BOP: A Bitset-based Optimization Paradigm for Content-based Event Matching Algorithms

Wei Liang[†], Wanghua Shi[§], Zhengyu Liao[†], Shiyou Qian[§*], Zhonglong Zheng[†*], Jian Cao[§] and Guangtao Xue[§]
[†]*Zhejiang Normal University, Zhejiang, China.* [§]*Shanghai Jiao Tong University, Shanghai, China.*
[*]*Corresponding authors, Email: qshiyou@sjtu.edu.cn, zhonglong,mlli@zjnu.edu.cn*

***Abstract*—Content-based publish/subscribe systems are widely used in many fields. Event matching is the core component to achieve fine-grained content-based data distribution. Many efficient algorithms have been proposed to improve event matching performance. However, in large-scale content-based publish/subscribe systems, event matching is still the performance bottleneck of the entire system due to the need to perform a lot of operations, such as additions, comparisons and bit-markings. In this paper, we explore to convert various non-logical operations into efficient logical ones, and propose a bitset-based optimization paradigm (BOP) for matching algorithms. On the one hand, BOP can eliminate expensive operations in the matching process, greatly improving matching performance. On the other hand, BOP can stabilize the performance of matching algorithms, ensuring the quality of service of data distribution. We apply BOP to optimize two existing matching algorithms, namely TAMA and REIN. The experimental results show that BOP shortens the matching time of TAMA and REIN by more than 60%. In addition, the performance of optimized versions is more stable than the original matching algorithms.**

***Index Terms*—Publish/subscribe, event matching, optimization, bitset**

I. Introduction

The publish/subscribe system has been widely used in many fields, such as online advertising [1], mobile message push [2] [3], information filtering [4], and content-based routing systems [5]. It provides a loosely coupled messaging architecture [6]. The publisher publishes messages (also called events) to the broker, and the subscriber submits his/her interest (often called subscriptions) to the broker. The broker matches each event with subscriptions and forwards the event to all interested subscribers. According to different subscription models, the publish/subscribe system can be roughly divided into two categories: topic-based and content-based. The first category is relatively simple, but the granularity is coarse and the expression ability is limited. The content-based publish/subscribe (CPS) system overcomes these shortcomings, and makes the system more flexible.

Event matching algorithm is the core component of CPS systems. Matching efficiency is critical, which directly affects the performance of the entire system. The design of high-performance event matching algorithm should consider time efficiency, subscription maintenance cost and memory consumption. For these three aspects, time efficiency is the major factor that affects the performance of event matching algorithm. Therefore, how to improve the efficiency of matching algorithm is a key problem.

With the continuous research and innovation of scholars, many efficient event matching algorithms have been proposed, such as PS-Tree [7], H-Tree [8], MO-Tree [9], TAMA [10], REIN [11], Siena [12], SCSL [13], HEM [14], PhSIH [15] and Comat [16]. These algorithms utilize different data structures, such as trees, tables and bloom filters, to index subscriptions to achieve high event matching speed. Different algorithms have their advantages and disadvantages.

According to the initial search target in the matching process, existing algorithms can be classified into forward matching algorithm such as TAMA [10] and OpIndex [17], and backward matching algorithm such as REIN [11], Ada-REIN [18] and GEM [19]. The forward matching algorithm directly searches matching subscriptions. They need to perform a lot of additions for counting-based algorithms and comparisons for tree-based algorithms. For example, TAMA [10] is built on an index table which is designed to locate all predicates that are satisfied by the given event value. For subscriptions containing each satisfied predicate, TAMA needs to perform a countering operation. On the other hand, the backward matching algorithm initially aims to search unmatching subscriptions. For example, REIN [11] first locates two lists of cells containing unsatisfied predicates for each event value and then traverses the cells to mark unmatching subscriptions in a bitset. Overall, these algorithms perform a large number of repetitive operations, which decreases the matching efficiency.

To improve the efficiency of existing matching algorithms, we propose a bitset-based optimization paradigm (BOP). The basic idea of BOP is manifested in two aspects. First, with bitsets, BOP can transform the expensive arithmetic operations in existing matching algorithms to efficient logical operations, which can make full use of the characteristics of hardware. Second, the status (matching or unmatching) of subscriptions can be marked in advance in bitsets, which avoids performing a large number of repetitive operations in the matching process. We formalize the optimization paradigm for forwarding matching algorithms. In addition, we extend it to support backward matching algorithm. Therefore, BOP is a general optimization method to improve the efficiency of most matching algorithms.

We apply BOP to optimize two existing matching algorithms, a forward matching algorithm TAMA [10] and a backward matching algorithm REIN [11], resulting in REIN-O and

DOI reference number: 10.18293/SEKE2023-142

TAMA-O respectively. Based on the original data structures of TAMA and REIN, BOP introduces bitsets to speed up event matching. We conducted extensive experiments to evaluate the effectiveness of BOP with different parameter settings, including subscription size, event size, predicate width and predicate attribute distribution. The experimental results show that REIN-O and TAMA-O achieve significant performance improvement. Compared with TAMA and REIN, REIN-O and TAMA-O reduce matching time by more than 60%. In addition, the performance of REIN-O and TAMA-O is more stable than that of TAMA and REIN.

The main contributions of this paper are as follows:

- We propose a bitset-based optimization paradigm called BOP, which aims to improve the efficiency of existing matching algorithms.
- We formalize the optimization paradigm and apply it to optimize two existing matching algorithms.
- We conduct a series of experiments to verify the optimization effect of BOP.

The remainder of this paper is organized as follows. Section II presents the background knowledge. Section III briefly discusses the related work. Section IV describes the design details of BOP. Section V analyzes the experimental results. Section VI concludes the paper.

II. Preliminaries

In this section, we provide the basic knowledge of the event matching problem.

Attribute: An attribute a_i represents the name of a data item, which appears in events and subscriptions. Each attribute in events has a value, expressed in the form of $< a_i, v >$ or a_i=v. The number of all attributes in the content space is called the dimensionality, denoted by d.

Event: Event is also known as message, consisting of multiple attribute-value pairs. Event is expressed as E=$\{a_1$=v_1, a_2=v_2,.....,a_m=$v_m\}$, where m represents the number of attributes in the event. m is called the event size in the paper. Note that each attribute appears only once in an event.

Predicate: A predicate defines a constraint on an attribute. An interval predicate is expressed as $P < a_i, lower, upper >$, where a_i is an attribute, and $lower$ and $upper$ represent the left and right boundaries of the predicate respectively.

Subscription: A subscription is composed of multiple interval predicates in the conjunctive normal form. A subscription S is expressed as $S = \{P_1 \wedge P_2 \wedge ..... \wedge P_k\}$, where k represents the number of predicates in the subscription. k is called the subscription size in the paper.

Match: Given an attribute-value pair a_i=v_i and a predicate $< a_j, lower, upper >$, if a_i=$a_j \wedge v_i \in [lower, upper]$, the attribute value satisfies the predicate. Given an event E and a subscription S, if each predicate in S is satisfied by the corresponding attribute value of E, S is a match of E.

Event Matching Problem: Give a set of n subscriptions and an event E, the event matching problem is to search all the matches of E from the subscription set.

III. Related Work

In this section, we review the related work and classify existing algorithms into two categories: forward matching and backward matching. Then, for each category, we briefly introduce the matching process and discuss a representative algorithm. Finally, we describe the difference between BOP and existing work.

A. Algorithm Classification

In order to achieve high matching performance, an effective data structure for indexing subscriptions is necessary. The classic data structures include match tree [20], match table [21], binary decision graph [22] and Bloom filter [23]. According to the initial search targets, the matching algorithms based on these data structures can be divided into two categories: forward matching algorithm and backward matching algorithms.

B. Forward Matching Algorithms

The forward algorithm aims at matching subscriptions during the matching process, which can be further divided into two sub-categories: counting-based algorithms such as TAMA [10] and OpIndex [17], and tree-based algorithms such as PS-Tree [7] and H-Tree [8].

The counting-based matching algorithm sets a counter for each subscription. In the matching process, for each predicate P satisfied by the event value, the counter of each subscription containing P will be incremented by one. For a subscription, if the number of satisfied predicates recorded in the counter is equal to the subscription size, the subscription is a match of the event. The basic idea of the tree-based matching algorithm is to use the pruning ability of trees, where subscriptions are usually stored in leaf nodes. In the matching process, internal nodes are used to filter unmatching subscriptions, finally generating a set of candidate subscriptions. These subscriptions are accurately evaluated to obtain matching results. To better understand the forward matching algorithm, we discuss a representative algorithm TAMA.

TAMA [10] is a forward and counting-based matching algorithm. Its overall structure can be divided into two layers. The first layer is indexed on attributes. All attributes are organized into a linear table. The second layer adopts the hierarchical discretization method, which divides each predicate into multiple levels. Specifically, the value space of each attribute is divided into cells level by level. Different levels have different granularity. The upper cell is evenly divided into two smaller cells in the lower level. Thus, at the i-th level, the attribute space is divided into 2^i cells, and each cell at the i-th level will be separated into two cells at the $(i+1)$-th level. Given an interval predicate P, from the level that P can contain the cell, the ID of the subscription containing P is stored in at most two cells that are covered by P at each level. Given an event value v, the cell containing v stores the subscriptions whose predicates are satisfied by v. The counter of these subscriptions will be incremented by one.

C. Backward Matching Algorithms

The backward matching algorithm initially targets at unmatching subscriptions during the matching process, such as REIN [11] and GEM [19]. Given the set of subscriptions, when the unmatching ones are determined, the matches can be easily obtained. Generally, the working principle of backward matching algorithms is to search all unsatisfied predicates. For each unsatisfied predicate P, all subscriptions containing P are marked as unmatching in a bitset. REIN is a representative backward matching algorithm, which is discussed in the following.

REIN transforms the matching problem into a rectangular intersection problem, and proposes an index structure to effectively solve the problem. REIN divides the value space of attributes into multiple cells and realizes the one-to-one mapping between predicate values and cells. When matching an event value, the list of cells containing unsatisfied predicates can be easily determined. These cells are iteratively processed by marking the unmatching subscriptions in a bitset. After processing all attributes, the unmarked bits represent matching subscriptions. Therefore, REIN mainly performs cell traversal and bit marking operations.

Existing algorithms perform a large number of repetitive operations, such as additions in TAMA. Compared with logical operations, these operations have higher costs from the perspective of hardware. To improve the performance of existing matching algorithms, we propose a general bitset-based optimization paradigm (BOP), which is applicable to most algorithms. BOP explores to convert costly non-logical operations into efficient logical operations by trading off space for time.

IV. Design of BOP

In this section, starting from the basic idea of BOP, we use bitset to optimize the forward algorithm first, and then extend it to optimize the backward algorithm.

A. Basic Idea

When the number of subscriptions is large, for most algorithms, matching an event needs to perform a large number of repeated operations, such as comparisons, additions or bit-markings. As discussed in Section III, the forward counting-based matching algorithm calculates the number of satisfied predicates for each subscription. When there is only one unsatisfied predicate in a subscription of size k, it is redundant and inefficient to count the $k-1$ satisfied predicates. Therefore, when the predicate width is wide and the subscription size is large, the performance of the counting-base matching algorithm will decline. The backward matching algorithm searches unsatisfied predicates and marks all subscriptions containing them. When there are multiple unsatisfied predicates in a subscription, the subscription will be marked repeatedly. Except for the first time, the subsequent marks are redundant and inefficient. Therefore, the backward matching algorithm performs poorly in high-dimensional content space and small-width predicates.

As a result, from the perspective of hardware, the basic idea of BOP is to convert a large number of expensive operations performed by the matching algorithm into more efficient logical operations. For this purpose, we introduce bitset as a caching mechanism to reduce some repeated operations. In addition, based on bitsets, BOP performs as many logical operations as possible to improve matching performance.

When designing BOP, two guidelines should be followed. First, BOP should be general, which means that BOP should be able to optimize backward and forward matching algorithms. Second, BOP should be non-intrusive, which means that BOP should not change the underlying data structure and matching process of existing algorithms.

B. BOP for Forward Counting-based Algorithm

Most forward counting-based algorithms index predicates defined on an attribute separately. This type of algorithms can only retrieve the satisfied or unsatisfied predicates on single attribute. When matching, the attributes in the event are processed one by one. For each attribute, when finding a satisfied predicate, all subscriptions containing the predicate can be marked in a bitset, instead of incrementing the counters. In this way, counting operations can be replaced with marking operations.

In addition, in the content space with d attributes, subscription size k is generally far smaller than d. In other words, subscriptions usually define predicates on some attributes, with most attributes in the content space having null predicates. From the semantics of event matching, the event value matches all null predicates. Therefore, for each attribute a_i $(i \in [1, 2, ..., d])$, BOP sets a bitset B^i_{null} to record the subscriptions that do not define predicate on a_i.

The matching process with BOP can be formalized as follows:

$$B_{result} = \cap_{i=1}^{d}(B^i_{null} \cup B^i_{match}) \tag{1}$$

where B_{result} represents the result bitset, d is the dimensionality of content space, B^i_{null} is the bitset used to mark subscriptions that have no predicate defined on attribute a_i, and B^i_{match} is the bitset that marks all subscriptions whose predicates defined on attribute a_i are satisfied.

C. Extension for Backward Marking-based Algorithm

BOP can also be extended to optimize the backward marking-based algorithm. Backward matching aims at finding all unsatisfied predicates and marking the corresponding subscriptions as unmatching in a bitset. Following the optimization paradigm defined in Formula (1), backward matching should be first transformed into forward matching, and then logical AND operations are performed. Specifically, the bitset associated with each attribute is initially set to all 1. In the matching process, all unmatching subscriptions found are marked as 0 in the bitset for each attribute. Then, logical AND operations are performed based on the bitset of all attributes. The bits with value 1 in the result bitset represent the matching subscriptions.

D. Implementation Considerations

To apply BOP, there are two implementation considerations. The first is to reduce the memory consumed by bitsets in the matching process, and the second is to efficiently process logical operations.

Intuitively, BOP needs to set d bitsets in the event matching process, and obtains the result bitset by performing logical operations on the d bitsets. This will consume a lot of memory in high-dimensional scenarios. By analyzing the principle of BOP, we find that only two bitsets are needed to iterate in the implementation. Actually, processing an attribute consists of two steps: marking all subscriptions containing satisfied predicates and performing a bitset AND operation. Therefore, it is not necessary to maintain one bitset for each attribute at the same time, but only two bitsets in the matching process. One bitset is used to store the result after processing all previous attributes, and the other bitset stores the subscription matching status on the current attribute.

An event of size m has no value on $d - m$ attributes, which are called null attributes. For each null attribute a_i, the corresponding B^i_{match} is all 0, thus no logical OR operation is required. Thus, there is no need to perform $d - m$ OR operations. BOP first performs a logical OR operation between B^i_{null} and B^i_{match} for each attribute in the event, so there will be m bitset OR operations. Then, BOP performs d bitset AND operations. A total of $d + m$ logical operations are required. However, in the implementation, we can eliminate bitset OR operations. For the m attributes contained in the event, we can copy B^i_{null} first, and mark the found matching subscriptions directly in B^i_{null}, without setting the bitset B^i_{match}, In this way, BOP only needs to perform d bitset AND operations.

E. Complexity Analysis

BOP needs to set a bitset for each attribute, which has the space complexity of $O(nd)$, where n is the number of subscriptions and d is the dimensionality of the content space. In addition, two bitsets are required in the matching process. For subscriptions of size k, the insertion time is increased by $O(d - k)$ due to the marking operations on the null attributes without defining predicates. In the matching process, counting operations are replaced with marking operations, so the number of operations does not change. BOP needs to perform the additional d bitset AND operations.

V. Experiments

To demonstrate the effectiveness of BOP for event matching algorithms, we select two different matching algorithms to optimize, namely REIN [11] and TAMA [10].

A. Test Benches

1) BOP on TAMA: According to the optimization paradigm defined in Formula (1), BOP can be directly applied to TAMA, which results in a new variant called TAMA-O. When inserting a subscription S, it is necessary for TAMA-O to mark S as matching in the bitset B_{null} of each null attribute on which S does not define a predicate. When matching an event E, TAMA-O processes the attributes in E one by one. First, a copy of B_{null} is made, denoted by B'_{null}, for each attribute. Then, based on the original data structure of TAMA, TAMA-O searches satisfied predicates for each event attribute and marks all subscriptions containing these predicates as matching in B'_{null}. Finally, a logical AND operation is performed between the result bitset B_{result} and B'_{null}. After processing all attributes in E and all null attributes, the unmarked bits in B_{result} represent the matching subscriptions of E.

2) BOP on REIN: Since REIN is a backward marking-based matching algorithm, we first consider transforming it into a forward matching, generating a variant called REIN-F. Specifically, REIN-F marks (set to 0) all subscriptions containing unsatisfied predicates in the bitset initialized to all 1, and then obtains subscriptions that match on all attributes through bitset AND operations. In the implementation, REIN-F does not need to actually perform AND operations. For all attributes, the same effect can be achieved by marking 0 on the same bitset, similar to REIN. Based on the data structure of REIN, BOP further optimizes REIN-F by searching satisfied predicates on each attribute. In this way, REIN-F works in the same way as the counting-based matching algorithm. Therefore, REIN-F can be optimized by BOP, resulting in a new variant called REIN-O.

TABLE I: Parameter settings in the experiments

Para.	Description	Values
d	Dimensionality	50
k	Subscription size	4, **6**, 8, 10
m	Event size	10, **20**, 30, 40
w	Predicate width	0.1,0.2,0.3,**0.4**,0.5,0.6,0.7,0.8,0.9
α	Attribute Zipf distribution	**0**, 0.25, 0.5, 0.75, 1.0
n	Number of subscriptions	**1M**, 2M, ..., 8M, 9M

B. Setup

For simplicity, we consider the value space of each attribute as [0,1]. Event values and predicate values are generated based on uniform distribution. Attributes are selected based on Zipf distribution with different setting of parameter α. Since the performance of matching algorithms is affected by many parameters, we evaluate the effects of subscription size k, event size m, predicate width w, predicate attribute distribution α, and the number of subscriptions n. The settings of these parameters are shown in Table I. By default, we set n to 1 million, d to 50, w to 0.4, k to 6, and m to 20. In each experiment, 1,000 events are matched and the average matching time is calculated. All the experiments were conducted on a server with 128 1.4GHZ CPUs, which runs Ubuntu 18.04.6 and Linux kernel 5.4.0-136. All code is written in C++ language and compiled by g++ with version 7.5.0.

C. Results and Analysis

1) Effect of Subscription Size k: We set k to 4, 6, 8 and 10 in the experiment. The experimental results of the four tested algorithms are shown in Figure 1. We can see that BOP significantly improves the performance of TAMA and

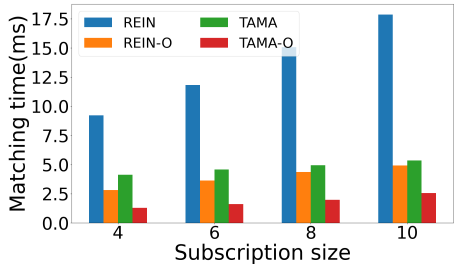

Fig. 1: Effect of subscription size k

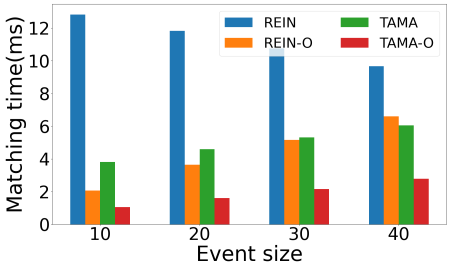

Fig. 2: Effect of event size m

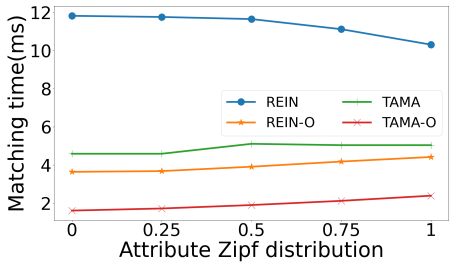

Fig. 4: Effect of attribute Zipf distribution α

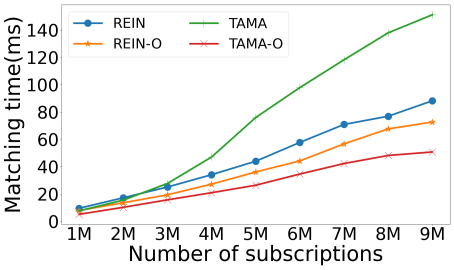

Fig. 5: Effect of the number of subscriptions n

REIN. Compared with TAMA, TAMA-O reduces the matching time by 61% on average. In addition, as k increases, TAMA-O performs more stable. Similarly, REIN-O is more efficient and stable than REIN, reducing the matching time by 71% on average. The reason why BOP achieves better performance is that it converts expensive non-logical operations into logical operations, which is more efficient for hardware.

2) Effect of Event Size m: In this experiment, we set m to 10, 20, 30 and 40. The experimental results are shown in Figure 2. Overall, the matching time of REIN decreases with the increase of m. When fixing the subscription size, events of larger size have more matching subscriptions, which is conductive to improving the performance of REIN. On the contrary, the average matching time of TAMA increases with m. With more matching subscriptions, TAMA needs to perform more counting operations. Compared with TAMA, TAMA-O reduces the matching time by 62% on average. Similarly, compared with REIN, REIN-O reduces the average matching time by 61%.

3) Effect of Predicate Width w: In this experiment, we set w from 0.1 to 0.9. The experimental results are shown in Figure 3. Overall, the matching time of TAMA increases with the increase of w, while REIN is the opposite. On average, the matching time of REIN-O is reduced by 66% compared with REIN. When w=0.1, the matching time is reduced by up to 89%. Similarly, the matching time of TAMA-O is much smaller than that of TAMA. When w=0.1, compared with TAMA, the matching time of TAMA-O is reduced by 79%. With the increase of the matching probability, there are more and more 0s set in the bitset. This is not conducive to the logical OR operations. Thus, the optimization effect of BOP continues to decline with the increase of w, resulting in an average reduction of 60% in the matching time.

4) Effect of Attribute Distribution α: In the experiment, attributes follow the Zipf distribution, where the parameter α is set to 0, 0.25, 0.5, 0.75 and 1. The experimental results are shown in Figure 4. Overall, the matching time of the four algorithms is basically unchanged with the increase of α. This is because the matching probability of subscriptions does not change. BOP can optimize TAMA, reducing the matching time by 60% on average. Similarly, REIN-O reduces the matching time by 65% compared with REIN.

5) Effect of Subscription Number n: In this experiment, we set n from 1M to 9M. The experimental results are shown in Figure 5. The matching time of the four algorithms increases with n because the total workload increases. TAMA-O and REIN-O perform better than their baselines, reducing the matching time by 69% and 66% respectively on average.

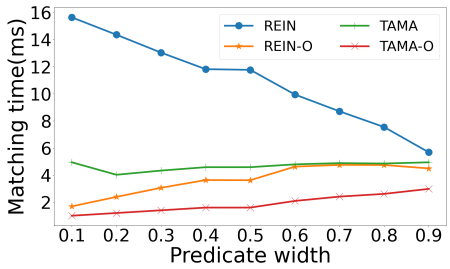

Fig. 3: Effect of predicate width w

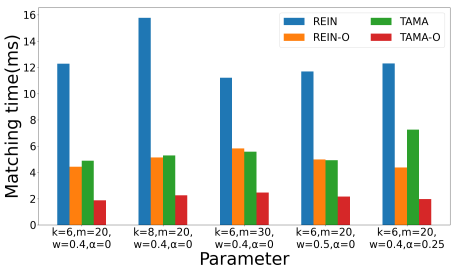

Fig. 6: The 95th percentile matching time

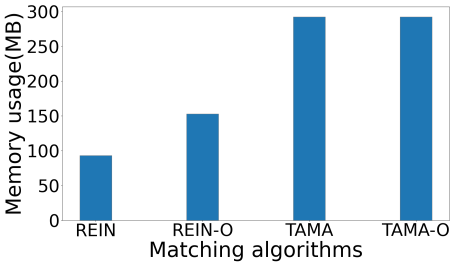

Fig. 7: Memory usage

6) Performance Stability: In this experiment, we evaluate the performance stability of the four algorithms in terms of the 95th percentile (P95) of matching time. With the default parameter settings, we change k, m, w and α respectively. The experimental results are shown in Figure 6. With BOP, TAMA-O and REIN-O perform more stable than their baselines. For example, in the first group of experiments, the P95 value of REIN, REIN-O, TAMA and TAMA-O is 12.27, 4.43, 4.90 and 1.87 ms respectively. Similar conclusions can be reached in other groups of experiments. Therefore, we verify that BOP is beneficial to improve and stabilize matching performance.

D. Memory Usage

In this experiment, we set each parameter as the default value to measure the memory consumption of the four algorithms. The experimental results are shown in Figure 7. Compared with TAMA, TAMA-O does not increase memory consumption. REIN-O increases memory usage by about 39% on the basis of REIN. This is mainly because a bitset is configured for each attribute. Considering that REIN-O reduces the matching time by more than 60%, BOP well achieves the goal of exchanging space for time.

VI. Conclusion

By summarizing the working principles of existing matching algorithms, we propose BOP to improve their performance. The basic idea of BOP is to convert a large number of non-logical operations into logical ones. The experimental results show that BOP can improve and stabilize the performance of matching algorithms. In the future, the optimization effect of BOP can be further improved by dividing attributes into groups and setting a bitset for each group.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61772334).

References

[1] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, "Aups: An open source authenticated publish/subscribe system for the internet of things," *Information Systems*, vol. 62, pp. 29–41, 2016.

[2] S. Nakamura, L. Ogiela, T. Enokido, and M. Takizawa, "An information flow control model in a topic-based publish/subscribe system," *Journal of High Speed Networks*, vol. 24, no. 3, pp. 243–257, 2018.

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

[4] T. W. Yan and H. Garcia-Molina, "Index structures for selective dissemination of information under the boolean model," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 2, pp. 332–364, 1994.

[5] A. Mitra, M. Maheswaran, and J. A. Rueda, "Wide-area content-based routing mechanism," in *IEEE Proceedings International Parallel and Distributed Processing Symposium (IPDPS)*, 2003, pp. 8–pp.

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

[7] S. Ji and H.-A. Jacobsen, "Ps-tree-based efficient boolean expression matching for high-dimensional and dense workloads," *Proceedings of the VLDB Endowment*, vol. 12, no. 3, pp. 251–264, 2018.

[8] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang, "H-tree: An efficient index structurefor event matching in content-basedpublish/subscribe systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 26, no. 6, pp. 1622–1632, 2014.

[9] T. Ding, S. Qian, J. Cao, G. Xue, Y. Zhu, J. Yu, and M. Li, "Mo-tree: an efficient forwarding engine for spatiotemporal-aware pub/sub systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 32, no. 4, pp. 855–866, 2020.

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

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

[12] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM*, 2003, pp. 163–174.

[13] T. Ding, S. Qian, J. Cao, G. Xue, and M. Li, "Scsl: Optimizing matching algorithms to improve real-time for content-based pub/sub systems," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 148–157.

[14] W. Shi and S. Qian, "Hem: A hardware-aware event matching algorithm for content-based pub/sub systems," in *Database Systems for Advanced Applications: 27th International Conference (DASFAA)*, 2022, pp. 277–292.

[15] Z. Liao, S. Qian, J. Cao, Y. Cao, G. Xue, J. Yu, Y. Zhu, and M. Li, "Phsih: A lightweight parallelization of event matching in content-based pub/sub systems," in *Proceedings of the 48th International Conference on Parallel Processing (ICPP)*, 2019, pp. 21:1–21:10.

[16] T. Ding, S. Qian, W. Zhu, J. Cao, G. Xue, Y. Zhu, and W. Li, "Comat: an effective composite matching framework for content-based pub/sub systems," in *IEEE International Conference on Parallel & Distributed Processing with Applications(ISPA)*, 2020, pp. 236–243.

[17] D. Zhang, C.-Y. Chan, and K.-L. Tan, "An efficient publish/subscribe index for e-commerce databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 8, pp. 613–624, 2014.

[18] S. Qian, W. Mao, J. Cao, F. L. Mouël, and M. Li, "Adjusting matching algorithm to adapt to workload fluctuations in content-based publish/subscribe systems," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 1936–1944.

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

[20] 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 18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1999, pp. 53–61.

[21] D. Zhang, C.-Y. Chan, and K.-L. Tan, "An efficient publish/subscribe index for e-commerce databases," *VLDB Endowment*, vol. 7, no. 8, pp. 613–624, 2014.

[22] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "Efficient filtering in publish-subscribe systems using binary decision diagrams," in *Proceedings of the 23rd IEEE International Conference on Software Engineering (ICSE)*, 2001, pp. 443–452.

[23] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *Proceedings of the 2nd ACM International Conference on Distributed Event-Based Systems (DEBS)*, 2008, pp. 71–81.