content distribution, as determined by the distribution function $D(x)$. This can be proved as follows: According to the definition of $median()$ function, the expected number of events that fall into all cells at a given level is the same. Since a cell at the i-th level is separated into two cells of equal probability at the $(i+1)$-th level, the expected number

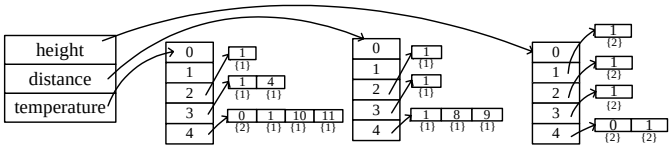

Fig. 5. The matching table of the example in Section III-D.

of events that fall into those two cells are half of that at the i-th level. Following the logic of the analysis in Section III-A, we find that the expected number of falsely matched events is reduced by half with one more level of discretization; this also applies to the false positive. So, this proves that using the separations produced by Algorithm 3 guarantees Theorem 3 for arbitrary attribute value distributions. Note that the separations can be calculated on-the-fly without consume extra memory.

D. Example

A simple example of event matching is described here. We first insert the following two subscriptions into the matching table with maximum level of 4:

1) $\{temperature, 10, 70\} \wedge \{distance, 100, 600\}$
2) $\{temperature, 0, 5\} \wedge \{height, 5, 100\}$

The first step to insert subscriptions is to normalize the attribute constraints. For simplicity of description, we set the value range of each attribute as follows:

- $temperature$: $[0, 100]$
- $distance$: $[0, 1000]$
- $height$: $[0, 100]$

Using uniform discretization, the subscriptions become:

1) $\{temperature, 0.1, 0.7\} \wedge \{distance, 0.1, 0.6\}$
2) $\{temperature, 0, 0.05\} \wedge \{height, 0.05, 1\}$

The resultant matching table is shown in Fig. 5. Given a event with assignments $\{temperature = 40\} \wedge \{distance = 200\} \wedge \{height = 70\}$, which becomes $\{temperature = 0.4\} \wedge \{distance = 0.2\} \wedge \{height = 0.7\}$ after normalization. The discretized event is $\{temperature \in \{0, 0, 1, 3, 6\}\} \wedge \{distance \in \{0, 0, 0, 1, 3\}\} \wedge \{height \in \{0, 1, 2, 5, 11\}\}$, which gives the IDs of the cells on level 0 through 4. The retrieved subscription IDs are $\{1, 1, 2\}$. Since subscription 1 appears twice, which is equal to its constraint count, it is matched by this event. Because subscription 2 appears once, it is not matched. After manual checking, we find that this result of Algorithm 2 is correct.

E. Implementation

1) Matching table organization: The implementation of the matching table aims to reduce memory access time, which determines the lookup speed. The first two levels of the table are implemented as linear tables. In order to convert the strings of attribute names into integer indexes, we maintain a global associative map of attribute names and indexes, which is called **attr_idx_table**. Given an attribute $name$, **attr_idx_table**[$name$] returns the index of the attribute's entries in the **match_table**. When a subscription is first received at a broker, the attribute names of its constraints are substituted by the corresponding indexes to save processing time for operations on other brokers (in a distributed pub/sub network) and later maintenance. The same operation is performed for an event when it is first received by a broker. The discretization levels (0 through $maximum\ level$) can be directly used as indexes. The final associative map of cell IDs and **subID**s are stored into a hash map as shown in Fig. 4.

2) Compact encoding of discretization results: It is very helpful to piggyback discretization results for subscriptions and events in order to reduce processing time, especially when using non-uniform separation. In the case where memory or bandwidth is limited, this naive optimization may not be acceptable. We introduce a compact representation of discretization results of subscriptions based on the unique property of the discretization process. We use a bit-vector to encode an attribute constraint. The first 6 bits are used to encode the index of the first level in discretization. Suppose the number of the first 6 bits is B, there are at most two cell IDs, and each can be encoded using at most B bits. The next trailing bit is used to indicate the number of cell IDs: "0" for 1 and "1" for 2. We further use $2(L-1)$ bits to encode the cell ID produced at two sides of the original range, where $(L-1)$ is the total number of levels used. Finally, a byte is appended before the vector to record the length of the bit-vector. The overall memory consumption is at most: $2 + 2 \times 4 + \frac{2 \times (L-1)}{8} = 10 + \frac{L-1}{4}$.

The decoding works as follows: suppose that the IDs of the left and right end cells at level i are l and r, respectively; if new cells are produced at level $i+1$, their IDs are $2l-1$ and $2(r+1)$ at the left and right sides, respectively. This can be seen from Fig. 3. The compact representation helps TAMA to reduce pre-processing time for subscriptions and events.

For an event's attribute assignments, the encoding is as follows: the first byte encodes the first level, and the next 4 bytes record the cell ID of the first level. Then, $L-1$ trailing bits are used to encode the following cell IDs. Note that the cell produced at the i-th level must be contained in the cell of the $(i-1)$-th level. We use "0" to indicate that the cell at the current level is the first sub-cell of the cell at the previous level, and "1" for the second sub-cell. A byte that records the length of the bit-vector is then appended at the beginning. As shown in Fig. 3, if the ID of an attribute assignment's cell at level i is n, the ID of the first sub-cell at level $(i+1)$ is $2n$ and the second is $(2n+1)$. This rule is used to decode the bit-vector to get the discretized event attribute assignments.

3) Dynamic matching: Eq. 1 gives a natural optimization in distributed content-based networks. We call those brokers directly connected with clients *edge brokers* and those only connected with other brokers *core brokers*. Edge brokers need to store a larger matching table with more levels to preserve a low FPR and reduce the wrong events received by end clients. Core brokers, on the other hand, could store a smaller table with less levels, as long as the event can match multiple filters for each interface. Another type of dynamic matching is to deliberately ignore a certain number

of higher levels of the matching table when at least one of the subscriptions associated with the interface is matched. Since false positives only occur at the maximum level, there will be no false positives in this case. This corresponds to the shortcut techniques used in *Siena* [8].

IV. Prototype implementation

We implement TAMA as a forwarding component in $Siena$ [6], [8]. Our implementation includes 3141 lines of C++ [23] code which implements these two interfaces. We use C++ STL containers and $\_\_gnu\_cxx :: hash\_map$ (a $hash\_map$ implementation provided by GNU GCC) to store TAMA's matching table.

We also implemented a prototype independent of $Siena$'s framework. It contains two programs running at brokers and clients. The broker side program has a **receive(addr)** function that is continuously listening on a local network interface specified by **addr**. The received UDP packet contains the source address that will be associated with the contained subscriptions in the packet as its outgoing interface. The subscriptions contained in the packets are put into the matching table using **insert(sub, addr)** function. **addr** is the associated outgoing interface of **sub**. On the client side, a **subscribe(sub, addr)** function registers the subscription **sub** to the broker residing at the network address indicated by **addr**. An accompanying **unsubscribe(sub, addr)** removes a subscription from the broker located at **addr**. Clients publish events using **notify(event, addr)**, where **addr** is the address of its associated broker. All network addresses are standard IPv4 address and port numbers. The binary layout of subscription and event storage are defined. But we do not give details here for lack of space.

V. Performance evaluation

This section presents the evaluation results. We summarize the results of message matching time, false positives of event and interface matching, processing efficiency and memory consumption. All experiments are conducted on a Linux workstation with 8 3GHz cores and 16GB memory. Parallelism is not used. All attribute values are normalized to $[0, 1]$. The parameters used are summarized in Table I.

parameter name	meaning	value range
$attr\_width$	width of each range constraints	$[0.01 - 0.1]$
N_{attr}	total number of attributes	$1,000$
N^s_{attr}	number of range constraints per subscription	$[1 - 10]$
N^e_{attr}	number of attribute assignments per event	$[100 - 1,000]$
N_{cons}	total number of primitive constraints	$[20,000 - 2 \times 10^7]$
N_{inf}	total number of interface per broker	$[10 - 2 \times 10^7]$
N_{sub}	total number of subscriptions in the matching table	$[1,000 - 10^6]$
N^{inf}_{sub}	number of associated subscriptions per interface	$[1 - 2 \times 10^5]$
N_{msg}	total number of messages sent to TAMA	$[10,000 - 10^8]$
F_e	distribution function of the attribute values of events	$\{uniform,\ pareto,\ power\ law\}$

TABLE I
THE PARAMETERS USED IN EXPERIMENTS.

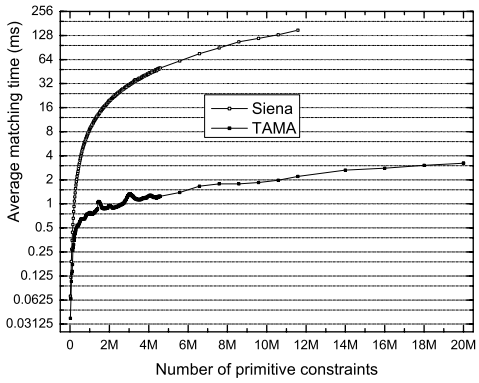

Fig. 6. Matching time per message in the degenerate case where the all matched subscriptions need to be returned for any input event. Each subscription has 10 range constraints with a width of 0.01. Each event has 100 attribute assignments.

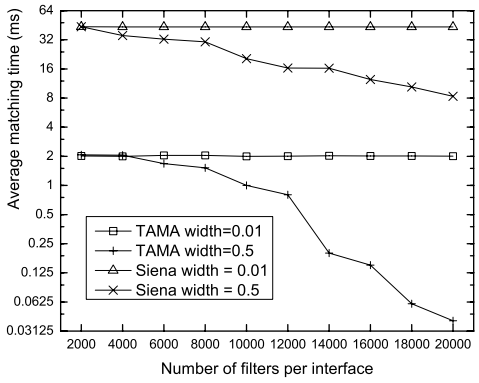

Fig. 7. Interface matching time per message. Each subscription has 10 range constraints with a width of 0.01 and 0.5; 20,000 subscriptions, or 4,000,000 primitive constraints, are stored in the matching table.

A. Matching time

Fig. 6 presents the matching time per message in the degenerate situation where each subscription is associated with a unique interface. Uniform distribution is used (the results of Pareto and power law distributions are close, so they are omitted to save space). We let $Siena$ perform 10 rounds of preprocessing. The x-axis represents the number of primitive constraints stored in the matching table. Note that each range constraint is counted as two primitive constraints. The y-axis represents the average matching time per-message in log-scale. In this experiment, the IDs of all matched subscription of every input event are returned.

As shown in the figure, TAMA outperforms $Siena$ with a large gap when the number of constraints exceeds 160k. A detailed profiling reveals that the querying phase of Algorithm 2 becomes a major bottleneck when the number of constraints is small, where TAMA only has a slightly smaller matching time compared to $Siena$. This phase almost spends a constant

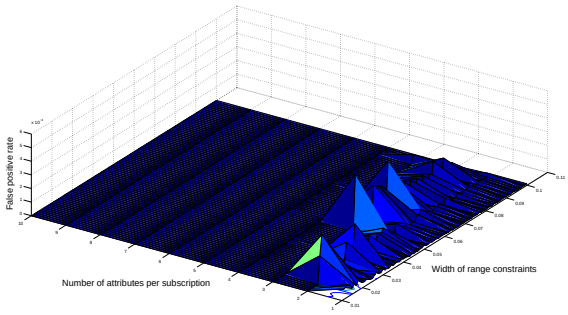

Fig. 8. 3D plot of the FPR of subscription matching. 20,000 subscriptions are stored in the matching table. The number of range constraints per subscription varies from 1 to 10. The width of range constraints varies from 0.01 to 0.1.

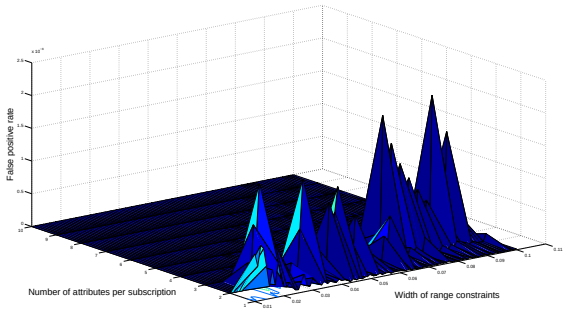

Fig. 9. 3D plot of the FPR of interface matching. The setting is the same as the experiment of Fig 8.

time (theoretically constant if using perfect hashing) when a large number of constraints are stored in the table. The slow growth of the matching time is resulted from more partially matched subscriptions. On the other hand, $Siena$ scanning all subscriptions (with shortcuts to avoid checking a fraction of subscriptions) would result in a linearly growing time complexity, which certainly would produce substantially degraded results when the number of constraints continues to grow. In the extreme case of 20-$million$ primitive constraints, TAMA sustains a per-message matching time of 3.23 milliseconds, which is far superior to $Siena$.

Shortcutting event matching when multiple filters are associated with each interface significantly reduces the matching time, as indicated in [8]. A similar experiment is conducted here, and the results are shown in Fig. 7. TAMA uses dynamic matching of Section III-E to reduce the number of levels needing to be matched. We note that shortcutting only works noticeably for the case that constraint width is 0.5. This is because when constraint width is small, and an input event can only match a very small number of subscriptions, TAMA and $Siena$ are forced to examine all matched subscriptions, so the matching time almost does not change. But, for wider constraints, more subscriptions are matched by the event, which provides more shortcutting to let both algorithms stop matching without checking all subscriptions.

B. False positive

Fig. 8 plots the FPR of subscription matching versus the number of attribute constraints per subscription and the width of each constraint. The FPR is calculated as follows:

$$\frac{matched\_sub\_count - truly\_matched\_sub\_count}{matched\_sub\_count} \quad (2)$$

We input 10,000 events to TAMA and calculate the overall FPR using Eq. 2. The results are presented in Fig. 8. The x-axis represents the number of attributes per subscription; the y-axis represents the width of each range constraint; and the z-axis is the average matching FPR. 20,000 subscriptions are stored in TAMA's matching table. All subscriptions are discretized up to level 20. This means that wider range constraints will have smaller FPRs since their effective discretization levels are larger, which is confirmed in the figure. Also, with the increasing number of attribute constraints per subscription, the FPR increases. Where when it exceeds a certain threshold, the probability that an event can match any subscriptions becomes extremely small, so that no subscriptions are matched at all. In this case, the FPR is 0. Actually, in real world applications, we seldom have more than 4 or 5 attributes. For example, when we search books in a library database, we usually only put release dates and book names in the query box, which is usually sufficient to get desirable results. The FPRs in Fig. 8 are always below 5×10^{-4} or 0.05%, and according to the analysis in Section III-B, the FPR for subscriptions that have 10 constraints is 0.5% in the worst case, which should be acceptable for real world applications. Additionally, since TAMA's approximate matching table preserves the proximity in content space, the falsely delivered events are considered very close to the required contents. This property makes it more favorable than the approximate matching approach used in [16], where an event may match subscriptions that are completely irrelevant to its content.

We present the interface matching FPR in Fig. 9. It is measured in a similar way to Eq. 2. The difference is that we count the number of selected interfaces instead of subscriptions. We feed 10^8 events to TAMA with 20,000 subscriptions, and 2,000 subscriptions are associated with each interface. The FPR is much smaller than that of subscription matching, which is below 2×10^{-8} in all cases and is generally negligible.

C. Subscription processing time

We list the subscription insertion time for 3 methods:

- *Direct insertion*: Each subscription is discretized before insertion;
- *Fully cached*: The discretization results are cached with each subscription and no discretization is needed;
- *Coding*: An encoded bit-vector is piggybacked with each subscription, and a decoding is needed before insertion.

Fig. 10 shows the results. We only present the results of using power law distribution in the optimal discretization. The power law distribution has a scaling factor of 1.0 and an exponent of 2. The reason to omit the results of using Pareto distribution is that it gives an almost identical results as that of the power law distribution. Using uniform distribution produces results that are close to those of the fully-cached

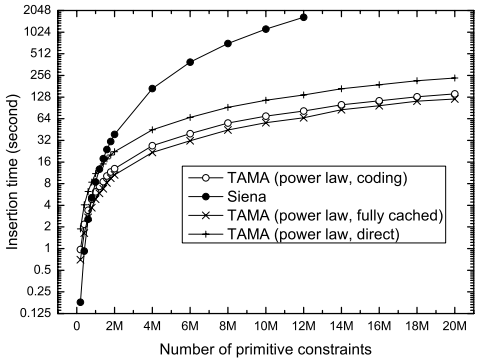

Fig. 10. The insertion time of different amounts of subscriptions that are expressed as the number of primitive constraints. Each subscription is composed of 10 range constraints. The width of each range constraint is 0.01. The effective discretization level is 14.

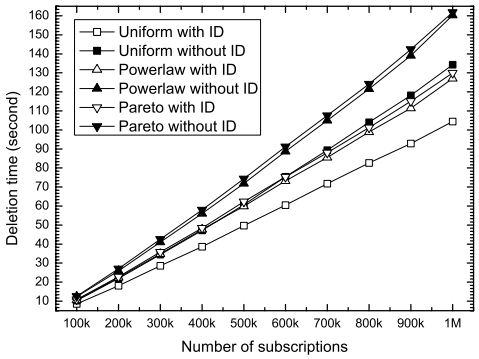

Fig. 11. The deletion time of different amounts of subscriptions. The setting is the same as the insertion test.

and direct-insertion, and using compact coding method does not provide noticeable improvement over direct-insertion since its computation is quite simple.

As shown in Fig. 10, the labels $coding$, $fully\ cached$ and $direct$ correspond to the three methods given above. We can see that using the fully-cached method achieves the fastest insertion speed. Where using the coding provides similar results. Both coding and fully-cached reduce nearly half of the insertion time of direct-insertion. Fig. 11 shows the time used to delete a subscription from the matching table. Deletion is not provided in $Siena$, so no results is given in the figure.

D. Memory consumption

We plot the memory consumption of TAMA in Fig. 12. The raw storage of each range constraint is 16 bytes, which corresponds to 2 $double$ floating-point numbers in our machine. TAMA's memory consumption is measured by the space needed to store all cell and subscription IDs in the matching table. In our implementation, cell and subscription IDs are 4-byte integers. As shown in the figure, the memory consumption grows linearly with the number of constraints. TAMA consumes 3-4 times memory of the raw storage. This is less than the worst case analytic results in Section III-A. Note that the effective discretization level numbers are 14 and 17 for the widths of 0.01 and 0.1, respectively.

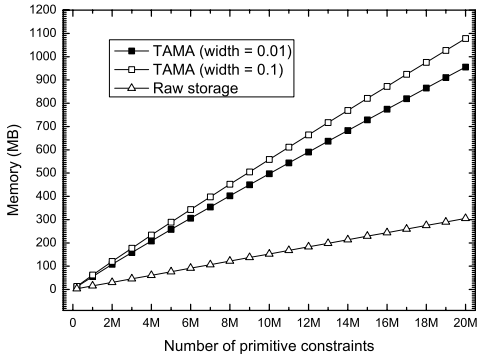

Fig. 12. The memory consumption counted in the memory space needed to store the cell IDs and **subID**s in the matching table.

VI. Related work

Content-based matching and forwarding are active research topics [7], [10], [27]. Except for content dissemination, it has extensive applications in web content caching [4], XML document filtering [9] and online advertisement [12]. In this section, we summarize selective papers on content-representation model and matching and routing techniques.

A. Content representation model

The canonical model uses Boolean expressions of conjunctive normal forms (CNF) to express subscriptions. This model is widely used in [2], [6], [8], [11], [28]. It has the advantages of uniform interfaces and the standard programming model, which makes it easy to implement. Besides, since it has a well-defined structure, many optimizations can be applied to achieve very fast event matching and forwarding speed. However, the problem of exponentially-increasing memory consumption of converting arbitrary Boolean expressions [21], [27] becomes severe and hinders the event processing throughput. The use of arbitrary Boolean expressions draws attention in [3], [5], [12]. Their works demonstrate that evaluating arbitrary Boolean expressions can be made efficient, which is comparable to the canonical model, but has smaller memory consumption. However, it is unlikely to sustain a similar performance in large scale systems of tens of millions of constraints, as we have done in this paper. Further, its excessive processing overhead is not well addressed in a dynamic environment.

B. Matching and routing algorithms

We mainly focus on the matching and routing using the canonical model. It is proved in [18] that content matching is as hard as partial matching [15]. $Siena$fast forwarding (SFF) extends the counting algorithm [4], [27] to the shortcut matching process for the disjunction of multiple subscriptions associated with a single network interface. The main problem of SFF is that it requires complex operations to update the matching table. And as indicated in this paper, its performance degrades substantially as the scale of the system grows. An extension to SFF is presented in [24], which aims to improve the storage and management of subscriptions. MICS [13]

intends to transform multi-dimensional ranges, which are used to express subscriptions into one-dimensional intervals. Although the matching speed is increased, it comes with a prohibitive memory consumption. The authors propose to use interval merging to mitigate this problem, which will result in additional FPRs. The implementation issue of content matching is studied in [11], where several optimizations are proposed to accelerate the execution speed of content matching algorithms.

A Bloom-filter-based matching scheme is presented in [16]. Bloom filters are used to store matched primitive constraints. Encoded subscriptions and intermediate matching results are stored in a novel data structure. An index structure of Boolean expressions is presented in [26] in order to reduce the memory consumption. A routing optimization based on recording intermediate matching results is proposed in [17]. It requires an identical matching table on all brokers in the network. Subscription covering [22] and subsumption [14], [20] reduces routing table size by only storing the most general subscriptions. Another similar approach is called subscription summarization [25], which proposes to use imperfect merging of subscriptions to reduce routing table size.

VII. Conclusion

In this paper, we present TAMA, an approximate matching and forwarding engine for large-scale CBNs. TAMA uses an hierarchical indexing structure to approximate the original subscriptions. This indexing structure provides fast approximate event matching, which causes false positives. Its novelty is that it can control the FPR of matching using a concept called hierarchical discretization, which trades-off between the memory consumption and the FPR of matching. Our analysis proves that TAMA can be carefully tuned to meet extensive performance requirements, i.e., by adjusting the maximum discretization level and using optimal discretization for specific attribute value distributions. Our experiment results show that TAMA outperforms a state-of-the-art technique with a substantial margin, and that the FPR is generally acceptable in all practical cases. We also present the implementation of TAMA in $Siena$, and a set of APIs for a prototype.

Acknowledgments

This research was supported in part by NSF grants CCF 1028167, CNS 0948184 and CCF 0830289.

References

[1] "RTI Data Distribution Service (DDS)." [Online]. Available: http://www.rti.com/

[2] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *Proc. of PODC '99*. New York, NY, USA: ACM, 1999, pp. 53–61.

[3] S. Bittner and A. Hinze, "The arbitrary boolean publish/subscribe model: making the case," in *Proc. of DEBS '07*. New York, NY, USA: ACM, 2007, pp. 226–237.

[4] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *In INFOCOM '99*, 1999, pp. 126–134.

[5] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "Efficient filtering in publish-subscribe systems using binary decision diagrams," in *Proc. of ICSE '01*.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[7] A. Carzaniga and A. L. Wolf, "Content-based networking: A new communication infrastructure," in *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, ser. Lecture Notes in Computer Science, no. 2538, Scottsdale, Arizona, oct 2001, pp. 59–68.

[8] ——, "Forwarding in a content-based network," in *Proc. of SIGCOMM '03*. New York, NY, USA: ACM, 2003, pp. 163–174.

[9] C. Chan, P. Felber, M. Garofalakis, and R. Rastogi, "Efficient filtering of xml documents with xpath expressions," *The VLDB Journal*, vol. 11, no. 4, pp. 354–379, 2002.

[10] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[11] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," in *Proc. SIGMOD '01*, anonymous, Ed. ACM, 2001, pp. 115–126.

[12] M. Fontoura, S. Sadanandan, J. Shanmugasundaram, S. Vassilvitski, E. Vee, S. Venkatesan, and J. Zien, "Efficiently evaluating complex boolean expressions," in *Proc. of SIGMOD '10*. New York, NY, USA: ACM, 2010, pp. 3–14.

[13] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "MICS: an efficient content space representation model for publish/subscribe systems," in *Proc. of DEBS '09*. ACM, 2009, pp. 1–12.

[14] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian, "Subscription subsumption evaluation for content-based publish/subscribe systems," in *Proc. of Middleware '08*. New York, NY, USA: Springer-Verlag New York, Inc., 2008, pp. 62–81.

[15] T. S. Jayram, S. Khot, R. Kumar, and Y. Rabani, "Cell-probe lower bounds for the partial match problem," *J. Comput. Syst. Sci.*, vol. 69, no. 3, pp. 435–447, 2004.

[16] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *Proc. of DEBS '08*, anonymous, Ed. ACM, 2008, pp. 71–81.

[17] ——, "Prefix forwarding for publish/subscribe," in *Proc. of DEBS '07*. New York, NY, USA: ACM, 2007, pp. 238–249.

[18] S. Kale, E. Hazan, F. Cao, and J. P. Singh, "Analysis and algorithms for content-based event matching," in *ICDCSW '05: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 363–369.

[19] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Proc. of SenSys '03*. New York, NY, USA: ACM, 2003, pp. 63–75.

[20] A. M. Ouksel, O. Jurca, I. Podnar, and K. Aberer, "Efficient probabilistic subsumption checking for content-based publish/subscribe systems," in *Proc. of Middleware '06*. New York, NY, USA: Springer-Verlag New York, Inc., 2006, pp. 121–140.

[21] P. R. Pietzuch and J. M. Bacon, "Hermes: A distributed event-based middleware architecture," *Distributed Computing Systems Workshops, International Conference on*, vol. 0, p. 611, 2002.

[22] Z. Shen, S. Aluru, and S. Tirthapura, "Indexing for subscription covering in publishsubscribe systems," in *Proc. of PDCS '05*, 2005.

[23] B. Stroustrup, *The C++ Programming Language*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[24] S. Tarkoma and J. Kangasharju, "Optimizing content-based routers: posets and forests," *Distributed Computing*, vol. 19, pp. 62–77.

[25] P. Triantafillou and A. Economides, "Subscription summarization: A new paradigm for efficient publish/subscribe systems," in *Proc. of ICDCS '04*, 2004, pp. 562–571.

[26] S. E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni, "Indexing boolean expressions," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 37–48, 2009.

[27] T. W. Yan and H. García-Molina, "Index structures for selective dissemination of information under the boolean model," *ACM Trans. Database Syst.*, vol. 19, no. 2, pp. 332–364, 1994.

[28] T. W. Yan and H. Garcia-Molina, "The SIFT information dissemination system," *ACM Trans. Database Syst.*, vol. 24, no. 4, pp. 529–565, 1999.