REIN: A Fast Event Matching Approach for Content-based Publish/Subscribe Systems

Shiyou Qian
Shanghai Jiao Tong University
qshiyou@sjtu.edu.cn

Jian Cao*
Shanghai Jiao Tong University
cao-jian@sjtu.edu.cn

Yanmin Zhu
Shanghai Jiao Tong University
yzhu@sjtu.edu.cn

Minglu Li
Shanghai Jiao Tong University
mlli@sjtu.edu.cn

***Abstract*—Event matching is the process of checking high volumes of events against large numbers of subscriptions and is a fundamental issue for the overall performance of a large-scale distributed publish/subscribe system. Most existing algorithms are based on counting satisfied component constraints in each subscription. As the scale of a system grows, these algorithms inevitably suffer from performance degradation. We present REIN (REctangle INtersection), a fast event matching approach for large-scale content-based publish/subscribe systems. The idea behind REIN is to quickly filter out unlikely matched subscriptions. In REIN, the event matching problem is first transformed into the rectangle intersection problem. Then, an efficient index structure is designed to address the problem by using bit operations. Experimental results show that REIN has a better matching performance than its counterparts. In particular, the event matching speed is faster by an order of magnitude when the selectivity of subscriptions is high and the number of subscriptions is large.**

I. Introduction

The content-based publish/subscribe paradigm is a flexible many-to-many communication model that meets the demands of many modern distributed applications, such as information filtering, selective content dissemination, location-based services, and workload monitoring and management. The publish/subscribe paradigm is attractive in that it realizes full decoupling of the communication parties in time, space and synchronization [1].

Event matching is the process of checking high volumes of events against large numbers of subscriptions and is a fundamental issue for the overall performance of a large-scale distributed publish/subscribe system. In order to distribute the load and be scalable, a content-based publish/subscribe system often uses a network to route and forward subscriptions and events, which consists of multiple connected brokers. Compared with the centralized architecture where all subscriptions and events are sent to a single broker, the distributed system is more flexible and failure-resistant. Whenever an event is received by a broker, event matching is immediately performed, returning the matched subscriptions and their routing information. In a large-scale publish/subscribe system, it is possible that there are millions of subscriptions maintained by brokers and millions of events to be matched every second. Therefore, to improve the matching speed of brokers is of great importance to large-scale content-based publish/subscribe systems.

*Corresponding author.

Different techniques have been employed to improve matching efficiencies in recent years. For example, efficient index structures have been proposed in [2–7]. Matching efficiencies can also be improved by reducing the number of subscriptions through covering, subsumption, merging and summarization of subscriptions [8–12]. However, the matching performance of these methods degrades when the scale of a system grows. In such a system, generally, there are a large number of subscriptions and each subscription consists of a large number of component constraints, which needs more efforts to be spent on checking their satisfactions.

In the paper, we present REIN (REctangle INtersection), a fast event matching approach for content-based publish/subscribe systems. The key idea behind REIN is to quickly filter out unlikely matched subscriptions rather than to determine whether a subscription is matched or not by counting its satisfied component constraints. In our research, it is assumed that each subscription is composed of multiple range constraints and each range constraint is a condition specified on an attribute with a low value and a high value. The range constraint is satisfied if an attribute value is located in the range formed by the low value and the high value. The attributes appearing in events form a high-dimensional space. In this space, a subscription is a high-dimensional rectangle (for short rectangle) and an event is a point. Therefore, the matching problem is equivalent to the point enclosure problem. We enlarge a point (event) into a high-dimensional cube (for short cube) to transform the point enclosure problem into the rectangle intersection problem. An efficient index structure is also designed to address the rectangle intersection problem by using bit operations.

Comparing with other matching algorithms, REIN has multiple features. Firstly, when matching an event, if there are more subscriptions that can match the event, the time to search these matchable subscriptions will increase in most existing matching algorithms. However, the matching time of REIN decreases with the number of matchable subscriptions. Secondly, the distributions of constraint values may cause the index structures of some existing methods skewed, such as in [3] and [6], which affects the matching performance. However, REIN is not affected by the distributions of constraint values. Thirdly, REIN is highly efficient to update (insert or delete) subscriptions in the index structure. Therefore, REIN is also applicable to very dynamic environments.

Extensive experiments were conducted to compare REIN with its counterparts. The parameters that have impacts on the performance of matching algorithms are first identified,

978-1-4799-3360-0/14/$31.00 © 2014 IEEE

including the selectivity of subscriptions, the width of range constraints, the number of subscriptions, and the distribution of constraint values. In the experiments, the number of subscriptions is up to 10 million and the number of constraints in subscriptions is up to 30. Experimental results show that REIN outperforms its counterparts to a large degree. In particular, the event matching speed is faster by an order of magnitude when the selectivity of subscriptions is high and the number of subscriptions is large. Our main contributions are:

- We transform the event matching problem into the rectangle intersection problem.
- An efficient index structure is proposed to support event matching implementation using bit operations.
- The performance of REIN has been evaluated through extensive experiments.

The rest of the paper is organized as follows. Section II discusses related work. Section III defines some basic terms and introduces the background knowledge. The design of REIN is introduced in Section IV. Section V presents the experimental results of performance evaluation. Section VI concludes the paper.

II. Related Work

Improving event matching efficiencies has been drawn great attentions in the last decade and different approaches have been proposed. These approaches can be roughly classified into two categories, (i) proposing new index structures and (ii) reducing the number of subscriptions using covering, subsumption and merging.

A. Proposing New Index Structures

New index structures proposed to improve matching efficiencies include those presented in [2, 3, 5, 6, 18]. The matching procedure based on these structures generally consists of three steps. In step 1, all satisfied constraints are quickly obtained through the index structures. In step 2, counting algorithms are used to sum up the number of satisfied constraints for subscriptions. In step 3, the number of satisfied constraints is compared with the number of constraints contained in subscriptions to judge whether a subscription is matched. The index structures of SIENA [3] and TAMA [6] are two representatives. The index structure of SIENA is a two-level forwarding table which is applicable to the situation where subscriptions change infrequently since every modification of subscriptions leads to rebuilding the whole table. TAMA constructs a two-layer matching table for approximate matching. One problem with these algorithms is that although a subscription will finally be decided as an unmatched one, it is counted multiple times in terms of its satisfied component constraints in the process of matching. Therefore, the time complexity of these counting algorithms is linear with the number of satisfied constraints. H-TREE is a hash table in nature by putting similar subscriptions in the same buckets [7]. When matching events, most unmatched subscriptions are filtered out to improve the matching speed. H-TREE is highly efficient in the case where the width of range constraints is smaller than the width of cells divided on the attributes' value domain.

B. Reducing the Number of Subscriptions

Besides proposing new index structures, reducing the number of subscriptions maintained by brokers is another way to improve the matching speed. The merging, summarization, covering, and subsumption of subscriptions are utilized to reduce the number of subscriptions [4, 8–12, 19–24]. Among the relationships of subscriptions, subsumption is the most efficient way to reduce the number of subscriptions [4, 10, 12, 21–23]. Space filling curves, such as Hilbert, are used to represent the content space for efficient checking of subscription subsumption [4, 23]. The merging and summarization of subscriptions are used to reduce the routing table size in [8, 9, 25]. Subscription covering is less efficient than subsumption but at a low cost, such as [19, 20]. These techniques are complementary to our proposed approach and it is beneficial to incorporate them in REIN. However, the cost of checking subscription subsumption is not trivial and it is not suitable for very dynamic systems. So far, these techniques are not used in our experiments.

III. Definitions and Background

In this section, we first define some basic terms of publish/subscribe systems. The data model is also described. Then the architecture and rationale of publish/subscribe systems are presented.

A. Definitions

Definition 1: Events

Clients who publish events are called publishers. An event is an observable occurrence, which is also called message, publication or notification in some literatures. Usually, an event is expressed as a conjunction of attribute-value pairs. As a convention, every attribute appears only once in an event expression. For example, $\{(temperature = 35), (humidity = 15)\}$ is an event describing the weather conditions. The set of attributes appearing in the event expression is defined as $A = \{a_1, a_2, ..., a_m\}$ and the number of attributes in the set A is denoted by m. It is assumed that the value of attributes is integer. Given the accuracy requirement, continuous values can be transformed into integer values by discretization.

Definition 2: Constraints

A constraint is a condition specified on an attribute selected from A. We consider range constraints in inclusive forms in the paper, which are represented as a 4-tuple of $\{attribute, value1, value2, type\}$. $Attribute$ is one of the attributes in A. $Value1$ and $value2$ are bounded by the value domain of the attribute and $value1$ is not larger than $value2$. $Type$ defines the data type of the attribute with a value domain, which can be any of $\{integer, double, string\}$. Given the value domain of the attribute, other forms of constraints can be transformed into range constraints. For example, if the data type of $temperature$ is integer with a value domain [0, 100], then a simple constraint $\{temperature, \geqslant, 20, integer\}$ can be transformed into a range constraint $\{temperature, 20, 100, integer\}$.

Definition 3: Subscriptions

Clients who issue subscriptions are called subscribers. A subscription is an expression of subscribers' interests in some

events, which is also used to route and forward events from the publishers to the target subscribers. Each subscription is identified by a unique subID and is specified as a conjunction of multiple range constraints. The number of range constraints contained in a subscription is not larger than m, where m is the number of attributes appearing in events. A subscription matches an event if all the range constraints contained in the subscription are satisfied when they are assigned the corresponding attribute values of the event.

Definition 4: Brokers

Brokers are also called servers or proxies. In non-P2P computing environments, a broker is a specialized server that is responsible for routing and forwarding subscriptions and events. In P2P environments, some clients also act as brokers.

Definition 5: Selectivity

The selectivity of subscriptions is the average probability that any subscription matches an event. The selectivity of constraints is the average probability that a constraint is satisfied by assigning the attribute value in an event. Given the prior distributions of the attribute values in events and the width of range constraints, the selectivity of subscriptions is deterministic. The selectivity of subscriptions affects the matching performance since a subscription can be decided as not matched as soon as one of its constraints is found to be not satisfied. Therefore, for most matching algorithms, the matching time increases with selectivity when matching an event.

Definition 6: Event Matching

Given a set of n subscriptions $S = \{s_1, s_2, ..., s_n\}$ and an event e, the task of event matching is to find all subscriptions from S which match e. The set of the matched subscriptions S_m is a subset of S, $S_m \subseteq S$.

$$S_m = \{s_i \mid s_i \in S \cap s_i \; matches \; e\} \tag{1}$$

B. Publish/Subscribe Systems

How to quickly distribute events from the publishers to the subscribers is a challenging problem for a large-scale system. In the extreme case where an event should be notified to all subscribers, broadcasting is the simple but efficient method to disseminate the event. On the other hand, if an event is only interested by a few subscribers, source routing mechanism is appropriate to transmit it. The publish/subscribe paradigm is applicable to the situation between these two extreme cases, where broadcasting and source routing mechanism are both not efficient.

A typical publish/subscribe system consists of subscribers, publishers, and a network of brokers. The publishers input events into the system while the subscribers submit subscriptions to the system. The core of publish/subscribe systems is to transmit events from the publishers to the target subscribers as quickly as possible. Usually, subscriptions will be broadcasted in the network of brokers [13, 14] or directed to some rendezvous brokers [15, 16]. In the former, event matching is performed at each broker to reversely transmit events to the subscribers according to the routing path constructed during the broadcasting process of subscriptions. As for the latter, event matching is carried out at the rendezvous brokers and events are transmitted to the interested subscribers by the rendezvous brokers. An example of a publish/subscribe system is shown in Fig. 1. There are 8 subscribers, 2 publishers and a network of brokers composed of 7 brokers in the system.

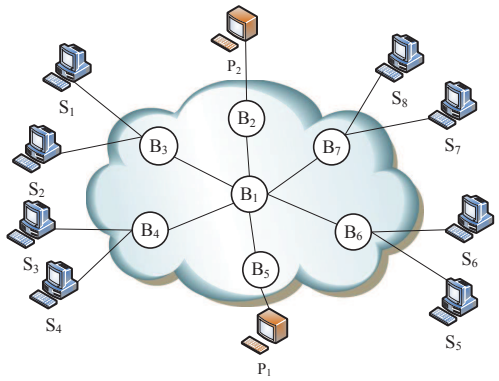


Fig. 1. A distributed publish/subscribe system composed of publishers, subscribers and brokers.

Whenever a broker receives an event from an interface, event matching is carried out to decide whether the event should be forwarded to the next-hop through other interfaces. For large-scale distributed publish/subscribe systems, there are millions of subscriptions maintained by brokers. Brokers with low matching speed are apt to be potential performance bottlenecks. For example, when the receiving rate of events is greater than the matching rate at a broker, the broker becomes a performance bottleneck. This is just like what happens at a toll station where thousands of vehicles are waiting to pass at peak hours. Therefore, improving the matching speed is critical for large-scale publish/subscribe systems.

IV. The Design of REIN

In this section, we explain the design of REIN. The matching problem is first transformed into the rectangle intersection problem. A concise index structure is also provided which is efficient to address the rectangle intersection problem by using bit operations. In addition, an example is given to illustrate the structure and the algorithm of REIN.

A. Overview

Most existing methods compute the matched subscriptions by counting the number of satisfied constraints, such as SIENA [14] and TAMA [6]. The satisfied component constraints are identified firstly. Then the matched subscriptions are picked out in terms of these satisfied constraints. On the contrary, REIN first marks unlikely matched subscriptions in a bit set by applying efficient bit operations. The unmarked bits represent the matched subscriptions.

As defined in Section II, given a set of subscriptions S and an event e, event matching is to find all subscriptions from S that match the event e. The set of attributes $A = \{a_1, a_2, ..., a_m\}$ appearing in events forms a m-dimensional space. In the space, events are points and subscriptions are rectangles. Therefore, the event matching problem is equivalent to the point enclosure problem [17], which finds all rectangles that contain a given point. We transform the point enclosure problem into the high-dimensional rectangle intersection problem. Therefore, the matched subscriptions are those rectangles

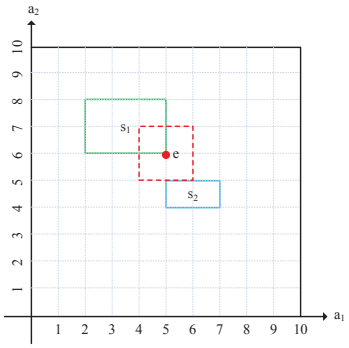


Fig. 2. Occurrence of a false positive by direct enlargement.

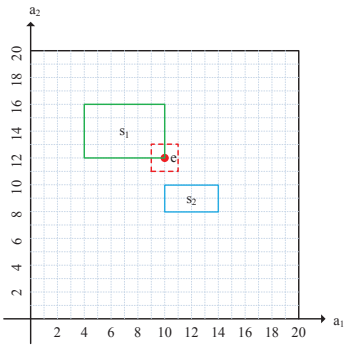


Fig. 3. Doubling of values to prevent false positives.

that intersect a given cube which is enlarged from a point representing an event. Based on the transformation, we design an efficient index structure to speed up the checking of rectangle intersection in the high-dimensional space.

B. Problem Transformation

Definition 7: Rectangle Intersection

In the m-dimensional space formed by the set of attributes $A = \{a_1, a_2, ..., a_m\}$ appearing in events, a rectangle r intersects another rectangle r' if r and r' have at least one point in common.

Our objective is to transform the point enclosure problem into the rectangle intersection problem. Due to the inclusive form of range constraints, a point lying on a side (or face) of a rectangle is contained by the rectangle. We enlarge a point in the m-dimensional space into a cube. Then all rectangles that contain the point intersect the cube. One problem with this direct enlargement of a point is the occurrence of false positives. As shown in Fig. 2, s_2 does not contain the point (denoted as a red dot). However, s_2 intersects the enlarged cube (denoted as a red dashed cube), causing the occurrence of a false positive.

REIN is an exact matching approach for content-based publish/subscribe systems. Therefore, some techniques should be applied to prevent the occurrence of false positives. It is assumed that the values of an attribute are integer taken from a value domain which is bounded by the minimum and the maximum values. The smallest unit of integer is 1. In order to prevent false positives, the value domain of each attribute is doubled. Constraint values and event values are processed accordingly. Fox example, in Fig. 2, the value domain of a_1 and a_2 is [0, 10]. The range constraints of s_1 are $\{a_1[2,\ 5] \wedge a_2[6,\ 8]\}$. The attribute values of the event e are $\{a_1 = 5,\ a_2 = 6\}$. After doubling, the value domain of a_1 and a_2 is [0, 20]. The values of s_1 and e are $\{a_1[4,\ 10] \wedge a_2[12,\ 16]\}$ and $\{a_1 = 10,\ a_2 = 12\}$ respectively, just as shown in Fig. 3.

The doubling of value domains and values is efficient to avoid false positives. Due to the effect of doubling, the values of range constraints and events are even integer. However, an enlarged cube is specified by odd integer. A rectangle either intersects or disjoints with an enlarged cube. If a rectangle r does not contain a point p, r will never intersect the cube enlarged from p, which prevents the happening of false positives. The prevention of false positives establishes a foundation of REIN.

C. Index structure

Based on the transformation from the point enclosure problem into the rectangle intersection problem, the matching problem can be solved by finding all rectangles which intersect a cube enlarged from a point. A strategy to deal with the rectangle intersection problem is to find out all rectangles intersecting a given cube. On the contrary, the basic idea behind REIN is to mark all disjoint rectangles first. The left unmarked rectangles are the solutions to the problem. This can be realized by setting all disjoint rectangles in a bit set through efficient bit operations. All intersected rectangles are represented by the unset bits in the bit set.

The index structure of REIN consists of multiple bucket lists. The number of bucket lists is $2m$, where m is the number of attributes appearing in events. For each attribute, two bucket lists are constructed. One bucket list is for the low values of the range constraints specified on the attribute. Another is for the high values of the range constraints. A bucket list is constructed by dividing the value domain of an attribute into cells and realizing the mapping from the cells to the buckets. All values belonging to a cell are mapped to the corresponding bucket. When a constraint value is mapped to a bucket, the constraint value and the corresponding subID are inserted into the bucket. The index structure is shown in Fig. 5, where 4 bucket lists are created for two attributes, a_1 and a_2. After doubling, the value domain of a_1 and a_2 is [0, 20] on which 4 cells are divided. Each cell maps to a bucket in which constraint values and subIDs are inserted.

The number of cells divided on a value domain is determined by multiple factors. One is the stability of subscriptions. For a publish/subscribe system where the subscriptions are relatively static, less cells can be divided and the items in each bucket can be sorted on the constraint values to obtain better performance. Otherwise, more cells are needed to reduce the cost of subscription modifications. Another factor is the number of subscriptions. In order to improve matching efficiencies, the size of buckets is important. Given the number of subscriptions, there is a critical point for the number of buckets. After the critical point, the performance of REIN degrades with more buckets.

D. Event Matching

The procedure of event matching in REIN is straightforward. When matching events, a bit set is initialized in which the number of bits equals the number of subscriptions ($line$ 3). All the unmatched subscriptions are marked in the bit set. Given an event, an enlarged cube is generated which is specified by $\{v_{11}, v_{12}, v_{21}, v_{22}, ..., v_{m1}, v_{m2}\}$, where v_{i1} is not larger than v_{i2} for all $1 \leqslant i \leqslant m$ ($line$ 4). For each attribute, we find out all the subscriptions that (i) the high value of the range constraint specified on the attribute is less than the low value of the attribute of the cube ($line\ 6-12$), and (ii) the low value of the range constraint specified on the attribute is larger than the high value of the attribute of the cube ($line\ 13-19$). The unset bits in the bit set represent the matched subscriptions ($line\ 21-25$). The event matching algorithm is shown in Fig.

4. Please note that comparison operations are only executed in one bucket for any attribute when matching an event. Other buckets are rapidly traversed in the course of matching.

Algorithm 1 Event Matching

```
1: input: event e
2: output: matched subIDs ID
3: initialize a bit set of size n;
4: generate a cube {v11, v12, v21, v22, ..., vm1, vm2} from e;
5: for (i = 1 to m) do
6:    map v_i1 to the bucket b in the bucket list for the high
      values of the range constraints specified on attribute i;
7:    for (each item t in b) do
8:       if t.val < v_i1 then
9:          mark the t.ID in the bit set;
10:      end if
11:   end for
12:   mark the bits for the IDs stored in the buckets before b;
13:   map v_i2 to the bucket b in the bucket list for the low
      values of the range constraints specified on attribute i;
14:   for (each item t in b) do
15:      if t.val > v_i2 then
16:         mark the t.ID in the bit set;
17:      end if
18:   end for
19:   mark the bits for the IDs stored in the buckets after b;
20: end for
21: for (each bit in the bit set) do
22:   if (the bit is unmarked) then
23:      add the corresponding subID to ID;
24:   end if
25: end for
```

Fig. 4. Event matching algorithm.

E. Example

In order to illustrate the index structure and the matching procedure of REIN, an example is presented which is shown in Fig. 5. There are 2 attributes, $A = \{a_1, a_2\}$. The value domain of each attribute is [0, 10]. After the doubling of the value domain, 4 buckets are constructed by evenly dividing the domain [0, 20]. For each attribute, two bucket lists are created, one for the low values of the range constraints, another for the high values. 10 subscriptions are stored in the system listed in Table I. When indexing a subscription, the range constraints contained in the subscription are processed one by one. For each range constraint, the low value and the high value specified on the attribute are used to determine the corresponding bucket to store the subID of the subscription. Please note that if there are k range constraints in a subscription, the subID of the subscription is stored $2k$ times. For example, when indexing s_1 in Table I, the low value on a_1 is $2*2 = 4$ which is located in the interval of bucket b_0, therefore s_1 is stored in b_0 shown in Fig. 5(a). The high value specified on a_1 is $2*6 = 12$ which is in the interval of bucket b_2, shown in Fig. 5(b). It is the same for the processing of the range constraint specified on a_2, s_1 is stored in b_1 and b_2, shown in Fig. 5(c) and Fig. 5(d) respectively.

The rectangles that represent these subscriptions are shown in Fig. 6 (a). Given an event e $\{a_1 = 3,\ a_2 = 5\}$,

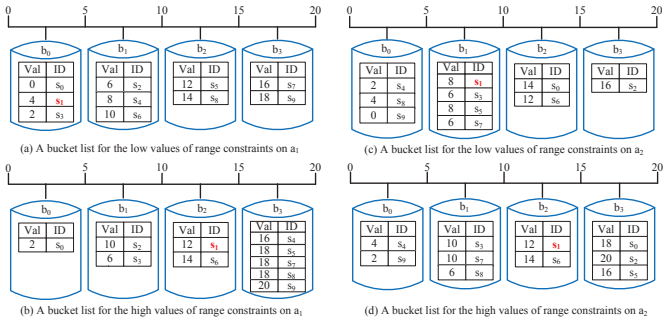


Fig. 5. The index structure of REIN composed of bucket lists.

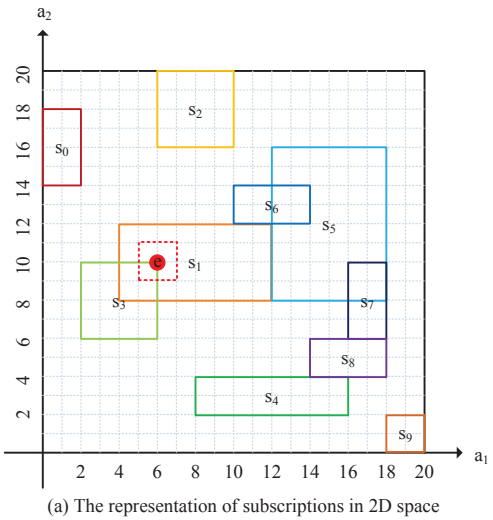
TABLE I. LIST OF 10 SUBSCRIPTIONS

SubID	a_1	a_2	SubID	a_1	a_2
s_0	$[0, 1]$	$[7, 9]$	s_5	$[6, 9]$	$[4, 8]$
s_1	$[2, 6]$	$[4, 6]$	s_6	$[5, 7]$	$[6, 7]$
s_2	$[3, 5]$	$[8, 10]$	s_7	$[8, 9]$	$[3, 5]$
s_3	$[1, 3]$	$[3, 5]$	s_8	$[7, 9]$	$[2, 3]$
s_4	$[4, 8]$	$[1, 2]$	s_9	$[9, 10]$	$[0, 1]$

which is denoted as a red point, it is enlarged as a cube $\{a_1 = [5, 7]\ a_2 = [9, 11]\}$ (shown as a red dashed cube after doubling). The matched subscriptions are those that intersect the cube. The rectangles that their right sides are on the left side of the cube are marked, namely s_0. The resulting rectangles for the right side of the cube are $s_4, s_5, s_6, s_7, s_8, s_9$. For the top side of the cube, s_0, s_2, s_6 are the resulting rectangles. s_4, s_8, s_9 are the rectangles for the bottom side. Obviously, some rectangles are marked multiple times. By checking the bit set, s_1 and s_3 are the matched subscriptions for the event e, which is shown in Fig. 6 (b).

V. PERFORMANCE EVALUATION

In order to comprehensively evaluate the matching performance of REIN, three time metrics are measured, which are matching time, insertion time and deletion time. Representative matching algorithms are chosen to be compared with REIN. All experiments were conducted on a Dell PowerEdge T710 running Ubuntu 11.10 with Linux kernel 3.0.0-12, which has



(a) The representation of subscriptions in 2D space

(b) The bitset status when matching an event e

Fig. 6. The representation of subscriptions and an event in a 2D space.

TABLE II. THE PARAMETERS USED IN THE EXPERIMENTS

Name	Meaning	Value
n	the number of subscriptions	200k $\sim$ 10M
m	the number of attributes in events	10 $\sim$ 46
k	the number of constraints in subscriptions	5 $\sim$ 30
w	the width of range constraints	0.01 $\sim$ 0.8
c	the number of cells in H-TREE	6
l	the number of indexed attributes in H-TREE	6
d	the discretization level in TAMA	13
b	the number of buckets in REIN	50 $\sim$ 20k

8 2.4GHz cores and 32GB memory. Parallelism was not used in all experiments. All code was written in C++ language. The evaluation results are presented in this section.

A. The Setting of Experiments

REIN is compared with three matching algorithms, namely SIENA, TAMA, and H-TREE. SIENA is a two-level forwarding table, the first level indexed on attributes, and the second level indexed on operators [3]. TAMA is an approximate matching and forwarding engine presented in [6]. The discretization level of TAMA is set to 13 in the experiments. H-TREE is an exact matching algorithm by putting similar subscriptions into the same bucket to improve the matching speed [7]. The cells and indexed attributes are set to 6 for H-TREE in the experiments.

The performance of matching algorithms is affected by many parameters which are shown in Table II. We observe their impacts on the matching performance by changing their values in the experiments. The value domain of attributes is normalized on [0, 1.0] for brevity. The constraint values and events values are generated from this domain with precision of 10^{-6}. For REIN, the real values are discretized into integer with the availability of value domain and precision.

B. Matching Time

Among the three time metrics, matching time is the most important to evaluate the performance of matching algorithms. Matching time is affected by many parameters and extensive experiments were conducted to observe the impacts of these parameters in different settings. 400 events were input to measure the averaged matching time in each experiment. In the experiments, constraint values, event values, and constraint attributes were randomly generated unless stated clearly.

1) Effects of the number of subscriptions: Generally speaking, matching time increases with the number of subscriptions. Compared with other three matching algorithms, the performance of REIN is least affected by the number of subscriptions. The experimental results are shown in Fig. 7, where $m = 20$, $k = 10$, $b = 1000$ and $w = 0.5$. Given the width of range constraints, the low values of range constraints were randomly generated from the value domain [0, 1-w] and the high values are from [w, 1.0]. In other words, the distribution of the high values is as same as the distribution of the low values when the width of range constraints is fixed.

When the number of subscriptions is 2000k, REIN is almost 14, 9, and 6 times faster than SIENA, TAMA, and H-TREE, respectively. Given the width of range constraints, the selectivity of subscriptions is deterministic. With more subscriptions, the matched subscriptions for events increase accordingly. The averaged matching time shown in Fig 7 just gives an overall evaluation. More insights are discovered by analyzing the details. For each event, the matched subscriptions and the matching time to search them were recorded in the experiment. 400 events were input to match against 2000k subscriptions and total 727894 subscriptions were matched. The averaged selectivity of subscriptions is 0.00091. By observing the relationships between the number of matched subscriptions and the matching time shown in Fig. 10, we found that the four algorithms behaved differently. The matching time of SIENA and TAMA increases logarithmically with the number of matched subscriptions. As for H-TREE, the matching time increases linearly. However, REIN decreases logarithmically, which is a pretty attractive feature. This can be explained that when more subscriptions are matched with an event, less bits in the bit set are marked in the course of matching, which reduces the matching time.

TABLE III. THE MAXIMUM, MINIMUM, AND STANDARD DEVIATION OF MATCHING TIME

	SIENA	TAMA	H-TREE	REIN
max	24.2147	19.1677	179.5545	2.0291
min	9.9990	8.4958	0.4445	1.0977
std	0.7280	1.7972	14.1006	0.1498

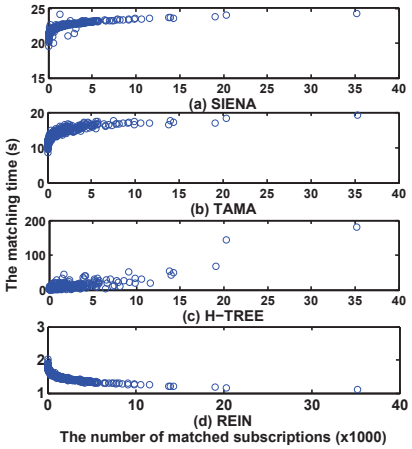


Fig. 10. Scatter between matching time and matched subscriptions.

The maximum, minimum, and standard deviation of the matching time are listed in Table III, where the number of subscriptions is 2000k. Overall, the matching time of H-TREE is the most volatile. The maximum is 179.5545s, in which the event matches 35194 subscriptions. The minimum is 0.4445s, where 30 matched subscriptions are returned. By observing the standard deviation, REIN is the most steady, which is also manifested in the difference between the maximum and the minimum.

2) Effects of the number of constraints: An experiment was also conducted to evaluate the effects of the number of constraints. In the experiment, the number of attributes m appearing in events was variable and the number of constraints k was half m. The results of the experiment are shown in Fig. 8, where $n = 1000k$, $b = 1000$, $w = 0.5$, and y-axis represents the matching time in log-scale. The number of matched subscription and the number of constraints are presented in Table IV. As mentioned above, the matching time of H-TREE is linear with the number of matched subscriptions. When the number of constraints increases, the selectivity of subscriptions decreases. Therefore, the matching time of H-TREE first drops quickly with the number of constraints.

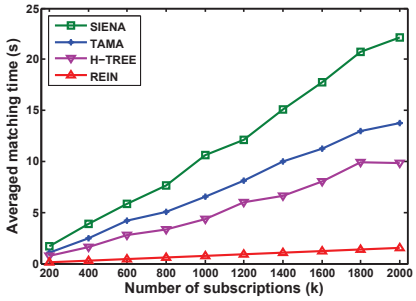


Fig. 7. Effects of the number of subscriptions.

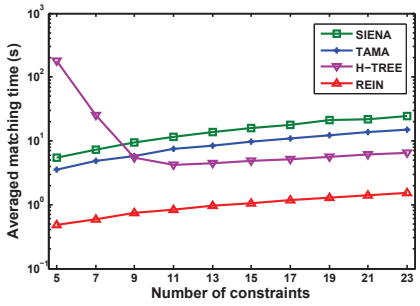


Fig. 8. Effects of the number of constraints.

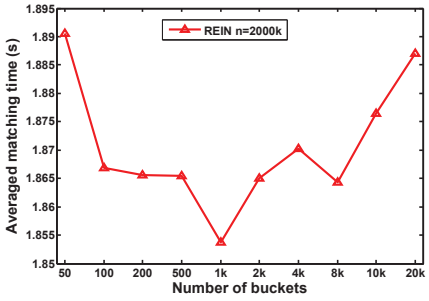


Fig. 9. Effects of the number of buckets.

TABLE IV. MATCHED SUBSCRIPTIONS WITH THE NUMBER OF CONSTRAINTS

Constraints	Matched subs	Constraints	Matched subs
5	11745128	15	10727
7	3015656	17	2769
9	752285	19	762
11	178837	21	251
13	43162	23	54

Then, the matching time of H-TREE is mainly affected by the number of constraints, which is similar to other three algorithms. In general, the matching time of the four algorithms increases with the number of constraints. When the number of constraints is 23, REIN is 16, 10, and 4 times faster than SIENA, TAMA, and H-TREE, respectively. The trend of maximum, minimum, and standard deviation is similar to Table III and is omitted due to the limit of space.

3) Effects of the number of buckets: For REIN, comparison operations are only executed in one bucket for an attribute. Obviously, the size of buckets affects the performance of REIN. When the number of subscriptions is large, more buckets are needed to improve the matching efficiencies. However, when the number of buckets reaches a critical point, the performance of REIN degrades. Although the size of buckets is reduced with more buckets, the cost to traverse buckets increases, which offset the benefits in the reduction of comparison operations. As shown in Fig. 9, when there are 2000k subscriptions, the critical point of buckets is 1000. The matching time first decreases with the number of buckets before reaching the critical point. After the critical point, the performance of REIN degrades with more buckets. Please note that the critical point changes with the number of subscriptions.

4) Effects of the distribution of constraint values: Ideally, subIDs should be evenly stored in the buckets of REIN, which is impractical in reality. REIN is almost unaffected by the distribution of the constraint values, which is another feature of REIN. When the number of buckets is enough, the effects of the distribution of the constraint values are eliminated since the size of buckets is small and comparison operations are only executed in one bucket. To evaluate the impacts of the distributions on matching time, we generated constraint values according to three different distributions, namely, Uniform, Normal, and Pareto. The mean and the variance were set to 0.5 and 0.02 respectively for the Normal distribution. For the Pareto distribution, the mean and the scale were set to 0.5 and 2 respectively. Event values and constraint attributes were generated randomly. Compared with the Uniform distribution, other two forms of distributions almost have the same results. Since there is very little difference in the results of the three distributions, it is not proper to show the results in a figure.

TABLE V. MATCHING TIME WITH THE DISTRIBUTION OF CONSTRAINT VALUES

	Uni	Nor	Par		Uni	Nor	Par
200k	0.1551	0.1554	0.1534	1200k	0.9146	0.9259	0.9290
400k	0.3061	0.3098	0.3106	1400k	1.0812	1.0739	1.0740
600k	0.4607	0.4680	0.4657	1600k	1.2423	1.2349	1.2484
800k	0.6163	0.6175	0.6174	1800k	1.3793	1.4166	1.4110
1000k	0.7708	0.7754	0.7743	2000k	1.5387	1.5765	1.5422

TABLE VI. MATCHED SUBSCRIPTIONS WITH WIDTH OF RANGE CONSTRAINTS

Width	Matched subs	Width	Matched subs
0.01	0	0.4	40166
0.05	0	0.5	378886
0.1	0	0.6	2329247
0.2	32	0.7	10843737
0.3	2369	0.8	42023835

The results are presented in Table V. It is the same for the distributions of event values. Due to the limit of space, the results are omitted.

5) Effects of the width of range constraints: Given the distribution of event values, the width of range constraints determines the selectivity of subscriptions, which increases with the width w. The results are shown in Fig. 11, where $n = 1000k$, $m = 20$, $b = 1000$, $k = 10$, and y-axis represents the matching time in log-scale. SIENA increases with the width of range constraints. When $w \leqslant 0.1$, the matching time of TAMA is less than the one of REIN. This can be explained that when $w \leqslant 0.1$, there is no matched subscription for events in the experiment shown in Table VI. With the increase of w, TAMA behaves asymptotically as SIENA. When $w \leqslant 0.3$, H-TREE behaves best in the four algorithms. More specifically, when $w \leqslant 0.1$, the matching time of H-TREE is invariable since the width of range constraints is less than the width of cells divided on the attributes' value domain. When $w > \frac{1}{c}$, a subscription is split into $\lceil \frac{w}{1/c} \rceil^l$ subscriptions with narrower range constraints, where c is the number of cells and l is the number of indexed attributes in H-TREE. When $w \geqslant 0.7$, 32GB memory is used up due to the exponential growth. Therefore, there is not result for $w = 0.7$ and $w = 0.8$ in the figure. The matching time of REIN decreases with w, which is of great importance in real applications. When $w = 0.6$, REIN is 17, 12, and 50 times faster than SIENA, TAMA, and H-TREE, respectively.

C. Insertion Time

The four matching algorithms all have their specialized index structures. We measure the time spent on inserting subscriptions into the index structures for these algorithms. For SIENA, since a range constraint is transformed into two simple

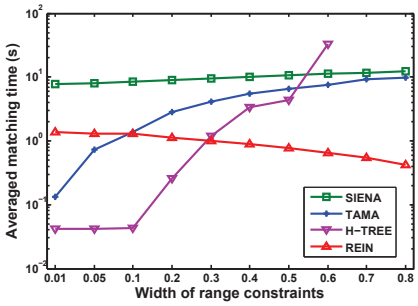


Fig. 11. Effects of the width of range constraints.

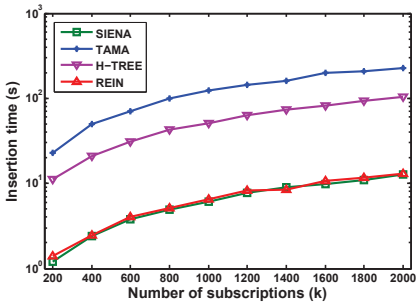


Fig. 12. Insertion time with number of subscriptions.

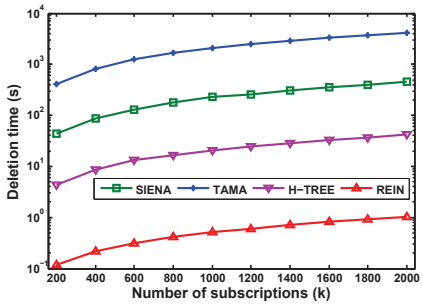


Fig. 13. Deletion time with number of subscriptions.

constraints, the subID of a subscription is stored $2k$ times in the index structure, where k is the number of range constraints in subscriptions. As for TAMA, the width of range constraints determines the times to store the subID of a subscription. When $w = 0.5$, the subID is stored at least $10k$ times. For H-TREE, the subID is stored $\lceil \frac{w}{1/c} \rceil^l$ times. The times to store the subID in REIN is $2k$, which is as same as SIENA. The results are shown in Fig. 12, where $m = 20$, $k = 10$, $b = 1000$, $w = 0.5$ and y-axis represents the insertion time in log-scale. As shown in the figure, SIENA spends least time when inserting subscriptions. When $n = 2000k$, REIN only spends 4% more time than SIENA. However, the insertion time of TAMA and H-TREE is 18, and 8 times of SIENA. Please note that the insertion time of SIENA and REIN is not affected by the width of range constraints. On the contrary, the insertion time of TAMA and H-TREE can be dropped when the width is reduced.

D. Deletion Time

The index structure of SIENA is two-level with one level indexed on attributes, and another level on operators. There are only two operators for each attribute, $\leqslant$ and $\geqslant$. Each operator is mapped to a bucket and the number of buckets is $2m$, where m is the number of attributes appearing in events. For each bucket, the size is n, where n is the number of subscriptions. When the number of subscriptions is large, deleting a subscription is very costly for SIENA. As for TAMA and H-TREE, the subID of a subscription is stored in multiple buckets. Deletion is also time-consuming. We measure the time when deleting 1000 subscriptions from different number of subscriptions. The results are shown in Fig. 13. By observing the insertion time and deletion time, the third feature of REIN is that REIN is applicable to very dynamic environments where subscription modifications occur frequently.

E. Memory Consumption

We also analyzed the memory consumption for the four matching algorithms. It is assumed that the values and subIDs are integer, each occupying 4 bytes. A bucket itself occupies 8 bytes. For SIENA, the number of buckets is $2m$, occupying $16m$ bytes. The memory storage for subscriptions is $16kn$ bytes. The number of constraints contained in each subscription is stored which occupies $4n$ bytes. When analyzing TAMA and H-TREE, the width of range constraints is 0.5. Only subIDs are stored in TAMA and H-TREE. For TAMA, the number of constraints contained in each subscription is also stored, occupying $4n$ bytes. The number of buckets is $m2^d$, occupying $8m2^d$ bytes. The memory for subscriptions is $40kn$ bytes. For H-TREE, the number of bucket is c^l, occupying $8c^l$ bytes. The memory for subscriptions is $4\lceil \frac{w}{1/c} \rceil^l n$ bytes. The number of buckets in REIN is b in the experiments, occupying 8b bytes. Constraint values and subIDs are stored in REIN. The memory for subscriptions is $16kn$ bytes. The details of the four algorithms are listed in Table VII. The counted memory consumption is shown in Fig. 14, where $m = 20$, $k = 10$, $b = 1000$, $w = 0.5$, $c = 6$, $l = 6$, $d = 13$ and the y-axis in log-scale. As shown in the figure, REIN consumes the least memory space.

TABLE VII. DETAILS OF MEMORY CONSUMPTION (B)

	SIENA	TAMA	H-TREE	REIN
Memory of buckets	$16m$	$8m2^d$	$8c^l$	$8b$
Memory of subIDs	$16kn$	$40kn$	$4\lceil \frac{w}{1/c} \rceil^l n$	$16kn$
Other	$4n$	$4n$	0	0

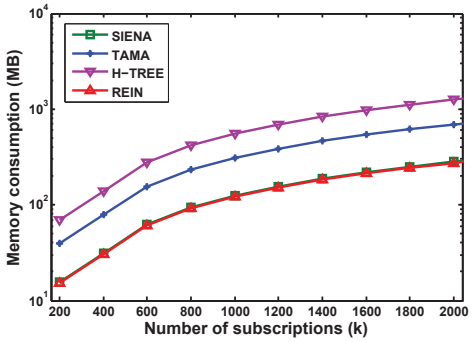


Fig. 14. Memory consumption counted in memory space.

VI. CONCLUSION

In this paper, we present REIN, a fast event matching approach for large-scale content-based publish/subscribe systems. The key idea behind REIN is to quickly filter out unlikely matched subscriptions rather than to identify matched subscriptions by counting satisfied component constraints. In the case of range constraints, a subscription is a rectangle in the space formed by the attributes appearing in events so that the matching problem is equivalent to the point enclosure problem. We then transform the point enclosure problem into the rectangle intersection problem by enlarging a point into a cube without leading to the occurrence of false positives. A concise index structure is also proposed to address the rectangle intersection problem through efficient bit operations. Comprehensive experiments were conducted to evaluate the performance of REIN. Experimental results show that REIN outperforms its counterparts to a large degree, especially in the case where the selectivity of subscriptions is high and the

number of subscriptions is large.

ACKNOWLEDGMENT

The work was supported by Morgan Stanley (No. CIP-A20110324-2). This work was also partially supported by NSFC (No. 61272438, and 61170238), Research Funds of Science and Technology Commission of Shanghai Municipality (No. 12511502704), China 973 Program (2014CB340303), Singapore NRF (CREATE E2S2), National 863 Program (2013AA01A601), SJTU SMC Project (201120), Shanghai Chen Guang Program (10CG11), MSRA funding for Urban Computing and for Star Track program, and Program for Changjiang Scholars and Innovative Research Team in University(IRT1158, PCSIRT), China.

REFERENCES

[1] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

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

[3] A. Carzaniga and A. Wolf, "Forwarding in a content-based network," in *SIGCOMM*. ACM, 2003, pp. 163–174.

[4] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "Mics: an efficient content space representation model for publish/subscribe systems," in *DEBS*. ACM, 2009, p. 7.

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

[6] Y. Zhao and J. Wu, "Towards approximate event processing in a large-scale content-based network," in *ICDCS*. IEEE, 2011, pp. 790–799.

[7] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang, "H-tree: An efficient index structure for event matching in content-based publish/subscribe systems," in *NETWORKING 2013*, 2013.

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

[9] G. Li, S. Hou, and H. Jacobsen, "A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams," in *ICDCS*. IEEE, 2005, pp. 447–457.

[10] A. Ouksel, O. Jurca, I. Podnar, and K. Aberer, "Efficient probabilistic subsumption checking for content-based publish/subscribe systems," *Middleware 2006*, pp. 121–140, 2006.

[11] X. Guo, J. Wei, and D. Han, "Efficient event matching in publish/subscribe: based on routing destination and matching history," in *NAS*. IEEE, 2008, pp. 129–136.

[12] K. Jayaram and P. Eugster, "Split and subsume: Subscription normalization for effective content-based messaging," in *ICDCS*. IEEE, 2011, pp. 824–835.

[13] P. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," in *ICDCS Workshops*. IEEE, 2002, pp. 611–618.

[14] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *TOCS*, vol. 19, no. 3, pp. 332–383, 2001.

[15] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: content-based publish/subscribe over p2p networks," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. Springer-Verlag New York, Inc., 2004, pp. 254–273.

[16] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 746–757.

[17] V. K. Vaishnavi, "Computing point enclosures," *Computers, IEEE Transactions on*, vol. 100, no. 1, pp. 22–29, 1982.

[18] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *DEBS*. ACM, 2008, pp. 71–81.

[19] Z. Shen, S. Aluru, and S. Tirthapura, "Indexing for subscription covering in publish-subscribe systems." *ISCA PDCS*, vol. 5, pp. 328–333, 2005.

[20] D. A. Tran and T. Nguyen, "A random projection approach to subscription covering detection in publish/subscribe systems," in *Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007. International Conference on*. IEEE, 2007, pp. 362–369.

[21] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian, "Subscription subsumption evaluation for content-based publish/subscribe systems," in *Middleware 2008*. Springer, 2008, pp. 62–81.

[22] X. Qin, J. Wei, W. Zhang, H. Zhong, and T. Huang, "A two-phase approach to subscription subsumption checking for content-based publish/subscribe systems," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 1278–1285.

[23] Z. Shen and S. Tirthapura, "Approximate covering detection among content-based subscriptions using space filling curves," *Journal of Parallel and Distributed Computing*, 2012.

[24] M. A. Tariq, B. Koldehofe, G. G. Koch, and K. Rothermel, "Distributed spectral cluster management: A method for building dynamic publish/subscribe systems," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. ACM, 2012, pp. 213–224.

[25] Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson, "Summary-based routing for content-based event distribution networks," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 5, pp. 59–74, 2004.