HEM: A Hardware-aware Event Matching Algorithm for Content-based Pub/Sub Systems

Wanghua Shi and Shiyou Qian(✉)

Shanghai Jiao Tong University, Shanghai, China
`s-whua@sjtu.edu.cn`, `qshiyou@sjtu.edu.cn`

Abstract. Content-based publish/subscribe (CPS) systems are widely used in many fields to achieve selective data distribution. Event matching is a key component in the CPS system. Many efficient algorithms have been proposed to improve matching performance. However, most of the existing work seldom considers the hardware characteristics, resulting in performance degradation due to a large number of repetitive operations, such as comparison, addition and assignment. In this paper, we propose a Hardware-aware Event Matching algorithm called HEM. The basic idea behind HEM is that we perform as many bit OR operations as possible during the matching process, which is most efficient for the hardware. In addition, we build a performance analysis model that quantifies the trade-off between memory consumption and performance improvement. We conducted extensive experiments to evaluate the performance of HEM. On average, HEM reduces matching time by up to 86.8% compared with the counterparts.

Keywords: Event matching · Hardware-aware · Bitset OR operation.

1 Introduction

As a flexible communication paradigm, content-based publish/subscribe (CPS) systems are widely applied in manifold applications, such as stock trading system [16] [1], recommendation system [7], social message dissemination system [2] and monitoring system [10]. There are three basic roles in CPS systems to achieve fine-grained data distribution: subscriber, publisher and broker. Subscribers commit subscriptions to brokers according to their interests and focuses. Publishers produce events and send them to brokers. For each incoming event, brokers execute event matching algorithms to search for matching subscriptions and forward the event to the subscribers to which those matches belong.

Apparently, matching algorithms play an irreplaceable role in CPS systems. With the growth of event arrival rate, subscription number and the size of event and subscription, it is inevitable for event matching algorithm to be a performance bottleneck of the entire CPS system. The urge to decrease the end-to-end data distribution latency and improve the throughput in CPS systems in the past decades has driven algorithm design in event matching.

Diverse algorithms have been proposed to boost event matching. According to the first search targets (matching or unmatching subscriptions), there are forward algorithms (such as TAMA [20] and OpIndex [19]) and backward algorithms (such as REIN [13] and GEM [6]). Each matching algorithm has its own design idea. For example, TAMA [20] designs an index table to locate all the predicates matching a given event value and maintains a counter for each subscription. REIN [13] locates two unmatching cell lists for each event value and traverses the cells to mark unmatching subscriptions. From the hardware perspective, TAMA and OpIndex do numerous addition operations one by one while REIN and GEM perform arduous traversal and marking operations bit by bit, thereby consuming much time to do repetitive operations. In effect, it is more efficient for computers to do in-flash bitwise operations because the overhead of data movement between caches and memory is mitigated [8]. In addition, as the memory price is falling and large capacity of memory becomes more common nowadays, cache mechanism becomes a potential method to trade space for time.

Motivated by the above discussion, we propose a novel hardware-aware event matching algorithm (HEM) based on bitset OR operations. HEM adopts a backward method to obtain matching results, similar to REIN [13] and GEM [6]. The basic idea is to fully utilize the hardware characteristic of doing a set of bitwise OR operations in the parallel way. Specifically, HEM uses a subscription pre-mark cache (SPC) mechanism to replace repeated traversal and marking operations with efficient bitwise OR operations in the matching process, following the concept of trading space for time. We build a theoretical model to quantify the trade-off between performance improvement and memory consumption.

Extensive experiments are conducted to evaluate the performance of HEM based on synthetic and real-world stock dataset. First of all, verification experiments validate the conclusion of the theoretical analysis: doubling the cache size halves the marking time. Secondly, compared with four counterparts, namely REIN [13], Ada-REIN [15], OpIndex [19] and TAMA [20], metric experiments show that HEM reduces matching time by 86.8%, 86.3%, 82.1% and 45.3% on average. The main contributions of our work are as follows:

- We propose a hardware-aware event matching algorithm called HEM which aims to reduce the time of repeated operations in the matching process.
- We propose a subscription pre-mark cache method to optimize the matching efficiency of HEM by trading space for time.
- We build a theoretical performance analysis model that quantifies the trade-off between memory consumption and performance improvement.
- We evaluate the performance of HEM through extensive experiments based on synthetic and real-world dataset.

2 Related Work

In the past few decades, improving matching performance has been one of the hot topics in the CPS system. Copious efficient event matching algorithms have been proposed, such as REIN [13], Ada-REIN [15], Comat [5], TAMA [20], OpIndex [19], H-Tree [14], MO-Tree [4], GEM [6], GSEC [18], PS-Tree [9].

Classification of Matching Algorithms. Matching algorithms can be classified from different perspectives. For instance, the work [11] classifies event matching algorithms according to whether the underlying data structure is subscription-grouping (such as H-Tree [14] and MO-Tree [4]) or predicate-grouping (such as REIN [13] and TAMA [20]). The work [21] reviews matching algorithms from three aspects: single-thread algorithms (such as Ada-REIN [15] and OpIndex [19]), parallel algorithms (such as PhSIH [11] and CCM [17]), and algorithms for elasticity (such as GSEC [18] and CAPS [12]).

Moreover, the work [6] divides matching mechanisms into single dimensional based and all dimensional based. The work [13] evaluates algorithms by whether it is forward matching or backward matching. Based on the search strategies, the work [3] regards the algorithms as filtering-based matching and counting-based matching. The work [4] provides a new perspective of whether the matching algorithm supports event matching and subscription matching.

Furthermore, event matching algorithms can also be distinguished from exact matching (such as REIN [13] and PS-Tree [9]) or approximate matching (such as Ada-REIN [15] and TAMA [20]), multi-algorithm composition matching (such as Comat [5]) or single algorithm matching.

Analysis of Operations in Matching Algorithms. In this paper, we assess a matching algorithm by whether it has a bottleneck on repeated operations. For example, REIN [13] is a backward matching algorithm indexing predicates. In the matching process, REIN mainly performs cell traversal and excessive repeated bit-marking operations. Each unmatching subscription is marked ψ'_S times for each event where ψ'_S is the number of unsatisfied predicates. REIN performs well with high matching probability of subscriptions because the workload of marking unmatches is small. GEM [6] is an analogous backward algorithm and has a similar problem. It designs a cache method to boost the removal operations, which is consistent with the idea of alleviating repeated operations.

TAMA [20] is a forward and counting-based matching algorithm. The core idea is to obtain the satisfied predicates rapidly and increase the counters of the corresponding subscriptions. TAMA performs well with low matching probability because the workload of counting satisfied predicates is small. However, all the satisfied predicates of both matching and unmatching subscriptions are counted. As a result, counting operations is a performance bottleneck. Differently, OpIndex [19] starts from the pivot attribute to search, which filters abundant unmatching subscriptions if the event does not contain the pivot attribute. If subscriptions including a set of certain interval predicates defined on different attributes are indexed together, we can replace multiple plus one operations with one direct addition operation. GSEC [18] constructs data structure in this way.

Distinct from the above algorithms suffering from repeatedly executing operations (such as assignment operations in REIN and addition operations in

TAMA) one by one, HEM stores subscription states in caches and mainly performs efficient bitwise OR operations during the matching process.

3 Problem Definition

For brevity, in model design and analysis, we regard the value domain of each attribute as [0, 1]. Let d be the number of attributes in the content space.

Definition 1. *An* event $E(e_1, e_2, ..., e_{\psi_E})$ *consists of* ψ_E *attribute-value pairs, which is a data point in the space.* $e_i(a_j, v)$ *means the* i^{th} *value* v *of* E *is defined on attribute* a_j*. Event size* ψ_E *is the number of nonempty attributes in* E*. Generally,* ψ_E *is much smaller than* d *in a high-dimensional space.*

Definition 2. *A* predicate $p(a_j, [l, h])$ *is an interval defined on attribute* a_j*, which includes a low value* l *and a high value* h *in the closed form. The width* w *of* $p(a_j, [l, h])$ *is* $h - l$. $p(a_j, [l, h])$ *matches an event value* $e(a_i, v)$ *only if* $a_j = a_i \wedge l \leq v \leq h$.

Definition 3. *A* subscription $S(p_1, p_2, ..., p_{\psi_S})$ *is composed by* ψ_S *predicates defined on distinct attributes. Usually* ψ_S *is much smaller than* ψ_E. S *matches* E *if each predicate of* S *matches the value of* E *on the same attribute.*

Definition 4. *Given an event and a set of subscriptions, event matching algorithm searches all the matches of the event from the set.*

4 Design

4.1 Overview

It is challenging to design matching algorithm to adapt to a wide range of application requirements. First, the number and size of subscriptions may vary widely. This change should not cause large fluctuations in the matching time. Second, the dimension of the content space may be between tens to tens of thousands. Both low-dimensional and high-dimensional situations should be adopted. Third, multiple event types should be supported and the size of the event should not seriously affect the matching performance. Fourth, the insertion and deletion of subscriptions should bring little overhead. Fifth, the skewed distribution of attributes, event values and predicate values should not cause the major performance loss. Sixth, the width of the predicates in subscriptions should not have much impact on the matching time. The first three points are regarded as hard parameters and the last two ones are soft parameters.

When the number of subscriptions is large, for most algorithms, matching an event requires lots of repeated operations, such as comparison, addition or bit marking. From a hardware point of view, bit OR operations are more efficient than the operations performed repeatedly by most existing matching algorithms. Therefore, replacing other types of operations with bit OR operations is a feasible solution to improve matching performance. Taking this idea into consideration, we design a hardware-aware data structure to index subscriptions.

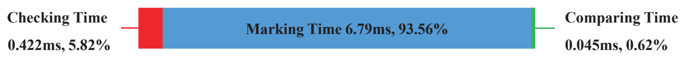

Fig. 1: The matching time distribution of REIN

4.2 Data Structure of HEM

The data structure of HEM consists of a three-level index layer and a collection of bitsets for each attribute. The first level of indexing is based on attributes. The second level is indexed by the low value end (LVE) and high value end (HVE) of the predicate. The third level is constructed by dividing the value domain of the attribute into c cells. Each cell maps to a bucket that stores the low or high value of the predicate and the corresponding subscription ID. Each LVE or HVE is associated with a collection of bitsets that are used to pre-mark certain subscriptions as mismatches. When inserting subscription S into the structure of HEM, the low/high value of each predicate in S is mapped to the cell responsible for the value at the LVE/HVE of the attribute respectively. The predicate value (low or high) and the subscription ID are stored as a pair in the cell. An example of the structure is shown in Fig. 2.

Similar to GEM [6] and REIN [13], HEM uses a backward matching method, first searching for unmatching subscriptions to obtain matches indirectly. Given the partitioned cells with an attribute, if an event value v falls into the cell c_j, all predicates with low values larger than v will definitely not match the event at LVE. Specifically, unsatisfied predicates are stored in the cells from c_{j+1} to the last cell. Similarly, at HVE, all predicates with high values less than v stored in the cells from c_1 to c_{j-1} should also be marked as unmatching. At LVE and HVE, the event should compare with the pairs in cell c_j one by one to determine unmatching subscriptions.

When matching event, REIN [13] iterates through each cell that contains unmatching subscriptions, and marks each mismatch in the bitset. These repeated marking operations account for most of the matching time, as shown in Fig. 1. HEM avoids marking mismatches as much as possible by a caching mechanism. Multiple cells are allocated to one group and each group has a bitset to record whether each subscription is in the group. HEM pre-marks the subscriptions in each group as mismatches in the corresponding bitset. This optimization is called the subscription pre-mark cache (SPC) method. In this way, HEM avoids costly traversal and marking operations, and instead uses bitwise OR operations that are more efficient for hardware.

Subscription Pre-mark Cache (SPC) Method Let g be the number of groups and c be the number of cells at LVE or HVE. In particular, each group g_i contains a list of continuous $i * \frac{c}{g}$ cells. Since the number of bitsets at LVE or HVE is equal to the number of groups, it is reasonable to let bitset B_i record the subscriptions in group g_i that contains the first/last $i * \frac{c}{g}$ cells for HVE/LVE. In this way, the coverage lengths of the bitsets constitute an arithmetic sequence with a common difference of $\frac{c}{g}$ cells.

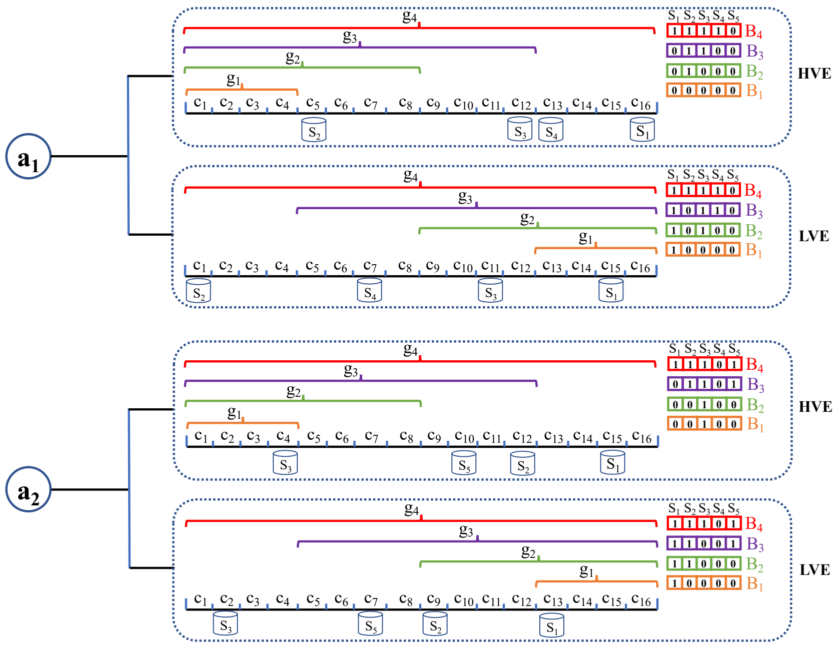

Fig. 2: HEM data structure

Table 1: Sample subscriptions

ID	$\mathbf{a_1}$	$\mathbf{a_2}$	ID	$\mathbf{a_1}$	$\mathbf{a_2}$	ID	$\mathbf{a_1}$	$\mathbf{a_2}$
S_1	[0.9, 0.95]	[0.8, 0.9]	S_2	[0.0, 0.3]	[0.5, 0.7]	S_3	[0.63, 0.69]	[0.1, 0.2]
S_4	[0.38, 0.76]	-	S_5	-	[0.4, 0.57]	-		

Fig. 2 shows an example structure of HEM which builds indexes on two attributes a_1 and a_2. The value domain of each attribute is divided into sixteen cells assigned to four groups. Specifically, at the LVE of each attribute, group g_1 includes cells from c_{13} to c_{16}, g_2 from c_9 to c_{16} and so on. In contrast, the way of grouping cells at the HVE starts from c_1. The length of the bitset associated with each group is equal to the number of subscriptions n. Each bitset is used to mark the subscriptions stored in the cells belonging to the corresponding group.

Insertion Algorithm Inserting a subscription S into HEM has two steps. Firstly, for each predicate in S, according to the attribute and low/high value, the predicate value and the subscription ID are stored as a pair in the cell that covers the value. Secondly, according to the cell grouping scheme, the subscription is marked in one or more bitsets. Algorithm 1 shows the pseudo code of insertion.

Fig. 2 also shows the state of the data structure indexing the five sample subscriptions listed in Table 1. For example, when inserting S_1, for its first

Algorithm 1: Insertion Algorithm

```
Input: Subscription $S$
1 for each predicate $p(a_i, [l, h])$ in $S$ do
2     Insert a pair $(l, S.ID)$ into the cell covering $l$ and mark $S$ in the
        corresponding bitsets of the groups that cover the cell at LVE;
3     Insert a pair $(h, S.ID)$ into the cell covering $h$ and mark $S$ in the
        corresponding bitsets of the groups that cover the cell at HVE;
```

predicate, $\lfloor 0.9*16+1 \rfloor = 15$ and $\lfloor 0.95*16+1 \rfloor = 16$, so S_1 is mapped to cell c_{15} and c_{16} at the LVE and HVE of a_1 respectively. Predicate values are omitted for brevity. Notice that c_{15} is in all the four groups at the LVE, so S_1 is marked in the four bitsets. At the HVE, c_{16} is only contained by group g_4, so S_1 is marked in the corresponding bitset B_4. The second predicate of S_1 is processed similarly.

4.3 Matching Procedure of HEM

HEM uses a bitset B to record the matching results. Each unmarked bit represents a matching subscription. Algorithm 2 gives the matching procedure of HEM, which can be divided into six steps. The first four steps are to process each attribute-value pair of the event. Step 1 performs comparisons in the cell into which the event value falls at LVE and HVE respectively and marks the unmatching subscriptions in B (lines 3-4). Step 2 selects the largest-size group that does not contain the cell into which the event value falls. Note that at LVE/HVE, when the cell ID is larger or equal than $\frac{c(g-1)}{g}$ / smaller or equal than $\frac{c}{g}$, no such group is available (line 5). Step 3 performs bit OR operations between B and the bitset of the selected group if available at LVE and HVE respectively for each nonempty attribute in the event (line 6). Step 4 marks the unmatching subscriptions in B that are stored in the cells not covered by the selected group (line 7). Step 5 does a series of bit OR operations between B and the bitset of the largest-size group for each null attribute of the event (line 8). When an event does not contain an attribute, all predicates defined on the attribute are not satisfied. Step 6 checks the unmarked bits in B to obtain the matching results (line 9).

Let event $E = \{(e_1(a_1, 0.64), e_2(a_2, 0.32)\})$ as an example. Based on the data structure shown in Fig. 2, the first attribute-value pair e_1 of E falls into the cell $\lfloor 0.64 * 16 + 1 \rfloor = 11$ on a_1. The low/high values of the predicates stored in cell c_{11} need to be compared with e_1 one by one at LVE/HVE respectively. Since the low value 0.63 of S_3 in c_{11} is smaller than the event value 0.64, it is not marked as unmatching in B. Next, $B_1(10000)$ and $B_2(01000)$ are selected to do bit OR operations with B since g_1 and g_2 are the largest-size group that does not cover c_{11} at the LVE and HVE of a_1 respectively. As a result, $B = (11000)$. Subsequently, cell c_{12} at LVE and cells c_9 and c_{10} at HVE should be traversed since they store unmatching subscriptions and are not covered by the selected groups. In this case, there are no subscriptions to be marked as unmatching in

Algorithm 2: Matching Algorithm

```
Input: Event $E$
Output: Matching results $B$
1 Initialize a zero bitset $B$ whose length is the number of subscriptions;
2 for each attribute-value pair $(a_j, v)$ in Event $E$ do
3     Find the cell $c_l$ at LVE and $c_h$ at HVE that the attribute value falls into;
4     Compare $v$ with the predicate values in $c_l$ and $c_h$, and marks the IDs of
        subscriptions as unmatching in $B$;
5     Select the largest-size group $g_l$ and $g_h$ not covering $c_l$ and $g_h$ at LVE and
        HVE respectively;
6     Do bit OR operations between $B$ and the bitsets of $g_l$ and $g_h$ to obtain all
        unmatching subscriptions of the groups if the group is not null;
7     Mark the IDs of subscriptions as unmatching for the rest unmatching cells
        not covered by $g_l$ and $g_h$ in $B$ at LVE and HVE respectively;
8 Do $(d - \psi_E)$ times of bit OR operations between $B$ and the bitset of the
    largest-size group to obtain the mismatches defined on null attributes of $E$;
9 Check the unmarked bits in $B$ to output matching results.
```

B. In REIN [13], cells from c_{12} to c_{16} at LVE and from c_1 to c_{10} at HVE need to be traversed one by one, but in HEM only three cells need to be traversed. The second attribute-value pair e_2 in E is processed similarly. At the comparing step, S_5 stored in cell c_7 at the LVE of a_2 is marked as unmatching in $B(11001)$. $B_2(11000)$ and $B_1(00100)$ at the LVE and HVE respectively of a_2 do bit OR operations with B. At the final checking step, $B(11101)$ has one unmarked bit, meaning that S_4 is the match of E.

HEM reduces the traversal and marking operations in the matching process by setting up a set of caches that pre-mark certain subscriptions as unmatching. Therefore, HEM mainly does bit OR operations when matching events. Generally, compared to marking each unmatching subscription by traversing each cell, it is more efficient to perform a bit OR operation to collectively mark the unmatching subscriptions stored in multiple cells. When the selected largest-size group covers all the cells that need to be traversed, the marking time of HEM can be optimized to be close to zero. For each attribute, the total number of cells to be traversed at LVE and HVE is a constant, namely $\frac{c}{g} - 1$.

5 Theoretical Analysis

5.1 Complexity Analysis

Time Complexity of Insertion Algorithm. For a subscription S with size ψ_S, it needs to insert $2\psi_S$ pairs into the corresponding cells, which takes $O(\psi_S)$. In addition, marking the subscription in one or more bitsets has a cost at most $O(g\psi_S)$. Therefore, the time complexity of insertion is $O(g\psi_S)$.

Time Complexity of Matching Algorithm. The matching procedure of HEM can be divided into six steps to analyze. The comparison step (Lines 3-4) checks pairs in two cells with average cost $O(\frac{n\psi_S}{dc})$. The cost of two bit OR operations (Lines 5-6) is $O(n)$. The marking step (Line 7) traverses $\frac{c}{g}-1$ cells for both value ends, so the cost is $O(\frac{n\psi_S}{dc} * (\frac{c}{g} - 1)) = O(\frac{n\psi_S}{dg})$. Given the event size ψ_E, these three steps have a cost $O(\psi_E n(1 + \frac{\psi_S}{dg}))$ since $g \leq c$. The time to process null attributes (Line 8) is $O((d - \psi_E)n)$. The final check step (Line 9) takes $O(n)$. Therefore, the time complexity of the matching algorithm is $O(dn + \frac{n\psi_E\psi_S}{dg})$.

Space Complexity. Given the dimensionality d of the content space and the number of cells c divided on each attribute, the total number of cells is dc. The total number of bitsets is $2dg$. For n subscriptions with size ψ_S, the total number of predicates is $n\psi_S$. Thus, the space complexity of HEM is $O(dc + dng + n\psi_S)$.

5.2 Performance Analysis

In this section, we build a analysis model for HEM to quantify the relationship between performance improvement and memory consumption. To explore the improvement on marking time, we take the HVE of one attribute as a breakthrough since the LVE and HVE of each attribute have similar characteristics.

Lemma 1. *Given the number of subscriptions n with size ψ_S, the marking time of HEM is proportional to $0.5n\psi_S$ without the SPC method.*

Proof. The marking time of HEM is proportional to the times of marking subscriptions one by one, which is computed as:

$$n\psi_S \int_0^1 (x - 0)dx = 0.5n\psi_S \tag{1}$$

where dx can be seen as a probability and x is the possible value of an event. The interval range to be traversed is $[0, x]$. □

Lemma 2. *Given n, ψ_S and g, the marking time of HEM is halved by doubling g with the SPC method.*

Proof. The adoption of the SPC method limits the number of traversing cells to $\frac{c}{g} - 1$. Hence, with the SPC method, the number of marked subscriptions is computed as:

$$\begin{aligned} & n\psi_S \sum_{i=0}^{g-1} \int_{\frac{i}{g}}^{\frac{i+1}{g}} (x - \frac{i}{g})dx \\ &= n\psi_S \int_0^1 xdx - \frac{n}{g}\psi_S \sum_{i=0}^{g-1} i \int_{\frac{i}{g}}^{\frac{i+1}{g}} 1dx \\ &= 0.5n\psi_S - \frac{(g-1)n\psi_S}{2g} = \frac{n\psi_S}{2g} \end{aligned} \tag{2}$$

Table 2: Parameter settings in the experiments

Name	Description	Experimental Values
$\mathcal{R}$	The value domain of attribute.	$[1, 1M]$
α	Parameter of Zipf.	**0**, 1 $\sim$ 5
d	Number of attributes.	**20**, 30, 100, 300 $\sim$ 900
n	Number of subscriptions.	0.3M, **1M**, 3M $\sim$ 9M
ψ_E	Event Size.	**20**, 30 $\sim$ 80
ψ_S	Subscription size.	5, **10**, 15 $\sim$ 30
w	Predicate width.	0.1, 0.2, **0.3** $\sim$ 0.9

where $x - \frac{i}{g}$ is the length of interval to be marked one by one for any event value $x \in [\frac{i}{g}, \frac{i+1}{g}], i \in [0, g-1]$. Since the marking time is proportional to the times of marking subscriptions one by one, doubling g halves the marking time. □

Theorem 1. *Given n, ψ_E, ψ_S, d and g, when $\psi_E = d, g > 1$ and the predicate values are uniformly distributed in the value domain $[0, 1]$, the improvement ratio of the marking time of HEM is $1 - \frac{1}{g}$ with the SPC method.*

Proof. Based on Lemma 1 and 2, the improvement ratio of the marking tasks of HEM is $1 - \frac{n\psi_S}{2g*0.5n\psi_S} = 1 - \frac{1}{g}$. □

Theorem 1 means that 50% or 96.875% of marking operations are avoided when g is set to 2 or 32 respectively, presenting an inverse proportional relationship. The ratio does not hold when $g = 1$ because it assumes that the unique bitset on LVE or HVE covers all cells. Nevertheless, we can configure that the unique bitset on LVE or HVE covers only half of the cells when $g = 1$. Thus, the improvement of $g = 1$ is equivalent to that of $g = 2$ when $\psi_E = d$.

6 Experiments

6.1 Setup

Workloads The event data comes from a real-world stock dataset with 50 attributes after being cleaned up. The dataset was collected from the Chinese stock market on June 8, 2018. The subscription data is generated based on the stock dataset. For high-dimensional testing, we synthesize event data. Table 2 lists the parameter settings where the default values are marked in bold.

Baselines We compare HEM with four event matching algorithms reviewed in Section 2, namely REIN [13], Ada-REIN [15], TAMA [20] and OpIndex [19]. Based on REIN, Ada-REIN ignores some marking tasks on attributes to reduce matching time, resulting in some false positive matches. REIN and Ada-REIN originally only support single-type event matching ($\psi_E = d$). We re-implemented them to match events with multiple types, namely $\psi_E \ll d$. Besides, they are set with the same c as HEM (1,000 by default). The false positive rate of Ada-REIN is set to 0.05. TAMA counts the matching predicates for each subscription which exist only in ψ_E attributes. The discretization level of TAMA is set to 13. We re-implemented TAMA to achieve exact matching with negligible overhead. OpIndex classifies subscriptions by their pivot attributes and only pivot attributes of events are processed.

Testbed All the algorithms are implemented in C++ language and compiled by g++ 9.3.0 with -O3 optimization enabled on Ubuntu 20.04 system. All the experiments are conducted on an AMD 3.7 GHz machine with 64 GB RAM.

Metrics We evaluate the performance of the five algorithms in terms of three metrics: matching time, insertion time and memory consumption. The matching time is measured from the beginning of matching an event to the end of obtaining the whole matching result. 500 events are processed to calculate the average matching time in each experiment. Insertion time refers to the time of inserting a subscription into the data structure. Memory consumption refers to the total memory used by the underlying data structure of the matching algorithm after inserting a subscription dataset.

6.2 Verification Experiments

We design a benchmark experiment to verify the performance analysis model in Section 5.2 and investigate the trade-off between matching time and memory usage. In this experiment, the parameters are set to the default values.

Fig. 3 presents the marking time of HEM with different number of groups g from 1 to 512. Starting from $g = 2$, the marking time of HEM halves for each time g is doubled. For example, the marking time is 0.60 ms and 0.31 ms for $g = 16$ and $g = 32$ respectively. In this experiment, we set $\psi_E = d$, so the group covering all the cells is not used. Consequently, the marking time for $g = 1$ is approximately equal to that for $g = 2$. The average marking time without the SPC method is 6.79 ms, smaller than twice of the marking time when $g = 2$. This is because the groups are statically divided and the predicates are not absolutely evenly distributed. When $g = 32$, the performance improvement ratio of HEM is $1 - \frac{0.31}{6.79} \approx 95.4\%$, which is close to the theoretical value 96.875% based on Theorem 1. Overall, the ratio of marking time in the total matching time decreases from 93.7% to 2.5% when g increases from 1 to 512, indicating that the bottleneck operation (Fig. 1) has been alleviated. In summary, the SPC optimization method is effective and Theorem 1 is validated.

Fig. 4 depicts the matching time and memory usage of HEM varying g from 1 to 512. The matching time dwindles exponentially and the memory usage grows exponentially with the exponential increase of g. When the SPC optimization method is not enabled, the matching time of HEM is 7.56 ms and the memory consumption is 152 MB. We finally set $g = 32$ to do the metric experiments since the matching time of HEM drops by 89.7% and the memory usage is nearly doubled, which makes a good trade-off.

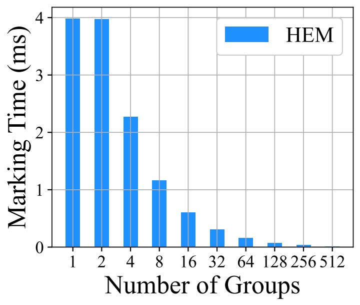

Fig. 3: Marking time of HEM with different g

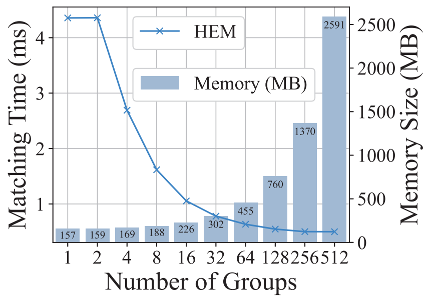

Fig. 4: Matching time and memory usage of HEM with different g

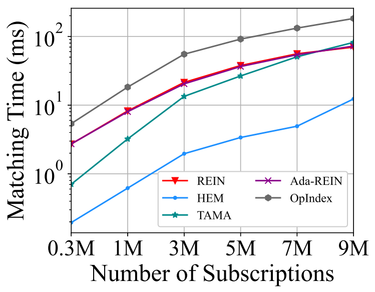

Fig. 5: Effect of n

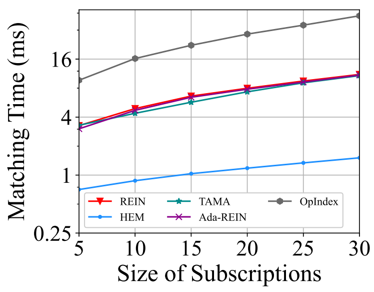

Fig. 6: Effect of ψ_S

6.3 Metric Experiments

The performance of the event matching algorithm is affected by many parameters. We change their settings to observe their effects in the experiments.

Number of Subscriptions n. The number of subscriptions is a core parameter to measure the workload, which has a vital impact on the matching time. As shown in Fig. 5, all algorithms have higher matching time as n increases. HEM performs best in all situations. Compared with REIN, TAMA, Ada-REIN and OpIndex, HEM reduces the matching time by 90.1%, 83.5%, 90.0% and 95.9% respectively on average. When n grows from 3M to 7M, the predicates become more densely distributed in the cells and the optimization space becomes larger. Hence, the matching time of HEM increases slower than before. When n is 9M, the matching time of HEM grows more quickly because the bit OR operations cost a lot. The matching time of TAMA increases faster than REIN and Ada-REIN because the workload of counting satisfied predicates is time-consuming. OpIndex performs worst because predicates are evenly distributed in attributes and all the attributes are elected as pivot attributes.

Subscription Size ψ_S. To measure the effect of ψ_S, we set $d = \psi_E = 30, w = 0.7$ and vary ψ_S from 5 to 30. From Fig. 6 we can see that the matching time of the five algorithms increases linearly with ψ_S. Therefore, ψ_S is more related to the real workload compared to n. The performance of Ada-REIN is almost the same as REIN. This is attributable to the low false positive rate, low matching probability and the uniform distribution of predicates. Compared with REIN, TAMA, Ada-REIN and OpIndex, HEM reduces the matching time by 83.7%, 82.6%, 83.1% and 95.2% respectively average.

Event Size ψ_E. In the event size experiment, we set $d = 80$ and vary ψ_E from 30 to 80. Generally, the event size is proportional to the matching time of the forward matching algorithms (TAMA and OpIndex) and inversely proportional to the matching time of the backward matching algorithms (REIN and Ada-REIN), as shown in Fig. 7. Nevertheless, HEM, as a backward matching algorithm, has a slowly increasing matching time with ψ_E. This is because both comparing and marking operations are avoided and only one bit OR operation is needed to process each null attribute. On average, the matching time of HEM is reduced by 88.5%, 86.5%, 86.3% and 50.3% compared with OpIndex, REIN, Ada-REIN and TAMA, respectively.

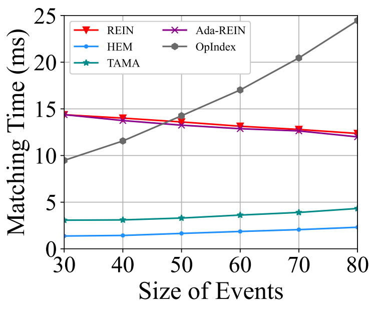

Fig. 7: Effect of ψ_E

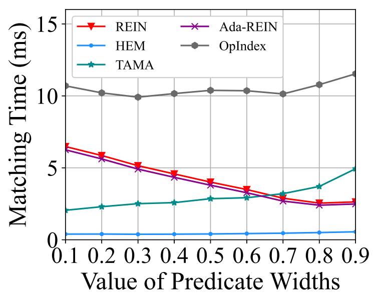

Fig. 8: Effect of w

Predicate Width w. The matching probability of subscriptions is relevant to ψ_S and w. Fig. 8 presents how w affects the matching time by varying w from 0.1 to 0.9. ψ_S is set to 5 to ensure a nonzero matching probability in the experiment. This setting makes the predicates sparsely distributed in cells. As a forward algorithm, TAMA takes a longer time with the increasing of w. The performance of OpIndex is not affected by w. As backward algorithms, REIN and Ada-REIN run faster with w. However, when $w = 0.9$, the predicates are dense and the comparing time of REIN and Ada-REIN becomes unnegligible so their matching time increase a little. HEM is nearly immune to w and exhibits a

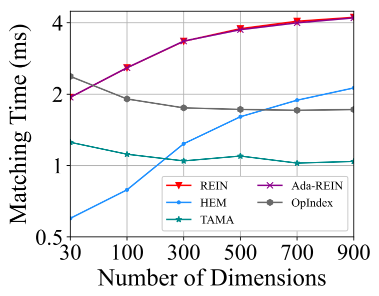

Fig. 9: Effect of d

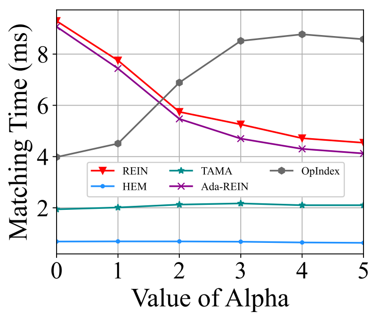

Fig. 10: Effect of α

steady performance because the number of cells to be marked is limited to $\frac{c}{g} - 1$ for each nonempty attribute. On average, in comparison with REIN, TAMA, Ada-REIN and OpIndex, HEM reduces the matching time by 88.0%, 85.0%, 87.3% and 95.9% respectively.

Number of Attributes *d* To simulate sparse workloads, we set $w = 0.5$ and vary d from 30 to 900. Fig. 9 indicates that all algorithms behave monotonically. TAMA performs best after d is up to 300. The two forward matching algorithms show a similar trend with d because they process each event value rather than each attribute and the workload decreases with the increase of d. However, the three backward matching algorithms have to mark all the unmatching subscriptions in each attribute. HEM replaces the marking task in a null attribute of an event with one bit OR operation. Unfortunately, that still costs a lot under high dimension and the memory consumption becomes large. As a result, d is a hard parameter for backward matching algorithms.

Attribute Distribution. There are two categories of the skewed distribution of algorithm input. One is the value distribution and the other is attribute distribution. Considering that the matching time basically keeps invariant with an uneven value distribution of subscriptions and events, we only give the experiment results under skewed attribute distribution of both subscriptions and events in Fig. 10, where $d = 50, \psi_S = 5$ and $w = 0.5$. In Zipf distribution, a larger α means a more serious skewed distribution and a more heavy workload. HEM and TAMA overcomes the skewed problem well while the other three counterparts fluctuate greatly with α. Ada-REIN skips from 1 attribute and about 26 k predicates to 40 attributes and about 5M predicates when α varies from 1 to 5. Thus its matching time is smaller than that of REIN.

6.4 Maintainability

Two experiments are conducted to test the maintenance cost of HEM. Fig. 11 reveals that the average insertion time of HEM increases by about 42.6% com-

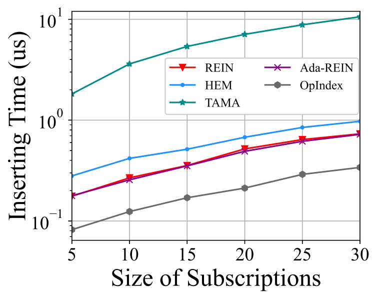

Fig. 11: Insertion time ψ_S

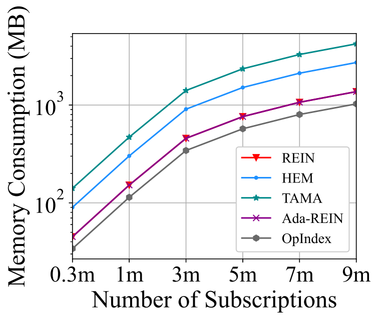

Fig. 12: Memory usage(MB) by n

pared to REIN because HEM needs to pre-mark a new subscription in one or more bitsets for each attribute on which the predicate is defined. TAMA inserts the subscription ID into a set of cells for each predicate, thereby resulting in the highest insertion time. The deleting time of HEM is very close to that of REIN because the time to find the deleted pairs in cells accounts a lot. The experiment results are omitted. Fig. 12 shows the memory usage of the five algorithms with different n. All the curves rise logarithmically. The memory usage of HEM is about twice of REIN's and 64.3% of TAMA's.

7 Conclusion

When processing a large number of subscriptions, most matching algorithms need to repeatedly perform a lot of operations, which becomes a performance bottleneck. In this paper, we propose a hardware-aware event matching algorithm termed HEM, aiming to execute efficient operations in the matching process. The experiment results reveal the superiority of HEM to its counterparts. HEM shows excellent performance in terms of matching time and stability under various conditions. In the future, we plan to design a state reduction method to optimize the bit OR operations of HEM under high dimension, and extend HEM to multi levels to accommodate input with multiple hotspots.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2019YFB1704400), the National Natural Science Foundation of China (61772334, 61702151), and the Special Fund for Scientific Instruments of the National Natural Science Foundation of China (61827810).

References

1. Barazzutti, R., Heinze, T., et al.: Elastic scaling of a high-throughput content-based publish/subscribe engine. In: ICDCS. pp. 567–576. IEEE (2014)

2. Chen, L., Shang, S.: Top-k term publish/subscribe for geo-textual data streams. The VLDB Journal **29**(5), 1101–1128 (2020)
3. Ding, T., Qian, S.: Scsl: Optimizing matching algorithms to improve real-time for content-based pub/sub systems. In: IPDPS. pp. 148–157. IEEE (2020)
4. Ding, T., Qian, S., Cao, J., Xue, G., Zhu, Y., Yu, J., Li, M.: Mo-tree: An efficient forwarding engine for spatiotemporal-aware pub/sub systems. IEEE Transactions on Parallel and Distributed Systems **32**(4), 855–866 (2021)
5. Ding, T., Qian, S., Zhu, W., et al.: Comat: An effective composite matching framework for content-based pub/sub systems. In: ISPA. pp. 236–243. IEEE (2020)
6. Fan, W., Liu, Y., Tang, B.: Gem: An analytic geometrical approach to fast event matching for multi-dimensional content-based publish/subscribe services. In: IEEE INFOCOM. pp. 1–9 (2016)
7. Fontoura, M., Sadanandan, S., Shanmugasundaram, J., et al.: Efficiently evaluating complex boolean expressions. In: ACM SIGMOD. pp. 3–14 (2010)
8. Gao, C., Xin, X., et al.: Parabit: Processing parallel bitwise operations in nand flash memory based ssds. In: IEEE/ACM MICRO-54. pp. 59–70 (2021)
9. Ji, S.: Ps-tree-based efficient boolean expression matching for high-dimensional and dense workloads. Proceedings of the VLDB Endowment **12**(3), 251–264 (2018)
10. Ji, S., Jacobsen, H.A.: A-tree: A dynamic data structure for efficiently indexing arbitrary boolean expressions. In: ACM SIGMOD. pp. 817–829 (2021)
11. Liao, Z., Qian, S., Cao, J., et al.: Phsih: A lightweight parallelization of event matching in content-based pub/sub systems. In: ICPP. pp. 1–10 (2019)
12. Ma, X., Wang, Y., Pei, X., Xu, F.: A cloud-assisted publish/subscribe service for time-critical dissemination of bulk content. Concurrency and Computation: Practice and Experience **29**(8), e4047 (2017)
13. Qian, S., Cao, J., Zhu, Y., Li, M.: Rein: A fast event matching approach for content-based publish/subscribe systems. In: IEEE INFOCOM. pp. 2058–2066 (2014)
14. Qian, S., Cao, J., Zhu, Y., Li, M., Wang, J.: H-tree: An efficient index structure for event matching in content-based publish/subscribe systems. IEEE Transactions on Parallel and Distributed Systems **26**(6), 1622–1632 (2015)
15. Qian, S., Mao, W., Cao, J., Mouël, F.L., Li, M.: Adjusting matching algorithm to adapt to workload fluctuations in content-based publish/subscribe systems. In: IEEE INFOCOM. pp. 1936–1944 (2019)
16. Sadoghi, M., Labrecque, M., Singh, H., Shum, W., Jacobsen, H.A.: Efficient event processing through reconfigurable hardware for algorithmic trading. Proceedings of the VLDB Endowment **3**(1-2), 1525–1528 (2010)
17. Shah, M.A., Kulkarni, D.: Multi-gpu approach for development of parallel and scalable pub-sub system. In: Computing, Communication and Signal Processing, pp. 471–478. Springer (2019)
18. Wang, Y.: A general scalable and elastic content-based publish/subscribe service. IEEE Transactions on Parallel and Distributed Systems **26**(8), 2100–2113 (2014)
19. Zhang, D., Chan, C.Y., Tan, K.L.: An efficient publish/subscribe index for e-commerce databases. Proc. VLDB Endow. **7**(8), 613–624 (2014)
20. Zhao, Y., Wu, J.: Towards approximate event processing in a large-scale content-based network. In: IEEE ICDCS. pp. 790–799 (2011)
21. Zhu, W., Qian, S., Xu, J., Xue, G., Cao, J., Zhu, Y., Li, W.: Lap: A latency-aware parallelism framework for content-based publish/subscribe systems. Concurrency and Computation: Practice and Experience p. e6640 (2021)