# PEM: A Parallel Ensemble Matching Framework for Content-based Publish/Subscribe Systems

Weidong Zhu[†], Yufeng Deng[§], Shiyou Qian [§*], Jian Cao[§] and Guangtao Xue[§]
[†]*Xuzhou University of Technology, Jiangsu, China.* [§]*Shanghai Jiao Tong University, Shanghai, China.*
[*]*Corresponding author, Email: qshiyou@sjtu.edu.cn*

***Abstract*—Content-based publish/subscribe systems are an effective paradigm for implementing on-demand event distribution. Each event needs to be matched against subscriptions to identify the target subscribers. To improve the matching performance, many novel data structures have been proposed. However, the predicates included in subscriptions are handled the same way in most existing data structures, which is not efficient given the matching probability of predicates. In this paper, we propose a parallel ensemble matching framework called PEM, which uses multiple algorithms with complementary behavior on predicate matching probabilities. To achieve the performance balance of parallel matching, we design an elastic subscription classification method. We implement a prototype of PEM based on two existing algorithms. The experiment results show that PEM improves the matching performance by 43%.**

## I. Introduction

In the face of data explosion, fine-grained data distribution services are required in many fields. For example, the stock market generates massive data every day, for which investors need to subscribe and receive information on events of interest [1] [2]. Another example is an intelligent transportation scenario, where a large number of devices are deployed to collect a large volume of data. Likewise, drivers need a mechanism to obtain timely congestion and accident information specific to their driving route [3]. These applications motivate the need for an efficient way of propagating data from publishers (sources) to subscribers (destinations).

Content-based publish/subscribe (pub/sub) systems are an effective paradigm for implementing on-demand event distribution. To express their interest in data, subscribers first define subscriptions that usually contain multiple predicates [4] and then issue them to the broker (server). The publisher generates events consisting of multiple attribute-value pairs and sends them to the broker. For each event, the broker needs to match it with subscriptions to identify the target subscribers to whom the event information should be forwarded. In this way, publishers and subscribers are loosely coupled, which is the most attractive feature of content-based pub/sub systems [5].

Obviously, event matching is a key component in a content-based pub/sub system. Given a high-dimensional space, an event represents a point and a subscription represents a rectangle. Event matching is essentially a point enclosure search problem in nature, which is expensive in high-dimensional spaces. To make matters worse, when the number of subscriptions is large, the matching performance degrades, becoming a potential performance bottleneck.

To improve matching performance, many new data structures for storing subscriptions have been proposed, such as trees [6] [7] [8] [9], tables [10] [11] [12] and bloom filters [13] [14]. These novel data structures support efficient event matching. However, most existing structures index predicates in the same way, regardless of their matching probability. As verified in [15], most matching algorithms suffer from predicate matching probabilities. Increasing or decreasing the matching probability will result in performance degradation. One problem of the matching algorithm with fluctuating performance is that it cannot guarantee fast and stable data distribution services.

In this paper, we propose a parallel ensemble matching algorithm called PEM, which aims to improve and stabilize the matching performance using a multi-thread strategy. Similar to COMAT [16] which builds a library with multiple behavior-complementary matching algorithms, the basic idea of PEM is to leverage each algorithm by indexing each subscription in the appropriate algorithm. When matching events, all algorithms run in parallel, one thread per algorithm. When designing PEM, two points need to be addressed. First, we need to classify subscriptions based on the behavior of each algorithm on the predicate matching probability. Second, to prevent the overload of a specific algorithm, we need to establish a dynamic feedback mechanism to ensure the performance balance between multiple algorithms. Considering the characteristics of algorithms and the balance of threads, we design an elastic subscription classification method in PEM.

We implemented a prototype of PEM and conducted extensive experiments to evaluate its effectiveness and performance. The experiment results well verify the ability of PEM to improve and stabilize the matching performance. Compared with COMAT [16] and the other two baselines REIN [12] and TAMA [10], the matching time of PEM is improved by 43%, 55% and 47% respectively on average.

The main contributions of this paper are as follows:

- We propose an effective parallel ensemble matching framework called PEM to take advantage of multiple algorithms.
- We design an elastic subscription classification mechanism to maintain the performance balance between multiple threads.

DOI reference number: 10.18293/SEKE2022-051

- We implement a prototype of PEM and evaluate its effectiveness and performance through extensive experiments.

The remainder of this paper is organized as follows. We briefly discuss the related work in Section II. We describe the design details of PEM in Section III. Section IV elaborates the implementation of PEM. Section V presents and analyzes the experiment results. We discuss and conclude the paper in Section VI and VII respectively.

## II. Related Work

In this section, we review the matching algorithms along two lines: sequential algorithms and parallel algorithms.

### A. Sequential Matching Algorithms

Most existing matching algorithms were initially proposed as sequential, such as TAMA [10], MO-Tree [6], OpIndex [11], HEM [17] and REIN [12]. To achieve high matching performance, an efficient data structure for indexing subscriptions is necessary and critical for the matching algorithm. Classical data structures include matching trees [18] [19] [7], matching tables [20] [10] [11], binary decision diagrams [21] [22] and bloom filters [14] [13]. The underlying data structure of the matching algorithm is responsible for maintaining subscriptions (inserts, deletes and updates) and supporting event matching. In most existing matching algorithms, predicates are treated in the same way, regardless of their matching probabilities.

Liao et al. proposed a parallelization method called PhSIH to optimize the performance and stability of the sequential matching algorithm [23]. They parallelize three sequential algorithms using PhSIH, namely TAMA [10], OpIndex [11] and REIN [12]. To achieve a good parallelization effect, the sequential matching algorithm should have three characteristics. First, the workload of matching an event can be divided into sub-tasks from a data structure perspective. Second, the workload of the sub-tasks should be uniform, and the existence of a few dominant sub-tasks should be avoided. Third, the synchronization cost between sub-tasks should be small.

### B. Parallel Matching Algorithms

Since most matching algorithms are executed sequentially, they cannot effectively utilize the parallel computing power of the hardware. Taking advantage of hardware development, such as multi-core CPUs, FPGAs and GPUs, some parallel matching algorithms have been proposed [24] [25] [26]. These algorithms typically parallelize event matching using a divide-and-conquer strategy, i.e. dividing the entire subscription set into subsets and building data structures on these subsets. While the divide-and-conquer approach is straightforward, it has limitations in flexibility and memory consumption.

The composite matching framework called COMAT is similar to our work [16]. In COMAT, each subscription is maintained in the data structures of multiple algorithms. When matching events, the matching time of all algorithms is estimated, and the optimal one is selected for event matching. COMAT can take advantage of different algorithms, but it has two limitations. First, each subscription is stored multiple times, which is not memory efficient. Second, in each algorithm, the predicate matching probability is not considered to optimize performance and stability.

Different from existing solutions, our work considers the effect of predicate matching probability on algorithm performance and stability. Considering the complementary behavior of different algorithms on predicate matching probability, subscriptions are maintained in the appropriate algorithm of PEM to take full advantage of the algorithm. Therefore, PEM can effectively improve and stabilize matching performance on the basis of existing algorithms.

## III. Design of PEM

### A. Overview

Obviously, our goal is to improve the matching performance for content-based pub/sub systems. Building on existing work, the design of the PEM framework is inspired by the idea of ensemble learning [27]. First, PEM uses a variety of algorithms to perform event matching. Subscriptions are classified according to their matching probability and assigned to the appropriate algorithm. Second, since subscription classification implies data parallelism, PEM allocates a thread to each algorithm to achieve parallel matching. The combination of subscription classification and parallel matching can greatly facilitate and stabilize matching performance.

*1) Subscription Classification:* As discussed in the work [15], most matching algorithms suffer from predicate matching probabilities. Subscriptions often contain multiple predicates with different matching probabilities. However, almost all existing matching algorithms treat predicates in their underlying data structures in the same way, regardless of the difference in matching probabilities. Intuitively, for subscriptions containing predicates with low matching-probability, it is efficient to use forward matching methods, such as TAMA [10] and OpIndex [11]. On the other hand, it is more efficient to use backward matching methods such as REIN [12] and GEM [28] to handle subscriptions with high matching probability. Therefore, we explore the idea of leveraging multiple algorithms with complementary behaviors in matching probability to improve performance.

*2) Parallel Matching:* Most existing matching algorithms are single-threaded. To speed up event matching, we can continuously optimize the performance of the single-threaded matching algorithm, but this performance improvement is generally difficult. With the development of computer hardware, multi-core CPUs and GPUs allow us to consider parallel matching, which has great potential to further improve matching performance. Therefore, we propose PEM based on subscription classification and multi-thread matching. PEM can greatly reduce the time to match events. In addition, PEM is beneficial to maintain the scalability of subscriptions and the stability of event matching under large-scale data.

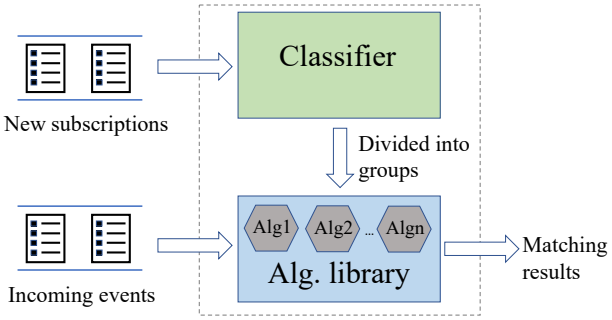


Fig. 1: The architecture of Ensemble Matching.

### B. The Architecture of PEM

The architecture of PEM is shown in Fig.1, which consists of two modules: algorithm library and classifier. The library consists of multiple algorithms with complementary behaviors in terms of predicate matching probabilities. For each new subscription, the classifier estimates its matching probability and chooses the most appropriate algorithm to insert. Therefore, each subscription is maintained by an optimal matching algorithm. When a new event arrives, all algorithms in the library are executed in parallel and their matching results are aggregated.

*1) Algorithm Library:*

The basis for implementing PEM is to build a library of multiple matching algorithms. To do so, we give three criteria for choosing matching algorithms.

i) The algorithms in the library should complement each other. In other words, algorithms should have different performance behaviors in terms of predicate matching probability.

ii) Candidate algorithms should have similar overall matching performance, aiming to achieve good ensemble effects.

iii) For better generality, the algorithms in the library should support different subscription data models.

*2) Classifier:*

*a) Quantification of Predicate Matching Probability:* The matching probability of predicates in a subscription can be estimated by the average width of interval predicates and the number of predicates. An interval predicate has a low value and a high value that forms an interval. Other forms of predicates can be transformed to interval ones. For simplicity, we assume that events are uniformly distributed. Given a subscription containing $K$ interval predicates, the average matching probability of the predicates can be estimated by

$$p = \frac{\sum_{i=1}^{K} w_i}{K} \tag{1}$$

where $w_i$ is the width of the $i^{th}$ interval predicate.

If the distribution of events is statistically available, the average matching probability of the predicates in the subscription can be calculated by

$$p = \frac{\sum_{i=1}^{K} \int_{l_i}^{h_i} p_e(x)\,dx}{K} \tag{2}$$

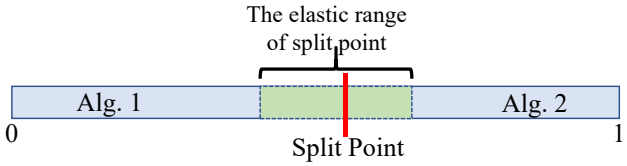


Fig. 2: Elastic classification of subscriptions considering algorithm characteristics and performance balance

where $l_i$ and $h_i$ represent the low value and high value of the $i^{th}$ interval predicate respectively, and $p_e(x)$ is the probability density function of events.

*b) Elastic Subscription Classification Method:* When designing a subscription classification method for PEM, two points need to be considered for good parallelism. First, the correspondence between algorithm characteristics and subscription probabilities should be considered to assign subscriptions to appropriate algorithms. Second, since each algorithm runs using a single thread, skewed subscriptions can cause severe performance imbalances. Therefore, we propose an elastic subscription classification method.

For simplicity of discussion, we normalize the value domain of predicate matching probability in [0, 1]. Let $L$ be the number of algorithms in the library. Since the algorithms in the library have complementary behaviors in terms of predicate matching probability, each algorithm has a range of applicable probability. The range of all algorithms collectively covers [0,1]. For example, assuming that the $i$th algorithm is optimal in the range [0.4, 0.5] and a subscription has a matching probability of 0.45, the subscription is assigned to the most suitable $i$th algorithm.

Given the $L$ algorithms in the library, we need to compute $L-1$ split points $SP_i$ $(1 \le i \le L-1)$ to separate the algorithms. $SP_i$ represents the split point between algorithm $i$ and algorithm $i+1$. The value of $SP$ indicates a matching probability in [0, 1]. Considering algorithm characteristics and performance balance, the split point $SP_i$ can fluctuate within a certain range $[R_{iMIN}, R_{iMAX}]$, which is called the elastic range. Specifically, $SP_i$ can be calculated by:

$$SP_i = R_{i_{MAX}} - \frac{t_i \times (R_{i_{MAX}} - R_{i_{MIN}})}{T} \tag{3}$$

where $t_i$ $(1 \le i \le L)$ is the matching time of algorithm $i$, $T = \sum_{i=1}^{L} t_i$ is the sum of the matching time of all algorithms, and $R_{i_{MAX}}$ and $R_{i_{MIN}}$ represent the elastic range of $SP_i$. Both $R_{i_{MAX}}$ and $R_{i_{MIN}}$ can be in the range of 0 to 1 and $R_{i_{MIN}} < R_{i_{MAX}}$. The initial value of $SP_i$ is determined according to the characteristics of each algorithm in the library. The value of $SP_i$ fluctuates within the elastic range $[SP_i \times (1-\alpha), SP_i \times (1+\alpha)]$. The value of $\alpha$ is set to 0.2 in the implementation.

We use the case of $L = 2$ to illustrate the concepts of split point and elastic range, as shown in Fig. 2. These two algorithms are suitable for subscriptions with different matching probabilities, where algorithm 1 (Alg. 1) is suitable for subscriptions with low matching probability, while algorithm 2 (Alg. 2) is suitable for subscriptions with high matching

**Algorithm 1:** Matching procedure of PEM

**Require:** an events $e$ and the event window size $\psi$.

```
j++;
Match e using all algorithms in the library;
Aggregate the output of each algorithms to obtain the matching results;
if j = ψ then
    j = 0;
    Compute the average matching time t_i of each algorithms in the current window;
    Adjust the value of all split points SP_i in the next window according to Eq. (3);
end if
```

probability. The value of $SP$ can fluctuate within an elastic range to maintain the performance of the two threads running the two algorithms separately.

The insertion process of PEM is straightforward. Given a new subscription, its matching probability $p$ is first quantified according to Eq. (2). Then, the first split point $SP_i > p$ is found. The subscription is assigned to the $i$th algorithm and maintained in the corresponding data structure.

*3) Matching Procedure of PEM:* The matching procedure of PEM is shown in Algorithm 1. For each event, all algorithms in the library are used for matching, and their outputs are aggregated to obtain matching results. After matching $\psi$ events in a time window, PEM adjusts the value of $SP_i$ according to Eq. (3), aiming to maintain the performance balance among all algorithms based on the matching time of each algorithm in the library.

## IV. Implementation

Similar to COMAT [16], we chose REIN [12] and TAMA [10] in the implementation to form the library. The two algorithms have similar overall performance and exhibit opposite behavior in terms of predicate matching probability. When matching events, TAMA uses a counter to record the predicate fulfillment condition for each subscription, while REIN uses a bitset to mark all unmatched subscriptions. Given an event, if a predicate evaluates to true, the counters of all subscriptions that contain the predicate are incremented by 1 in TAMA. Conversely, if a predicate evaluates to false, the bits representing all subscriptions that contain the predicate are marked in REIN. Therefore, REIN can achieve higher performance if more predicates evaluate to true, while TAMA may have lower performance, and vice versa.

Each algorithm in PEM is assigned to a thread. In the implementation, we need to balance the running time of REIN and TAMA. Since the split point of PEM is the criterion for subscription classification between REIN and TAMA, we design a dynamic feedback mechanism to maintain performance. After $\psi$ events are matched in each window, the split point is adjusted according to the matching time ratio of REIN and TAMA. In the next window, the adjusted split point is applied. $\psi$ is set to a small value of 20 in the implementation because the adjustment cost is almost negligible. Frequent tuning ensures that the two threads always reach a performance balance.

TABLE I: Parameters used in the experiments

| Note | Description | Values |
|---|---|---|
| $N$ | Number of subscriptions | **1M**, 2M, 3M, 4M,5M |
| $M$ | Event size | **20**, 50, 100 |
| $K$ | Subscription size | 5, **10**, 15, 20 |
| $W$ | Width of interval predicates | 0.1, 0.3, **0.5**, 0.7, 0.9 |
| $SP$ | Split point of PEM | 0.4, **0.5**, 0.6 |
| $\psi$ | Window size | 20 |

## V. Experiments

### A. Experiment Setup

All the experiments were conducted on a server with 16 2.3GHz vCPUs and 32GB RAM, which runs Ubuntu 18.04 with Linux kernel 4.15.0-111. All code is written in the C++ language and compiled by g++ with version 7.5.0 and -O3 optimization.

We compare PEM with three baselines: COMAT [16], TAMA [10] and REIN [12]. According to the papers, the discretization level of TAMA is set to 17, and the number of buckets in REIN is set to 1000. The neural networks used in COMAT are implemented by the TensorFlow library with version 2.2.0-rc1. The matching time of COMAT is the sum of the matching time of the algorithm selected from REIN and TAMA and the prediction time of the neutral networks. The matching time of PEM is the time to execute REIN and TAMA in parallel using two threads. We run each experiment 10 times and obtain the average result.

In the experiments, by default, the number of subscriptions $N$ is set to 1 million, the width of predicates $W$ to 0.5, the number of predicates in subscriptions (subscription size) $K$ to 10, and the number of attribute-value pairs in events (event size) $M$ to 20. Subscriptions and events are randomly generated based on $W$, $K$ and $M$. In each experiment, 500,000 events are matched and the average matching time is calculated. With large-scale datasets, we can better compare the performance of the tested matching algorithms. Table I lists the parameters used in the experiments. Bold values represent default settings.

### B. Matching Performance

As shown in Fig. 3 and Fig. 4, PEM always performs best with different numbers of subscriptions $N$, followed by COMAT. TAMA appears to be more sensitive to $N$ in terms of matching time and the standard deviation (Std) of matching time. From Fig. 3, we can see that TAMA outperforms REIN when the number of subscriptions is less than 3 million. REIN outperforms TAMA when the number of subscriptions becomes larger. When $N$ = 1M, PEM achieves 43.7%, 68.6% and 46.9% performance improvements over COMAT, REIN and TAMA respectively on matching time. When $N$ = 5M,

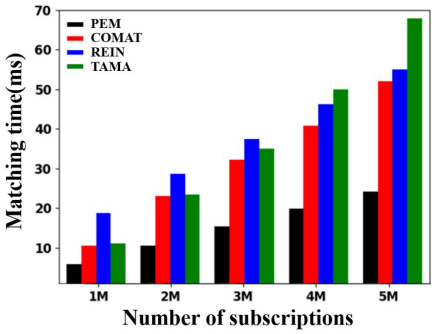


Fig. 3: Matching time with different numbers of subscriptions $N$

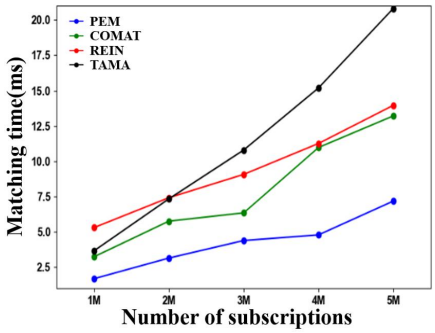


Fig. 4: Std of matching time with different numbers of subscriptions $N$

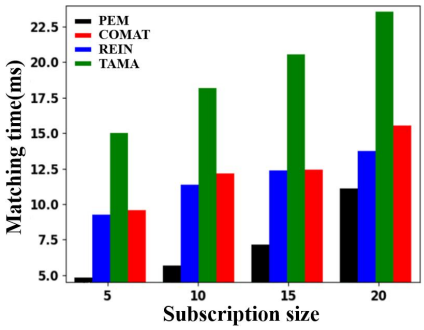


Fig. 5: Matching time with different subscription sizes $K$

the performance improvement of PEM over COMAT, REIN and TAMA is 53.3%, 55.8% and 64.2% respectively. Overall, PEM is around 40% faster than the second-best algorithm in all the experiments. In addition, PEM also has the lowest standard deviation of matching time, as shown in Fig. 4, which indicates that PEM has good performance stability.

In addition to $N$, the performance of the matching algorithm is also affected by several parameters, including $K$, $M$, and $W$. In the following subsections, we vary the settings of these parameters and evaluate their impact on the matching performance of PEM, COMAT, REIN and TAMA.

*1) Effect of Subscription Size $K$:* Figure 5 shows the effect of $K$ on matching time. In the experiments, the subscription size is randomly generated in the range [1, $K$]. A larger $K$ means that more predicates are stored in the structure of REIN and TAMA, and subscriptions have a lower matching probability. REIN is more susceptible to an increase in $K$ as it needs to repeatedly mark more subscriptions as the matching probability of subscriptions decreases. When $K = 5$, PEM is 47.7%, 67.7% and 49.4% higher that COMAT, REIN and TAMA respectively. When $K = 20$, the improvement of PEM over COMAT, REIN and TAMA is 19.3%, 52.9% and 25.87% respectively. The performance of COMAT improves when $K$ increases. This is because COMAT chooses TAMA more frequently than REIN as the number of predicates contained in each subscription increases. TAMA is better suited to subscriptions with a large number of predicates, reducing the performance difference between COMAT and PEM.

*2) Effect of Event Size $M$:* In this experiment, we set $M$ from 20 to 100 and the results are shown in Fig. 6. A larger $M$ means that there are more attribute-value pairs in events that the matching algorithm needs to process. We can find that REIN is more affected as $M$ increases. This shows that under uniform distribution, REIN is more sensitive to event size. When $M = 20$, PEM achieves an improvement of 39.1%, 67.5% and 54.1% over COMAT, REIN and TAMA respectively. When $M = 100$, the improvement of PEM over COMAT, REIN and TAMA is 17.6%, 72.6% and 23.7% respectively.

*3) Effect of the Width of Predicates $W$:* The width of predicates is a key parameter that directly affects the performance of REIN and TAMA. REIN performs well at a wider width, while TAMA does the opposite. As shown in Fig. 7, when $W = 0.1$, PEM improves by -2.2%, 80.9% and 4.5% over COMAT, REIN and TAMA respectively. In this case, PEM, COMAT and TAMA perform almost identically, with COMAT slightly better than PEM. TAMA's performance is much better REIN. Conversely, when $W = 0.9$, PEM improves by 12.4%, 13.6% and 47.1% over COMAT, REIN and TAMA respectively. In this case, REIN significantly outperforms TAMA. When $W = 0.1$, almost all subscriptions are more suitable for TAMA. So, the two threads of PEM are unbalanced, and the results of PEM, COMAT and TAMA are very close. The situation is similar when $w = 0.9$. The results of PEM, COMAT and REIN are very close. When $W = 0.5$, PEM still outperforms the three baselines significantly. We believe that the performance of PEM is generally optimal when the predicate width varies.

### C. Effectiveness of Feedback-based Adjustment Method

As shown in Fig. 8, the strategy of dynamically adjusting the split point consistently achieves the best performance when increasing the number of subscriptions. When $N = 200,000$, this strategy achieves 50.5%, 27.3%and 36.4% improvement over SP=0.4, SP=0.5 and SP=0.6 respectively. When $N = 1000,000$, the strategy achieves 31.2%, 6.6%and 20.8% improvement over SP=0.4, SP=0.5 and SP=0.6 respectively. We can see that the performance of SP=0.5 is better than SP=0.4 and SP=0.6. This is because the data is evenly distributed, with roughly an equal amount of data on both sides of 0.5. But the performance of dynamical adjustment is still better than SP=0.5, which reflects the effectiveness of the feedback-based update method in PEM.

## VI. Discussion

The time it takes to insert a subscription in PEM is the sum of the subscription insertion time. Only one insertion is required per subscription in PEM. In our implementation, since the insertion time of REIN is much smaller than that of TAMA, the insertion time of PEM is close to that of TAMA. Subscription deletion is similar to subscription insertion.

Ideally, the dual-threaded parallel algorithm has a performance improvement of 100% compared to the single-threaded algorithm. In practice, we cannot have the exact same running

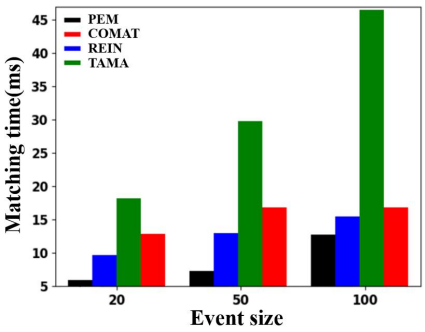


Fig. 6: Matching time with a different total number of attributes $M$.

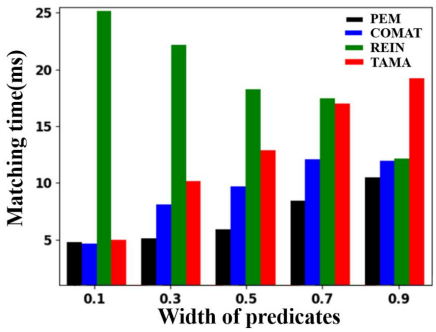


Fig. 7: Matching time with a different width of predicates $W$.

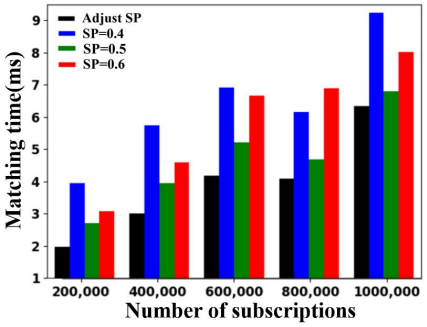


Fig. 8: Matching time with different split points.

time for both algorithms, and we also need time to classify the subscriptions. These factors take some time. In general, the performance improvement of PEM is still very significant.

## VII. Conclusion

In this paper, we explore the idea of using multiple matching algorithms to improve and stabilize the performance of matching algorithms. The main idea of PEM is to classify subscriptions, choose the most appropriate algorithm for each subscription, and then perform parallel matching on multiple algorithms. To evaluate the performance of PEM, we conducted a series of experiments. The experiment results show that PEM can greatly improve and stabilize the matching performance under different parameter settings.

## Acknowledgments

This work was supported by the National Key Research and Development Program of China (2019YFB1704400) and the National Natural Science Foundation of China (61772334).

## References

[1] S. Qian, W. Mao, J. Cao, F. Le Mouël, and M. Li, "Adjusting matching algorithm to adapt to workload fluctuations in content-based publish/subscribe systems," in *IEEE INFOCOM*, 2019, pp. 1936–1944.

[2] 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 IPDPS*, 2020, pp. 148–157.

[3] N. Dasanayaka, C. Wang, D. Jayalath, and Y. Feng, "Publish-subscribe communications for V2I safety applications in intelligent transportation systems," in *IEEE VTC Fall*, 2019, pp. 1–6.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems (TOCS)*, vol. 19, no. 3, pp. 332–383, 2001.

[5] 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.

[6] 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, 2021.

[7] 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, 2015.

[8] S. Ji and H. Jacobsen, "A-tree: A dynamic data structure for efficiently indexing arbitrary boolean expressions," in *ACM SIGMOD*, 2021, pp. 817–829.

[9] M. Sadoghi and H.-A. Jacobsen, "Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space," in *ACM SIGMOD*, 2011, pp. 637–648.

[10] Y. Zhao and J. Wu, "Towards approximate event processing in a large-scale content-based network," in *IEEE ICDCS*, 2011, pp. 790–799.

[11] 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.

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

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

[14] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *ACM DEBS*, 2008, pp. 71–81.

[15] S. Qian, J. Cao, W. Mao, Y. Zhu, J. Yu, M. Li, and J. Wang, "A fast and anti-matchability matching algorithm for content-based publish/subscribe systems," *Computer Networks*, vol. 149, pp. 213–225, 2019.

[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 ISPA*, 2020, pp. 236–243.

[17] W. Shi and S. Qian, "HEM: A hardware-aware event matching algorithm for content-based pub/sub systems," in *DASFAA*, 2022, pp. 277–292.

[18] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *ACM PODC*, 1999, pp. 53–61.

[19] M. Sadoghi and H.-A. Jacobsen, "Analysis and optimization for boolean expression indexing," *ACM Transactions on Database Systems (TODS)*, vol. 38, no. 2, pp. 1–47, 2013.

[20] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *ACM SIGCOMM*, 2003, pp. 163–174.

[21] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "Efficient filtering in publish-subscribe systems using binary decision diagrams," in *IEEE ICSE*, 2001, pp. 443–452.

[22] G. Li, S. Hou, and H.-A. Jacobsen, "A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams," in *IEEE ICDCS*, 2005, pp. 447–457.

[23] 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 *ICPP*, 2019, pp. 21:1–21:10.

[24] A. Farroukh, E. Ferzli, N. Tajuddin, and H.-A. Jacobsen, "Parallel event processing for content-based publish/subscribe systems," in *ACM DEBS*, 2009, pp. 1–4.

[25] K. Tsakalozos, M. Tsangaris, and A. Delis, "Using the graphics processor unit to realize data streaming operations," in *ACM Middleware Doctoral Symposium*, 2009, pp. 1–6.

[26] A. Margara and G. Cugola, "High-performance publish-subscribe matching using parallel hardware," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 1, pp. 126–135, 2014.

[27] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, pp. 5839–5847, 2018.

[28] W. Fan, Y. Liu, and B. Tang, "Gem: An analytic geometrical approach to fast event matching for multi-dimensional content-based publish/subscribe services," in *IEEE INFOCOM*, 2016, pp. 1–9.