MO-Tree: An Efficient Forwarding Engine for Spatiotemporal-aware Pub/Sub Systems

Tianchen Dian, Shiyou Qian, *Member, IEEE,* Jian Cao, *Member, IEEE* Guangtao Xue, *Member, IEEE,* Yanmin Zhu, *Member, IEEE,* Jiadi Yu, *Member, IEEE,* and Minglu Li, *Member, IEEE*

Abstract—For large-scale spatiotemporal-aware publish/subscribe systems, it is critical to design an efficient forwarding engine to achieve fast matching and maintenance of events and subscriptions. For this goal, we propose a novel data structure called MO-Tree to index both subscriptions and events in a unified way. The design philosophy behind MO-Tree is to keep the data structure concise, which manifests in three aspects: limiting the height of MO-Tree, trading space for time, and avoiding node merging and splitting. The difficulty in designing MO-Tree is how to efficiently index width-variable intervals. We present a multi-level cell-overlapping partition scheme and build a theoretical model to optimize the cell width in each level. To evaluate the performance of MO-Tree, a series of experiments is conducted on real-world trace datasets. The experiment results show MO-Tree significantly outperforms the state-of-the-art in terms of matching speed and maintenance cost.

Index Terms—Spatiotemporal-aware, publish/subscribe, event matching, event maintenance, subscription matching, subscription maintenance.

1 Introduction

With the popularization of mobile devices such as smart phones, many applications continuously generate huge volumes of location-tagged and time-sensitive data, such as vehicular networks [1] [2], the internet of things [3] and mobile e-commerce [4]. These applications give rise to the demands on real-time spatiotemporal-aware data dissemination.

Fig. 1 illustrates a mobile e-commerce scenario. On the one hand, businesses periodically publish time-sensitive sales promotions in the form of events, such as {brand=Huawei, model=P30, price=¥3200, capacity=128GB, until=12/5/2020, lon=121.4419, lat=36.1628}. These events contain the coordinates of businesses, namely lon and lat, and the validity of promotions expressed by $until$. Each event may last for some time and satisfy the preferences of late-joining customers. On the other hand, customers holding a mobile device wish to receive sales promotions that satisfy their preferences expressed in the form of subscriptions, such as {brand=Huawei $\wedge$ model=P30 $\wedge$ price<¥3300 $\wedge$ distance<1km}. Customers specify their desirable notification areas centered at the consumer's current location by $\{distance<1\text{km}\}$, depicted by the red square in Fig. 1. As customers are moving, their subscriptions should be continually updated based on their changing locations.

This scenario exhibits three characteristics of spatiotemporal-aware data dissemination, namely fine-grained expressiveness, temporal sensitivity and location-awareness. Firstly, the sales promotions of products or services usually have tens of attributes. A high-dimensional content space allows great expressive power through Boolean expressions. Secondly, each event has a validity after which it is expired. Thirdly, each subscription specifies a spatial notification area which moves with the consumer.

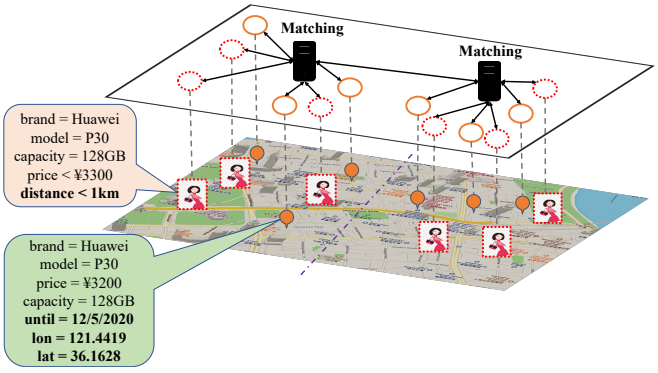

Fig. 1. A scenario illustrating spatiotemporal-aware event dissemination.

Four kinds of operations are critical to realize spatiotemporal-aware data dissemination. 1) Subscription maintenance: all subscriptions issued by customers should be efficiently maintained to promote event matching performance, which involves subscription insertion, deletion and update. 2) Event maintenance: similarly, all unexpired events should be efficiently maintained to improve subscription matching performance, which involves event insertion and deletion. 3) Subscription matching: when receiving a new subscription, it should be matched against the set of unexpired events in order to send the customer the events of interest. 4) Event matching: when an event is published, it should be matched against the set of subscriptions in order to send the event of interest to customers.

It is very challenging to guarantee rapid spatiotemporal-aware event dissemination when the number of subscrip-

- *T. Dian, S. Qian, J. Cao, G. Xue, Y. Zhu and J. Yu are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. S. Qian is the corresponding author. E-mail: { dtcccc, qshiyou, cao-jian, gt_xue, yzhu, jiadiyu}@sjtu.edu.cn*
- *M. Li is with the College of Mathemat-ics and Computer Science, Zhejiang Normal University, Zhejiang, China. E-mail: mlli@sjtu.edu.cn*

Manuscript received April 19, 2020; revised August 26, 2020.

tions is large and the incoming rate of events is high[1]. Some pub/sub systems have been proposed to deal with this challenge. For example, the work in [2] proposes a prediction-based opportunistic spatiotemporal event processing system. Hong et al. build a spatiotemporal-aware pub/sub middleware called Grus and present a composite subscription language [3].

However, to the best of our knowledge, none of the existing pub/sub systems presents a high-performance overall solution which supports the efficient matching and maintenance operations of both events and subscriptions. For this goal, we propose a novel data structure called MO-Tree to index both events and subscriptions in a unified way. MO-Tree is a tree-like data structure in which subscriptions and events are stored in the leaf nodes. A pruning-based search strategy is employed to support event matching and subscription matching.

The design philosophy behind MO-Tree is to keep it as concise as possible, which manifests in three aspects. Firstly, the shape of MO-Tree is flattened by limiting the height of the tree and widening the fan-out of nodes. Secondly, the number of items stored in each leaf node is limited by utilizing the strategy of trading space for time. Thirdly, node splitting and merging are avoided, which is different from traditional trees, such as R-Tree [5] and QuadTree [6]. In addition, information on children is not stored in the internal nodes of MO-Tree.

One difficulty in designing MO-Tree is to index subscriptions which are composed of interval predicates of different widths. We present a hybrid partition scheme to divide the value domain of attributes into multi-level overlapping cells to deal with the variety of interval widths. Furthermore, we build an optimization model to compute the width of overlapping cells in each level.

To evaluate the overall performance of MO-Tree, we conduct a series of experiments on real-world trace datasets. MO-Tree is compared with three state-of-the-art event matching algorithms, TAMA [7], R*-Tree [8] and OpIndex [9], and two subscription matching algorithms, BEQ-Tree [10] and R*-Tree. The matching time and maintenance cost of all the tested algorithms are evaluated in different settings. The experiment results show MO-Tree significantly outperforms the state-of-the-art in terms of matching speed and maintenance cost.

The main contributions of this paper are as follows:

- We propose a novel data structure called MO-Tree to index both events and subscriptions in a unified way.
- We present a multi-level cell-overlapping partition scheme for indexing width-variable interval predicates.
- We conduct comprehensive experiments to evaluate the performance of MO-Tree based on real-world datasets.

The rest of the paper is organized as follows. We define the data model and the problem in Section 2. We review related work in Section 3. The design of MO-Tree is detailed in Section 4. We describe the optimization model to compute the width of cells in Section 5. The performance evaluation of MO-Tree is presented in Section 6. We conclude the paper in Section 7.

1. For example, the Miaojie App launched by Alibaba provides massive discount information for millions of consumers, reaching 52 million total downloads in Feb. 2020.

2 Data Model and Problem Definition

2.1 Data Model

Definition 1. ***Event*** *An event* $E = \{A_1 = V_1 \wedge A_2 = V_2 \wedge ... \wedge A_M = V_M\}$ *is a conjunction of multiple attribute-value pairs.*

The set of attributes in all events forms the space in which subscribers can specify sub-spaces of interest. In the spatiotemporal-aware scenario, the location of publishers is automatically appended to events in the form of $\{lon = 121.4419, lat = 36.1628\}$. Meanwhile, the validity of events is specified by the publishers when events are generated, such as $until = 12/5/2020$.

Definition 2. ***Interval Predicate*** *An interval predicate* $I = \{A, V_l, V_h\}$ *defines a condition on attribute* A*, where* V_l *and* V_h *are two operands specifying the low value and high value of an interval respectively, and* $V_l \leq V_h$*.*

Given the value domain and numeric precision of attributes, other types of predicates defined on relational operators $\{>, \geq, <, \leq, =\}$ can be easily transformed to interval predicates. Given a value of an attribute V, the output $I_i(V)$ of an interval predicate I_i defined on the same attribute is either true or false, indicating whether the predicate is satisfied or not by the value.

Definition 3. ***Subscription*** *A subscription* $S = \{I_1 \wedge I_2 \wedge ... \wedge I_K\}$ *is a conjunction of multiple interval predicates.*

A subscription is used to express a subscriber's interest in events. The notification area of a subscription is represented by a square in this paper, which is the conjunction of two spatial interval predicates defined on lon and lat. When subscribers issue subscriptions, they specify the maximum notification distance r which is automatically transformed to two spatial interval predicates according to their current location.

Definition 4. ***Match*** *A subscription* S *is a match of an event* E *denoted as* $S \sim E$ *only*

$$\forall I_i \in S, \exists (A_j = V_j) \in E : Att(I_i) = A_j \wedge I_i(V_j) = true$$

where $Att(I_i)$ *returns the attribute defined in* I_i*.*

Definition 5. ***Event matching*** *Given an event* E *and a set of* N_s *subscriptions* $\mathbb{S} = \{S_1, ..., S_{N_s}\}$*, event matching finds the set of matches* $\mathbb{S}_m$ *of* E *from* $\mathbb{S}$*.*

$$\mathbb{S}_m = \{S_i \in \mathbb{S} \wedge S_i \sim E\}$$

Definition 6. ***Subscription matching*** *Given a subscription* S *and a set of* N_e *events* $\mathbb{E} = \{E_1, ..., E_{N_e}\}$*, subscription matching finds the set of matches* $\mathbb{E}_m$ *of* S *from* $\mathbb{E}$*.*

$$\mathbb{E}_m = \{E_i \in \mathbb{E} \wedge E_i \sim S\}$$

2.2 Problem Definition

In this paper, we focus on designing an efficient forwarding engine for spatiotemporal-aware content-based pub/sub systems. The engine should support high-performance event matching and subscription matching with a low cost for maintaining subscriptions (insertion, deletion and update) and events (insertion and deletion). Our objective is to propose a novel data structure which is able to efficiently index both events and subscriptions.

3 Related Work

Existing work related to our study include location-aware keyword search, location-aware pub/sub systems and content-based pub/sub systems.

3.1 Location-aware Keyword Search

Based on canonical R-Tree [5], R*-Tree [8] or Quad-Tree [6], many efficient data structures have been proposed to retrieve items of interest from a large spatio-keyword database, such as KR^*-Tree [11], IR^2-Tree [12], IR-Tree [13] and IUR-Tree [14]. Felipe et al. present IR^2-Tree which combines an R-tree with text signatures to find the top-k answers [12]. Lu et al. design a hybrid index tree called IUR-tree (Intersection-Union R-Tree) that effectively combines location proximity with textual similarity [14]. Cong et al. present a location-aware textual search model by integrating an R-tree and inverted files [15]. Zhou et al. integrate inverted files and R^*-trees to handle both textual- and location-aware queries [16]. Chen et al. propose a new hybrid index structure called Inverted File Quad-tree (IQ-tree) to match Boolean range continuous queries over geo-textual objects in real time [17].

3.2 Location-aware Pub/Sub Systems

To improve the event matching performance of location-aware pub/sub systems, different data structures have been proposed, such as R^t-Tree [18], AP-Tree [19], R^I-Tree [20], RP-Tree [21], BEQ-Tree [4], PT-QuadTree [10] and IQ-Tree [17]. Zhao et al. propose an R^t-Tree to index subscriptions, which adds some keywords into R-tree nodes [18]. The work in [21] proposes RP-trees to index Boolean expressions, which integrates an R-tree and a Boolean expression index. In [4], Guo et al. design a Boolean Expression QuadTree (BEQ-Tree) based on a Quad-Tree and the counting-based matching strategy [22] to support efficient subscription matching and event update. The work in [10] proposes the PT-QuadTree which adds the textual descriptions into the quadtree nodes.

3.3 Content-based Pub/Sub Systems

In the previous few decades, researchers have proposed many efficient data structures to promote the event matching efficiency of content-based pub/sub systems, including Be-Tree [23], OpIndex [9], TAMA [7], H-Tree [24] and Rein [25]. TAMA [7] and OpIndex [9] are counting-based matching algorithms which adopt a two-phase matching procedure. H-Tree [24] and Be-Tree [23] utilize the pruning capability of trees to filter unmatching subscriptions in the searching process and verify each candidate to obtain real matches. The basic idea behind Rein [25] is to quickly search unmatching subscriptions to indirectly obtain the matches.

TABLE 1
Comparison of existing matching algorithms

Data Structure	Eve. Mat.	Sub. Mat.	Mat. Sem.
KR^*-Tree [11]	×	✓	KW
IR^2-Tree [12]	×	✓	KW
IR-Tree [13]	×	✓	KW
IUR-Tree [14]	×	✓	KW
IQ-Tree [17]	×	✓	KW
BEQ-Tree [4]	×	✓	BE
PT-QuadTree [10]	✓	×	KW
AP-Tree [19]	✓	×	KW
R^t-Tree [18]	✓	×	KW
R^I-Tree [20]	✓	×	KW
RP-Tree [21]	✓	×	BE
H-Tree [24]	✓	×	BE
Rein [25]	✓	×	BE
OpIndex [9]	✓	×	BE
Be-Tree [23]	✓	×	BE
TAMA [7]	✓	×	BE
MO-Tree	✓	✓	BE

[1] Eve., sub., mat., and sem. represent event, subscription, matching and semantic respectively.
[2] BE and KW stand for Boolean expression and keyword respectively.

3.4 Summary

As summarized in Table 1, existing solutions have three shortcomings in the spatiotemporal-aware data dissemination scenario. Firstly, most of them only support keyword matching so they are unable to provide fine-grained expressiveness for subscribers. Secondly, most of them focus on either event matching or subscription matching, without providing a high-performance overall solution for the scenario of spatiotemporal-aware event dissemination. Thirdly, although traditional data structures like R-Tree and R*-Tree can support both types of matching, they have poor maintenance performance due to nodes split or merge with insertion or deletion.

Based on the existing work, we design a novel data structure to efficiently index subscriptions and events in a unified way. The structure support high-performance matching and maintenance operations of both subscriptions and events.

4 Design of MO-Tree

4.1 Design Philosophy

MO-Tree is a tree-like data structure which indexes subscriptions and events in a unified way. An item, either a subscription or an event, is mapped to and stored in only one leaf node (bucket) of an MO-Tree. To realize a high overall performance, our design philosophy is to keep MO-Tree as concise as by: 1) keeping the shape of MO-Tree fat, 2) limiting the number of items stored in the leaf nodes, 3) avoiding node merging and splitting, and 4) orchestrating matching and maintenance operations.

Firstly, the shape of MO-Tree is flattened by constraining the tree height and widening the fan-out of nodes. MO-Tree adopts a two-stage procedure to perform both event and

subscription matching: filtering and verification. As most unrelated leaf nodes are filtered out in the first stage, a small number of leaf nodes need to be further verified in the second stage. Thus, the pruning capability of a fat MO-Tree is fully utilized to promote matching performance.

Secondly, MO-Tree is targeted to index a large number of subscriptions and events, usually at the scale of millions. To limit the size of buckets (the number of subscriptions or events stored in leaf nodes), sufficient number of leaf nodes are constructed in an MO-Tree. As subscriptions or events are sparsely stored in the buckets, the performance of matching and maintenance can be improved at the cost of wasting some space consumed by empty buckets.

Thirdly, to further improve the maintenance performance, node merging and splitting are avoided in MO-Trees. Although merging and splitting are beneficial to balance the tree, the cost is not negligible. As matching is paused during the process of node merging or splitting, the matching throughput decreases. In addition, the internal nodes of an MO-Tree do not maintain any information on their children.

Fourthly, based on a common data structure for events and subscriptions, subscription matching and event deletion can be orchestrated. In the verification stage of subscription matching, expired events are deleted incidentally, which avoids taking specialized deletions for expired events. This orchestration not only amortizes the deletion cost, but also mitigates the impact of deletion on matching throughput.

Each non-leaf layer of an MO-Tree is built on an attribute called an indexing attribute. The value domain of each indexing attribute is partitioned into multiple cells. A one-to-one map from a value or an interval to a cell is realized. Based on the mapped cell of each indexing attribute, it is easy to identify the target bucket into which an event or a subscription is stored. Thus, constructing an MO-Tree involves three steps: 1) selecting some indexing attributes from the content space, 2) partitioning the value domain of indexing attributes to build a mapping rule, and 3) stacking multiple indexing attributes to form a tree. These three aspects are described in details in the following subsections.

4.2 Selection of Indexing Attributes

In this paper, we use $\mathbb{A}_I$ to denote the set of indexing attributes selected from the set of content space. In general, attributes with good selectivity, popularity and uniformity are preferred. We give three guidelines for the selection of indexing attributes, which are summarized as follows:

1) Popularity: Attributes that frequently appear in subscriptions can be first selected as candidates. Popular attributes are more likely to build a balanced MO-Tree than infrequent ones.
2) Selectivity: More cells can be partitioned on the value domain of indexing attributes with a high selectivity, which contributes to balancing MO-Tree.
3) Uniformity: For intervals with a given width, if their centers are uniformly distributed, they can be evenly mapped to the cells. This uniformity is beneficial to balance MO-Tree.

In the mobile e-commerce scenario, the longitude and latitude of coordinates, and the name, price, brand and model of products (or services) can be selected as indexing attributes since they are popular and selective. Of these attributes, the type of longitude, latitude and price is numerical while the type of the others is a string. For a string-type attribute, we can select the first one or more letters to form the value domain to which the proposed partition scheme can be applied. In addition, when designing MO-Tree, we fully take advantage of the locality of subscriptions by choosing longitude and latitude of coordinates as indexing attributes. This locality implied in MO-Tree makes it easy to build a distributed data structure to support efficient processing of events and subscriptions.

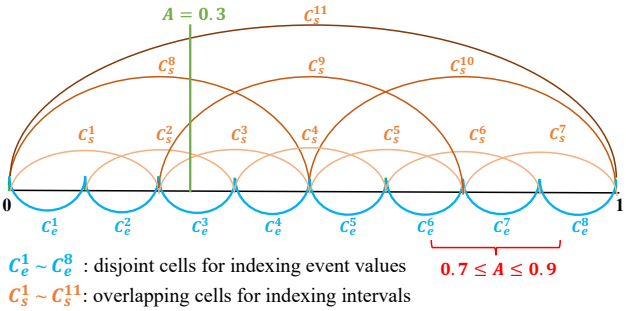

Fig. 2. Hybrid partition scheme of attribute value domain.

4.3 Partitioning Scheme

In order to index both single values of events and interval predicates of subscriptions in a unified way, we propose a hybrid partition scheme. The value domain of indexing attributes is partitioned into disjoint cells and multi-level overlapping cells for indexing values and intervals, respectively.

4.3.1 *Disjoint Partition for Values*

The partitioning scheme for indexing event values is straightforward, just like R-Tree [5] and Quad-Tree [6]. The value domain of each indexing attribute is first uniformly partitioned into a certain number of disjoint cells (the number of cells is a parameter). Then, one-to-one mapping can be easily undertaken between a value and a cell.

An example is shown in Fig. 2, where the value domain of attribute A is partitioned into 8 disjoint cells denoted by $\{C_e^1, ..., C_e^8\}$. To deal with the case where some events do not contain an indexing attribute (equivalent to a null value), we create a special null cell denoted by C_e^n. When an interval predicate is matched against the event values stored in the disjoint cells, only those cells that intersect with the interval contain candidate matches. As shown in Fig. 2, the interval $0.7 \le A \le 0.9$ intersects with C_e^6, C_e^7 and C_e^8, which will be further verified to obtain the real matches.

4.3.2 *Multi-level Overlapping Partition for Intervals*

Indexing subscriptions is more complex, as intervals have different widths and centers. The challenge is that we need a way to partition the value domain into a certain number of cells and build a one-to-one mapping rule between an interval and a cell. We are inspired by the overlapping

partition scheme of H-Tree [24] and address its disadvantage to only index intervals of a narrow width.

To index intervals of different widths, we propose a multi-level overlapping partition scheme to divide the value domain into cells. From top to bottom, the number of cells in each level increases and the width of the overlapping cells decreases level by level. Specifically, the topmost level contains only one cell covering the total value domain. This cell is used to store predicates with a large width and to cope with the case where subscriptions do not contain predicates defined on the attribute. With this multi-level cell-overlapping structure, an interval can map to the most suitable cell that is the smallest one completely containing the interval. In a level, if an interval can be contained by several neighboring overlapping cells, it is mapped to the cell whose center is the nearest to the center of the interval.

For example, Fig. 2 shows a three-level partition scheme with total 11 cells $\{C_s^1, ..., C_s^{11}\}$. The first (bottom) level ($C_s^1 - C_s^7$) is designed to store intervals whose width is smaller than $1/4$. The second level ($C_s^8 - C_s^{10}$) is for intervals with a width between $1/4$ and $1/2$. The top level (C_s^{11}) stores intervals with a width larger than $1/2$ in addition to the case where no predicate is defined on the attribute. When matching a value against intervals, all the cells that are stabbed by the value contain candidate matches. As shown in Fig. 2, when processing an attribute value $A = 0.3$, it stabs $C_s^2, C_s^3, C_s^8, C_s^9$ and C_s^{11}. These cells will be further checked, which is similar to subscription matching. In the matching process, all other cells are filtered out since they must not contain matches. In this example, the filtering efficiency on a single attribute is 1-5/11=54.5%.

From this example, we can see that the overlapping partition has two meanings. Firstly, two neighboring cells in the same level overlap each other, such as C_s^1 and C_s^2. Secondly, two cells in different levels may also overlap, such as C_s^1 and C_s^8. By constructing multi-level overlapping cells, we realize a one-to-one map from an interval with different widths and center to a cell.

The multi-level overlapping partition scheme has three parameters, i.e., the number of levels L, the number of cells c_i and the width of cells w_i in each level. There is a trade-off among these parameters. Generally, to obtain a good filtering effect, the number of levels and the width of cells should not be too high such that the number of cells stabbed by a value is limited. Given L and c_i in each level, we build an optimization model to compute w_i under the constraint that the total number of cells stabbed by a value is not larger than a threshold η. This optimization model is presented in Section 5.

4.4 Stacking Multiple Indexing Attributes

An MO-Tree can be constructed by stacking multiple indexing attributes whose value domains are divided by the hybrid partition scheme. In an MO-Tree, each layer corresponds to an indexing attribute.

Based on the partition scheme shown in Fig. 2, an MO-Tree is exemplified in Fig. 3 by stacking two indexing attributes, A_1 and A_2. For each indexing attribute, 9 disjoint cells are divided for indexing values. Please note C_e^9 is specialized for the case where events do not contain the attribute (null). Meanwhile, 11 overlapping cells are partitioned to index intervals in 3 levels. Thus, there are a total of 81 and 121 buckets to store events and subscriptions, respectively. Please note subscriptions and events are not mixed in a bucket.

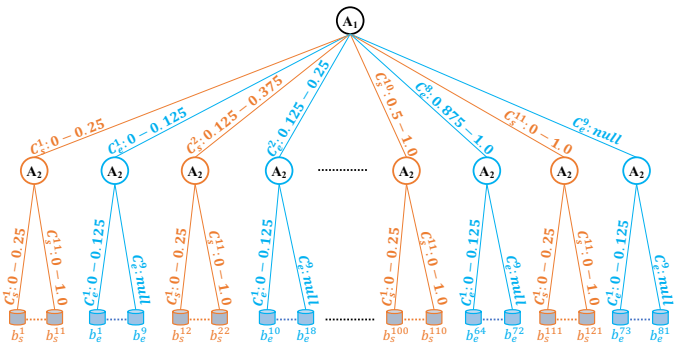

Fig. 3. An example of a two-layer MO-Tree.

TABLE 2
Illustration of indexing subscriptions and events in MO-Tree

ID	Definition	A_1 Cell	A_2 Cell	Bucket
S_1	$A_1[0.2, 0.3] \wedge A_2[0.5, 0.8]$	C_s^2	C_s^{10}	B_s^{20}
S_2	$A_1[0.3, 0.7]$	C_s^9	C_s^{11}	B_s^{99}
S_3	$A_3[0.5, 0.8]$	C_s^{11}	C_s^{11}	B_s^{121}
E_1	$A_1 = 0.3 \wedge A_2 = 0.8$	C_e^3	C_e^7	B_e^{21}
E_2	$A_1 = 0.4$	C_e^4	C_e^9	B_e^{36}
E_3	$A_3 = 0.6$	C_e^9	C_e^9	B_e^{81}

Suppose there are three attributes $\{A_1, A_2, A_3\}$ in the content space, Table 2 gives an illustration to show the indexing of subscriptions and events in the MO-Tree. As the two intervals in S_1 are mapped to cell C_s^2 and C_s^10 on A_1 and A_2 respectively, S_1 is stored in bucket B_s^{20}. S_2 does not contain a predicate defined on A_2, which maps to cell C_s^{11} on A_2. S_3 contains none of the two indexing attributes, which maps to cell C_s^{11} on both A_1 and A_2. The mapping of event values to cells is similar.

For any value, since there are at most 5 cells stabbed by the value, the filtering efficiency of event matching is 1-25/121=79.3% for the MO-Tree shown in Fig. 3. As for the filtering efficiency of subscription matching, it depends on the width of the interval predicates.

4.5 Matching Algorithms

Based on MO-Trees, we can efficiently perform event matching and subscription matching in a two-stage way: filtering and verification.

4.5.1 *Subscription Matching*

Algorithm 1 shows the procedure of subscription matching, where $\mathbb{A}_I$ represents the set of indexing attributes. For each interval predicate of subscription S defined on an indexing attribute A, we use $IntersectCell_A$ to record the cells which intersect with the interval (line 2-6). If S does not contain a predicate defined on A, we search all the cells of the attribute (line 7-9). After obtaining the intersecting cells for all indexing attributes, the target buckets

Algorithm 1: Subscription Matching

Input: S, a new subscription
Output: $\mathbb{R}_s$, the set of matching events of S

1 $\mathbb{R}_s = \emptyset$;
2 **foreach** *interval predicate* $(A, V_l, V_h) \in S$ **do**
3 **if** $A \in \mathbb{A}_I$ **then**
4 $lowCell = \lfloor V_l * totalCell \rfloor$;
5 $highCell = \lceil V_h * totalCell \rceil$;
6 add $cell \in [lowCell, highCeil]$ to $IntersectCell_A$;
7 **foreach** $A \in \mathbb{A}_I$ **do**
8 **if** $IntersectCell_A == \emptyset$ **then**
9 add all cells to $IntersectCell_A$;
10 DFS($rootnode$) and add target buckets to $BucketList$ according to $IntersectCell$;
11 **foreach** $bucket \in BucketList$ **do**
12 **foreach** *event* $E \in bucket$ **do**
13 **if** E *is expired* **then**
14 remove E from $bucket$;
15 **else**
16 **if** S *matches* E **then**
17 add E to $\mathbb{R}_s$;

Algorithm 2: Event Matching

Input: E, a new event
Output: $\mathbb{R}_e$, the set of matching subscriptions of E

1 $\mathbb{R}_e = \emptyset$;
2 **foreach** *attribute-value pair* $(A, V) \in E$ **do**
3 **if** $A \in \mathbb{A}_I$ **then**
4 determine cells stabbed by V and add them to $StabCell_A$;
5 **foreach** $A \in \mathbb{A}_I$ **do**
6 **if** $StabCell_A == \emptyset$ **then**
7 add the last cell to $StabCell_A$;
8 DFS($rootnode$) and add target buckets to $BucketList$ according to $StabCell$;
9 **foreach** $bucket \in BucketList$ **do**
10 **foreach** $S \in bucket$ **do**
11 **if** S *matches* E **then**
12 add S to $\mathbb{R}_e$;

that need to be checked can be determined by the depth-first search (DFS) method (line 10), which ends the filtering stage. The verification step is simple. Events in the target buckets are checked to determine the real matches of S. Meanwhile, expired events are removed from the target buckets incidentally in the verification process (line 11-17). In this way, we do not need to perform specialized deletions for expired events, amortizing the deletion cost in a series of subscription matching operations. By orchestrating subscription matching and event deletion, the processing efficiency of MO-Trees can be improved.

4.5.2 Event Matching

Algorithm 2 shows the event matching procedure which is similar to subscription matching, as they are based on a common data structure and adopt the same matching procedure. There are two differences from subscription matching. Firstly, given the value of an indexing attribute in an event, all cells stabbed by the value are determined (line 4). Secondly, if an event does not contain an indexing attribute, the last cell covering the whole value domain is added (line 7).

4.6 Maintenance Algorithms

According to the indexing rules, each subscription or event is stored in only one bucket. As the bucket size is limited to a small value, it is very efficient to maintain events or subscriptions in an MO-Tree.

4.6.1 Event Insertion and Deletion

When inserting a new event, the first step is to determine the target bucket. For each indexing attribute with a value in the event, the value is mapped to the cell that contains it. If the event does not contain an indexing attribute, it is mapped to the specialized null cell. Based on the mapped cell of each indexing attribute, it is easy to identify the target bucket into which the event is inserted.

There is no need to undertake specialized deletion for expired events in MO-Trees, which is done during the subscription matching. This kind of orchestration improves performance by amortizing event deletion cost in a series of subscription matching operations and mitigating the impact of deletion on matching throughput.

4.6.2 Subscription Insertion, Deletion and Update

The procedure of inserting a subscription is similar to inserting an event with two differences. Firstly, mapping an interval predicate to the cell is more costly than mapping an attribute value, since we need to identify the smallest cell that completely contains the interval from the bottom to the top level. Secondly, if a subscription does not contain an interval predicate defined on an indexing attribute, the subscription is mapped to the cell covering the whole value domain.

Since subscriptions stored in each bucket are not sorted, deletion will take a little more time than insertion, but following a similar processing step. After determining the target bucket, all subscriptions stored in the bucket are traversed to find the one to be deleted.

When updating a subscription, we need to differentiate two cases. If the update does not change the storing bucket, we can directly update the subscription, taking a little less time than deleting a subscription. Otherwise, the procedure of an update is equivalent to the combination of a deletion and an insertion.

4.7 Theoretical Analysis

4.7.1 Subscription Matching

The time cost of subscription matching is mainly determined by the number and width of interval predicates contained in subscriptions. The filtering stage calculates

TABLE 3
Description of symbols

Symbol	Meaning
x	Random variable of the widths of intervals
$P_w(x)$	Probability of the widths of intervals
y	Random variable of the low values of intervals
$P_l(y\|x)$	Probability of the low values of intervals of width x
z	Random variable of the attribute values of events
$P_e(z)$	Probability of the attribute values of events
H	Height of MO-Tree (number of layers)
L	Number of levels in one layer
η	Threshold on the number of cells stabbed by a value
c_i	Number of cells in the i^{th} level
w_i	Width of cells in the i^{th} level
o_i	Overlapping width of neighboring cells in the i^{th} level
ρ_i	Number of cells stabbed by a value in the i^{th} level
γ_i	Interception amount in the i^{th} level
r_i	Success rate of interception in the i^{th} level x
N_e	Number of events
N_s	Number of subscriptions
M	Dimension of content space
K_e	Number of attributes in events
K_s	Number of predicates in subscriptions
r	Notification range of subscriptions

the cells at each layer, which takes little time, whereas the verification stage checks all candidates in the target buckets, consuming the major part of the matching time. On average, the time complexity of subscription matching is $O(N_e \prod_{i\in\mathbb{A}_I}(V_{hi}-V_{li}))$, where N_e is the number of events stored in an MO-Tree and $\prod_{i\in\mathbb{A}_I}(V_{hi}-V_{li})$ is the product of the width (normalized to [0,1]) of all interval predicates defined on the indexing attributes, which represents the matching probability of a subscription.

4.7.2 Event Matching

The time complexity of event matching is $O(N_s(\frac{\eta}{\sum_{i=1}^{L} c_i})^H)$, where N_s is the number of subscriptions stored in an MO-Tree, $\sum_{i=1}^{L} c_i$ is the total number of cells divided on each indexing attribute, η is the threshold limiting the number of cells stabbed by an attribute value, and H is the height (i.e., the number of indexing attributes) of an MO-Tree.

4.7.3 Maintenance

The time cost of event and subscription insertion is quite low, since simple calculations are done to determine one cell at each layer and finally only one target bucket is determined. So, the time complexity of insertion is $O(H)$. The time complexity of subscription deletion is $O(H + \frac{N_s}{(\sum_{i=1}^{L} c_i)^H})$, where $\frac{N_s}{(\sum_{i=1}^{L} c_i)^H}$ is the average size of buckets for indexing subscriptions. The time complexity of subscription update and event deletion is similar to that of subscription insertion.

5 Partition Optimization

In this section, we describe the model for optimizing multi-level overlapping partitions to index interval predicates. The symbols used in this paper are described in Table 3.

5.1 Problem Statement

For simplicity, we normalize the value domain of attributes to [0,1], and let $P_w(x)$, $P_l(y|x)$ and $P_e(z)$ represent the probability density function of the widths of intervals, the low values of intervals of width x, and the attribute values of events, respectively. Let L be the number of total levels divided on each indexing attribute, and c_i and w_i be the number and the width of cells respectively in the i^{th} level for $1 \le i \le L$.

In order to handle intervals of various widths, each level i partitioned on an indexing attribute is devoted to index some predicates whose widths are within the range specified, denoted by $[t_{i-1}, t_i]$ for $1 \le i \le L$, as the example shown in Fig. 2. For convenience, we define $t_0 = 0$ and $t_L = 1$. The core of dividing multi-level overlapping cells is to decide the widths of cells w_i in each level i with two objectives: i) to maximize the amount of target intervals intercepted by each level; and ii) to satisfy a threshold η on the total number of cells stabbed by an attribute value in all levels.

5.2 Optimization Model

In the optimization model, we set $c_i = L-i+1$ for $1 \le i \le L$ in the partition scheme.

Lemma 1. *For two neighboring cells in the i^{th} level, their overlapping width is $o_i = \frac{c_i * w_i - 1}{c_i - 1}$.*

Proof. The total sum of the cell widths in the i^{th} level is $c_i * w_i$. So $c_i * w_i - 1$ is the total sum of the overlapping widths. Since there are in total $c_i - 1$ overlaps between neighboring cells in the i^{th} level, we get the overlapping width between neighboring cells as $o_i = \frac{c_i * w_i - 1}{c_i - 1}$. □

Lemma 2. *Suppose the widths of predicates follow the Zipf distribution, to evenly store predicates in the cells of the L levels, the ideal range t_i in the i^{th} level should satisfy $t_i = \frac{i}{L}$.*

Proof. To evenly store predicates in all the cells of the L levels, the following constraint should be satisfied:

$$\frac{1}{c_1}\int_0^{t_1} P_w(x)dx = \frac{1}{c_2}\int_{t_1}^{t_2} P_w(x)dx = \cdots = \frac{1}{c_L}\int_{t_{L-1}}^{1} P_w(x)dx \quad (1)$$

where $P_w(x)$ is the probability density function of the widths of intervals. Suppose the widths of predicates follow the Zipf distribution and $c_i = L - i - 1$ for $1 \le i \le L$, we can roughly compute $t_i = \frac{i}{L}$ according to Eq. (1). □

Lemma 3. *In the i^{th} level, the maximum value of w_i is $W_i^{max} = \frac{t_i(c_i-1)+1}{c_i}$, and the minimum value of w_i is $W_i^{min} = t_i$.*

Proof. Since the i^{th} level targets predicates with a width not larger than t_i, the width of cells should not be smaller than t_i, i.e., $W_i^{min} = t_i$. On the other hand, because predicates with a width not larger than o_i are guaranteed to be stored in the i^{th} level, W_i^{max} is the solution of $o_i = t_i$, i.e., $W_i^{max} = \frac{t_i(c_i-1)+1}{c_i}$. □

Let ρ_i represent the number of cells stabbed by an attribute value on average in the i^{th} level. It is the expected

number of cells to be visited for an attribute value of events, and is calculated by:

$$\rho_i = \int_0^1 P_e(z) k_i(z)\, dz \tag{2}$$

where $P_e(z)$ is the probability density function of the attribute values of events, and $k_i(z)$ is a function to count the cells containing the value z by checking whether z is in the j^{th} cell for $j = 0, 1, ..., c_i - 1$, which is computed by

$$k_i(z) = count\left(z \in [\frac{1-w_i}{c_i-1} * j,\ \frac{1-w_i}{c_i-1} * j + w_i]\right). \tag{3}$$

If $P_e(z)$ fits the uniform distribution, Eq. (2) is the sum of the cell widths, namely $\rho_i = c_i * w_i$.

Let γ_i be the interception rate in the i^{th} level, which shows the percentage of predicates with a width in the target range that are successfully intercepted and stored in each level. If a level fails to intercept a target predicate, the predicate goes to the next level which has wider cells. To calculate γ_i, we introduce the success rate of interception $r_i(x)$ for a given width x, which is computed by

$$r_i(x) = \int_0^{1-x} P_l(y|x) b_i(y, x)\, dy \tag{4}$$

where $P_l(y|x)$ is the probability density function of the low value y of intervals with width x, and $b_i(y, x)$ is an indicator function to decide whether predicates with a low value y and width x can be stored in the i^{th} level

$$b_i(y,x) = \begin{cases} 1 & y \in [\frac{1-w_i}{c_i-1} * j, \frac{1-w_i}{c_i-1} * j + w_i - x] \\ 0 & others \end{cases}$$

where $j = 0, 1, ..., c_i - 1$. During subscription insertion and matching, $b_i(y, x)$ is used to determine the cells to be visited.

Lemma 4. *When $P_l(y|x)$ follows the uniform distribution, the success rate of interception for a given width x is*

$$r_i(x) = \min\left(\frac{c_i(w_i - x)}{1-x}, 1\right) \tag{5}$$

Proof. $\frac{1-w_i}{c_i-1}$ is the distance between the left endpoints of two neighboring cells, and $[\frac{1-w_i}{c_i-1} * j, \frac{1-w_i}{c_i-1} * j + w_i - x]$ is the range in which the low value of a predicate with width x could lie if the predicate can be stored in the j^{th} cell. If $P_l(y|x)$ fits the uniform distribution, $c_i(w_i - x)$ is the sum of the widths of these ranges. The denominator $1 - x$ in Eq. (5) is the width of range $[0, 1 - x]$ in which all possible low values y of predicates with width x lie. □

According to Eq. (5), γ_i can be calculated by:

$$\gamma_i = \int_{t_{i-1}}^{t_i} r_i(x) P_w(x)\, dx \tag{6}$$

Since the aiming range of t_i for each level has been decided by Lemma 2, the optimization problem is to intercept as many predicates as possible in each level, while the total number of cells visited during event matching is no more than a threshold η. i.e.,

$$\arg\max_{w_i} \sum_{i=1}^{L} \gamma_i \tag{7}$$

$$s.t. \quad \begin{cases} \sum_{i=1}^{L} \rho_i \le \eta \\ w_i \in [W_i^{min}, W_i^{max}],\ 1 \le i \le L \end{cases}$$

The threshold η is a super parameter and can be adjusted according to real applications.

For the uniform distribution, the problem in one level can be approximated as $\arg\max \frac{c_i(w_i-x)}{1-x} \quad s.t. \quad c_i * w_i = \rho_i \le \eta_i$, where η_i is the threshold on number of cells stabbed by a value in the i^{th} level. So c_i should be a small number. This is the reason why c_i is set to grow linearly (but not exponentially) with the rising of the levels.

Theorem 1. *Given γ_i for $1 \le i \le L$, the solution to the optimization problem in Eq. (7) is*

$$\begin{cases} \frac{\partial F}{\partial w_i} = 0, \quad w_i \in [W_i^{min}, W_i^{max}],\ 1 \le i \le L \\ \frac{\partial F}{\partial \lambda} = \left(\sum_{i=1}^{L} \rho_i - \eta\right) = 0 \end{cases} \tag{8}$$

Proof. The constraint $\sum_{i=1}^{L} \rho_i \le \eta$ can be transformed to $\sum_{i=1}^{L} \rho_i = \eta$, because increasing ρ_i means increasing w_i, and thus increasing γ_i. So, γ_i has a positive correlation with ρ_i, and we set the sum of ρ_i to reach the maximum limit η. Then, we introduce Lagrange multiplier λ and generate the following equation:

$$\lambda\left(\sum_{i=1}^{L} \rho_i - \eta\right) = 0$$

So, we get the new target function to maximize,

$$F(w_1, ..., w_L, \lambda) = \sum_{i=1}^{L} \gamma_i + \lambda\left(\sum_{i=1}^{L} \rho_i - \eta\right) \tag{9}$$

We derive F with respect to w_i and λ, and get $L + 1$ one-degree equations in Eq. (8). □

5.3 An Example

Suppose we set $L = 4$, $t_i = \frac{i}{L}$ and $\eta = 6$. The low values and high values of the predicates and attribute values of events are generated in a uniform distribution, i.e.,

$$\begin{aligned} P_w(x) &= 2(1-x) && x \in [0,1] \\ P_l(y|x) &= \frac{1}{1-x} && y \in [0, 1-x] \\ P_e(z) &= 1 && z \in [0,1] \end{aligned}$$

According to these distributions, γ_i can be calculated by

$$\gamma_i = -a(w_i - W_i^{max})^2 + \int_{t_{i-1}}^{t_i} P_w(x)\, dx,\ \ w_i \in [W_i^{min}, W_i^{max}]$$

where a is a positive coefficient. γ_i is a parabola which has a vertex of $\left(W_i^{max}, \int_{t_{i-1}}^{t_i} P_w(x)\, dx\right)$ and passes through the point $\left(W_i^{min}, \int_{t_{i-1}}^{t_i} r_i(x|w_i = W_i^{min}) P_w(x)\, dx\right)$. Here $r_i(x|w_i = W_i^{min})$ is the success rate $r_i(x)$ in the condition of $w_i = W_i^{min} = t_i$, so that $r_i(t_i|w_i = W_i^{min}) = 0$, and γ_i reaches its minimum value.

In this example, the solution is $w_1 = \frac{3}{8}$, $w_2 = \frac{11}{18}$, and $w_3 = \frac{5}{6}$. Note that the width of the last level w_4 is always 1 so it can store all the remaining predicates. In addition, if no limit is set on η (i.e., $\eta = \infty$), the solution would be $w_i = W_i^{max}$.

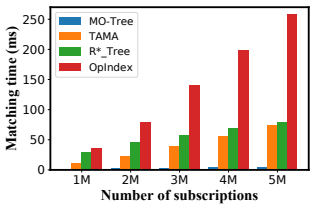

Fig. 4. Event matching time with N_s

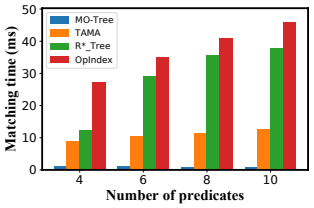

Fig. 5. Event matching time with K_s

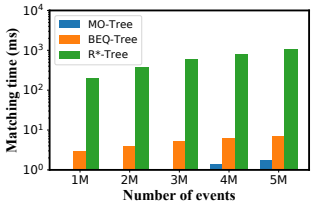

Fig. 6. Sub. matching time with N_e

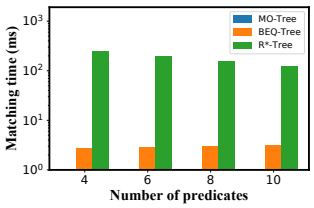

Fig. 7. Sub. matching time with K_s

TABLE 4
Parameters set in the experiments

Parameters	Values	Parameters	Values
N_e	**1M** - 5M	r	0.5-2km
N_s	**1M** - 5M	H	6
M	20	L	4
K_e	15-20	η	6.5
K_s	4, **6**, 8, 10,	α	0, 0.25, 0.5, 0.75, **1**

6 Experiments

In this section, we evaluate the performance of MO-Tree and analyze the experiment results. The source code of MO-Tree is publicly available on GitHub[2].

6.1 Experiment Setup

All experiments were conducted on a server running Ubuntu 18.04 with Linux kernel 4.15.0-118, which has 6 3.6GHz Intel cores and 32GB RAM. All code is written in the C++ language and compiled using g++ 7.5.0 with -O3 optimization. In each experiment, 10,000 events or subscriptions are processed and the average of the performance metrics is computed.

6.1.1 Datasets

We employ about 114,152,144 taxi trajectories [26] collected in Shanghai, China on April 1, 2015 to generate subscriptions and events. From the traces, about 933,794 passenger pick-up points are extracted and 10,000 hot trading areas are clustered by k-means. These passenger pick-up points are used to generate subscriptions and the trading areas are used to generate events.

6.1.2 Parameters

The settings of the parameters in the experiments are listed in Table 4. The bold values represent the default settings in the experiments. Parameters specified by ranges without bold texts mean they are randomly generated. The attributes of events and subscriptions are generated in the Zipf distribution with $\alpha = 1$ by default. The low and high values of the interval predicates in the subscriptions and the attribute values in the events are generated in uniform distribution.

6.1.3 Compared Methods

To evaluate the performance of event matching and subscription maintenance, we compare MO-Tree with three state-of-the-art algorithms, namely TAMA [7], R*-Tree [8] and OpIndex [9]. TAMA is an approximate matching and a forwarding engine which uses a two-layer matching table building on attributes and discretization levels. R*-Tree improves query performance on the basis of R-Tree by incorporating a combined optimization of area, margin and overlap of each enclosing rectangle. OpIndex is a counting-based matching algorithm that adopts a two-level index structure, building on pivot attributes and operators.

To evaluate the performance of subscription matching and event maintenance, MO-Tree is compared with BEQ-Tree [4] and R*-Tree. BEQ-Tree adopts a two-layer partitioning scheme, one for the spatial attribute and the other for the predicates in the Boolean expression.

Besides the implementation of R*-Tree by libspatialindex[3], we implemented TAMA, OpIndex and BEQ-Tree. The parameters of these four compared methods are optimally set according to the papers in which they are proposed. In the experiments, the number of disjoint cells divided for event values is 10 and the other parameters of MO-Tree are tuned as $H = 6, L = 4$ and $\eta = 6.5$, so there are 1M buckets for storing events and subscriptions respectively.

6.2 Experiment Results

6.2.1 Event Matching

In this experiment, we vary the number of subscriptions N_s from 1M to 5M and the results are shown in Fig. 4. On average, when $N_s = 5M$, MO-Tree is about 17.2, 18.5 and 60.4 times faster than TAMA, R*-Tree and OpIndex respectively, which contributes to the conciseness of MO-Tree and the selective indexing attributes, especially the longitude and latitude attribute. When searching subscriptions in MO-Tree, only a small part of the buckets needs to be further verified. On the contrary, since TAMA and OpIndex use the counting-based method, they need to process a large number of partially matching subscriptions that have one or more predicates satisfied. R*-Trees perform badly in high-dimensional space.

The number of predicates K_s contained in subscriptions has an effect on the performance of event matching. We vary K_s in the experiment and the results are depicted in Fig. 5. When $K_s = 10$, MO-Tree is respectively about 15.6, 47.2 and 57.3 times faster than TAMA, R*-Tree and OpIndex on average. Given the number of subscriptions, increasing K_s will result in more popular attributes, which is beneficial to MO-Tree. We can see from the figure that the event matching performance of MO-Tree improves with more predicates. On the contrary, the performance of TAMA, R*-Tree and OpIndex degrades with more predicates, as subscriptions

2. https://github.com/xizeroplus/matching-algorithm

3. https://github.com/libspatialindex/libspatialindex

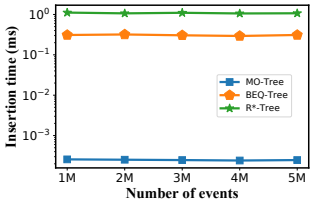

Fig. 8. Event insertion time

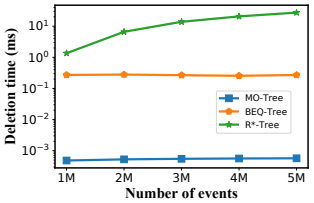

Fig. 9. Event deletion time

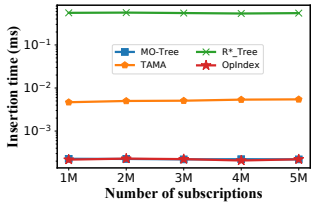

Fig. 10. Subscription insertion time

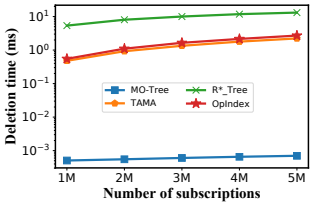

Fig. 11. Subscription deletion time

with more predicates are stored in more buckets and more partially matching subscriptions need to be checked.

6.2.2 Subscription Matching

Similar to the settings in the experiments for event matching, we vary N_e and K_s to evaluate the subscription matching performance of the three tested algorithms and the results are shown in Fig. 6 and Fig. 7 respectively.

Overall, the subscription matching time of the three compared algorithms grows with the number of events. When $N_e = 5M$, MO-Tree is about 3.9 and 628.0 times faster than BEQ-Tree and R*-Tree respectively, which demonstrates the good performance of MO-Tree in subscription matching. BEQ-Tree has a good subscription matching performance, since events stored in buckets are sorted. However, sorting greatly increases the insertion time of BEQ-Tree.

Fig. 7 shows the impact of K_s on subscription matching performance. When a subscription contains more predicates, the matches of the subscription decreases. This is beneficial to MO-Tree by reducing the number of candidate events that need to be verified. The same is for R*-Tree. But the matching time of BEQ-Tree increases slowly with more predicates. When $K_s = 10$, MO-Tree is almost 17.5 and 668.7 times faster than BEQ-Tree and R*-Tree respectively.

6.2.3 Effect of Attribute Distribution

As MO-Tree selects popular attributes as the indexing ones, it may skew with the change of the attributes' popularity. In this experiment, we vary the setting of α in the Zipf distribution to change the popularity of attributes and evaluate its effect on the matching performance. The results are listed in Table 5. When the popularity of attributes decreases, the matching performance of MO-Tree degrades. As for the other four matching algorithms, their performance either improves or remains constant. In the setting $\alpha = 0$ which corresponds to the uniform distribution of attributes, MO-Tree is 2.1 and 96.5 times faster than BEQ-Tree and R*-Tree respectively in terms of subscription matching time. As for the event matching time, MO-Tree is 8.1, 6.5 and 26.3 times faster than TAMA, R*-Tree and OpIndex respectively.

6.2.4 Event Insertion and Deletion

In this experiment, we insert or delete 10,000 events into or from the data structures with different bases from 1M to 5M to measure the time spent on inserting or deleting an event on average. The results are shown in Fig. 8 and 9, where the y-axis represents time in log scale.

The insertion of MO-Tree is quite efficient. After determining the cells of the H indexing attributes, the event is inserted in only one bucket without sorting. When an attribute is not contained in an event, it is treated as being defined in the whole value domain, so the event point may become an area, which makes the insertion time of R*-Tree the highest. In addition, R*-Tree is not suitable in a space of high dimension. As BEQ-Tree sorts events in the buckets, it is also costly to insert an event. As shown in Fig. 8 , MO-Tree is 1,235 and 4,345 times faster than BEQ-Tree and R*-Tree respectively on average in terms of event insertion time.

During deletion, since the size of buckets is limited in MO-Tree, there are few events in each bucket on average. So, MO-Tree takes almost the same time to insert and delete an event. Due to sorting, the cost of deleting an event from BEQ-Tree is similar to that of inserting. For R*-Tree, deleting an event is 12.8 times slower than inserting an event. As shown in Fig. 9, MO-Tree is about 497 and 25,817 times faster than BEQ-Tree and R*-Tree respectively in terms of event deletion time.

TABLE 5
Effect of attribute distribution on matching time (ms)

α	Subscription matching			Event matching			
	MO-T	BEQ	R*-T	MO-T	TAM	R*-T	OpI
1.0	0.99	2.81	192.17	0.96	10.26	29.02	35.07
0.75	1.08	2.79	139.59	1.05	10.07	21.02	34.29
0.5	1.17	2.82	133.74	1.13	10.06	13.19	33.39
0.25	1.26	2.83	133.64	1.21	10.20	10.26	33.38
0	1.32	2.79	127.17	1.27	10.46	8.37	33.71

6.2.5 Subscription Insertion, Deletion and Update

The time spent on inserting and deleting a subscription is shown in Fig. 10 and 11 respectively. For MO-Tree, deleting a subscription takes almost the same time as insertion. As for TAMA and OpIndex, subscription deletion takes more time than insertion with a growth of up to 412 and 5,618 times respectively since they need remove a subscription from multiple buckets, which is very costly. On average, MO-Tree is 22.9, 2,476 and 0.98 times faster than TAMA, R*-Tree and OpIndex respectively in terms of insertion time. As for deletion time, MO-Tree is 2,238, 16,019 and 2,701 times faster than TAMA, R*-Tree and OpIndex respectively.

If updating a subscription does not change its storing bucket, the time cost of updating is similar to the deletion time. Otherwise, updating a subscription is equivalent to the combination of a deletion and an insertion.

6.3 Memory Consumption

As each empty bucket uses 4 bytes by a pointer in the implementations, an MO-Tree that contains 1M buckets occupies about 380MB memory, which is acceptable for most

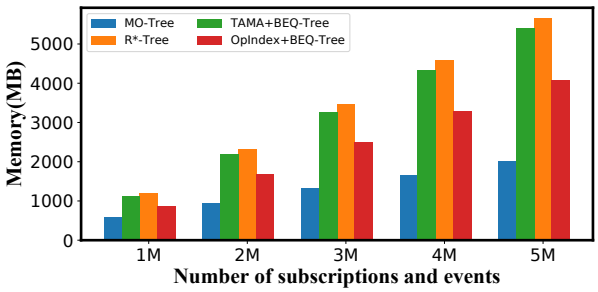

Fig. 12. Memory consumption.

scenarios. In the experiments, MO-Tree and R*-Tree store the same amount of both subscriptions and events together, which is equivalent to a composition of an event matching algorithm and a subscription matching algorithm. There are two compositions for the other three algorithms, namely TAMA+BEQ-Tree and OpIndex+BEQ-Tree. Their memory consumption is shown in Fig. 12. When $N_s = N_e = 1M$, MO-Tree is about 96.1%, 111.6% and 52.4% more efficient than R*-Tree, TAMA+BEQ-Tree and OpIndex+BEQ-Tree respectively in terms of memory consumption. When $N_s = N_e = 5M$, the improvement is about 168.9%, 181.6% and 103.5%. Thus, MO-Tree is efficient from the perspective of joint memory consumption of subscriptions and events.

6.4 Experiment Summary

From the above experiment results and analysis, we verify that MO-Tree has a high overall performance in both the matching and maintenance of events and subscriptions, which contributes to the three design principles embodying the conciseness philosophy, namely keeping the shape of MO-Tree fat, limiting the size of buckets, and avoiding node merging and splitting.

Compared with existing solutions, MO-Tree has three attractive features. i) Solution integrity: both events and subscriptions can be indexed in MO-Tree. ii) High matching performance: MO-Tree is efficient in terms of matching speed. iii) Excellent maintainability: the cost of maintaining events or subscriptions in MO-Tree is very low.

7 Conclusion

To provide an efficient matching and maintenance engine for spatiotemporal pub/sub systems, in this paper, we propose a novel data structure called MO-Tree to index events and subscriptions in a unified way. We design a hybrid partition scheme to divide the value domain of attributes into multi-level overlapping cells to index interval predicates of various widths. We also build a theoretical model to optimize the partition scheme. The experiment results based on real-world trace datasets demonstrate the state-of-the-art overall performance of MO-Tree in the matching and maintenance operations of events and subscriptions.

Acknowledgments

This work was supported by National Key R&D Program of China (2018YFB2101100), the National Science Foundation of China (61772334, 61702151), and the Joint Key Project of the National Natural Science Foundation of China (U1736207).

References

[1] I. Leontiadis and C. Mascolo, "Opportunistic spatio-temporal dissemination system for vehicular networks," in *Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking*. ACM, 2007, pp. 39–46.
[2] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Opportunistic spatio-temporal event processing for mobile situation awareness," in *ACM DEBS*, 2013, pp. 195–206.
[3] B. Jin, W. Zhuo, J. Hu, H. Chen, and Y. Yang, "Specifying and detecting spatio-temporal events in the internet of things," *Decision Support Systems*, vol. 55, no. 1, pp. 256–269, 2013.
[4] L. Guo, D. Zhang, G. Li, K.-L. Tan, and Z. Bao, "Location-aware pub/sub system: When continuous moving queries meet dynamic event streams," in *ACM SIGMOD*, 2015, pp. 843–857.
[5] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *ACM SIGMOD*, 1984, pp. 47–57.
[6] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.
[7] Y. Zhao and J. Wu, "Towards approximate event processing in a large-scale content-based network," in *IEEE ICDCS*, 2011, pp. 790–799.
[8] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *ACM SIGMOD*, vol. 19, no. 2, 1990, pp. 322–331.
[9] D. Zhang, C.-Y. Chan, and K.-L. Tan, "An efficient publish/subscribe index for e-commerce databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 8, pp. 613–624, 2014.
[10] M. Yu, G. Li, and J. Feng, "A cost-based method for location-aware publish/subscribe services," in *ACM CIKM*, 2015, pp. 693–702.
[11] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *IEEE SSDBM*, 2007, pp. 16–16.
[12] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *IEEE ICDE*, 2008, pp. 656–665.
[13] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009.
[14] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *ACM SIGMOD*, 2011, pp. 349–360.
[15] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 373–384, 2010.
[16] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *ACM CIKM*, 2005, pp. 155–162.
[17] L. Chen, G. Cong, and X. Cao, "An efficient query indexing mechanism for filtering geo-textual data," in *ACM SIGMOD*, 2013, pp. 749–760.
[18] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *ACM SIGKDD*, 2013, pp. 802–810.
[19] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang, "Ap-tree: Efficiently support continuous spatial-keyword queries over stream," in *IEEE ICDE*, 2015, pp. 1107–1118.
[20] J. Hu, R. Cheng, D. Wu, and B. Jin, "Efficient top-k subscription matching for location-aware publish/subscribe," in *International Symposium on Spatial and Temporal Databases*. Springer, 2015, pp. 333–351.
[21] P. Zhao, H. Jiang, J. Xu, V. S. Sheng, G. Liu, A. Liu, J. Wu, and Z. Cui, "Location-aware publish/subscribe index with complex boolean expressions," *World Wide Web*, vol. 20, no. 6, pp. 1363–1384, 2017.
[22] T. W. Yan and H. García-Molina, "Index structures for selective dissemination of information under the boolean model," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 2, pp. 332–364, 1994.
[23] M. Sadoghi and H.-A. Jacobsen, "Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space," in *ACM SIGMOD*, 2011, pp. 637–648.
[24] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang, "H-tree: An efficient index structurefor event matching in content-basedpublish/subscribe systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1622–1632, 2014.

[25] S. Qian, J. Cao, Y. Zhu, and M. Li, "Rein: A fast event matching approach for content-based publish/subscribe systems," in *IEEE INFOCOM*, 2014, pp. 2058–2066.
[26] B. Cheng, S. Qian, J. Cao, G. Xue, J. Yu, Y. Zhu, M. Li, and T. Zhang, "Stl: Online detection of taxi trajectory anomaly based on spatial-temporal laws," in *International Conference on Database Systems for Advanced Applications*. Springer, 2019, pp. 764–779.

Tianchen Dian is currently pursuing a Master degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests lie in matching algorithms of publish/subscribe systems.

Shiyou Qian received a PhD in computer science from Shanghai Jiao Tong University in 2015. He is currently an associate researcher with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include event matching for content-based publish/subscribe systems, resource scheduling for Hybrid-Cloud, and driving recommendation with vehicular networks. He is a member of the IEEE.

Jian Cao received a PhD from the Nanjing University of Science and Technology in 2000. He is currently a professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His main research topics include service computing, network computing, and intelligent data analytics. He is a member of the IEEE.

Guangtao Xue received a PhD from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2004. He is a professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include vehicular ad hoc networks, wireless networks, mobile computing, and distributed computing. He is a member of the IEEE, IEEE Computer Society, and the IEEE Communication Society.

Yanmin Zhu is a professor with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. He obtained a PhD in computer science from Hong Kong University of Science and Technology in 2007, and BEng from Xian Jiao Tong University in 2002. His research interests include ad hoc sensor networks, vehicular networks, and mobile computing and systems. He is a member of the IEEE, the IEEE Communication Society and the IEEE Computer Society.

Jiadi Yu received a PhD in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2007. He is currently an associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. His research interests include cyber security and privacy, mobile and pervasive computing, cloud computing, and wireless sensor networks. He is a member of the IEEE and the IEEE Communication Society.

Minglu Li received a PhD from Shanghai Jiao Tong University in 1996. He is currently a distinguished professor with the College of Mathematics and Computer Science at Zhejiang Normal University and a professor with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His main research topics include cloud computing, image processing, and e-commerce.