An Efficient Location-Aware Publish/Subscribe Index with Boolean Expressions

Hanhan Jiang[1], Pengpeng Zhao[1](✉), Victor S. Sheng[2], Guanfeng Liu[1], An Liu[1], Jian Wu[1], and Zhiming Cui[1]

[1] School of Computer Science and Technology, Soochow University, Suzhou, China
20145227025@stu.suda.edu.cn,
{ppzhao,gfliu,anliu,jianwu,szzmcui}@suda.edu.cn
[2] Computer Science Department, University of Central Arkansas, Conway, USA
ssheng@uca.edu

Abstract. A location-aware publish/subscribe (pub/sub) system is gaining more and more interest in both industry and academia with the rapid progress of mobile Internet and the rising popularity of smart-phones. Nowadays, with the booming of E-commerce, Object-to-Object (OTO) services are gaining more and more popularity, which results in millions of products with different structured descriptions and locations. To meet this requirement, a pub/sub system should handle subscriptions with location-aware boolean expressions to present users' interests. In this paper, we propose an efficient location-aware pub/sub index for boolean expressions, called RP-trees. RP-trees integrates an R-tree index and a boolean expression index together, can efficiently and simultaneously prune boolean expressions and spatial dimensions. Our experimental results show that RP-trees achieves better performance on both a synthetic dataset and a real-world dataset.

Keywords: Location-aware pub/sub · Boolean expression · OTO

1 Introduction

With the rapid progress of mobile Internet and the growing popularity of smart-phones equipped with GPS, the location-aware pub/sub have recently attracted significant attention from both industrial and academic communities [2,5–7,16,17]. The subscribers subscribe their interests as subscriptions while a publisher publishes messages as events which are delivered to subscribers whose interests are matched with the messages. Location-aware pub/sub systems can be applied in many real-world domains, such as Groupon, Twitter and Foursquare. Existing location-based pub/sub systems [6,11] can support subscriptions with textual descriptions well. Users on Twitter can register their interests with a textual descriptions and a spatial constraint (e.g.,"Fresh seafood","$distance \leqslant 3\,\mathrm{km}$"). For each tweet with textual description and location information (e.g.,"Newly stocked seafood: crayfish","31.129025, 120.468226"), the system has to ensure

J. Wang et al. (Eds.): WISE 2015, Part I, LNCS 9418, pp. 216–231, 2015.
DOI: 10.1007/978-3-319-26190-4_15

a timely delivery of textual and spatial matching events to the corresponding subscribers.

However, thanks to the booming of e-commerce and mobile Internet. OTO services are gaining more and more popularity, which bring millions of products with many different attributes, values and geo-locations. This leads to a requirement for structuring descriptions to capture such information. Existing unstructured pub/sub works [2,5,6,17] using textual descriptions cannot accurately represent users interests. Existing structured pub/sub works [4,7,12,13,15] using boolean expressions cannot efficiently process spatial constraints. To meet this requirement, we model a new location-aware structured pub/sub problem in which subscriptions are represented by a combination of boolean expressions and spatial information. To explain this, we have the following example as a working scenario.

We can model services or products in an OTO platform like Groupon[1] *as events and their users as subscribers. Users register their interested regions, attributes and values of products as subscriptions. As the* Fig. 1 *shows, a subscription* $\{(B = 3 \wedge A \in (3,2,5) \wedge C \geqslant 2), R_1\}$ *is a combination of a boolean expression and a spatial constraint. A product is represented by a list of attribute-value pairs with a spatial point* $\{(A = 3 \wedge B = 3 \wedge C = 5), P_1\}$. *Users will be notified when a new inserted product matches their subscriptions.*

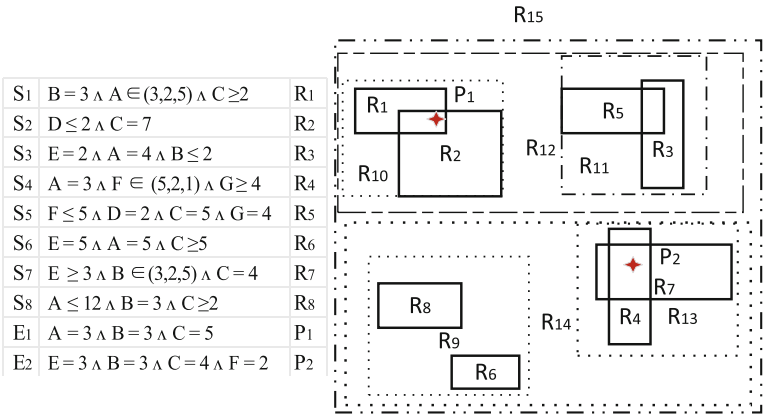


Fig. 1. An example of subscriptions and events

In the following, we present two real-world applications for which a location-aware pub/sub with boolean expression system may benefit.

– City-wide online trading OTO platforms, such as 58 city[2], have a large amount of products with multi-attributes and geo-locations. Location information to

[1] http://www.groupon.com.
[2] http://www.58.com.

users is important since users on this online platform tend to trade in a nearby area. A traditional way for users is searching in the system and making a choice from a great number of products. It is obvious that a location-aware pub/sub system may improve the service quality for users. After users register their interests and regions in this system, they can receive a timely notification when a match occurs.
– Group-buying platforms such as LivingSocial[3] and RetailMenot[4] have the location-aware pub/sub gene in nature. Customers in a Group-buying system can register their interests in the form of boolean expressions and a location constraint. When new products are continually being inserted, customers can receive products information timely instead of repeatedly searching the system to find their interested products.

There are several challenges in location-aware boolean expression pub/sub. First, it needs to handle millions of boolean expressions with a large amount of attributes and values. Second, when a spatial constraint is added to subscriptions, its computation cost in both indexing and matching becomes rather larger. Third, its matching process must be efficient enough to ensure a high event arrival rate. Thus, more efficient filtering techniques that support location-based pub/sub with boolean expressions are necessary.

To improve the indexing efficiency, we propose a two-step partitioning R-tree based index structure (called RP-trees since then). In its first step, we partition subscriptions according to their sizes (number of predicates in a subscription). In its second step, we select the most representative attribute of each subscription, and group subscriptions with the same size and the same selected attribute together. To index the spatial constraints, we build an R-tree for each group of subscriptions. Our experiments on large synthetic datasets and real-world datasets show that our method achieves higher performance.

To summarize, we make the following contributions.

– We propose a new problem, i.e. location-aware pub/sub with boolean expressions.
– We propose a new index structure RP-trees,which can efficiently index location-aware boolean expressions.
– We conduct experiments on a synthetic dataset and a real-world dataset to evaluate the efficiency of our proposed index structure RP-trees.

The remaining of this paper is organized as follows. We formalize the problem in Sect. 2. In Sect. 3, we review related works on the pub/sub system. In Sect. 4, we propose two preliminary solutions by extending a well-known existing solution as baselines for comparisons. In Sect. 5, we propose an RP-trees index and present its corresponding index construction algorithm. In Sect. 6, we present our subscription matching algorithm. Extensive experiments results are reported in Sect. 7, and we conclude this paper in Sect. 8.

[3] http://www.livingsocial.com.
[4] http://www.retailmenot.com.

2 Problem Formulation

In a location-aware pub/sub system, subscribers register their interests as subscriptions. A subscription includes a boolean expression and a spatial region, denoted by *s= (B, R)*.

A boolean expression is a combination of predicates in Conjunctive Normal Form. A predict contains three elements: an attribute denoted by A, an operator denoted by f_{op} and a value denoted by v. That is, $P_{(A,f_{op},v)}$ denotes a predicate. We can support six operators $(=, \leqslant, \geqslant, <, >, \in)$ in this paper. A boolean expression of a subscription s is modeled as follows:

$$s.B : P_1(x) \wedge P_2(x) \wedge P_3(x) \wedge ... \wedge P_m(x)$$

Thus, a subscription s is modeled over a boolean expression and a spatial constraint as follows:

$$s : \{P_1(x) \wedge P_2(x) \wedge P_3(x) \wedge ... \wedge P_m(x), Region\}$$

An information publisher publishes an event e that contains a collection of value-pairs denoted as $e.V$ and a geo-position denoted as $e.P$. $e.V$ is represented in the form of conjunction of equality predicates and the size of $e.V$ is the number of value-pairs denoted by m. Thus, an event can be denoted as follows:

$$e : \{(A_1 = v_1) \wedge (A_2 = v_2) \wedge (A_3 = v_3) \wedge ... \wedge A_m = v_m, Point\}$$

Definition 1. Predicate Match

A value-pair appears in e.V is denoted by V_n,$V_n.A$ donates the attribute of the value-pair and $V_n.V$ denotes the value. For a predicate P_m appears in a subscription, we said that there is a predicate match if $P_{m(V_n.A,f_{op},V_n.V)} = true$.

Definition 2. Boolean Expression Match

A boolean expression s.B is said to match a collection of value-pairs e.V if each of the predicates in s.B has a match in e.V.

Definition 3. Subscription Match

Given a collection of subscriptions $S = \{s_1, s_2...s_m\}$, each subscription has a boolean expression and a spatial constraint denoted by s.B and s.R respectively. For each event e in an event stream with a collection of value-pairs and a point message e.V and e.P, we said that $s_i \in S$ match e if $(s_i.R \cap e.P) \neq \varnothing$ and $s_i.B$ matches e.V.

Let us revisit the example we mentioned before in Fig. 1. There are 8 subscriptions and 2 events and their locations. For the event $E_1 = \{(A = 3 \wedge B = 3 \wedge C = 5), P_1\}$, the subscription S_1 matches E_1. However, the subscription S_4 doesn't match E_1 as there is no value-pair in E_1 that matches the predicate $G \geqslant 4$. The subscription S_8 is not an answer either, as its region R_8 has no overlap with P_1. That is, the answer of the event E_1 is S_1. Similarly, the answer of E_2 is S_7.

3 Related Works

The pub/sub system with boolean expressions has been studied for many years while the location-aware pub/sub system is a much more hotter issue in both academia and industry. However, existing works on pub/sub with boolean expressions [1,4,12,15] do little for location matches. Recently, there is a structured pub/sub work focuses on location-aware pub/sub with boolean expression over dynamic event streams [7] which mainly deal with the moving features of subscriptions, it adapt a disk-based method to handle the moving feature of events which result in less efficiency in event matching compared to our memory-based method. In contrast, our work is mainly focus on efficiently filter subscriptions with boolean expression and locations. In addition, location-based pub/sub systems are mainly for unstructured information [5,6,8–10,14,16,18]. Traditional pub/sub systems with boolean expressions mainly focus on indexing support to efficiently identify matching subscriptions. Two newly proposed representative indexing methods for boolean expressions are BE-tree [12] and Opindex [4]. It is known that Opindex can achieve better matching performance with much smaller construction cost under different situations.

Opindex is consisted of a two-level index structure [4]. In its first level, subscriptions are grouped together according to selected pivot attributes (i.e., the attributes with minimum frequency) from each subscription. In its second level, subscriptions are divided by the operators ($e.g., \leqslant, =, \geqslant$) of predicates. In each group, predicates are divided by operators. Signature elements are used to map the predicates by a hash function. For each group of subscriptions, there are a corresponding counter arrays to track the matching predicates. The attributes of value-pairs in events are used as keys to grouped subscriptions that are divided by pivot attributes. Given an event, if an attribute in the event is indeed a pivot attribute, then find the corresponding group of predicates and enumerate each value-pair of the event to search predicate lists that are divided by operators.

Opindex is an efficient index for boolean expressions with a large amount of attributes. We can add the predicates for location information into subscriptions. However, the newly added predicates are apparently a burden for Opindex. In the respect of index efficiency, it is a waste to abandon the pruning ability of location information. The existing unstructured pub/sub systems, however, can only support textual descriptions. Thus, it is not accurately enough to represent items with multi-attributes and values. In contrast, our approach can support items with a great amount of boolean expressions with spatial information and achieve a high matching rate.

4 Preliminary Solutions

In this section, we first propose two preliminary solutions by making some extensions for a well-known index Opindex for pub/sub systems to handle the new problem, i.e., location-aware pub/sub with boolean expressions. Note that these two preliminary solutions are used as baselines for evaluating our advanced solution discussed in Sect. 5.

4.1 Extension with Location Predicates

As we know, there exists a well-known index called Opindex for pub/sub systems. Opindex does not contain predicates for locations. If we apply Opindex to location-aware pub/sub systems, we can extend it straightforwardly by just adding predicates into subscriptions to present spatial constraints. For example, for a region ($lat_1 = 51.25144123$, $long_1 = -0.14251263$, $lat_2 = 51.335125445$, $long_2 = 0.12142324$), we can use operators ($\leqslant, \geqslant$) to present region constraints as ($lat_1 \geqslant 51.25144123$, $long_1 \geqslant 0.14251263$, $lat_2 \leqslant 51.335125445$, $long_2 \leqslant 0.12142324$). And then we can straightly utilize Opindex to solve the problem of location-aware pub/sub index with boolean expressions. This extension of Opindex is called Opindex-loc in this paper. Obviously, this method abandons the ability of pruning spatial constraints which leads to high computation cost.

4.2 Extension Using R-tree

R-tree is a well-known index structure to index spatial information, in which can use a minimum bounding rectangular (MBR) to denote spatial regions. The basic idea of this extension is to utilize R-tree as an index for spatial information and organize boolean expressions in the Opindex structure. First, we build an R-tree to index the spatial constraints in subscriptions. Then we organize the subscriptions in the form of the Opindex structure. We append a counter array for each grouped subscription partitioned by a pivot attribute. We initialize the counter array with the sizes of boolean expressions in the subscriptions. Then, for processing event matching, we get the value-pairs of an event to validate the boolean expressions of the subscriptions, for each matched predicate in a subscription, the corresponding element in its counter array is decremented by one. And candidate subscriptions are generated when their corresponding counter elements in counter arrays goes to zero. Then, using the spatial information of the event as an input, we search the R-tree. For each satisfied region, we get its identification to validate the candidates to find the final matched subscriptions to return. Since boolean expressions are first visited to generate candidates, then we called this extension as BF-Opindex.

5 Hybrid Index Structure

As we discussed in Sect. 4.1, both the two mentioned method result in poor performance in event matching since the Opindex-loc abandons the pruning ability of spatial information and the BF-Opindex generates many candidates whose spatial constraints are not satisfied. To improve the performance in event matching, we present a novel partitioning index structure RP-trees, which organizes the regions of subscriptions into disjointed R-trees. In the RP-trees index, each subscription has a representative attribute termed as a pivot attribute denoted by δ_A. The subscriptions are partitioned by two steps. First, the subscriptions are partitioned based on the sizes of subscriptions. And then subscriptions are

further partitioned by their pivot attributes. Thus, subscriptions with the same sizes and the same pivot attributes are grouped together using inverted lists. To index the spatial constraints, we build an R-tree using the spatial regions of subscriptions in each list. Given an input event, we search the corresponding R-tree to generate candidates and validate boolean expressions of these candidates. For convenience, we summarize the key notations used in this paper as shown in Table 1.

Table 1. Notation

s	A subscription
e	An event
δ_A	The pivot attribute of a subscription s
$s.R$	The spatial region of a subscription s
$s.B$	The boolean expression of a subscription s
$e.P$	The spatial point of an event e
$e.V$	The value-pairs of an event e
k	The number of predicates of a subscription s
m	The number of attributes in an event
S	The entire subscription dataset
E	The entire events dataset
$L_{(k)}$	List of subscriptions partitioned by k
$L_{(\delta_A)}$	List of subscriptions partitioned by δ_A
$L_{(k,\delta_A)}$	List of subscriptions partitioned by k and δ_A
C	The counter array correspond with $L_{(k,\delta_A)}$
C_i	The counter element in C

5.1 Boolean Expression Index

We index the boolean expressions of subscriptions in two partitioning steps. In the first step, subscriptions are partitioned into disjointed subscription lists based on the sizes of the subscriptions k as follows:

$$S = L_{(k_1)} \cup L_{(k_2)} \cup L_{(k_3)} \cup .... \cup L_{(k_m)}$$

If a boolean expression $s.B$ in a subscription s matches the value-pairs $e.V$ in an event e, then the size of the event m must not less than the size of subscription k. Obviously, if k is larger than m, then, there must be a predicate of any subscription s in $L_{(k)}$ which cannot be matched. According to Definition 2, e is guaranteed not to match with subscriptions in $L_{(k)}$.

In the second step,subscriptions with the same size are further partitioned according to the selected pivot attributes δ_A as follows:

$$L_{(k_i)} = L_{(\delta_{A1})} \cup L_{(\delta_{A2})} \cup L_{(\delta_{A3})} \cup .... \cup L_{(\delta_{Am})}$$

From the Definition 1, we can conclude that if an event e matches a subscription s, then all the attributes in s must appear in e. Obviously, if there is an attribute in s which is not in e, e will be concluded not match s. Thus, given an event e, we only consider the subscriptions whose pivot attributes appears in e. Attributes with less frequency bring more probability to filter subscriptions because attributes with less frequency has less probabilities to appear in subscriptions. Thus, we chose the least frequency attribute in a subscription as pivot attribute.

For each subscription s in $L_{(k,\delta_A)}$, we establish a predicate storage structure based on standard operators $(=, \geqslant, \leqslant)$ of the subscription. Other operators $(<, >)$ are treated similarly. And the collection operator $\in$ is rewritten using the standard operator. For example, $F \in (5, 2, 1)$ can be rewritten into $F = 5 \vee F = 2 \vee F = 1$. Using a counting-based algorithm, the counter arrays can detect matched subscriptions for an input event. For each subscription s_i in $L_{(k,\delta_A)}$, there is a counter array C to track the number of predicates in $s_i.B$ that has not been matched during an event searching process. Each counter array is initialized as the sizes of its corresponding subscriptions, and the corresponding counter element in counter array is decremented by one whenever a value-pair in $e.V$ of a searching event e matches a predicate in $s_i.B$. Thus, $s_i.B$ matches $e.V$ when the element C_i for S_i in the counter array goes to zero.

The boolean expression index structure for the running example subscriptions in Fig. 1 is shown in Fig. 2. In the first step subscriptions are partitioned by their sizes into three lists L_2, L_3 and L_4. Next, a pivot attribute is selected according to the appearance frequency in the datasets. Thus, A, D, E and G are selected as the pivot attributes respectively. Given an event $E_1 = \{(A = 3 \wedge B = 3 \wedge C = 5), P_1\}$, according to Definition 2, subscriptions in $L_{(4)}, L_{(3,G)}, L_{(2,D)}$ are guaranteed not to match with E_1. Figure 2 also shows the counter arrays and predicates storages structure of S_2, S_5, S_7.

Subscription	Size	Pivote Attribute
S_1	3	A
S_2	2	D
S_3	3	E
S_4	3	G
S_5	4	G
S_6	3	E
S_7	3	E
S_8	3	A

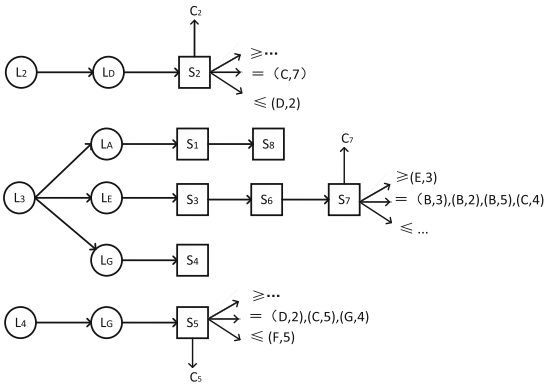


Fig. 2. Boolean expression index structure

5.2 Combine Boolean Expressions and Spatial Index

In the rest of this section, we present the structure of RP-trees. The RP-trees contains three components that are shown in Fig. 3. The first component is a collection of the inverted lists of boolean expressions derived from the two-step partitioning method that have been described above. The second component is a set of corresponding R-trees for each partitioned subscription list $L_{(k,\delta_A)}$. The corresponding R-trees are used to filter the spatial information, and generate candidate subscriptions whose spatial constraints are satisfied by events. The third component is a collection of counter arrays, corresponding to the boolean expression lists $L_{(k,\delta_A)}$.

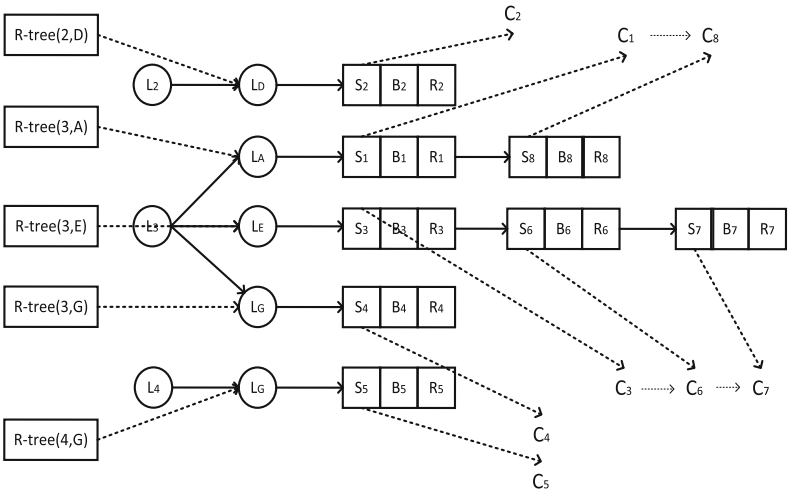


Fig. 3. RP-trees index structure

Algorithm 1 shows an algorithm to insert a new subscription s_i into the RP-trees index. We transfer the collection operator $\in$ into the standard operators as described in Sect. 5.1. Next, we determine the size k_i and the pivot attribute δ_{Ai} of s_i, append a counter element C_i for s_i in counter array C and initial C_i with the size of $s_i.B$. Then the spatial information $s_i.R$ is inserted into the newly built R-tree or an existing R-tree corresponding with $L_{(k,\delta_A)}$.

To explain how a RP-trees index is built, we revisit our running example here. The RP-trees index for the subscriptions in Fig. 1 is shown in Fig. 3. The boolean expressions of the subscriptions are first partitioned and are inserted into lists L_2, L_3 and L_4. And then, each boolean expression of a subscription list is further partitioned into several lists $L_{(2,D)}, L_{(3,A)}, L_{(3,E)}, L_{(3,G)}, L_{(4,G)}$ according to their pivot attributes (i.e., A, E, D and G). For each boolean expression list above, we append an R-tree to index their spatial regions and there are five counter arrays for the eight subscriptions in Fig. 1.

Algorithm 1. *Subscription Insert(s)*

```
1: Determine the size k of s
2: Get the pivot attribute δ_A from s
3: Get predicates p,and the region R from s
4: if L_(k) not exists then
5:     Define L_(k), L_(k,δ_A), R − tree_(k,δ_A), C, Ci
6: else
7:     if L_(k,δ_A) not exists then
8:         Define L_(δ_A), R − tree_(k,δ_A), C, Ci
9:     else
10:        for each(R, p) ∈ s do
11:            Insert(R) into R − tree_(k,δ_A)
12:            Insert (P) into L_(k,δ_A)
13:            Initial C_i with k
```

6 Query Processing

Algorithm 2 provides the process of how RP-trees return matching subscriptions for an input event e. Before the process of searching the RP-trees index, we initialize the set of matching subscriptions R to be empty, and set each value in corresponding counter array to its respective subscription size. First, we calculate the size of e. For each boolean expression list $L_{(k_j)}$, we scan $L_{(k_j)}$ if the size of e is not less than k_j. For each boolean expression list $L_{(k_j,\delta_A)_i}$, we enumerate the attributes A_j from the input event e. If A_j is a pivot attribute, we use a spatial point $e.P$ as a query on the corresponding R-tree to filter out candidate subscriptions. Then, the boolean expressions of the candidate subscriptions are refined by the value-pairs in e. When a predicate stored in $s_j.B$ is matched by a value-pair, the corresponding value in counter array is decremented by one. If the counting value goes to zero, we have a matching subscription for e, which is added to the result set R.

Let us revisit our running example here. Consider the process of the event $E_1 = \{(A = 3 \wedge B = 3 \wedge C = 5), P_1\}$ using the RP-trees index in Fig. 3. The size of the event is 3, so the subscription list $L_{(4)}$ is pruned. For the list $L_{(2)}$, there is no pivot attribute in E_1, thus, $L_{(2)}$ is pruned. For the list $L_{(3)}$, only attribute A is contained in E_1. Therefore, only the $R - tree_{(3,A)}$ should be searched for the regions overlapped with P_1. And the region R_1 does overlap with P_1, then, candidate boolean expression $S_1.B$ is scanned, and the value-pairs of E_1 satisfies $S_1.B$. Thus, S_1 is added to the result set R.

7 Experiments

In this section, we report our experimental results of evaluating the performances of our proposed RP-trees index. We compare it with two preliminary methods, which are described in Sect. 4. The implementation of Opindex is implemented according to the algorithms proposed by its authors. All the indexes are memory

Algorithm 2. *Matching Algorithm*

Require: *an event e.*
Ensure: *result set of subscriptions R.*

Initialize :$R \leftarrow \{\}$
Get the size m of e
Get the distinct value-pairs $A_j = V_j$ and the point location $e.P$ from e.
for each $L_{(k_i)}$ in the RP-trees index **do**
  if $m \geqslant k_i$ **then**
    for each $L_{(\delta_{A_i})}$ in $L_{(k_i)}$ **do**
      if $A_j = \delta_{Ai}$ **then**
        Search $e.P$ in $RP - tree_{(k_i,\delta_{A_i})}$
        if $e.P$ overlap s.$R \in RP - tree_{(k_i,\delta_{A_i})}$ **then**
          for each predicate p in $s.B \in L_{(k_i,\delta_{A_i})}$ **do**
            if $A_j = V_j$ satisfies p **then**
              $C_i \leftarrow C_i - 1$
              if $C_i = 0$ **then**
                Add the subscription s to the result set R.
                return R

resident and implemented in Java. We conduct the experiments on a server with 256 G memory, 64 KB L1 cache and 512 KB L2 cache, running centos 5.6.

7.1 Data Generator

To generate the synthetic dataset, we implement a data generator, which can generator attributes, operators and values, uniformly distributed or follows Zipf distribution. Three main operators $(=, \geqslant, \leqslant)$ are supported. we generate 5 k to 20 k distinct attributes, following Zipf distribution from 0.2 to 1.0. We vary the number of subscriptions from 1 M to 10 M to test the scalability. The max size of each generated subscription k varies from 4 to 20 while the size of events m varies from 5 to 20. Default parameters for testing the scalability on a real-world dataset (58 city) are set as follows: The size of each subscription is from 2 to 10, the size of events is from 4 to 12, the number of distinct attributes is set to 5,000, and a zipf law is used in the distribution of attributes with an alpha value 0.8. For the spatial information, we generate 1 M longitudes and latitudes to compose the point locations for an event and a region for a subscription.

Besides the synthetic datasets, we also design a data generator from real-world datasets. The dataset is extracted from 58 city. It is a series products information with its name, price, address and other attributes of the products. Based on the address, we extract the coordinates in the form of longitude and latitude from Baidu API[5]. And then we generate MBRs by selecting a location of a product as the center and extending a random width and height with a upper bound. Similarly, we generate the value of attributes by select the value of the attribute in a product information as mid-value and extending a random value to add to or subtract from the mid-value as a value of predicate in a subscription.

[5] http://developer.baidu.com.

In summary, we totally generates 10 M synthetic subscriptions corresponding with 0.1 M events for matching tests. For the real-world dataset, we generate 5 M subscriptions and 0.1 M events from 58 city. Table 2 summarizes the parameters and their settings of the two datasets.

Table 2. Parameters and settings

Parameters	Synthetic dataset	58 city
Number of subscription	2 M, 4 M, 6 M, 8 M, 10 M	1 M, 2 M, 3 M, 4 M, 5 M
Max subscription size	4∼20	2∼10
Max event size	5 25	4 12
Zipf	0.2, 0.4, 0.6, 0.8, 1.0	Uniformly distributed
Number of distinct attribute	5 k, 10 k, 15 k, 20 k, 25 K	10 K
Node capacity of R-tree	40	40

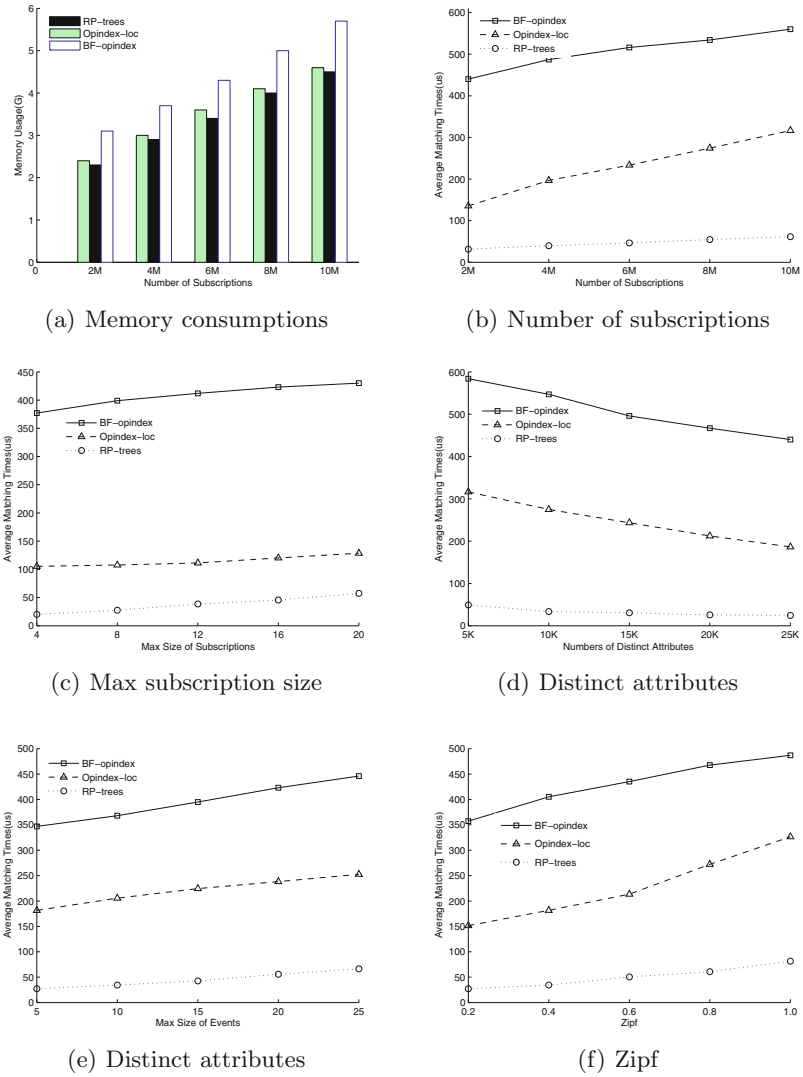
7.2 Experimental Results

In this section, we will evaluate three indexes on synthetic datasets and real-world datasets. For synthetic datasets we evaluate the performance of three indexes from different perspectives, the memory consumption, a varied number of subscriptions and distinct attributes, varied max size of subscriptions, varied max size of events, varied Zif distributions

Memory consumption: Since all the three indexes are memory-resident, we first evaluate the memory consumption for the three indexes by varying the number of subscriptions. The experimental results are shown in Fig. 4(a). We can see that our RP-trees consumes less memory, comparing with Opindex-loc and BF-Opindex. This is because RP-trees need not to generate signature elements while the other two methods needs signature elements to track the predicates for each subscription.

Matching time on different number of subscriptions: From Fig. 4(b), we can see that the our RP-trees index achieves the best event matching time. This is because partitioning subscriptions using pivot attributes is not as efficient as using the hybrid partitioning of the sizes and the pivot attributes.Furthermore, R-tree with a small amount of regions partitioned by the sizes and the pivot attributes can efficiently generate candidate subscriptions. Compared with Opindex-loc, it avoids a large amount of unnecessary boolean expressions whose spatial constrains are not satisfied to calculate. In summary, our RP-trees can get the least amount of candidates for both spatial information and boolean expressions.

Matching time on different size of subscriptions: The average event matching time on different size of subscriptions for the three indexes are reported in



(a) Memory consumptions (b) Number of subscriptions

(c) Max subscription size (d) Distinct attributes

(e) Distinct attributes (f) Zipf

Fig. 4. Evaluation for the three indexes on synthetic dataset

the Fig. 4(c). As we can see from the Fig. 4(c), RP-trees index scales better compared to Opindex-loc and BF-Opindex. The reason is, with the increasing k, the pruning ability for the first partitioning step of RP-trees become more and more powerful since events only visit the subscriptions whose size are not large than that of events.

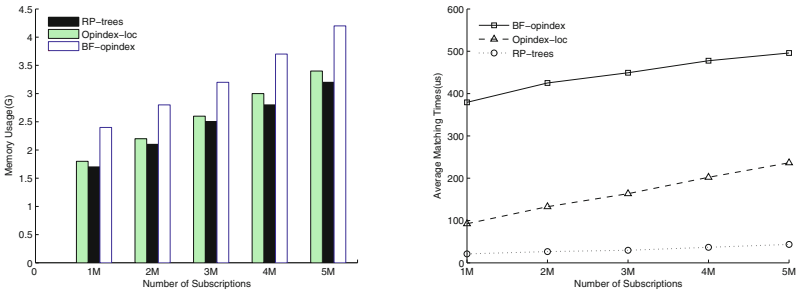
Matching time on different number of distinct attributes: The number of distinct attributes is an important parameter for the three indexes, because all of the indexes are used pivot attributes to partition subscriptions. An obvious

observation from the Fig. 4(d) is that when the number of attributes increases, the matching times of the three indexes decrease because all of the three indexes can generate more narrowed partitions with the increment of the number of attributes. And the matching time of our RP-trees decreases apparently with the increasing number of attributes.

Matching Time under Different sizes of Events: The experimental results are shown in Fig. 4(e). An obvious observation is that our RP-trees scale similarly to Opindex-loc and BF-Opindex. Both RP-trees and the other two indexes are sensitive to the size m of the event. With the size m increases, The collection of candidate subscriptions grows at the same time. There are two reasons for this. First, the pruning ability for the first step partitioning of RP-trees directly decreases since the chance for a larger than m becomes smaller with the increasing of m. Second, events will access more candidate subscriptions partitioned by the second step of partitioning, since there are more attributes in an event with the increasing of m, which increases the probability to access subscriptions partitioned by pivot attributes.

Matching Time under Different Alpha Values of Zipf: We also evaluate the matching time by changing the alpha value from 0.2 to 1.0 in Zipf distribution. The experimental results are shown in Fig. 4(f). An obvious observation is that the matching times for the three indexes all grow with the increasing alpha value of Zipf. Our RP-trees index scales better than both Opindex-loc and BF-Opindex, because of the powerful partitioning ability of RP-trees.

For the real-world dataset from 58 city, we varied the number of the subscription to evaluate the scalability and the memory consumption. We can see from the Fig. 5, our RP-tree achieves the minimum event matching and memory consumption, and with the increases of subscriptions, the event matching time increased nonlinearly. This is because with number of subscriptions increased, RP-trees still pruned large amount of unnecessary subscriptions.



(a) Memory consumptions (b) Varied number of subscriptions

Fig. 5. Evaluation for the three indexes on 58 dataset

8 Conclusion

In this paper, we tackled the problem of location-based matching in pub/sub for boolean expression which is significant for location-based pub/sub system with a large amount attributes and values. Facing the challenge of efficiently delivering events to the corresponding subscribers, we propose a novel index structure called RP-trees, which using a two-step partitioning method to organize the boolean expression and the state-of-art R-tree to index spatial information. Extensive experiments conducted in both synthetic and real datasets demonstrate the effectiveness of our algorithms. In the future, we plan to support top-k pub/sub matching.

Acknowledgment. This work was partially supported by Chinese NSFC project (61402312, 61170020, 61402311, 61440053), and the US National Science Foundation (IIS-1115417).

References

1. Bittner, S.: Supporting arbitrary boolean subscriptions in distributed publish/subscribe systems. In: Proceedings of the 3rd International Middleware Doctoral Symposium (MDS 2006) (2006)
2. Chen, L., Cong, G., Cao, X., Tan, K.L.: Temporal spatial-keyword top-k publish/subscribe. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 255–266 (2015)
3. Cugola, G., Margara, A.: High-performance location-aware publish-subscribe on GPUs. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 312–331. Springer, Heidelberg (2012)
4. Zhang, D., Chan, C.Y., Tan, K.L.: An efficient publish/subscribe index for E-commerce databases. Proc. VLDB Endow. **7**(8), 613–624 (2014)
5. Eugster, G.: Location-based publish/subscribe. In: 2013 IEEE 12th International Symposium on Network Computing and Applications, pp. 279–282 (2005)
6. Li, G., Wang, Y., Wang, T.: Location-aware publish/subscribe. IEEE Trans. Knowl. Data Eng. **27**(4), 950–963 (2013)
7. Guo, L., Zhang, D., Li, G., Tan, K.L., Bao, Z.: Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (2015)
8. Huang, Y., Garcia-Molina, H.: Publish/subscribe in a mobile environment. Wireless Netw. **10**(6), 643–652 (2004)
9. Kai Zheng, P.C.F., Zhou, X.: K-nearest neighbor search for fuzzy objects. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (2010)
10. Lai, S., Wang, G.: P2p streaming media resource location algorithm based on publish/subscribe. Henan Science (2012)
11. Naicken, S.M.: Trusted content-based publish/subscribe trees. University of Sussex (2012)
12. Sadoghi, M., Jacobsen, H.-A.: Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space. In: ACM Conference on Management of Data, pp. 637–648 (2011)

13. Sadoghi, M., Jacobsen, H.A.: Location-based matching in publish/subscribe revisited. In: Proceedings of the Posters and Demo Track (2012)
14. Tam, D., Azimi, R., Jacobsen, H.-A.: Building content-based publish/subscribe systems with distributed hash tables. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) DBISP2P 2003. LNCS, vol. 2944, pp. 138–152. Springer, Heidelberg (2004)
15. Whang, S.E., Brower, C., Shanmugasundaram, J.: Indexing boolean expressions. Stanford InfoLab **2**(1), 37–48 (2009)
16. Xiang Wang, Y.Z., Xuemin Line, W.W.: Ap-tree: efficiently support continuous spatial-keyword queries over stream. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 1107–1118 (2015)
17. Yu, M., Li, G., Wang, T., Feng, J., Gong, Z.: Efficient filtering algorithms for location-aware publish/subscribe. IEEE Trans. Knowl. Data Eng. **27**(4), 1 (2015)
18. Zheng, K., Su, H., Zheng, B., Shang, S., Xu, J., Liu, J., Zhou, X.: Interactive top-k spatial keyword queries. IEEE (2015)