GEM-Tree: Tree-based Analytic Geometrical Multi-dimensional Content-based Event Matching

Wenhao Fan, *Member, IEEE,* Peng Xiong, Fan Wu, and Yuan'an Liu, *Member, IEEE*

***Abstract*—In large content-based multi-attribute publish /subscribe systems, event matching is a key component, which is in charge of finding all subscriptions that match events. However, the increasing user scale, the enriching message diversity, and the aggravating QoS demands make the performance of event matching severely challenged. Most existing event matching schemes cannot efficiently sustain in these scenarios. The performance rapidly drops as the system load rises. In this paper, we propose GEM-Tree (Geometrical Event Matching Tree), a novel tree-based analytic geometrical index structure for highly efficient event matching in large-scale content-based publish/subscribe systems. To further improve the event matching speed, a local-adjustment mechanism is designed to determine the deployment for each new subscription registering into the GEM-Tree, and a global-adjustment mechanism is designed to optimize the locations of the subscriptions already inserted in GEM-Tree. The experiment results in 8 scenarios demonstrate that GEM-Tree is superior to 3 state-of-the-art reference schemes(BE-Tree, OP-Index, and TAMA). Especially, the leading advantage of GEM-Tree is more significant in matching time for a large number of subscriptions.**

***Index Terms*—event matching, event filter, publish/subscribe, data dissemination**

I. Introduction

Publish/subscribe(pub/sub) is an anonymous, many-to-many asynchronous messaging model with full decoupling of the communication parties in time, space and synchronization [1]. Pub/sub is widely deployed in various scenarios, such as online advertising [2], information filtering [3], mobile message push systems [4]–[6], IOT transferring [7]–[9], content-based routing protocol [10], etc.

In a pub/sub system, subscribers can express their interests in the form of subscriptions. Publishers can send events to the system. When an event arrives at the system, it is then delivered to subscribers whose subscriptions match the event. The process of searching for subscriptions that match the event is called event matching. The event matching is the core component of pub/sub systems, and it is the key factor that determines the system performance.

Pub/sub systems can be divided into three categories based on the message type they aiming at: theme-based, type-based and content-based. The differences between content-based and the others are as follows: Firstly, the subscription is described as a combination of multiple attributes, and each attribute represents a unique aspect of the subscription's characteristics. Secondly, event matching is driven by content rather than other factors (such as IP addresses, information theme) of events and publishers; Thirdly, theme-based and type-based systems are weak in expressing complex subscriptions, in contrast, content-based systems have stronger expression ability, which can describe the content of subscriptions in detail.

Event matching in content-based pub/sub systems is meeting some challenges such as exponential increasing user scale, enriching message diversity, and aggravating QoS demands. Besides, in large-scale systems, the registration of subscriptions and the input of events occur with a very high frequency, which means the load of the systems always keep at a high level. In this case, event matching becomes the bottleneck since the search space expands and the search complexity increases correspondingly. Therefore, how to match events accurately and efficiently is the vital problem that needs to be solved in the design and optimization works of pub/sub systems.

Many event matching schemes have been proposed to address the event matching problem in content-based pub/sub systems in recent years. Proposing efficient index structures is the mainstream approach to accelerate event matching. Some research works are based on tree index structures [11]–[13], which shrinks the search space of event matching by exploiting the similarity between subscriptions, in order to reduce the event matching time. Some works propose novel and efficient index structures based on the characteristics of subscriptions [14]–[19], which improve the matching performance by reducing unnecessary search works. Partial event matching is carried out in each dimension firstly, then the single-dimensional matching results are summarized to obtain the final matching results. Besides, reducing the number of subscriptions is another way to improve the matching speed [20]–[24]. The merging, summarization, covering and subsumption of subscriptions are utilized to reduce the number of subscriptions. However, the performance of most schemes, especially the event matching speed, deteriorates when the system scale increases, since their structures, schemes are still sensitive for search works in a large number of subscriptions, and no targeted optimization techniques are used. Therefore, investigating a new event matching scheme has significant meanings for large-scale pub/sub systems.

In this paper, we propose GEM-Tree (Geometrical Event Matching Tree), a high-speed content-based multi-attribute event matching scheme that supports exact event matching (different with the approximate event matching such as TAMA [15]) and unequal-attribute event matching (the set of attributes of events includes that of subscriptions).

Wenhao Fan, Peng Xiong, Fan Wu, and Yuan'an Liu are with the School of Electronic Engineering, and Beijing Key Laboratory of Work Safety Intelligent Monitoring, Beijing University of Posts and Telecommunications, Beijing, China. e-mail: whfan@bupt.edu.cn.

Manuscript received XXXX; revised XXXX.

GEM-Tree uses a novel tree-based structure consisting of multiple right triangle structure to efficiently organize the subscriptions over multi-dimensional attribute space. A partitioning method is adopted based on analytical geometrical techniques to fast partition matched subscriptions from the structure. When a new subscription comes, its location in GEM-Tree is determined by a local-adjustment mechanism, which uses the ranking function to select the highest-ranking node to insert. And the ranking function is based on the matching time cost. When there are too many subscriptions in the node, a split attribute is selected as the criteria for subscription migration. The selection of the split attribute also takes into account the matching time. When GEM-Tree stores a large number of subscriptions, a global-adjustment mechanism is then used to globally adjust the location of the inserted subscription, based on the analysis of the subscription distribution in the right triangle structure. Combined with the highly efficient filtering mechanisms of the index structure, the partitioning method, and the adjustment mechanism, GEM-Tree can shrink the search space rapidly, further to boost the event matching speed.

Extensive experiments are conducted to compare GEM-Tree with 3 state-of-the-art reference schemes (BE-Tree [11], TAMA [15], OP-Index [16]). We use multiple criteria to measure the performance of these schemes, including the number of subscriptions, the number of attributes, the distribution of subscriptions, the cardinalities of attribute values, the matching rate of subscriptions, the proportion of equivalent predicates, insertion time and memory consumption. Performance of GEM-Tree is superior to other reference schemes under matching time criteria, especially in the case of high workloads.

In this paper, our main contributions are:

1) A novel tree-based analytic geometrical index structure is designed, which can provide high-speed event matching in the scenario that scales to millions of subscriptions, hundreds of dimensions, and dozens of predicates per subscription and event. Through the hierarchical structure of GEM-Tree, non-matched subscriptions can be rapidly excluded layer by layer.
2) A local-adjustment mechanism is proposed to dynamically determine the location of subscriptions in GEM-Tree during the subscription insertion process, which uses the ranking function to select the highest-ranking node to insert. It guarantees the system can support massive workload without deterioration of performance.
3) A global-adjustment mechanism is then used to globally adjust the location of the inserted subscription, based on the analysis of the subscription distribution in the right triangle structure, further improving the event matching speed.
4) We evaluate the performance of GEM-Tree in 8 scenarios with multiple parameters, and it is compared comprehensively with BE-Tree, OP-Index, and TAMA.

Our paper is organized as follows. Section 2 discusses the related works on content-based multi-attribute event matching schemes. Section 3 defines several relevant terms used in GEM-Tree. Section 4 shows the design and analysis of GEM-Tree. Section 5 shows the experiment results and evaluates the performance of event matching by comparing GEM-Tree with the reference schemes. Section 6 concludes the work in our paper.

II. Related Works

Recently, improving event matching performance is the focus of attention and multiple representative schemes have been proposed. They can be generally classified into two categories, (i) Increasing match efficiency by using a novel index structure; (ii) Increasing match efficiency by reducing the number of subscriptions.

A. Increasing match efficiency by using a novel index structure.

Designing a fitted index structure based on the characteristics of the subscription has been a hot issue. In recent years, many excellent index structures have been proposed, and they can be roughly divided into two categories: one is a tree structure and the other is a single-dimensional structure, which means that subscriptions are matched separately according to attribute dimension, and each dimension has same index structure. The final matching result is obtained by summarizing the results of single-dimensional matching.

BE-Tree [11] is a dynamic tree index structure designed to efficiently retrieve subscriptions from a high-dimensional discrete space. It utilizes a novel two-phase space-cutting technique. Space partitioning can make a division of subscriptions by using the attributes of the subscription. For each partition, space clustering determines the best grouping of subscriptions by the range constraints of the attribute. The entire domain of attribute is divided into multiple bisected intervals. Through the filtering of attributes and value constraints, BE-Tree can eliminate a large number of unsatisfied subscriptions, thus reducing the number of subscriptions that will eventually perform event matching. However, since the division of the value domain is a halving method, BE-Tree needs to perform event matching on the subscriptions of multiple nodes from top to bottom. Also, the performance of event matching is related to the insertion order of the subscriptions. Because BE-Tree dynamically adjusts the index structure based on the inserted subscriptions, the extra costs brought by the above interval division will have negative influences on event matching.

H-Tree [12] is a hash table in nature which is a combination of hash lists and hash chaining. After the value domain of each indexed attribute is divided into several partially overlapped cells, the hash lists of all indexed attributes are chained into a hash tree. Hence, similar subscriptions are hashed into the same bucket based on the center of multiple range constraints. When matching events, 2 or 3 cells is figured out for each indexed attribute. Because of this structural feature of H-Tree, when the length of subscription increases, the number of buckets will increase exponentially, which will increase the cost of retrieval.

REIN [14] transforms the event matching problem into the rectangle intersection problem. For each attribute, two bucket

lists are constructed. One bucket list is for the low values of range constraints. Another is for the high values of the range constraints. When matching events, a bit set is initialized. For each attribute, REIN finds out non-matched subscriptions by traversing the bucket lists and marks them in the bit set. The unset bits in the bit set represent the matched subscriptions. The above matching mechanism causes the matching result of the previous attribute to have no acceleration on the matching of the next attribute. Additionally, the matched subscriptions need to obtain by traversing the bit set.

TAMA [15] uses a hierarchical index table to store subscriptions. It bisects the range of each attribute into multiple cells from the top to the bottom of the multilayer index structure. TAMA places a subscription in the corresponding cells according to its constraints. During event matching, for each attribute, TAMA obtains the results of the partial matching and then summarizes them by the counting algorithm to obtain the final matching result. TAMA's hierarchical index structure allows a subscription to be stored in multiple cells, which results in additional memory consumption. TAMA utilizes a counting algorithm to integrate the partial matching results, which means that subscriptions that do not match the previous attribute are not filtered out. Besides, TAMA only supports approximate matching, which means that its matching result is false positive. Although the false positive rate of matching events can be adjusted by tuning the size of the matching table, it requires lots of extra cost.

OP-Index [16] consists of two components. The first component is a two-level partitioning scheme, which consists of a predicate partitioning level and an operator partitioning level. The second component is a collection of counter arrays, corresponding to the collection of subscription lists. The counter arrays are used by a counting-based algorithm to detect matched subscriptions for an event. And it leads to low efficiency of event matching performance since non-matched subscriptions cannot be filtered out in time. Besides, due to the list storage structure of OP-Index, the length of the list has a great influence on the matching speed.

Among the above five schemes, BE-Tree and H-Tree are tree structures. They use the combination of attributes and constraints to find out candidate subscriptions that match the event, which reduces extensive unnecessary event matching. REIN, TAMA, and OP-Index first perform partial matching of each attribute, then summarize the results of partial matching to obtain the final matching result.

GEM-Tree is built on a tree structure. When the subscription is inserted, the right triangle structure is responsible for the division of the attribute values, and the local-adjustment mechanism dominates the location of subscriptions in GEM-Tree. The combination of them gives an efficient index structure. Distinguished with H-Tree and REIN, which strictly require the same length of subscription and event, GEM-Tree has comprehensive and realistic application scenarios. Different from the TAMA and OP-Index that use the counting algorithm to summarize the partial matching results to obtain the final matching result, GEM-Tree can directly obtain the result in the matching process through the recursive matching method. Compared with BE-Tree, GEM-Tree uses a hash-like method to locate attribute values, which can undoubtedly improve the matching speed of the scheme.

B. Increasing match efficiency by reducing the number of subscriptions

The matching time is decreased when the number of subscriptions is reduced. Therefore, reducing the number of subscriptions is another way to improve matching efficiencies. Schemes in [20]–[24] explore similarities between subscriptions and reduce the number of subscriptions by merging, summarizing, covering, and subsuming subscriptions. [20], [21] utilize space filling curves to represent content space, simplifying multidimensional search space into one-dimensional space, for merging subscriptions with inclusion relationships. To make summaries more compact and effective, [22] compresses the routing table size by merging and summarizing subscriptions. [23] relies on a divide and conquer strategy to wipe off subscriptions with subsumption relationships. [24] proposes a novel approximate approach of covering detection at a low cost without losing accuracy. However, subscription subsumption is more efficient than covering. The preprocessing of subscriptions reduces the overall system load and generates traffic since the covered subscriptions are not propagated in pub/sub systems. Hence, if preprocessing is carried out before the subscriptions are inserted, the performance of event matching will be improved. But the preprocessing of subscriptions is not the focus of this paper, so it is not used in our paper.

III. RELEVANT TERMS

An attribute represents a characteristic of an object, which is composed of multiple characteristics. In addition, each attribute corresponds to a value or interval. We call the combination of attribute and value interval a predicate. Attribute value can be integer or decimal. The interval is defined as the left boundary value, the left boundary type, the right boundary value, and the right boundary type. Boundary type can be $CLOSED$ or $OPEN$. Assuming a, b are two values in the attribute value interval, and $a \leq b$, so an interval can be partitioned into 4 types based on its boundaries: left-open and right-open (a, b), left-open and right-closed $(a, b]$, left-closed and right-open $[a, b)$, left-closed and right-closed $[a, b]$. The range of value is expressed as$[0, R_m]$, where R_m is the maximum value. Attribute set is defined as $A = \{a_1, a_2, ..., a_M\}$, where M is the number of dimensions.

An object published by a publisher is called event. An event is expressed as a conjunction of attribute-value pairs. That is, an event can have multiple attributes. And each attribute corresponds to an exact value, which is denoted as v_l. As a convention, each attribute can only occur once in an event. In this paper, an event with L attributes is denoted by $E = \{v_1, v_2, ..., v_L\}$.

A subscription shows subscriber's interest in certain events. In the pub/sub systems, subscribers can submit subscriptions or unsubscribe. A subscription is expressed as a conjunction of attribute-interval pairs. And it's defined as $S_n = \{p_n^1, p_n^2, ..., p_n^K\}$, where K is the number of attributes. The

subscription set with N subscriptions is denoted by $S = \{S_1, S_2, ..., S_N\}$.

If all the attributes in the S_n appear in the E, and the value of the E falls into the interval of the corresponding attribute, we say S_n matches E. In this paper, our scheme supports unequal-attribute event matching. In other words, the set of attributes of events includes that of subscription.

IV. The Design and Analysis of GEM-Tree

The design and analysis of GEM-Tree are in this section. Firstly, an overview of GEM-Tree is presented to give a rough description of our scheme. Secondly, we introduce GEM-Tree in detail, including the overall structure of GEM-Tree, the attribute value interval is mapped to the right triangle structure, which is the essential part of the event matching performance improvement, the ranking function, which plays a key role in the dynamic construction of GEM-Tree. Thirdly, the insertion algorithm and matching algorithm of GEM-Tree are introduced. They are the cornerstone of implementing GEM-Tree. Finally, a global-adjustment mechanism is proposed to obtain a more outstanding performance.

A. Overview of GEM-Tree

GEM-Tree is a tree structure, and it consists of two types of nodes. One is the attribute partition node(aNode), whose role is to store subscriptions and maintain split attributes. The other is a value mapping node(vNode), which contains a right triangle structure and is bound with an attribute. The right triangle structure is divided into multiple cells. And the interval of each attribute is mapped to a cell. With the help of the local-adjustment mechanism, the index structure will be dynamically generated with the insertion of subscriptions.

The local-adjustment mechanism is composed of two parts, one is the selection strategy of the split attribute, and the other is the ranking function. The capacity of a bucket is limited. When it overflows, we need to separate a part of the subscriptions into the new node, and the condition of the move is that the subscription has a split attribute. The local-adjustment mechanism proposes a selection strategy of split attribute based on the matching time cost. The ranking function is critical to the generation of the GEM-Tree. When the subscription is inserted, the choice of inserting the branch will have a great impact on the event matching performance. Over-concentration and decentralization of subscriptions slow down the matching of events. To this end, we propose the ranking function based on the matching time cost. Each attribute has a rank obtained by the ranking function. And the higher the rank, the more beneficial the event matching. Therefore, the attribute with the highest-ranking should be selected.

At the same time as the subscriptions are inserted, we collect the distribution information of subscriptions. And a global-adjustment mechanism is proposed that exploits the information we collect to sort the attributes. We can rely on the priority sequence to optimize the locations of the subscriptions already inserted in GEM-Tree. This optimization method has an improvement in the performance of the pub/sub system.

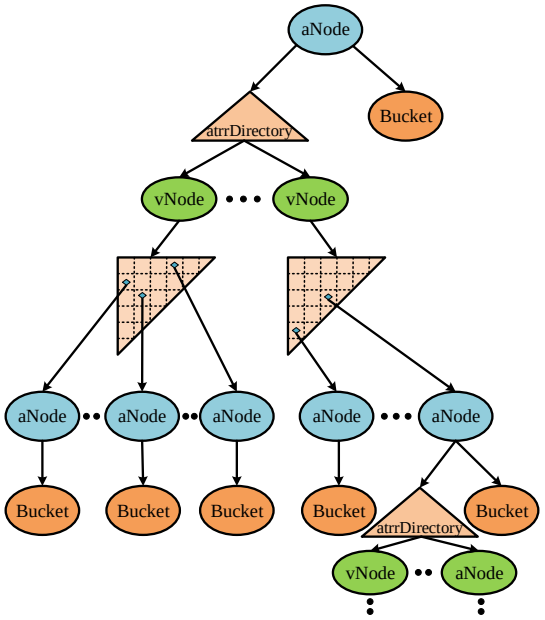

Fig. 1. The GEM-Tree structure

For each incoming event, the entire GEM-Tree is searched from the root node. For each aNode on the search path, all subscriptions in aNode will be matched. Finally, we obtain a subscription set that matches the event.

B. Analysis of GEM-Tree

1) GEM-Tree structure: In general, the root node of GEM-Tree is an aNode, and each aNode has a bucket for storing subscriptions. aNode and vNode alternate on each path. The entire tree is dynamically generated during the subscription insertion process. The index structure of GEM-Tree is shown in Fig. 1.

GEM-Tree is a multi-tree, which stores subscriptions inside nodes. We distinguish between two classes of nodes: one attribute partition node and the other value mapping node. aNode is mainly composed of attribute directory(attrDirectory) and subscription container(Bucket). The former's role is to maintain the already split attributes, the latter is used to store the actual subscriptions. The role of vNode is to divide the range of attribute value. The right triangle structure in vNode is divided into cells, each of which can be represented by a coordinate. Thus the value interval of each attribute can be replaced by a point on the two-dimensional plane. Moreover, vNode is actually bound with an attribute, and there is an aNode in each cell. In fact, each attribute in the aNode's attrDirectory is a vNode, and each cell in the right triangle structure is an aNode. That's to say, aNode and vNode will alternate in each layer of GEM-Tree.

Initially, an empty aNode performs as the root node of the GEM-Tree. When the subscription arrives, the subscription is inserted into the bucket of the root node until the number of subscriptions in the bucket exceeds its maximum capacity. Next, a new attribute is selected from the set of subscription attributes in the overflowing bucket as the new split attribute. That is, a vNode will be generated, which will be placed in the

aNode's atrrDirectory. After selecting the new split attribute, all the subscriptions with the split attribute in the overflowing bucket are moved to the new vNode. If the split strength of the split attribute is not powerful enough, it may result in a small number of subscriptions in the newly generated vNode. It causes the performance of the index structure to deteriorate. So we need to set a threshold, which means that the split attribute can only be selected if the number of subscriptions with split attribute in the subscriptions in the overflowing node is greater than it. If there are no attributes that satisfy the criteria, we should increase the capacity of that node.

2) Right triangle structure(RTS) of GEM-Tree: Mathematical expressions have a close relationship with geometric. For example, in the one-dimensional coordinate system, a directed line represents the range space, the points on the line reflect different values. It is natural to associate the interval with line segments. When it comes to the interval matching in pub/sub systems, the value set can be described by a line segment. In this case, the problem of dividing subscriptions by interval on a single attribute is transformed into the division of the line segment. Based on the above analysis, we introduce a new structure, the right triangle structure as follows.

Every interval has two bounds, lower and upper bounds, such as$[a, b]$. If we take the lower bound as the vertical, and take the upper bounds b as horizontal, the interval $[a, b]$ can be represented by a point (b, a) in a two-dimensional cartesian coordinate system, which is mapped to range space. On this basis, we design RTS to organize the interval points. RTS is a planar triangular structure, which is used to further divide subscriptions in ranges on a single attribute. RTS is constructed with multiple cells. The number of cells is configurable based on different granularities. For example, Fig. 2(a) illustrates an RTS consisting of 36 cells. We define the maximum number of cells in a row or a column as λ. We set $\lambda = 8$ in $Fig.2$(a). We describe the position of a cell in GEM by column c and row r. Therefore, we use (r, c) to denote the index of the cell. In the RTS, c starts from 0, and increases by 1 from the left to the right; r starts from 0, and increases by 1 from the bottom to the top. The index of each cell is shown in Fig. 2(a).

For an attribute, we get the index of cell by its value interval. For the simplicity of the expression, we formulate several notations, the maximum value of the value range(R_m), the left boundary value of the interval ($left$), and the right boundary value of the interval ($right$). The location algorithm computes the c and r by

$$c = \begin{cases} \frac{left}{R_m/\lambda} & \text{left} \neq R_m \\ \lambda - 1 & \text{left} = R_m \end{cases} \quad r = \begin{cases} \frac{right}{R_m/\lambda} & \text{right} \neq R_m \\ \lambda - 1 & \text{right} = R_m \end{cases}$$

$Fig.2$(b) illustrates the event matching implemented on RTS of a single attribute. From the definition of matched subscriptions on value domain, we know that $p_n^m.left \leq v_m$ and $p_n^m.right \geq v_m$. Lines l_1, l_2 , generated from v_m, divide the triangle area into three subareas, A, B, and C. Cells in the RTS are divided into two types, some cells are contained in the area A or B or C, the other is the cells penetrated by l_1 and l_2, like area A' and B'. It is obvious that the subscriptions in area C are matched on this single attribute. The subscriptions located in A' and B' need to carry out further matching. Accordingly, the subscriptions in subarea A and B are non-matched areas. In this case, we get the areas consist of C, A' and B' that cover all candidates of matched subscriptions. Subscriptions in cells contained in A' and B' need further matching. RTS can conveniently and efficiently get the candidate matching area at one time, instead of traversing the cells one by one, which will cost much more time.

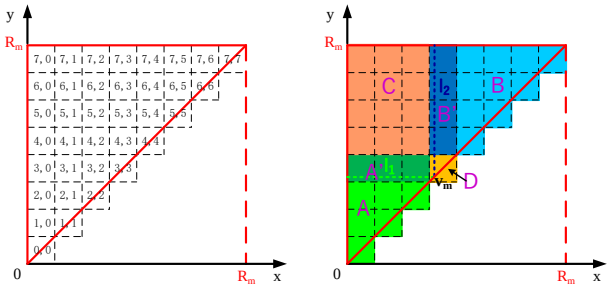

Fig. 2. (a) The right triangle structure with 36 cells,(b) partitioning of cells by graph division method

3) Local-adjustment mechanism of GEM-Tree: In order to maintain a stable performance of GEM-Tree under different workloads, and the performance will not deteriorate rapidly with the increase of subscriptions, we propose a local-adjustment mechanism to cripple the drawback, including ranking function and selection strategy of split attribute. For each RTS, we get the total number of subscriptions(n) it has stored internally and the total number of attributes(h) in the subscriptions that have not occurred in the path from the root node to the current node. And we call the number of attributes in the event the length of the event(len^e), and the number of attributes in the search path from the root node to the current node is called the length of the path(len^p). Each RTS has a matching rate(m) for the attribute corresponding to the event, which can be obtained by low-cost calculation. If the subscriptions follow the Uniform distribution, it is effortless to infer $m = \frac{2v_m(R_m - v_m)}{(R_m)^2}$. The cost of event matching is directly related to the number of predicates that need to find out whether the value of the event falls into the interval of the subscription attribute value. So the cost of matching all subscriptions in an RTS can be approximated as

$$cost = mn\alpha + mh$$

The former is the matching cost of subscriptions in the intersection area, where α is a constant, and the latter is the matching cost of the attributes that cannot be matched by the graph partitioning matching method. And we know that a vNode contains a RTS. For the ith vNode, it can be considered as the root node of the GEM-Tree subtree, then the total cost of matching all subscriptions in the subtree is

$$cost(v_i) = \begin{cases} m_i n_i \alpha + m_i h_i & \text{if } vNode_i \text{ is a leaf} \\ & \text{node or } len^e{=}len^p \\ \sum (m_i cost(v_j)) & \\ +m_i n_i \alpha + m_i h_i & \text{otherwise} \end{cases}$$

Algorithm 1 InsertSub

INPUT:sub, aNode, vNode
01. $foundAttr$ = false
02. **if** aNode.atrrDirectory.size != 0 **then**
03. **for each** $attr$ **in** sub **do**
04. **if** !inPath($attr$) **then**
05. **for each** $node$ **in** attrDirectory **do**
06. **if** $node$.attr == $attr$ **then**
07. $foundAttr$ = true
08. **if** $node$.ranking > $vNode^{optimal}$.ranking **then**
09. $vNode^{optimal}$ = $node$
10. **if** $foundAttr$ **then**
11. c = location(sub[$vNode^{optimal}$.attrID].left)
12. r = location(sub[$vNode^{optimal}$.attrID].right)
13. $aNode^{next}$ = $vNode^{optimal}$.RTS[r][c]
14. InsertSub(sub, $aNode^{next}$, $vNode^{optimal}$)
15. UpdateRanking($aNode^{next}$)
16. **else**
17. aNode.storage(sub)
18. **if** aNode.isOverflowing **then**
19. $attr^{split} \leftarrow$ selectMostPopularAttribute
20. Call $MoveSubs$ ($attr^{split}$,$aNode$,$vNode$,$vNode^{new}$)
21. aNode.attrDirectory.storage($vNode^{new}$)

Algorithm 2 MoveSubs

INPUT:attr^{split}, aNode, vNode, vNode^{new}
1. **for each** sub **in** aNode **do**
2. **for each** $attr$ **in** sub **do**
3. **if** $attr$ == attr^{split} **then**
4. c = location(sub[attrID].left)
5. r = location(sub[attrID].right)
6. vNode^{new}.RTS[r][c].storage(sub)
7. vNode.RTS[r][c].erase(sub)
8. break

where v_j is immediate descendants of v_i. And The total number of subscriptions in the subtree is

$$num(v_i) = \begin{cases} n_i & \text{if } v_i \text{ is leaf node} \\ \sum \big(num(v_j)\big) + n_i & \text{otherwise} \end{cases}$$

Therefore, the average cost of matching a subscription in the branch where v_i is the root node is

$$rank(v_i) = \frac{cost(v_i)}{num(v_i)}$$

We name $rank(v_i)$ the ranking function, which computes a ranking for each vNode over k events. $rank(v_i)$ is essentially the matching cost, that is, the lower the matching cost of the node, the higher it's ranking. Hence, we ought to give preference to the node with high ranking during subscriptions insertion.

Another pivotal factor affecting the performance of GEM-Tree is the choice of split attributes. We can also handle the problem using a cost-based analysis similar to the above. The difference is that here we only consider the change in matching cost caused by the new node because the distribution of subscriptions in other nodes has not changed. Suppose the parameters of the triangle area with an overflowing node are n and h. After selecting a split attribute and generating a new node, the parameters of the original triangle area are $n^{original}$ and $h^{original}$. The corresponding parameters for the new triangle area are n^{new}, h^{new} respectively. There are equations relation between them.

$$n = n^{new} + n^{original} \qquad h = h^{new} + n^{new} + h^{original}$$

The matching cost after generating a new node is

$$cost = m^{original}n^{original}\alpha + m^{original}h^{original} + m^{new}n^{new}\alpha + m^{new}h^{new}$$

What is obtained by combining the above three equations is

$$cost = m^{original}n\alpha + m^{original}h + (m^{new} - m^{original})h^{new} + (m^{new}\alpha - m^{original}\alpha - m^{original})n^{new}$$

m is determined by the value of events, which is not a constant, and we mainly consider the impact of the number of subscriptions moved to the new node on the matching performance. So we weaken the importance of m and consider that m^{new} and $m^{original}$ are approximately equal, that is, $(m^{new} - m^{original}) \approx 0$. Therefore, when selecting split attributes, we prefer the most popular attribute.

C. GEM-Tree implementation

The algorithm for inserting subscriptions dominates the dynamic variability of GEM-Tree, which is the basis for efficient and feasible index structure. In $Algorithm1$, we take sub, $aNode$, and $vNode$, vNode is the parent of aNode, as input. Initially, the aNode is the root node and the vNode is $null$. In general, the insertion is recursively implemented in two steps. Firstly, we search the attribute in atrrDirectory, which did not appear in the path from the root node to the current node(line 4). If there are several candidates, we choose the one with the highest ranking(line 5-9). When the attribute is found, we also get an optimal vNode. And then the location algorithm is used to locate the cell that the interval of the attribute belongs to in the RTS(line 13). The insertion is not accomplished yet. The sub, aNode corresponding to the cell, and the optimal vNode are passed to the $InsertSub$. $Algorithm1$ is executed again from scratch again(line 14).

On the other hand, if the attribute is not found, the current aNode maintains this subscription(line 17). When the number of subscriptions in aNode exceeds its maximum capacity, GEM-Tree will generate a new vNode(line 19). Primarily, the most popular attribute is selected as the split attribute in the overflowing node. The subscription including the split attribute in the overflowing node is then moved to the new vNode(line 20) and the corresponding subscription in the original vNode is deleted. The pseudo code in $Algorithm2$ embodies this process.

The $MatchEvent$ algorithm can be divided into two steps. One is the matching subscription(line 1) and the other is a recursive call $MatchEvent$. We take $event$, $aNode$, $vNode$, $matchedSub$, and $intersectionAttr$ as input. $matchedSub$ is used to store matched subscriptions, and $intersectionAttr$ aims to mark attributes that need to be further matched. During

Algorithm 3 MatchEvent

INPUT: event, aNode, vNode, matchedSub, intersectionAttr
OUTPUT:matchedSub
01. Call $CheckaNode$ with inputs
02. **for each** $attr$ **in** event **do**
03. **if** aNode.attrDirectory.contains($attr$) **then**
04. $index$ = location(event[attrID].value)
{match in the partial matching cell}
05. **for** (r=$index+1$;r<λ;r++) **do**
06. **for** (c=0;c<$index$;c++) **do**
07. $aNode^{new}$=aNode.attrDirectory[attrID].RTS[r][c]
08. $vNode^{new}$=aNode.attrDirectory[attrID]
09. Call $MatchEvent(event, aNode^{new}, vNode^{new}, matchedSub, intersectionAttr)$
{match in the intersecting cell}
10. intersectionAttr.storage(attr)
11. **for** (r=$index$;r<λ;r++) **do**
12. $aNode^{new}$=aNode.attrDirectory[attrID].RTS[r][$index$]
13. Call $MatchEvent(event, aNode^{new}, vNode^{new}, matchedSub, intersectionAttr)$
14. **for** (c=0;c<$index$;c++) **do**
15. $aNode^{new}$=aNode.attrDirectory[attrID].RTS[$index$][c]
16. Call $MatchEvent(event, aNode^{new}, vNode^{new}, matchedSub, intersectionAttr)$

Algorithm 4 CheckaNode

INPUT: event, aNode, vNode, matchedSub, intersectionAttr
OUTPUT: matchedSub
01. **for each** sub **in** aNode **do**
02. $isMatched$ = true
03. **for each** $attr$ **in** sub **do**
04. **if** event.contains($attr$) **then**
05. $value$ = event[attrID].value
06. **if** !inPath($attr$) **then**
07. **if** !sub[attrID].interval.covers($value$) **then**
08. $isMatched$ = false
09. break
10. **else**
11. **if** intersectionAttr.contains($attr$) **then**
12. **if** !sub[attrID].interval.covers(value) **then**
13. $isMatched$ = false
14. break
15. **else**
16. $isMatched$ = false
17. break
18. **if** $isMatched$ **then**
19. matchedSub.storage(sub)

the matching process of $CheckaNode$, each attribute in the subscription is checked. If the event does not contain any of the attributes in the subscription, then it can be directly determined to be non-matched(line 16). Alternatively, if the attribute is found in the subscription(line 4), it will be more complicated. First, if the attribute is not in the path from the root node to the current node(line 6), the result of the partial match is based on whether the value of the event falls into the interval of the attribute value(line 7). If the attribute appears in the path and $intersectionAtrr$ doesn't contain it(line 11), it is a partial match. Otherwise, a further match is performed. Finally, if each attribute in the subscription matches partially, then the subscription matches the event(line 18). The detail of the above process is shown in $Algorithm4$.

The second part of the MatchEvent algorithm is to continue exploring potential matching subscriptions. For each attribute in the event, if the attrDirectory of aNode contains it(line 3), then there may be subscriptions that match the event in the subtree. It's easy to obtain candidate cells by the value of attribute. And they are divided into two categories, one is the partial matching cell, and all subscriptions in it are partial matching with the event without any additional calculations(line 5-9). The other is the intersecting cell(line 10-16), which must be further evaluated at $CheckaNode$(line 10-17).

D. Global-adjustment mechanism of GEM-Tree

In essence, the order of attributes on the search path has a big impact on matching performance. The local-adjustment mechanism simply makes a locally optimal choice on the attribute because of the shortage of subscription distribution information. And if we obtain more information about the distribution of all inserted subscriptions in each dimension, we can get a better order of attribute selection.

There are some notions relevant to our global-adjustment mechanism of GEM-Tree: the average length of the subscription(l^{avg}), the RTS is divided into k cells(k), the number of subscriptions(n_i) and predicates(p_i) in the ith RTS on the insertion path, the matching rate of ith RTS(m_i), and dispersion(d_i), which is the standard deviation of the number of subscriptions in a cell. For RTS, the worst case is that each cell has a child aNode and each vNode layer is a same attribute. Under this assumption, we analyze the matching cost of a path of length l^{avg}. The matching cost of the ith attribute of the path from top to bottom is

$$PartCost_i = (p_i - in_i + \alpha n_i)\frac{1}{k^{i-1}}k^{i-1}\prod_{l=1}^{i} m_l \frac{\beta}{d_{i-1}}$$

$p_i - in_i$ is the matching cost of the attributes that cannot be matched by the graphics division. αn_i is the matching cost of the intersection area. The ith layer has a maximum of k^{i-1} RTS. After each RTS filter on the path, the matching rate on the ith layer is $\prod_{l=1}^{i} m_l$. The greater the d_i, the more concentrated the subscription, the fewer branches are generated and the corresponding matching cost is reduced. It indicates that the matching cost is inversely proportional to the d_i. And the relationship between n_i and p_i can be approximated as $p_i = \rho n_i$. So,

$$PartCost_i = (\rho - i + \alpha)n_i \prod_{l=1}^{i} m_l \frac{\beta}{d_{i-1}}$$

And the cost of matching on a path of length l^{avg} is

$$Cost = \sum_{i=1}^{l^{avg}} PartCost_i$$

This shows that the matching cost is mainly determined by $n_i \prod_{l=1}^{i} m_l$ and d_i. It is proportional to $n_i \prod_{l=1}^{i} m_l$ and inversely proportional to d_i. The larger i of $\prod_{l=1}^{i} m_l$ is, the smaller $\prod_{l=1}^{i} m_l$ will be. The number of preceding RTS is less, which causes it to be more likely to overflow, so the d_i has a greater impact on the attributes in the back. Thus, we conclude that it is more advantageous to reduce the cost of matching by arranging attributes with a larger sum of n_i and d_i in later spot.

When the subscription is inserted, an RTS is built for each attribute, and each attribute of the subscription is traversed for insertion. Through the sum of n_i and d_i, we can get an order of attribute selection. All inserted subscriptions are inserted by a new priority order again.

V. Experiment results and performance evaluation

Our experiments are implemented in 8 scenarios with multiple parameters so that the performance of GEM-Tree can be evaluated extensively. GEM-Tree is compared with 3 state-of-the-art reference schemes, BE-Tree [11], TAMA [15] and OP-Index [16]. We also simulate the scheme that doesn't use global optimization, which is used GEM-Tree (noOpt) to represent. We compare the matching time of the various schemes under different criteria because matching time is a critical performance indicator for the pub/sub systems. The insertion time and memory consumption are tested as well. In the experiments, we implement GEM-Tree in the C++ programming language. The programs are executed on server with Linux 3.8.0-29-generic, Ubuntu 12.04.3 LTS, gcc 4.6.3, and 64GB RAM.

In our experiments, we generate a subscription dataset and event dataset for the performance evaluations for different purposes. The workload is determined by the number of subscriptions, which is given by N. And the number of attribute dimensions is denoted as M. The attribute values of interval in subscriptions and events are generated from the value domains of multiple attributes with varying cardinalities, which are denoted by τ. The number of attributes in subscriptions is defined as ξ. The property of the ratio of event matching subscriptions in the system is called match rate, which is donated as θ. If the left boundary value of the interval equals the right boundary value of the interval, it is called the single value interval. And the ratio of single value interval in subscriptions is given by μ. The settings of the above parameters are summarized in Table. I.

TABLE I
Parameters used in experiments

Name	Meaning	Value
N	the number of subscriptions	[1M, 3M]
M	the number of dimensions	[20, 100]
τ	the cardinality of attribute values	[16, 80]
ξ	the number of attributes in subscriptions	[4, 20]
θ	the match rate	[0.01, 0.1]
μ	the single value interval rate	[0.1, 0.6]
λ	the maximum number of cells in a row or a column	8
d	the discretization level in TAMA	13
w	the number of bits in a segment signature	31

A. Event matching time

The matching time is the most important metric to evaluate the performance of matching schemes. It is influenced by many parameters and comprehensive experiments are conducted to observe the impacts of these parameters. 500 events are sent to measure the average matching time in each experiment. In the experiments, interval value, event value and attributes are generated uniformly unless stated clearly.

1) Different number of subscriptions: As is shown in Fig. 3, N increases from 1 million to 3 million, while the other parameters are $M = 50$, $\tau = 32$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$. In general, matching time increases with the number of subscriptions and GEM-Tree has the best performance. When the number of subscriptions is 1 million, GEM-Tree is almost 1.6, 13.3 and 17.2 times faster than BE-Tree, OP-Index and TAMA, respectively. As the subscriptions increase, the superior performance of GEM-Tree is more obvious. Moreover, GEM-Tree improves the matching time of GEM-Tree(noOpt) by 16%.

2) Different distribution of subscriptions: In reality, the distribution of subscriptions is probably not uniform, but rather close to the Zipf distribution. This brings about an increase in the concentration of the subscription distribution, and a large number of subscriptions are stored in a local area of the index structure, which will undoubtedly increase matching cost. Under Zipf distribution, we measure the event matching time of the 4 schemes with the increase of N from 1 million to 3 million, and the other parameters are $M = 50$, $\tau = 32$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$. As shown in Fig. 4, the matching time of all schemes is increased.

The reason that GEM-Tree receives the least impact on subscription distribution is its adjustment mechanism. Because the local-adjustment mechanism is a local optimal adjustment for the index structure, under the Zipf distribution, the frequency of overflowing node increases. GEM-Tree selects the optimal attribute through the local-adjustment mechanism to generate a new node to minimize the negative impact.

3) Different number of dimensions: From the results in Fig. 5, where M grows from 20 to 100, and other settings are $N = 1M$, $\tau = 32$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$, M has a great impact on all schemes. Because as M increases, subscriptions are distributed more uniformly. On the one hand, for the tree index structure, it reduces the height of the tree. On the other hand, the number of subscriptions in the TAMA bucket is reduced, and the list in OP-Index is shortened, which is beneficial for event matching. When $M = 40$, the performance of OP-Index and TAMA is reversed. Because when M is small, the list in OP-Index is very long, and the search in the list is very time consuming. However, the length of the list rapidly decreases with an increase of dimensions, and the performance of the OP-Index also recovers.

4) Different cardinalities of attribute value: As τ grows, the value of subscriptions and events is more diverse. In other

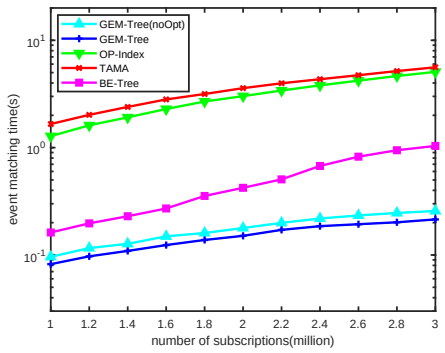

Fig. 3. The event matching time with different number of subscriptions.

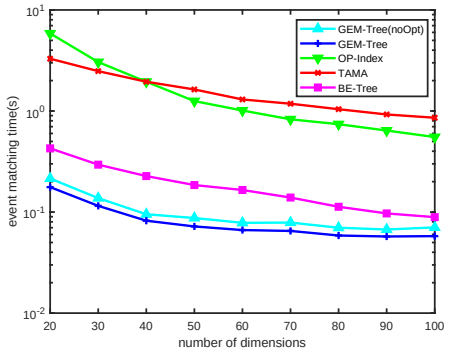

Fig. 5. The event matching time with different number of dimensions.

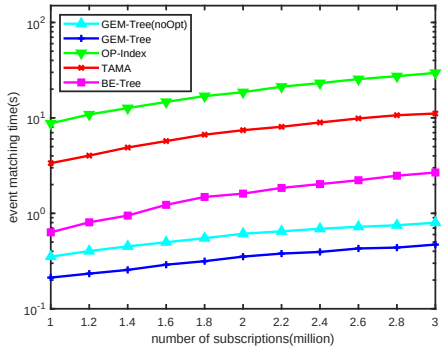

Fig. 4. The event matching time with different number of subscriptions under Zipf distribution.

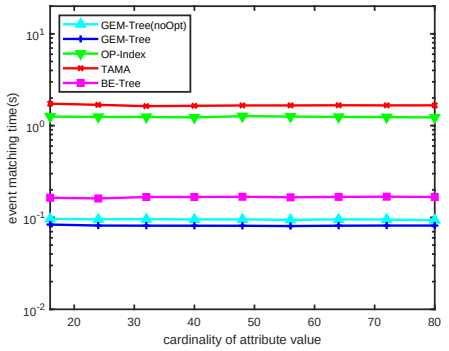

Fig. 6. The event matching time with different number of cardinalities of attribute value.

words, there are more options for the left and right boundary value of the subscription interval and the value of the event. As is shown in Fig. 6, we measure the matching time with different cardinalities of attribute value. In the experiments, τ increases from 16 to 80, while the other settings are $N = 1M$, $M = 50$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$. Generally speaking, the curve has almost no fluctuations and the effect of τ parameter on all schemes can be ignored.

5) Different number of attributes in subscriptions: The number of attributes in subscriptions also affects the performance of the event matching. Increasing the number of attributes increases the workload because, to find all the matched subscriptions, more attributes in the subscriptions need to be matched. In the experiments, the average number of attributes per subscription is set from 4 to 20, and other parameters are $N = 1M$, $M = 50$, $\tau = 32$, $\theta = 0.01$ and $\mu = 0.3$, and the performance of schemes are shown in Fig. 7. TAMA and OP-Index is more sensitive to subscription size than BE-Tree and GEM-Tree. When the number of attributes is low, the performance of OP-Index is just below the BE-Tree and GEM-Tree, as the average number of attributes in subscriptions increases, it is obvious that event matching time grows rapidly. Because unlike BE-Tree and GEM-Tree divide the subscriptions on both attribute and value domain, OP-Index divides subscriptions on their typical attribute and operators. This method is destined to cost more time on excluding non-matched subscriptions under more attributes in subscriptions. It is no surprise that OP-Index has poor performance when the size of attributes in subscriptions extend. Among the schemes in the experiments, GEM-Tree has the best performance with the extension of attributes size.

6) Different match rate: The matching rate is another indispensable factor that affects the event matching time. When the match rate increases, the number of candidate subscriptions also increases. It means more time will be paid on event matching. In order to fully evaluate schemes' performance, the number of subscriptions in the dataset is set to $N = 1M$, dimensionality is $d = 50$, the number of attributes in subscriptions is $\xi = 7$ and the single value interval rate is $\mu = 0.3$. Under a workload with medium expressiveness, while keeping the match rate below 10%, GEM-Tree outperforms other schemes. For the match rate below 4%, OP-Index is more outstanding than TAMA. Larger match rate means more subscriptions that match event. Because the OP-Index counting algorithm is performed after the predicate of the subscription partially matched event, which is different from TAMA's centralized integration of all partial matching results. Therefore, OP-Index is more sensitive to match rate than TAMA. The experiment results are shown in Fig. 8.

7) Different single value interval rate: The lowest cost of checking whether the predicate of subscription partially matched the event is the graph division method, and the worst

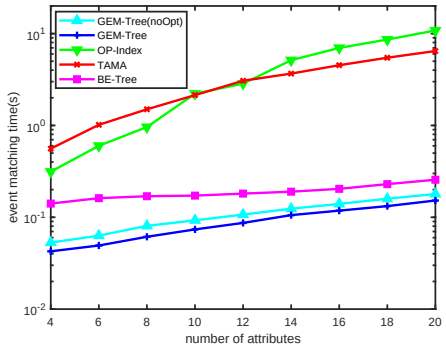

Fig. 7. The event matching time with different number of attributes.

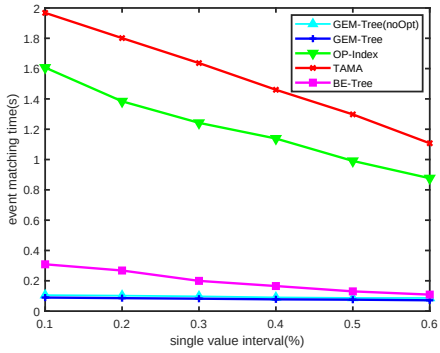

Fig. 9. The event matching time with different single value interval rate of subscriptions.

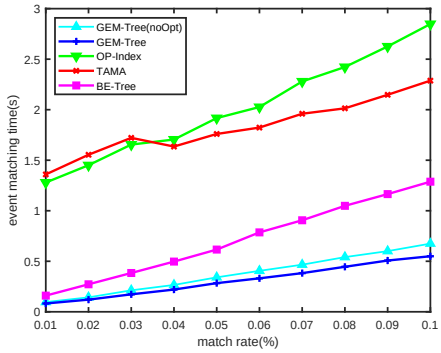

Fig. 8. The event matching time with different match rate of subscriptions.

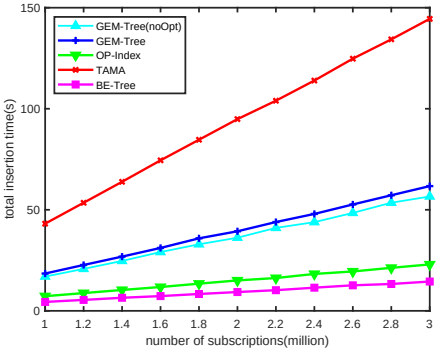

Fig. 10. The total insertion time with different number of subscriptions.

is the left and right boundary verification of the value interval. However, if the value interval is a single value interval, only one check is required. That means that the matching cost is halved, so the single value interval rate helps to reduce the matching time. In the experiments, we set $N = 1M$, $M = 50$, $\xi = 7$, $\theta = 0.01$ and $\tau = 32$, respectively.And μ is equidistantly chosen from [0.1, 0.6]. The experimental result in Fig. 9 also confirms that increasing the single value interval rate can speed up event matching.

B. Maintenance costs

1) Insertion time: Event matching schemes organize the subscriptions by their unique structures. The time of constructing their index structure is also the parameter to evaluate the schemes. When the number of subscriptions increases, construction time increases as well. As is shown in Fig. 10, experiments are carried out to measure the event matching time of the 4 schemes with different N from 1M to 3M, where the other settings are $M = 50$, $\tau = 32$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$. From Fig. 10, GEM-Tree is no longer the best scheme in terms of insertion time, it only surpasses TAMA. The reason for the slow construction of TAMA is its multi-layered index structure. The amount of calculation for each subscription insertion is large, which translates into time cost. Building the index structure of GEM-Tree is more expensive because the structure used to organize subscriptions in GEM-Tree is more complicated than structures in BE-Tree and OP-Index.

2) Memory consumption: In addition to insertion time, memory is another metric for evaluating the maintenance costs. The results are shown in Fig. 11, where N increases from 1M to 3M, and other settings are $M = 50$, $\tau = 32$, $\xi = 7$, $\theta = 0.01$ and $\mu = 0.3$. The average memory consumptions of GEM-Tree, OP-Index, TAMA, and BE-Tree per subscription are 1.53KB, 1.0KB, 2.0KB, 0.80KB, respectively. For TAMA, each subscription is stored in multiple dimensions and stored multiple times in each dimension, which is why it consumes a lot of memory. And the larger d, the more memory consumption. Theoretically, the space complexity of OP-Index for storing N subscriptions with M attributes is O(1.5NM), because, on each attribute, a subscription is stored once or twice. OP-Index storage is relatively memory-saving. However, GEM-Tree memory consumption is very large. The reason is that the cells in the RTS are not all used, which causes a lot of space waste, but it is precisely because of the special structure of the RTS that the GEM-Tree matching time is much faster than other schemes.

3) Performance analysis of the global-adjustment mechanism: The local-adjustment mechanism only makes a local optimal adjustment for GEM-Tree because the available information is deficient. However, if the subscription distribution

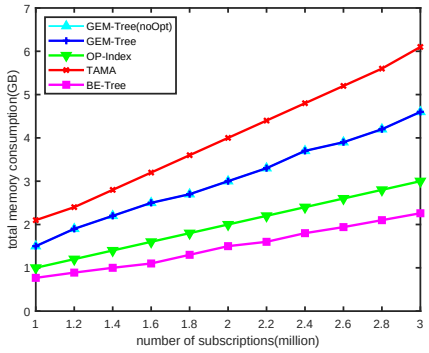

Fig. 11. The memory consumption with different number of subscriptions.

information is recorded when inserting the subscription, we can analyze and extract feature information to further adjust the index structure. Therefore, a global-adjustment mechanism is proposed to achieve a global adjustment to GEM-Tree to improve its performance. In all experiments, we also evaluate the performance of the GEM-Tree optimized by the global-adjustment mechanism, and the parameters are the same as other schemes. It can be seen from the experimental results that the optimization of GEM-Tree is always the best in matching time. Compared to GEM-Tree without optimization, its matching speed can be increased by about 16%. This is due to the fact that the purpose of the global-adjustment mechanism is to find a global optimal index structure without being affected by the subscription insertion order. But this adjustment definitely increases the insertion time of the subscription. Surprisingly, however, the extra memory consumption is light. Because each cell in the RTS only needs to store the number of subscriptions without storing any other information. So each cell consumes 4 bytes, and the space complexity of the extra memory is $O(\frac{k(k+1)}{2}M) = O(k^2M)$. Compared to the memory occupied by GEM-Tree, the extra memory required by the optimization mechanism is negligible. As shown in Figure. 11, the curves of GEM-Tree(noOpt) and GEM-Tree coincide.

VI. Conclusion

This paper introduces GEM-Tree, an efficient index structure to improve the event matching for multi-dimensional content-based pub/sub services. GEM-Tree is implemented by tree structure and right triangle structure. Overall, GEM-Tree is a tree structure. When inserting subscriptions, the local-adjustment mechanism assists in the dynamic generation of the index structure, enabling GEM-Tree to have an excellent performance. And right triangle structure shrinks search space rapidly, which excludes non-matched subscriptions hierarchically, the two of them realize high performance of GEM-Tree among other event matching schemes. Moreover, the global-adjustment mechanism of GEM-Tree makes a global adjustment to GEM-Tree by analyzing all the inserted subscriptions. The event matching speed of GEM-Tree is further improved at an acceptable cost. Extensive experiments are conducted to evaluate the performance of GEM-Tree and experimental results show that GEM-Tree outperforms its counterparts to a large degree, especially in the case where the number of subscriptions is large.

Acknowledgment

This work was supported in part by National Natural Science Foundations of China (61821001, 61502050), Yang-Fan Innovative & Entrepreneurial Research Team Project of Guangdong Province.

References

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

[2] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, "Scalable ranked publish/subscribe," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 451–462, 2008.

[3] 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.

[4] G. Muhl, A. Ulbrich, and K. Herrman, "Disseminating information to mobile clients using publish-subscribe," *IEEE Internet Computing*, vol. 8, no. 3, pp. 46–53, 2004.

[5] M. Jergler, K. Zhang, and H.-A. Jacobsen, "Multi-client transactions in distributed publish/subscribe systems," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 120–131, IEEE, 2018.

[6] E. Mamas, "Wireless messaging services using publish/subscribe systems," Mar. 7 2019. US Patent App. 16/181,332.

[7] C.-Y. Huang and S. Liang, "Ahs model: Efficient topological operators for a sensor web publish/subscribe system," *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, p. 54, 2017.

[8] P. Moraes, R. Reale, and J. Martins, "A publish/subscribe qos-aware framework for massive iot traffic orchestration," *arXiv preprint arXiv:1806.03157*, 2018.

[9] C. Esposito, A. Bruno, G. Cattaneo, and F. Palmieri, "On the optimal tuning and placement of fec codecs within multicasting trees for resilient publish/subscribe services in edge-iot architectures," *Future Generation Computer Systems*, vol. 88, pp. 140–150, 2018.

[10] T. Xue and R. Zhou, "A content-based publish-subscribe routing protocol in mobile ad hoc network," in *International Conference on Internet Technology & Applications*, 2010.

[11] M. Sadoghi and H.-A. Jacobsen, "Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 637–648, ACM, 2011.

[12] 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.

[13] B. Qiao, N. Jiang, Z. Wang, and F. Gao, "A new matching structure and interval division on content based publish/subscribe system," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 641–646, IEEE, 2015.

[14] 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-IEEE Conference on Computer Communications*, pp. 2058–2066, IEEE, 2014.

[15] Y. Zhao and J. Wu, "Towards approximate event processing in a large-scale content-based network," in *2011 31st International Conference on Distributed Computing Systems*, pp. 790–799, IEEE, 2011.

[16] 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.

[17] W. Fan, Y. Liu, and B. Tang, "Gem: An analytic geometrical approach to fast event matching for multi-dimensional content-based publish/subscribe services," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[18] J. Yang, J. Fan, and S. Jiang, "Doco: An efficient event matching algorithm in content-based publish/subscribe systems," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 200–207, IEEE, 2016.

[19] G. Cao, "An event matching algorithm of attribute value domain division for content-based publish/subscribe system," in *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, pp. 177–182, IEEE, 2018.

[20] Z. Shen and S. Tirthapura, "Approximate covering detection among content-based subscriptions using space filling curves.," in *ICDCS*, p. 2, Citeseer, 2007.

[21] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "Mics: an efficient content space representation model for publish/subscribe systems," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, p. 7, ACM, 2009.

[22] Y.-m. Wang, L. Qiu, C. E. Verbowski, D. Achlioptas, G. Das, and P.-A. Larson, "Summary-based routing for content-based event distribution networks," Apr. 3 2007. US Patent 7,200,675.

[23] K. Jayaram and P. Eugster, "Split and subsume: Subscription normalization for effective content-based messaging," in *2011 31st International Conference on Distributed Computing Systems*, pp. 824–835, IEEE, 2011.

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