GEM: An Analytic Geometrical Approach to Fast Event Matching for Multi-dimensional Content-based Publish/Subscribe Services

Wenhao Fan, Yuan'an Liu, Bihua Tang
School of Electronic Engineering, Beijing Key Laboratory of Work Safety Intelligent Monitoring
Beijing University of Posts and Telecommunications, China
whfan@bupt.edu.cn

***Abstract*—Event matching is vital in multi-dimensional content-based publish/subscribe services, which are widely employed for data dissemination in various scenarios. Existing mechanisms suffer from performance degradation in high-dynamic large-scale systems. To this end, we present GEM (Geometrical Event Matching), an analytic geometrical approach to fast event matching. GEM offers a very high event matching speed, and it also has low costs for subscription insertion/deletion operations and memory usage. In GEM, subscriptions are organized efficiently by a triangle-like index structure. A graph partitioning matching method and a selection matching method are jointly used for single-dimensional matching (SDM). Optimized by a decision algorithm for each incoming event, the event matching process is carried out in a pipeline consisting the SDM for each dimension. The search space shrinks continuously as the process goes, so that the event matching performance is promoted adaptively. A cache method is also designed to boost the first SDM in the pipeline. We implement extensive experiments to evaluate the performance of GEM in comparison with 3 state-of-the-art reference algorithms (TAMA, H-TREE and REIN). The results show that, the event matching time, subscription insertion/deletion time and memory consumption of GEM is on average 53.9%, 42.3%/49.5% and 31.8% lower than the best in other 3 algorithms, respectively. The event matching time of GEM is reduced efficiently via the cache method. The superiority of GEM appears more significantly as system scale and dynamic grow, and its performance also maintains in a high stability.**

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

I. Introduction

Event matching is a key issue in publish/subscribe (pub/sub) communication systems, which are widely employed for data dissemination in various scenarios, such as mobile message push systems [1], algorithmic trading systems [2], content-based routing systems [3], deep packet inspection systems [4], and computational advertising systems [5], etc. In a pub/sub system, receivers (called subscribers) register their predicates in the form of subscriptions, and senders (called publisher) submit events. The system forwards an event to the subscribers whose subscriptions matches the event. The process for finding out all subscriptions that match a certain event is called event matching.

Recently, event matching is challenged by a number of new trends. Firstly, both the quantity and quality of contents are growing rapidly. The contents of events are generated by publishers at very high speed. Twitter users submit over 0.3 million tweets and Facebook users share nearly 2.5 million pieces of contents per minute on average. The descriptions of events' contents and subscriptions' contents are also enriched in multiple dimensions. For example, the air quality can be described by time, location, temperature, PM2.5 value, PM10 value, CO value, NO_2 value, SO_2 value, etc., and a subscription from a user contains the predicates for the values on these dimensions. Secondly, the pub/sub environments are becoming time-varying. The sessions of most users in social networks only last for a few minutes. The mobility of users and the instability of wireless networks may change and interrupt the network connections between users and the system. Hence, the pub/sub associations in the system vary in time. Thirdly, the real-time performance is turning into the most prior criterion for pub/sub services. More and more applications that employ pub/sub services require the event matching to provide immediate decisions, and the event matching performance should keep stable for different incoming events. Uber needs to push the mobile clients' requests to their matched taxi drivers as soon as possible, or else the associations between them may be expired only after a short time due to the changes of the drivers' locations and the clients' locations.

The above trends are prompting event matching to have high and stable real-time performance for the pub/sub systems with dense event inputs, massive user scales, multi-dimensional contents, and time-varying environments. The key is to offer a high-efficiency event matching mechanism to search all matched subscriptions for a certain event sent by a publisher, from the whole search space consisting of all subscriptions submitted by subscribers. Otherwise, the event matching has to traverse a large number of irrelevant subscriptions to collect all matched ones, and the event matching cost may fluctuant sharply for different events. Moreover, the index structure of the event matching mechanism should support fast subscription insertion and deletion operations. If not, the frequent changes of subscriptions not only take extra operation costs, but also deteriorate the event matching performance.

Several event matching mechanisms are proposed in recent years. They focus on promoting the event matching performance for multi-dimensional content-based pub/sub systems through reducing unnecessary searching in the event matching process. Some mechanisms [6]–[9] first handle the matching for each dimension (called single-dimensional matching, SDM) assisted by their designed index structures, then integrate the matching result for each dimension (called partial

总结需求
要构造一个什么样的算法（解决方法）
QOA
现有方法第一类
定义
事件匹配
三个需求

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

matching result, $\widetilde{\mathbf{S}}$) through specific technologies to generate the matching result for the event (called final matching result, $\widehat{\mathbf{S}}$). The ideas of other mechanisms [10]–[13] lie in treating all dimensions as a combination. According to the similarities of subscriptions, their index structures partition the search space into multiple subspaces, where the event matching are carried out. However, most of them still can not meet the requirements on event matching performance, and are hard to balance between performance and system scalability. They suffer from performance degradation for the event matching with high dimensionality, due to the limitations of their matching algorithms. Moreover, their index structures are designed to boost the event matching speed, although, the performance of subscription insertion and deletion operations worsens at the same time, and the memory usages increase as well.

Motivated by the advantages of the single-dimensional and the all-dimensional event matching mechanisms, in this paper, we propose an analytic geometrical approach to fast event matching for multi-dimensional content-based pub/sub services, called GEM (Geometrical Event Matching). GEM offers a very high event matching speed, at the same time, it also has very low costs for subscription insertion/deletion operations and memory usage. Our approach supports an expressive multi-dimensional content-based pub/sub model [14]–[16], and can be applied to both the centric and distributed pub/sub networks. The main innovations of GEM can be summarized as follows:

1) An analytic geometry-based index structure is designed, which maps the predicates of subscriptions into a triangle area for each dimension. The area is structured by a 2-dimensional isometric array to store and organize subscriptions. The index structure can provide very fast subscription insertion/deletion operations with a constant time and its memory usage is low, because subscriptions are stored without being duplicated. Therefore, GEM is very suitable for high-dynamic systems.

2) Two candidate matching methods are designed to jointly handle the SDM for each dimension. One is a graph partitioning method, which partitions the triangle area and excludes all non-matched subscriptions in a high-efficiency way; the other is a selection method, which directly filters out all non-matched subscriptions from $\widetilde{\mathbf{S}}$. The event matching for each incoming event is optimized by a decision algorithm, then is carried out by pipelining each SDM. The search space shrinks efficiently as the process goes, thus, the event matching speed of GEM is quite fast even for the large-scale systems with massive subscriptions and high dimensionality.

3) A cache method is designed to buffer multiple subsets from the subscription set of the system. It can efficiently decrease the number of subscriptions in the traversing needed by the first SDM in the event matching process. Through the method, the event matching speed of GEM can be further boosted as the number of caches grows. It is also configurable for balancing the costs among event matching, subscription insertion/delection operations and memory usage.

Extensive experiments are implemented to evaluate the performance of GEM on multiple criteria. GEM is compared with 3 state-of-the-art reference algorithms (TAMA [6], H-Tree [10] and REIN [7]). The experiment results show that GEM outperforms the reference algorithms on event matching time, subscription insertion/deletion time and memory consumption by over 53.9%, 42.3%/49.5% and 31.8% on average, respectively, and GEM's performance is quite stable for different events. The event matching speed of GEM is promoted efficiently via the cache method, and a 78.2% reduction on average is reached with 20 caches. Additionally, the superiority of the decision algorithm is validated through comparing it with the corresponding optimal algorithm.

The rest of this paper is organized as follows: Section 2 presents the design of GEM. Section 3 shows the performance evaluation and experiment results for GEM. Section 4 reviews and discusses the related works. Section 5 concludes our work.

II. Design of GEM

A. Multi-dimensional Content-based Data Model

GEM is based on a multi-dimensional content-based data model. The dimensionality of contents is expressed by the dimension set $\mathbf{A} = \{a_1, a_2, ..., a_M\}$, where M is the number of dimensions. The subscription set of the system with N subscriptions is denoted by $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_N\}$, where subscription $\mathbf{S}_n$ ($\mathbf{S}_n \in \mathbf{S}$) is a conjunction of its predicates on all dimensions in $\mathbf{A}$. $\mathbf{S}_n = \{p_n^{(1)}, p_n^{(2)}, ..., p_n^{(M)}\}$, where $p_n^{(m)}$ is the predicate of $\mathbf{S}_n$ on a_m ($a_m \in \mathbf{A}$). The predicates with different types ($>, \geq, <, \leq, =, \neq$) can be all converted into interval-based expressions. For the interval of $p_n^{(m)}$, its low value, low boundary type, high value and high boundary type are defined as $p_n^{(m)}$.lowValue, $p_n^{(m)}$.lowType, $p_n^{(m)}$.highValue and $p_n^{(m)}$.highType, respectively. The value of a boundary type is chosen from INCLUSIVE and EXCLUSIVE. We denote the value range of a_m as $[0, R_m]$ for simplicity, where R_m is the upper bound. An event $\mathbf{E}$ is a combination of the values from the value range of each dimension. The value of $\mathbf{E}$ on a_m is defined as v_m, thus, $\mathbf{E} = \{v_1, v_2, ..., v_m\}$.

We say $\mathbf{S}_n$ partially matches $\mathbf{E}$ on a_m if v_m falls into the interval of $p_n^{(m)}$. If $\mathbf{S}_n$ partially matches $\mathbf{E}$ on all dimensions, then we say $\mathbf{S}_n$ matches $\mathbf{E}$ and it is a matched subscription, otherwise, $\mathbf{S}_n$ does not match $\mathbf{E}$ and it is a non-matched one.

For instance, in an air quality monitoring system, the sensors act as publishers who publish events containing the values on 3 dimensions: time (a_1), temperature (a_2), PM2.5 (a_3). The users perform as subscribers who submit their subscriptions to the system. We assume a subscription $\mathbf{S}_1$ contains a user's predicates to events, whose time interval is $[\text{July 2 2015}, \text{July 20 2015}]$ ($p_1^{(1)}$), temperature interval is $(25°\text{C}, 30°\text{C}]$ ($p_1^{(2)}$), PM2.5 interval is $[0\mu\text{g/m}^3, 100\mu\text{g/m}^3)$ ($p_1^{(3)}$). For an event published by a sensor, $\mathbf{E} = \{\text{July 3 2015 } (v_1), 28°\text{C } (v_2), 56\mu\text{g/m}^3 \ (v_3)\}$, the user will receive $\mathbf{E}$ since $\mathbf{S}_1$ matches $\mathbf{E}$. For another event $\mathbf{E} = \{\text{July 12 2015 } (v_1), 25°\text{C } (v_2), 153\mu\text{g/m}^3 \ (v_3)\}$, the user will not receive $\mathbf{E}$ because $\mathbf{S}_1$ does not partially match $\mathbf{E}$ on a_2 ($v_2 = 25°\text{C} \notin p_1^{(2)} = (25°\text{C}, 30°\text{C}]$), and a_3 ($v_3 = 153\mu\text{g/m}^3 \notin p_1^{(3)} = [0\mu\text{g/m}^3, 100\mu\text{g/m}^3)$).

B. Overview of GEM

The event matching process can be further improved by hybridizing the advantages of the single-dimensional and the

all-dimensional event matching. For different events, we notice that the cost of SDM varies for different dimensions. The event matching speed can be promoted if multiple candidate methods are designed, and the suitable method with the lowest cost is employed for each SDM. We also notice optimization can be done among SDMs. If the non-matched subscriptions are filtered out in a previous SDMs as early as possible, the number of subscriptions needed to be traversed in the next SDM can decrease efficiently. In this way, the search space shrinks as the event matching process goes, hence, the event matching speed can be boosted. Besides, the design of index structure is another key issue. If a subscription are stored in the index structure without being duplicated, the memory usage can be efficiently controlled since no extra memory is consumed. Thus, the subscription insertion/deletion operations can be done very fast, even in a constant time.

GEM is designed based on the above ideas. The index structure of GEM is inspired from the analytic geometry theories. GEM creates a triangle area in a plane coordinate for each dimension, the predicate of a subscription on a dimension is mapped to a point in the area. A 2-dimensional isometric array is designed to structure the triangle area. According to a hash algorithm, the predicate of a subscription is stored into its corresponding cell in the array.

The event matching of GEM is a pipelined process where each SDM acts as a machining unit in a pipeline. Every SDM is equipped by two candidate methods: a graph partitioning matching method and a selection matching method. The former partitions the triangle area and exclude all non-matched subscriptions efficiently, whereas, the latter directly filter out non-matched subscriptions from $\widetilde{\mathbf{S}}$. The above two methods have different matching costs even for the same dimension. Based on a decision algorithm, GEM chooses the suitable method for each SDM and decides the sequence in the pipeline for each SDM. In the event matching process, the SDM for each dimension is executed sequentially. The $\widetilde{\mathbf{S}}$ output from a previous SDM is used as the input of the next one. Finally, $\widehat{\mathbf{S}}$ is the $\widetilde{\mathbf{S}}$ output by the SDM at the end of the pipeline. In such way, the search space shrinks continuously as the event matching goes from the first SDM to the last one in the pipeline.

We found that the matching cost of the first SDM in the pipeline occupies a large proportion in the total event matching cost. To this end, a cache method is designed to preparatively cache multiple partitions of the system's subscription set. For a certain event, the suitable cache is chosen, then, the $\widetilde{\mathbf{S}}$ of the first SDM is obtained by shallowly cloning the cache and slightly revising the subscriptions in the cloned cache with a small amount of traversing. The number of the caches can be adjusted to balance the performance among event matching time, subscription insertion/deletion time, and memory consumption.

C. Analytic Geometry-based Index Structure

For a certain dimension $a_m \in \mathbf{A}$ and a certain subscription $\mathbf{S}_n \in \mathbf{S}$, GEM maps the predicate $p_n^{(m)}$ of $\mathbf{S}_n$ on a_m into an area in a plane coordinate. The mapped $p_n^{(m)}$ is formulated by

$$p_n^{(m)}.x = p_n^{(m)}.\text{lowValue} \qquad p_n^{(m)}.y = p_n^{(m)}.\text{highValue} \quad (1)$$

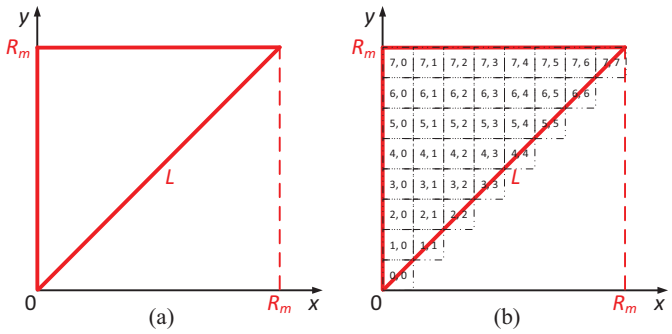

Fig. 1. (a) The isosceles right angled triangle area for a_m, (b) An indexed 2-dimensional array with 36 cells in the area.

where $p_n^{(m)}.x$ and $p_n^{(m)}.y$ are the mapped values of $p_n^{(m)}$ on x-axis and y-axis, respectively.

According to all possible values of predicates from the value range $[0, R_m]$ of a_m, all mapped predicates fall into a right angled triangle area, as is shown in Fig. 1(a), where the formulation of the long side L is $y = x$. L actually ensures $p_n^{(m)}.x \leq p_n^{(m)}.y$ since $p_n^{(m)}.\text{lowValue} \leq p_n^{(m)}.\text{highValue}$ holds.

GEM uses a triangle-like 2-dimensional isometric array, which is very easy to be implemented by multiple programming languages, to structure the right angled triangle area. The array divides the area into multiple square cells. The number of cells is configurable based on different granularities. For example, Fig. 1(b) illustrates an array consisting of 36 cells. We define the maximum number of cells in a row or a column as τ, so $\tau = 8$ for the array in Fig. 1(b). The index of a cell includes its row index $i^{(r)}$ and column index $i^{(c)}$, thus, we name such a cell as $cell(i^{(r)}, i^{(c)})$. In the array, $i^{(r)}$ starts from 0, and increases by 1 from the bottom to the top; $i^{(c)}$ starts from 0, and increases by 1 from the left to the right. Fig. 1(b) also shows the indices of the cells in the array.

Each mapped predicate falls into its corresponding cell in the array, and a cell stores all predicates that correspond to it. The searching for a cell is completed via a hash algorithm only with $O(1)$ time complexity. For $p_n^{(m)}$, the algorithm computes the row and column indexs of its corresponding cell by

$$i^{(r)} = \begin{cases} \dfrac{p_n^{(m)}.y}{R_m/\tau}, & p_n^{(m)}.y \neq R_m \\ \tau - 1, & p_n^{(m)}.y = R_m \end{cases}, \; i^{(c)} = \begin{cases} \dfrac{p_n^{(m)}.x}{R_m/\tau}, & p_n^{(m)}.x \neq R_m \\ \tau - 1, & p_n^{(m)}.x = R_m \end{cases} \quad (2)$$

In GEM, there are m arrays built up for all m dimensions. When inserting a subscription, its predicate on each dimension is inserted into the corresponding cell in the array for the dimension; when deleting a subscription, its predicate on each dimension is deleted from the corresponding cell in the array for the dimension. Therefore, no predicates need to be duplicated in the index structure of GEM, moreover, the subscription insertion/delection operations can be completed very fast in a constant time due to the hash algorithm.

D. Methods for Single-dimensional Matching

1) Graph Partitioning Matching Method: For a_m, the target of the method is to find out all subscriptions whose predicates

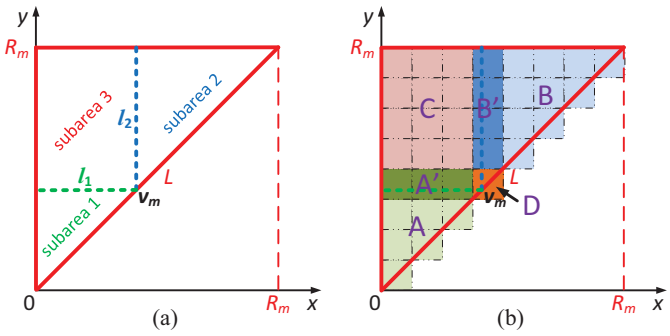

Fig. 2. (a) the subareas partitioned by l_1 and l_2, (b) the related cells that needs to be handled by the graph partitioning method.

on a_m do not satisfy the v_m of an event $\mathbf{E}$ on a_m, then exclude them from the current $\widetilde{\mathbf{S}}$, which is output from the previous SDM. Thus, when the current SDM completes, the subscriptions remaining in $\widetilde{\mathbf{S}}$ all partially match $\mathbf{E}$ on a_m.

We have the following theorems to describe all situations that a subscription does not partially match an event on a dimension:

Theorem 1: Given a_m, v_m of $\mathbf{E}$ and $p_n^{(m)}$ of $\mathbf{S}_n$, we say $p_n^{(m)}$ does not satisfy v_m when 1) $p_n^{(m)}$.lowType = EXCLUSIVE and $p_n^{(m)}$.lowValue $\geq v_m$, or 2) $p_n^{(m)}$.lowType = INCLUSIVE and $p_n^{(m)}$.lowValue $> v_m$.

Theorem 2: Given a_m, v_m of $\mathbf{E}$ and $p_n^{(m)}$ of $\mathbf{S}_n$, we say $p_n^{(m)}$ does not satisfy v_m when 1) $p_n^{(m)}$.highType = EXCLUSIVE and $p_n^{(m)}$.highValue $\leq v_m$, or 2) $p_n^{(m)}$.highType = INCLUSIVE and $p_n^{(m)}$.highValue $< v_m$.

Proof: $p_n^{(m)}$ does not satisfy $\mathbf{E}$ on a_m when v_m is not in the interval of $p_n^{(m)}$. Thus, the 4 situations listed in *Theorem 1* and *Theorem 2* include all possibilities that the interval of $p_n^{(m)}$ does not contain v_m. ■

Then, we have the following theorem to decide whether a subscription does not match an event.

Theorem 3: If at least one of the 4 situations in *Theorem 1* and *Theorem 2* holds on a_m, then $\mathbf{S}_n$ do not match $\mathbf{E}$.

Proof: $\mathbf{S}_n$ matches $\mathbf{E}$ only if every predicate in $\mathbf{S}_n$ satisfies the value of $\mathbf{E}$ on the corresponding dimension. If at least one of the conditions in *Theorem 1* and *Theorem 2* holds, it means $p_n^{(m)}$ does not satisfy v_m on a_m, so $\mathbf{S}_n$ do not match $\mathbf{E}$. ■

The graph partitioning matching method maps the v_m of $\mathbf{E}$ on a_m into the right angled triangle area according to **Formula** (1). The values of the mapped v_m on x-axis and y-axis are equal, namely, $v_m.x = v_m.y = v_m$, so the mapped v_m always locates on L since L's formulation is $y = x$. Consequently, the row index and column index of the cell where the mapped v_m locates are equal. They are both denoted by $i^{(v)}$, which can be obtained by **Formula** (2). As shown in Fig. 2(a), we draw two lines starting from the mapped v_m (point t): a horizontal line $l_1 : y = v_m$ and a vertical line $l_2 : x = v_m$, so that the area is partitioned into 3 subareas: subarea 1, 2 and 3.

It can be observed that the subscriptions whose mapped predicates are in subarea 1 do not partially match $\mathbf{E}$, because *Theorem 2* holds in subarea 1; the subscriptions whose mapped predicates are in subarea 2 do not partially match $\mathbf{E}$ because *Theorem 1* holds in subarea 2. The graph partition matching method excludes the above subscriptions from $\widetilde{\mathbf{S}}$ since they do not match $\mathbf{E}$ according to *Theorem 3*.

The graph partitioning matching method is carried out relying on the analytic geometry-based index structure. As shown in Fig. 2(b), the subscriptions whose predicates are mapped to the cells in the subareas labeled as A and B can be excluded through directly traversing all subscriptions in these cells. For the subscriptions in the cells locating in the subareas labeled as A' and B', which intersect with l_1 or l_2 only, extra examinations via *Theorem 1* or *Theorem 2* are needed in the exclusions. The subscriptions in the cell labeled as D need to be checked by both *Theorem 1* and *Theorem 2*, because both l_1 and l_2 intersect with the cell.

Based on the above explanations, the process of the graph partitioning matching method is shown in **Algorithm 1**.

Algorithm 1 The Graph Partitioning Matching Method

INPUT: v_m of $\mathbf{E}$ on a_m, $\widetilde{\mathbf{S}}$ of the previous SDM.
OUTPUT: $\widetilde{\mathbf{S}}$ for a_m.
01. Compute the row/column index $i^{(v)}$ of the mapped v_m via **Formula** (2);
02. *Handle the cells in the subarea labeled as A:*
03. **for**$(i \leftarrow 0; i < i^{(v)}; i \leftarrow i + 1)$
04. **for**$(j \leftarrow 0; j \leq i; j \leftarrow j + 1)$
05. Traverse all subscriptions in $cell(i, j)$, and remove them from $\widetilde{\mathbf{S}}$;
06. **end for**
07. **end for**
08. *Handle the cells in the subarea labeled as B:*
09. **for**$(i \leftarrow i^{(v)} + 1; i < \tau; i \leftarrow i + 1)$
10. **for**$(j \leftarrow i^{(v)} + 1; j \leq i; j \leftarrow j + 1)$
11. Traverse all subscriptions in $cell(i, j)$, and remove them from $\widetilde{\mathbf{S}}$;
12. **end for**
13. **end for**
14. *Handle the cells in the subarea labeled as A':*
15. **for**$(j \leftarrow 0; j < i^{(v)}; j \leftarrow j + 1)$
16. Examine the subscriptions in $cell(i^{(v)}, j)$ via *Theorem 1*, and remove the non-matched ones from $\widetilde{\mathbf{S}}$;
17. **end for**
18. *Handle the cells in the subarea labeled as B':*
19. **for**$(i \leftarrow i^{(v)} + 1; i < \tau; i \leftarrow i + 1)$
20. Examine the subscriptions in $cell(i, i^{(v)})$ via *Theorem 2*, and remove the non-matched ones from $\widetilde{\mathbf{S}}$;
21. **end for**
22. *Handle the cell labeled as D:*
23. Examine the subscriptions in $cell(i^{(v)}, i^{(v)})$ via both *Theorem 1* and *Theorem 2*, and remove the non-matched ones from $\widetilde{\mathbf{S}}$.

2) Selection Matching Method: The selection matching method is carried out directly in the current $\widetilde{\mathbf{S}}$, so it requires no index structures.

We have the following theorems to describe all situations that a subscription partially matches an event on a dimension:

Theorem 4: Given a_m, v_m of $\mathbf{E}$ and $p_n^{(m)}$ of $\mathbf{S}_n$, we say $\mathbf{S}_n$ partially matches $\mathbf{E}$ on a_m when 1) $p_n^{(m)}$.lowValue $< v_m$ and $p_n^{(m)}$.highValue $> v_m$ or 2) $p_n^{(m)}$.lowValue $= v_m$ and $p_n^{(m)}$.lowType = INCLUSIVE or 3) $p_n^{(m)}$.highValue $= v_m$ and $p_n^{(m)}$.highType = INCLUSIVE.

Proof: $\mathbf{S}_n$ partially matches $\mathbf{E}$ on a_m when v_m is in the interval of $p_n^{(m)}$. The condition 1) describes that v_m locates in the interval of $p_n^{(m)}$, and the condition 2) and 3) describe that v_m locates at the low and high inclusive boundary of $p_n^{(m)}$'s interval, respectively. ■

If all of the above 3 situations do not hold, then $\mathbf{S}_n$ does not partially match $\mathbf{E}$ on a_m, so $\mathbf{S}_n$ does not match $\mathbf{E}$.

The process of the SDM via the selection matching method lies in traversing the subscriptions in the current $\widetilde{\mathbf{S}}$. Each subscription is examined via *Theorem 4*. If a subscription does not match $\mathbf{E}$, the method excludes it from $\widetilde{\mathbf{S}}$. Hence, when the current SDM completes, the subscriptions left in $\widetilde{\mathbf{S}}$ all partially match $\mathbf{E}$ on a_m.

Based on the above explanations, the process of the selection matching method is shown in **Algorithm 2**.

Algorithm 2 The Selection Matching Method

INPUT: $v_{m\widetilde{}}$ of $\mathbf{E}$ on a_m, $\widetilde{\mathbf{S}}$ of the previous SDM.
OUTPUT: $\widetilde{\mathbf{S}}$ for a_m.
01. **for** each subscription $\mathbf{S}_i$ in $\widetilde{\mathbf{S}}$ **do**
02. Examine $\mathbf{S}_i$ via *Theorem 4*. If $\mathbf{S}_i$ does not match $\mathbf{E}$, remove it from $\widetilde{\mathbf{S}}$;
03. **end for**

E. Event Matching Process

The event matching process of GEM can be compared to a pipeline with m machining units. A machining unit corresponds to the SDM for a dimension. The machining units are connected sequentially, that is, the output of a previous machining unit is connected to the input of the next one. $\widetilde{\mathbf{S}}$ is a workpiece that is processed by the SDMs continuously from the beginning of the pipeline to the end. At the start of the event matching process, GEM submits $\mathbf{S}$ to the pipeline, where $\mathbf{S}$ is used as the input of the first SDM. When all SDMs in the pipeline complete, $\widehat{\mathbf{S}}$ is the $\widetilde{\mathbf{S}}$ output from the SDM at the end of the pipeline.

In the pipeline, both the sequence of the SDM for each dimension, and the matching method chosen from the graph partitioning method and the selection method for each SDM are configurable. Assisted by a decision algorithm, which is near-optimal but low-cost, GEM decides the suitable sequences and matching methods to carry out the event matching for each incoming event adaptively. In this way, the event matching performance of GEM is promoted efficiently.

We define a sequence vector as $\mathbf{C} =< c_1, c_2, ..., c_M >$, where $c_k \in \mathbf{C}$ is the index of the dimension whose SDM is arranged at the kth location in the pipeline; we also define a method vector as $\mathbf{D} =< d_1, d_2, ..., d_M >$, where $d_k \in \mathbf{D}$ represents the chosen matching method for the dimension whose SDM is at the kth location in the pipeline. The graph partitioning matching method is chosen if $d_k = 0$; the selection matching method is chosen if $d_k = 1$. According to the above definitions, the event matching process of GEM is described in **Algorithm 3**.

Algorithm 3 The Event Matching Process of GEM

INPUT: $\mathbf{E}$, $\mathbf{S}$.
OUTPUT: $\widehat{\mathbf{S}}$.
01. Determine $\mathbf{C}$ and $\mathbf{D}$ via the decision algorithm;
02. $\widetilde{\mathbf{S}} \leftarrow \mathbf{S}$;
03. **for** $k \leftarrow 1 : M$ **do**
04. **if** $|\widetilde{\mathbf{S}}| = 0$ then // quit the pipeline if no subscriptions left in the current $\widetilde{\mathbf{S}}$
05. **break**;
06. **end if**
07. **if** $d_k = 0$ **then** // choose the graph partitioning method
08. Carry out the SDM for a_{c_k} via **Algorithm 1** with inputs v_{c_k} and $\widetilde{\mathbf{S}}$;
09. **else** // $d_k = 1$, choose the selection method
10. Carry out the SDM for a_{c_k} via **Algorithm 2** with inputs v_{c_k} and $\widetilde{\mathbf{S}}$;
11. **end if**
12. **end for**
13. Now $\widetilde{\mathbf{S}}$ is $\widehat{\mathbf{S}}$.

F. Decision Algorithm

The main work of the SDM for a dimension via the graph partitioning matching method lies in traversing all non-matched subscriptions, so the cost can be approximated as the number of these subscriptions. We normalize the cost on a_m by $|\mathbf{S}|$, and formulate it by $\beta_m^{(\mathbf{E})} = \delta_m^{(\mathbf{E})}/|\mathbf{S}|$, where $\delta_m^{(\mathbf{E})}$ is the number of non-matched subscriptions on a_m for $\mathbf{E}$. $\delta_m^{(\mathbf{E})}$ can be obtained by the counters which increase or decrease along with the subscription insertion or deletion operations. If the subscriptions on a_m follow the Uniform distribution, $\beta_m^{(\mathbf{E})}$ can be approximated as the proportion of the sum of subarea 1 and 2 in the triangle area, namely, $\beta_m^{(\mathbf{E})} = \frac{\text{Subarea1+Subarea2}}{\text{TriangleArea}} = 1 - \frac{\text{Subarea3}}{\text{TriangleArea}} = 1 - \frac{2v_m(R_m - v_m)}{(R_m)^2}$.

In the process of the SDM for a dimension via the selection matching method, all subscriptions in the input $\widetilde{\mathbf{S}}$ are traversed, so the cost can be approximated as the number of the subscriptions in $\mathbf{S}$. We normalize the cost on a_m by $|\mathbf{S}|$, and formulate it by $\gamma_m^{(\mathbf{E})} = |\widetilde{\mathbf{S}}|/|\mathbf{S}|$, here, $|\widetilde{\mathbf{S}}|$ is the output of the previous SDM, and it can be obtained via a counter for $\widetilde{\mathbf{S}}$.

We define the event matching rate $\rho_m^{(\mathbf{E})}$ to indicate the proportion of the partially matched subscriptions on a_m for $\mathbf{E}$. Actually, $\rho_m^{(\mathbf{E})}$ is the normalized number of the partially matched subscriptions, so $\rho_m^{(\mathbf{E})} = 1 - \beta_m^{(\mathbf{E})}$. Thus, for a_{c_k} whose SDM is arranged at the kth location in the pipeline, the number of subscriptions in the $\widetilde{\mathbf{S}}$ output by the SDM can be estimated by $|\widetilde{\mathbf{S}}^{(c_k)}| = \rho_{c_k}^{(\mathbf{E})}|\widetilde{\mathbf{S}}^{(c_{k-1})}|$.

Now we can compute the cost of the SDM on a_{c_k}. If the graph partitioning matching method is employed, $\beta_{c_k}^{(\mathbf{E})} = \delta_{c_k}^{(\mathbf{E})}/|\mathbf{S}|$; if the selection matching method is employed, $\gamma_{c_k}^{(\mathbf{E})}$ can be obtained by multiplying the event matching rates of a_{c_k}'s previous SDMs, that is, $\gamma_{c_k}^{(\mathbf{E})} = |\widetilde{\mathbf{S}}^{(c_{k-1})}|/|\mathbf{S}| = \rho_{c_{k-1}}^{(\mathbf{E})}|\widetilde{\mathbf{S}}^{(c_{k-2})}|/|\mathbf{S}| = ... = \rho_{c_{k-1}}^{(\mathbf{E})}...\rho_{c_1}^{(\mathbf{E})}|\mathbf{S}|/|\mathbf{S}| = \prod_{l=1}^{k-1}\rho_{c_l}^{(\mathbf{E})}$.

Based on the above definitions, the cost of the event matching for $\mathbf{E}$ can be formulated by

$$u^{(\mathbf{E})} = \sum_{k=1}^{M}\left((1-d_k)\beta_{c_k}^{(\mathbf{E})} + d_k\gamma_{c_k}^{(\mathbf{E})}\right) \tag{3}$$

GEM optimizes the event matching performance for $\mathbf{E}$ by providing suitable $\mathbf{C}$ and $\mathbf{D}$. GEM attempts to minimize the cost of the event matching for $\mathbf{E}$, namely, the target of GEM is $\min_{\mathbf{C},\mathbf{D}} u^{(\mathbf{E})} = \min_{\mathbf{C},\mathbf{D}}\left(\sum_{k=1}^{M}\left((1-d_k)\beta_{c_k}^{(\mathbf{E})} + d_k\gamma_{c_k}^{(\mathbf{E})}\right)\right)$. The above problem belongs to a NP-hard combinatorial optimization problem [17]. Actually, solving the problem directly is very complex and expensive since a large number of feasible solutions needs to be examined. To this end, we propose a decision algorithm to find out a near-optimal solution with very low computation cost ($O(M\log M + M)$ time complexity). The algorithm is based on the following observations:

Observation 1: In the optimal solution, the graph partitioning matching method is always employed by the first SDM in the pipeline.

Observation 2: If both methods are employed in the optimal solution, the SDMs in the pipeline are combined by a pre-part and a post-part. The matchings in the pre-part all employ the graph partitioning matching method, whereas, the matchings in the post-part all employ the selection matching method.

Observation 3: If the selection matching method is applied in the optimal solution, the dimensions on which the method is employed are sorted in ascending order of event matching rate.

The proofs of the above observations are not provided here due to the page limitation of the paper. We present the decision algorithm in **Algorithm 4**. The main idea of the algorithm is to carry out the SDM for the dimension with low event matching rate in prior, and the *Observation 1-3* are obeyed at the same time. It is actually a greedy algorithm to shrink the search space in the event matching process.

Algorithm 4 The Decision Algorithm

INPUT: $\mathbf{E}$, $\mathbf{A}$ and $\mathbf{S}$.
OUTPUT: $\mathbf{C}$ and $\mathbf{D}$.
01. Compute the $\rho_m^{(\mathbf{E})}$ for each a_m in $\mathbf{A}$. According to *Observation 3*, sort $\mathbf{A}$ in ascending order of event matching rate via a sorting algorithm with $O(M\log M)$ time complexity;
02. According to *Observation 1*, choose the first dimension a_m in the sorted $\mathbf{A}$, then let $c_1 = a_m$ and $d_1 = 0$;
03. $k \leftarrow 2$, $\epsilon \leftarrow 1$;
04. **while** $k \leq |\mathbf{A}|$ **do**
05. Choose the kth dimension a_m in the sorted $\mathbf{A}$;
06. $\epsilon \leftarrow \epsilon \cdot \rho_{c_{k-1}}$; // multiplying the previous event matching rate to compute $\gamma_m^{(\mathbf{E})}$
07. **if** $\epsilon > \beta_m^{(\mathbf{E})}$ **then** // if $\gamma_m^{(\mathbf{E})}$ is greater than $\beta_m^{(\mathbf{E})}$
08. Let $c_k \leftarrow a_m$ and $d_k \leftarrow 0$; // the pre-part described in *Observation 2*
09. **else**
10. **jump to** line 14;
11. **end if**
12. $k \leftarrow k + 1$;
13. **end while**
14. **while** $k \leq |\mathbf{A}|$ **do**
15. Choose the kth dimension a_m in the sorted $\mathbf{A}$;
16. Let $c_k \leftarrow a_m$ and $d_k \leftarrow 1$; // the post-part described in *Observation 2*
17. $k \leftarrow k + 1$;
18. **end while**

G. Cache Method

In the event matching process of GEM, the cost of the first SDM in the pipeline takes a considerable proportion in the total cost of event matching. In large systems, this becomes severe because there are millions of subscriptions in $\mathbf{S}$ needing to be traversed by the graph partitioning matching method for the first SDM. To this end, we propose a cache method to enhance the graph partitioning matching method for the first SDM, which can efficiently decrease the number of subscriptions needed in the traversing. A cache is a subset of $\mathbf{S}$, which contains all subscriptions in a rectangle area from the triangle area of GEM's index structure. Instead of traversing in $\mathbf{S}$, the cache method first chooses and clones the proper cache for $\mathbf{E}$ from ω candidate caches with different locations on L. The location of the chosen cache is the nearest to the mapped v_{c_1} of $\mathbf{E}$. Then, the output $\widetilde{\mathbf{S}}$ can be obtained by carrying out some extra traversing for the unshared subareas between the rectangles of the cache and the mapped v_{c_1}.

An instance with $w = 3$ is shown in Fig. 3(a), where 3 caches represent 3 rectangle areas in the triangle area, and their locations on L are t_1, t_2 and t_3, respectively. For a_{c_1}, the cache-enhanced graph partitioning matching method is employed for the SDM. Firstly, the method chooses cache 2 which is the nearest to the mapped v_{c_1} of $\mathbf{E}$, then clones the cache as the input $\widetilde{\mathbf{S}}$ for a_{c_1}. The cloning belongs to a shallow copy process [18], so it can be completed very fast and it only takes a low memory cost. Secondly, the method finds out the rectangle of the mapped v_{c_1}, which is partitioned from the triangle area via l_1 and l_2. Thirdly, the method locates the unshared subareas labeled as A and B between the rectangles of cache 2 and the mapped v_{c_1}. For the subscriptions in the subarea labeled as A, which locates outside of the cache's rectangle but inside of the mapped v_{c_1}'s rectangle, they all partially match $\mathbf{E}$, so the method examines the related cells and adds these subscriptions to the cloned cache. For the subscriptions in the subarea labeled as B, which locates inside of the cache's rectangle but outside of the mapped v_{c_1}'s rectangle, they all do not match $\mathbf{E}$, so the method examines the related cells and removes these subscriptions from the cloned cache. When the above operations are done, the cloned cache can be taken as the output $\widetilde{\mathbf{S}}$ from a_{c_1}. The details of the cache method are shown in **Algorithm 5**.

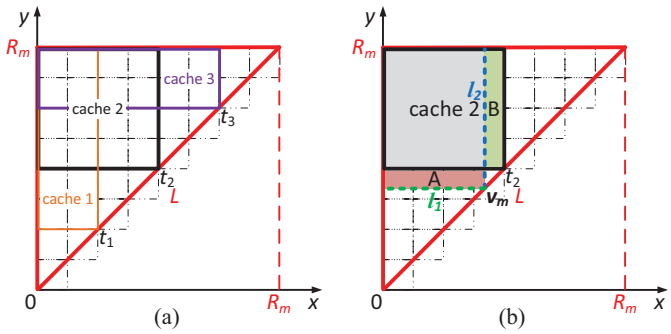

Fig. 3. (a) 3 caches covering 3 rectangle areas in the triangle area, (b) the unshared subareas of the rectangles of cache 2 and the mapped v_{c_1}.

Algorithm 5 The Cache Method

INPUT: v_{c_1} of $\mathbf{E}$ on a_{c_1}.
OUTPUT: $\widetilde{\mathbf{S}}$ for a_{c_1}.
01. Find out the cache whose location t_k is the nearest to the mapped v_{c_1};
02. $\widetilde{\mathbf{S}} \leftarrow$ the cache;
03. Compute the row/column index $i^{(v)}$ of the mapped v_m, and the row/column index $i^{(t)}$ of t_k via **Formula** (2);
04. **if** $v_{c_1} < t_k$ **then**
05. **for**$(i \leftarrow i^{(v)}; i \leq i^{(t)}; i \leftarrow i + 1)$ // handle the shared subarea labeled as A
06. **for**$(j \leftarrow 0; j \leq i^{(v)}; i \leftarrow j + 1)$
07. Add the subscriptions, which partially match $\mathbf{E}$ in $cell(i, j)$, to $\widetilde{\mathbf{S}}$;
08. **end for**
09. **end for**
10. **for**$(i \leftarrow i^{(t)}; i < \tau; i \leftarrow i + 1)$ // handle the shared subarea labeled as B
11. **for**$(j \leftarrow i^{(v)}; j \leq i^{(k)}; j \leftarrow j + 1)$
12. Remove the subscriptions, which do not match $\mathbf{E}$ in $cell(i, j)$, from $\widetilde{\mathbf{S}}$;
13. **end for**
14. **end for**
15. **else if** $v_{c_1} > t_k$ **then**
16. **for**$(i \leftarrow i^{(t)}; i \leq i^{(v)}; i \leftarrow i + 1)$ // handle the shared subarea labeled as A
17. **for**$(j \leftarrow 0; j \leq i^{(k)}; j \leftarrow j + 1)$
18. Remove the subscriptions, which do not match $\mathbf{E}$ in $cell(i, j)$, from $\widetilde{\mathbf{S}}$;
19. **end for**
20. **end for**
21. **for**$(i \leftarrow i^{(v)}; i < \tau; i \leftarrow i + 1)$ // handle the shared subarea labeled as B
22. **for**$(j \leftarrow i^{(k)}; j \leq i^{(v)}; j \leftarrow j + 1)$
23. Add the subscriptions, which partially match $\mathbf{E}$ in $cell(i, j)$, to $\widetilde{\mathbf{S}}$;
24. **end for**
25. **end for**
26. **end if**

As the cache method involves, the subscription insertion/delection operations also need to be carried out for the related caches. Additionaly, a cache itself consumes an extra amount of memory to backup the duplicates of subscriptions. Thus, although the event matching speed can be promoted efficiently by the cache method, it prolongs the subscription insertion/deletion time and consumes more memory, which deteriorate along with the increase of ω. The balance of the performance among these criteria can be done by adjusting ω.

III. Performance Evaluation and Experiment Results

In order to evaluate the performance of GEM extensively, experiments are implemented in several scenarios with multiple different parameters. GEM with different number of caches is compared with 3 state-of-the-art reference algorithms, TAMA [6], H-Tree [10] and REIN [7]. In the experiments,

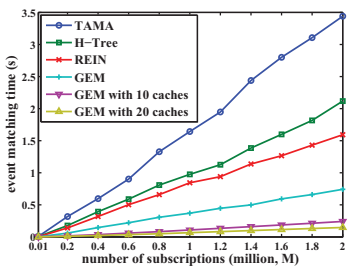

Fig. 4. The event matching time with the increase of N

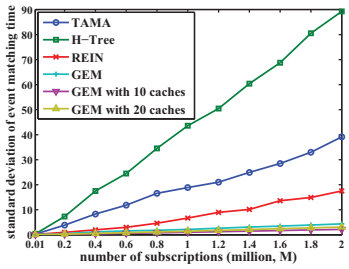

Fig. 5. The standard deviation of the event matching time

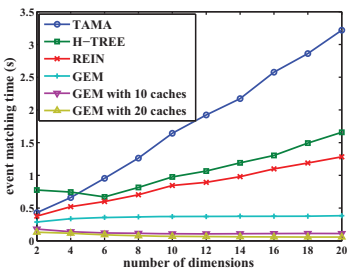

Fig. 6. The event matching time with the increase of M

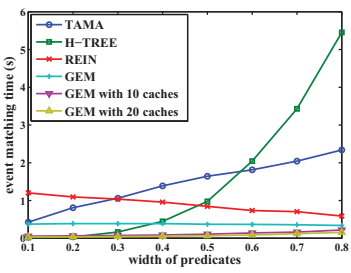

Fig. 7. The event matching time with the increase of θ

TABLE I
THE PARAMETERS USED IN EXPERIMENTS

Name	Meaning	Value
N	the number of subscriptions	$[0.01\text{M}, 2\text{M}]$
M	the number of dimensions	$[2, 20]$
θ	the width of a predicate's interval	$[0.1, 0.8]$
τ	the maximum number of cells in a row or a column in the array of the GEM's index structure	80
ω	the number of caches used in GEM	$0, 10, 20$
π	the recommended number of buckets in REIN [7]	1000
ξ	the recommended number of discretization levels in TAMA [6]	13
η	the recommended number of cells in H-Tree [10]	4
λ	the recommended number of indexed dimensions in H-Tree [10]	6

the main criteria for event matching are evaluated, which include event matching time, subscription insertion/deletion time and memory consumption. Additionally, our near-optimal decision algorithm is compared with the optimal decision algorithm. We implement TAMA, H-Tree, REIN and GEM in Java programming language. The programs are executed on a Dell PowerEdge R720 server with a 2.6GHz Intel Xeon CPU and 192GB RAM. The server runs Linux OS (kernel 3.13.0) with a Java runtime (JavaSE-1.8) installed.

In the experiments, $R_1, R_2, ..., R_M$ are all normalized to 1 for simplicity. The low and high values of an event's interval on a_m are randomly generated from $[0, R_m]$, where the low value is from $[0, 1-\theta]$, and the high value is from $[\theta, 1]$. The boundary types of an interval are uniformly chosen from "( )", "( ]", "[ )", and "[ ]". If the cache method is used ($\omega > 0$), the locations of ω caches split L into $\omega + 1$ equal segments.

The result of each experiment is averaged from the results of repeated measurements with 500 random events. The parameters are configured based on TABLE I unless stated clearly.

A. Event Matching Time

1) Event Matching Time with Variable Number of Subscriptions: Fig. 4 gives the variation in event matching time as we increase N, where $M = 10$, $\theta = 0.5$. The reductions in event matching time using the GEM without caches ($\omega = 0$) over TAMA, H-Tree and REIN are from 75.4% to 81.1%, 60.3% to 68.3% and 47.9% to 57.2% with averages 78.2%, 63.5% and 53.9%, respectively. Further, the event matching time via the GEMs with $\omega = 10$ and $\omega = 20$ are 24.8% to 75.1% and 41.2% to 84.9% faster (the averages are 66.5% and 78.2%) than via the GEM without caches, respectively. It is also observed that, with the increase of N, the performance degradation of GEM ($\omega = 0, 10, 20$) is the lowest among all algorithms. The reason for the superiority of GEM is that the non-matched subscriptions are filtered out as early as possible in the event matching process, so that the search space shrinks efficiently and thus the event matching time is reduced to a large extent.

The stability of event matching time is measured as well. The results with the increase of N are shown in Fig. 5, where $M = 10$, $\theta = 0.5$. In general, the standard deviations of all algorithms grow with the increase of N. The averages of TAMA, H-Tree, REIN and GEM ($\omega = 0, 10, 20$) are 18.7, 43.3, 7.4, 2.2, 1.0, 1.4, respectively. We found that the standard deviation of GEM ($\omega = 0, 10, 20$) is the lowest for all N, and it is nearly flat with the increase of N. Therefore, GEM can provide very stable performance in event matching time for different event since optimization is done for every incoming event to guarantee a near-optimal event matching for the event.

2) Event Matching Time with Variable Number of Dimensions: Fig. 6 shows the increase of event matching time as M grows, where $N = 1\text{M}$, $\theta = 0.5$. The average event matching time of TAMA, H-Tree, REIN and GEM ($\omega = 0, 10, 20$) are 1.7704s, 1.0697s, 0.8494s, 0.3605s, 0.1210s, 0.0784s, respectively, and GEM ($\omega = 0, 10, 20$) is the best among them. Overall, the event matching time of all algorithms grows as M rises, while, the curves of the GEMs with $\omega = 0, 10, 20$ are almost flat or slightly declining. The main reason lies in that the event matching time via GEM is in major relevance to the front dimensions in the pipeline, and the dimensions with low event matching rates for the event can be found more easily as M increases. The matched subscriptions decreases continuously as the event matching goes, thus, $|\widetilde{\mathbf{S}}|$ could be very small for the dimensions in the rear of the pipeline, the costs of the SDMs for these dimensions only take a very low proportion in the total cost. We noticed that the event matching time of H-Tree first decreases slightly when $M < 6$, then increases when $M > 6$. This is because the dropping of event matching rate dominates in the former, whereas, in the latter, the expansion of search space is the leading factor.

3) Event Matching Time with Variable Widths of Predicates: θ is a characteristic that represents the selectivity of a predicate. The wider θ is, the higher possibility that the predicate satisfies a certain event, and vice versa. The event matching time via the 4 algorithms is measured in the experiments with the increase of θ, and the results are given in Fig. 7, where $N = 1\text{M}$, $M = 10$. The average

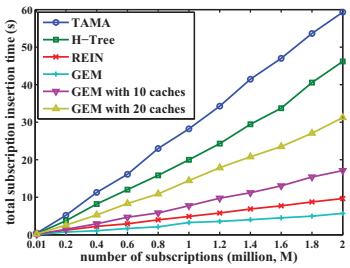

Fig. 8. The total subscription insertion time with the increase of N

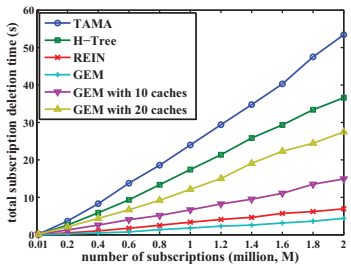

Fig. 9. The total subscription deletion time with the increase of N

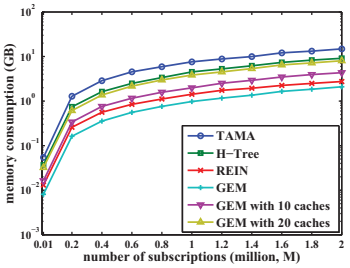

Fig. 10. The memory consumption with the increase of N

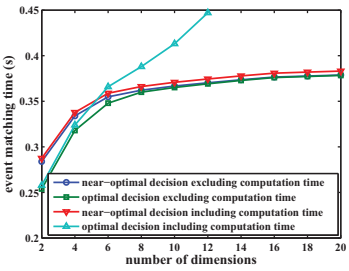

Fig. 11. The two decision algorithms with the increase of M

event matching time of TAMA, H-Tree, REIN and GEM ($\omega = 0, 10, 20$) is 1.44s, 1.5705s, 0.8949s, 0.3712s, 0.1131s, 0.0743s, respectively, and GEM is the best among them. It can be seen that the performance of the algorithms is diverse with the increase of θ. The event matching time of TAMA and GEM ($\omega = 0, 10, 20$) increases linearly as θ grows, but the rising amplitude of GEM ($\omega = 0, 10, 20$) is the lowest, because when θ is low, the dimensions with low event matching rates are chosen preferentially, which makes $\widetilde{\mathbf{S}}$ small enough to be quickly processed by the selection method; when θ is high, the matching cost is reduced by the graph partitioning method since the amount of non-matched subscriptions is small. The performance of H-Tree worsens exponentially since the predicates with wide θ fall into more cells so that the number of cells needing to be traversed increases exponentially. REIN provides short event matching time when θ is wide, because the number of non-matched subscriptions, which need to be traversed and excluded, decreases as θ increases.

B. Subscription Insertion/Deletion Time

In the implementation of GEM, the data structure of LinkedList in Java programming language is adopted to store the references of subscriptions in a cell. A LinkedList object is a linked list structure, which offers subscription insertion operations with $O(1)$ time complexity theoretically and supports subscription traversing. Fig. 8 and Fig. 9 show the total subscription insertion time and deletion time with the increase of N, respectively, where $M = 10$, $\theta = 0.5$. The average subscription insertion time of TAMA, H-Tree, REIN and GEM ($\omega = 0, 10, 20$) per subscription is 0.2981ms, 0.2118ms, 0.0557ms, 0.0321ms, 0.0813ms, 0.1490ms, respectively, and their average subscription deletion time per subscription is 0.2289ms, 0.1636ms, 0.0329ms, 0.0166ms, 0.0732ms, 0.1258ms, respectively. The GEM with $\omega = 0$ is the lowest among these algorithms since subscriptions are stored without being duplicated. Whereas, a subscription may be stored in multiple cells in TAMA and H-Tree. The high value and low value of a predicate are stored into two separated bucket lists in REIN. It can be seen that the subscription insertion/deletion time of GEM worsens with the increase of ω, because the reference of a subscription is duplicated and stored into ω caches for each dimension. Thus, the cache method promotes the event matching speed of GEM, although, the deterioration in subscriptions insertion/deletion time must be considered and balanced for high-dynamic systems.

C. Memory Consumption

Fig. 10 illustrates the memory consumptions of the 4 algorithms for storing N subscriptions, where $M = 10$, $\theta = 0.5$, and y-axis is in log-scale. The average memory consumptions of TAMA, H-Tree, REIN and GEM ($\omega = 0, 10, 20$) per subscription are 7.12KB, 4.27KB, 1.38KB, 0.94KB, 1.98KB, 3.68KB, respectively. We found that the memory consumption of the GEM without caches ($\omega = 0$) is the lowest among all algorithms, because the employed LinkedList consumes a relatively smaller amount of memory than other data structures, and the subscriptions are stored without being duplicated, whereas, as was aforementioned, subscription duplicating is needed by other algorithms. Theoretically, the space complexity of the GEM without caches ($\omega = 0$) is $O(NM)$ for storing N references of subscriptions into M dimensions. The memory consumption of the GEM with caches ($\omega > 0$) rises with the increase of ω. When $\omega = 10$ and $\omega = 20$, the memory consumptions are on average 110.6% and 291.4% larger than that without caches ($\omega = 0$), respectively. Thus, extra memory consumption is another drawback of the cache method, ω should be further considered to avoid reaching the memory limitation of the system.

D. Performance of the Decision Algorithm

The optimality and computation cost of our decision algorithm is evaluated in the experiments, and its performance is compared with that of the optimal decision algorithm, which computes the optimal solution through traversing and examining all feasible solutions that satisfy the Observation 1-3. The optimal decision algorithm has an exponential time complexity with M as a parameter. The experiment results for the event matching time via the two algorithms are shown in Fig. 11, where $N = 1$M, $\theta = 0.5$, $\omega = 0$ and M grows from 2 to 20. If the computation time of the two algorithms is not considered, the difference between the two algorithms decreases as M increases. The event matching time via our near-optimal decision algorithm is very close to that via the optimal decision algorithm, and the two curves almost coincide after $M = 10$. However, the computation cost of the decision algorithm is a part of the total cost of event matching, so the computation time of the two algorithms must be considered. When $M > 6$, we can see that the event matching time via the optimal decision algorithm is much higher than that via our near-optimal algorithm, and the gap between them increases exponentially as M rises. Therefore, although the optimal

decision can be obtained via the optimal decision algorithm, its high computation complexity rapidly worsens the total event matching cost. Conversely, the solution via our near-optimal decision algorithm is quite close to the optimal, meanwhile, it only brings a relatively negligible computation cost.

IV. Related Works

As were aforementioned, the related works of event matching mechanisms can be generally divided into two categories: the single-dimensional-based event matching and the all-dimensional-based event matching. Several representative works are described and analyzed here.

The single-dimensional-based event matching is studied in [6]–[9]. TAMA [6] builds up a layered index structure containing multiple bisected cells for each dimension. The $\widetilde{\mathbf{S}}$ of a SDM is iteratively collected in the corresponding cells from the top layer to the bottom. Then $\widehat{\mathbf{S}}$ is integrated via a counting algorithm. The main drawbacks of TAMA lie in that it only supports approximate matching, and it needs extra storage since a subscription may be stored into multiple cells, and it also uses the counting algorithm which can not effectively reducing unnecessary searching. In REIN [7], the index structure uses two bucket lists for each dimension to store the low values and high values of the predicates on that dimension. The non-matched subscription are excluded in the SDM, and a bit set is employed as $\widetilde{\mathbf{S}}$. $\widehat{\mathbf{S}}$ is generated from the bit set when all SDMs are done. However, a subscription is stored twice on each dimension, and the event matching is quite rigid since there are no optimizations among the SDMs. The bit set is less applicable because the matched subscriptions still need to be accessed by traversing the whole bit set.

The all-dimensional-based event matching is investigated in [10]–[13]. H-TREE [10] is based on a tree-like index structure, which associates several specified dimensions as layers. In each layer, the range of the dimension is partitioned into several partially overlapped cells, where subscriptions are mapped based on their center locations and the widths of their predicates. In this way, H-TREE groups similar subscriptions, so that event matching only needs to be carried out in related groups. BE-Tree [11] partitions subscriptions defined on a high dimensional space using space partitioning and space clustering technologies, to group the predicates with respect to the ranges of dimensions. However, the performance of both H-TREE and BE-Tree is limited by the widths of predicates. On one hand, subscriptions with wide predicates is split and stored into too many cells/clusters, which leads to the increase of storage costs; on the other hand, too many subscriptions fall into the same cell/cluster if the cell/cluster criterion is too wide, which deteriorates the event matching performance.

V. Conclusion

This paper introduces GEM, an analytic geometrical approach to fast event matching for multi-dimensional content-based pub/sub services. The high performance of GEM is realized by the innovative design of a triangle-like index structure and a pipelined event matching process consisting of the SDM for each dimension with two candidate matching methods: the graph partitioning method and the selection method. The event matching of GEM is optimized for every incoming event via a decision algorithm and can be further boosted by a cache method. The results from extensive experiments demonstrate the superiority and stability of GEM, especially in high-dynamic large-scale systems.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (Grant No. 61502050, 61170275), Civil Aerospace Science and Technology Project, YangFan Innovative & Entrepreneurial Research Team Project of Guangdong Province, and Fundamental Research Funds for the Central Universities.

References

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

[2] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H. A. Jacobsen, "Efficient event processing through reconfigurable hardware for algorithmic trading.," *Proceedings of the Vldb Endowment*, vol. 3, no. 12, 2010.

[3] A. Mitra, M. Maheswaran, and J. A. Rueda, "Wide-area content-based routing mechanism," in *Parallel and Distributed Processing Symposium, International*, p. 246b, 2003.

[4] C. Krügel, T. Toth, and C. Kerer, "Decentralized event correlation for intrusion detection," in *Information Security and CryptologyICISC 2001*, pp. 114–131, Springer, 2002.

[5] M. Fontoura, S. Sadanandan, J. Shanmugasundaram, S. Vassilvitski, E. Vee, S. Venkatesan, and J. Zien, "Efficiently evaluating complex boolean expressions," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 3–14, ACM, 2010.

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

[7] S. Qian, J. Cao, Y. Zhu, and M. Li, "Rein: A fast event matching approach for content-based publish/subscribe systems," in *INFOCOM, 2014 Proceedings IEEE*, pp. 2058–2066, IEEE, 2014.

[8] A. Margara and G. Cugola, "High-performance publish-subscribe matching using parallel hardware," *IEEE*, vol. 25, no. 1, pp. 126–135, 2014.

[9] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei, "A scalable and elastic publish/subscribe service," in *Proceedings of the 2011 IEEE International Parallel and Distributed Processing Symposium*, pp. 1254–1265, 2011.

[10] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang, "H-tree: An efficient index structure for event matching in content-based publish/subscribe systems," *IEEE Transactions on Parallel and Distributed Systems*, p. 1, 2014.

[11] M. Sadoghi and H. A. Jacobsen, "Be-tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space," *Sigmod Conference*, pp. 637–648, 2011.

[12] V. Muthusamy and H. A. Jacobsen, "Infrastructure-free content-based publish/subscribe," *Networking IEEE/ACM Transactions on*, vol. 22, no. 5, pp. 1516 – 1530, 2014.

[13] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pp. 802–810, ACM, 2013.

[14] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, pp. 53–61, ACM, 1999.

[15] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Achieving scalability and expressiveness in an internet-scale event notification service," *Proceedings of the Nineteenth Annual Acm Symposium on Principles of Distributed Computing*, pp. 219–227, 2000.

[16] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114–131, June 2003.

[17] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

[18] Object Copying, https://en.wikipedia.org/wiki/Object_copying.