Adjusting Matching Algorithm to Adapt to Workload Fluctuations in Content-based Publish/Subscribe Systems

Shiyou Qian[1,2], Weichao Mao[1,2], Jian Cao[1,2*], Frédéric Le Mouël[3], and Minglu Li[1]
1 Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
2 Shanghai Institute for Advanced Communication and Data Science, Shanghai Jiao Tong University, Shanghai, China
3 CITI-INRIA Lab, INSA-Lyon, University of Lyon
** Corresponding Author*
{*qshiyou, maoweichao, cao-jian, mlli*}*@sjtu.edu.cn*, frederic.le-mouel@insa-lyon.fr

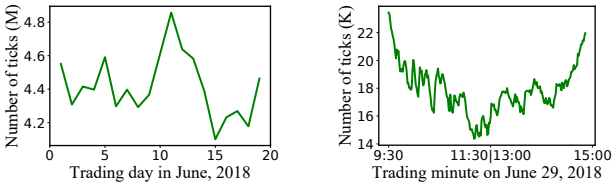
***Abstract*—When facing fluctuating workloads, can the performance of matching algorithms in a content-based publish/subscribe system be adjusted to adapt to the workloads? In this paper, we explore the idea of endowing matching algorithms with adaptability. The prerequisite for adaptability is to enable the matching algorithm to possess the ability to dynamically and quantitatively adjust its performance. We propose PSAM, a Predicate-Skipping Adjustment Mechanism that realizes dynamic performance adjustment by smoothly switching between exact matching and approximate matching, following the strategy of trading off matching precision in favor of matching speed. The PSAM mechanism is integrated into an existing matching algorithm, resulting in a performance-adjustable matching algorithm called Ada-Rein. To collaborate with Ada-Rein, we design PADA, a Performance Adjustment Decision Algorithm that is able to make proper performance adjustment plans in the presence of fluctuating workloads. The effectiveness of Ada-Rein and PADA is evaluated through a series of experiments based on both synthetic data and real-world stock traces. Experiment results show that adjusting the performance of Ada-Rein at the price of a small false positive rate, less than 0.1%, can shorten event latency by almost 2.1 times, which well demonstrates the feasibility of our exploratory idea.**

***Index Terms*—Pub/sub, workload, adaptability, adjustability**

I. Introduction

The publish/subscribe (pub/sub) system is widely deployed in many domains to disseminate high volumes of events from the sources (publishers) to the interested users (subscribers) [1]. To provide scalability, an overlay topology composed of multiple servers (brokers) is usually constructed as the middleware to provide an event dissemination service, with each broker performing event matching. Whenever a broker receives an event, it is first matched with the subscribers' interests (subscriptions) registered at the broker. According to the matching results, the event is forwarded to the next-hop neighbors (brokers or subscribers) or discarded directly.

Although a content-based pub/sub system provides subscribers with fine-grained expressiveness, the operation of event matching is costly and prone to be a performance bottleneck in some dynamic environments, such as social networks [2] and stock exchanges [3], which witness workload fluctuations over time. Specifically, when the arrival rate of events is higher than the matching rate of a matching algorithm running at a broker, events will be congested at the broker, which greatly affects the transfer latency of events.



(a) Daily distribution (b) Minute distribution

Fig. 1. The rate of producing ticks changes over time.

A typical scenario of such a dynamic content-based pub/sub system is the dissemination of real-time stock ticks. To guarantee latency requirements, stock exchanges send real-time ticks, while subscribers register their interest in ticks related to their investment strategies, e.g., when the price for a given stock goes over or under a specified threshold. In such a setting, the rate at which ticks are produced depends on the activities of the stock exchange and exhibits a dynamic property which manifests on two levels: 1) the total number of ticks varies from day to day. For example, Fig. 1(a) shows the daily number of ticks of the Shanghai Stock Exchange for 19 trading days in June, 2018, with fluctuations up to 18.4%; and 2) the rate at which ticks are produced also changes at various times in a day. Fig. 1(b) shows the rate at which ticks are produced minute-by-minute on June 29, 2018, with fluctuations up to 63.2% during the trading periods (9:30-11:30 and 13:00-15:00).

A common solution to this problem is to use a provisioning technique to adjust the number of brokers to adapt to the churn workloads [3] [4] [5]. However, this solution has several limitations. First, the proper provisioning of brokers depends on the accurate prediction of workload. However, the workload experienced by a pub/sub system is by nature difficult to predict because the rate at which events can be produced varies significantly over time, as shown in Fig. 1(b). Second, as discussed in [5], adjusting the number of brokers will take

tens of seconds, which may not satisfy the strict timeliness requirements in stock market scenarios. Over-provisioning adequate brokers in advance is simple but not cost-efficient.

The root cause of performance bottleneck is the time-consuming event matching operation performed by brokers. Naturally, one question is, can the performance of matching algorithms be adjusted to adapt to workload fluctuations to a certain degree and save critical preparation time for broker provisioning? In general, almost every matching algorithm has its own parameters that can be configured to optimize its performance. For example, the discretization lever of Tama can be configured to tradeoff between matching speed and matching precision [6]. The cache size of Gen is adjustable to balance between matching speed and storage cost [7]. Nevertheless, most existing matching algorithms lack the adaptability to dynamically respond to changing workloads.

In this paper, we explore the idea of endowing matching algorithm with this adaptability, trying to address workload fluctuations from the perspective of matching algorithms. To realize this idea, the matching algorithm should first possess adjustability which implies the algorithm can adjust its performance dynamically and quantitatively. This adjustability is the prerequisite to achieving adaptability. Furthermore, a decision algorithm is required to make proper adjustment plans for the matching algorithm in line with the fluctuating workloads. The combination of adjustability and decision-making gives birth to the desired adaptability which can deal with workload fluctuations.

For adjustability, we propose PSAM, a Predicate-Skipping Adjustment Mechanism that adopts the strategy to switch between exact matching and approximate matching in order to realize instantaneous performance adjustment by trading off matching precision in favor of matching speed. Specifically, we deduce two models that quantify the relationship between matching speed and matching precision. As for decision-making, we design PADA, a Performance Adjustment Decision Algorithm that is able to recommend proper performance adjustment plans for matching algorithms.

The PSAM mechanism is integrated into an existing matching algorithm Rein [8], giving rise to a variant named Ada-Rein. To evaluate the effectiveness of Ada-Rein, a series of experiments are conducted on a single machine using synthetic data. In addition, a test bed is set up to verify the adjustment effect of PADA in terms of event transfer latency based on real-world stock tick traces. Experiment results show that adjusting the performance of Ada-Rein at the price of a small false positive rate, less than 0.1%, can shorten event latency by almost 2.1 times, which well demonstrates the feasibility of our exploratory idea.

The main contributions of this paper are as follows:

- We put forward the idea of enhancing matching algorithms with adaptability in order to instantaneously respond to changing workloads.
- We propose a predicate-skipping adjustment mechanism (PSAM) and a performance adjustment decision algorithm (PADA) to realize this adaptability.
- We integrate PSAM and PADA into an existing algorithm and conduct a series of experiments to evaluate the adjustment effectiveness.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III describes the whole framework. Section IV details the design and evaluation of PSAM. Section V presents PADA. Section VI details the evaluation results on a test bed. Finally, Section VII concludes the paper.

II. Related Work

In this section, we review the related work from two aspects: parametric matching algorithms and techniques dealing with fluctuating workloads.

A. Parametric Matching Algorithms

Since the performance of matching algorithms is critical to a content-based pub/sub system, many efforts have been devoted to this research area, resulting in a large number of efficient matching algorithms, such as H-Tree [9], Be-Tree [10], Gryphon [11], OpIndex [12], Tama [6], Siena [13], K-Index [14], Rein [8], DEXIN [15], Gem [7] and Mics [16]. Generally, almost every matching algorithm has its own parameters that can be configured to optimize its performance. For example, the discretization lever of Tama can be configured to tradeoff between matching speed and matching precision [6]. The number of segments split on the attributes' space is an optimization parameter for OpIndex [12]. The number of indexed attributes and the number of cells divided on the attributes' value domain are two key parameters for H-Tree [9]. The cache size of Gem is adjustable to balance between matching speed and storage cost [7].

Nevertheless, there are two shortcomings in relation to the configurability of the existing matching algorithms. First, the configuration of most existing matching algorithms can not be performed dynamically. For example, when needing to adjust the matching precision of Tama [6], the underlying data structure will be reconstructed. Second, none of the existing matching algorithms has a parameter that can be used to quantitatively adjust the matching performance. To overcome these shortcomings, we propose an adjustment mechanism that can realize dynamic and quantitative performance adjustment.

B. Techniques Addressing Fluctuating Workloads

Many researchers explore churn workloads from the perspective of the overlay topology, by either provisioning more brokers or reconstructing the overlay. The work in [5] proposes GSEC, a general scalable and elastic pub/sub service based on the cloud computing environment, which uses a performance-aware provisioning technique to adjust the number of brokers to adapt to churn workloads. The works similar to GSEC include CAPS [4] and E-STREAMHUB [3]. In [17], a distributed system called OMen is proposed for dynamically maintaining overlays for topic-based pub/sub systems, which supports the churn-resistant construction of topic connected overlays. Other works utilize a load balancing strategy to deal

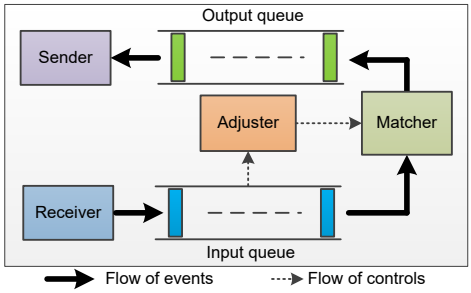


Fig. 2. The component framework of a broker.

with the workload churn problem through consistent hashing [18] [19].

There are two limitations in relation to adjusting the provisioning of brokers to adapt to churn workloads. First, the proper provisioning of brokers depends on the accurate prediction of workloads which is by nature difficult to predict in a pub/sub system since the rate at which events are produced can vary significantly over time. A typical example is stock ticks. Second, this kind of method can not satisfy the strict timeliness requirements of some applications. For example, as discussed in [5], adjusting the capacity of GSEC will take tens of seconds to adapt to the churn workloads. Although over-provisioning adequate brokers in advance can be a simple solution, it is not cost-efficient.

Differing from these works, we address this problem from the perspective of matching algorithms. The solution proposed in this paper to deal with fluctuating workloads is not to replace the existing provisioning techniques, but is a beneficial complement.

III. Whole Framework

A broker needs three components to collaboratively undertake event matching, namely a receiver, matcher and sender, as shown in Fig. 2. The receiver is responsible for receiving incoming events from upstream peers and injecting them into the input queue, and the sender is in charge of transmitting events from the output queue to the next-hops (brokers or subscribers). The matcher runs a matching algorithm that performs selective event filtering according to the subscriptions and is the connector between the input and the output queue. Please note, to obtain high throughput and low latency, there might be multiple instances running at the same time for all or part of these three components, which is the parallelization problem and out of the scope of this paper.

Ideally, the three components should have equivalent processing capacity, so events will get through the receive-match-send procedure without any delay. However, when the event arrival rate is higher than the event matching rate, events will be congested in the input queue and the matcher becomes a performance bottleneck. To adapt to fluctuating workloads, a new component called **adjuster** is added to the framework, as shown in Fig. 2. The adjuster is responsible for making performance adjustment plans for the matcher according to the changing workloads.

To make this framework function, two points are critical. First of all, the matcher, namely the matching algorithm, should have the ability to dynamically and quantitatively adjust its performance. In addition, the adjuster should recommend proper performance adjustment plans for the matcher in line with the fluctuating workloads. In the remainder of this paper, we focus on addressing these two points, proposing a predicate skipping adjustment mechanism (PSAM) to augment the adjustability of matching algorithm and presenting a performance adjustment decision algorithm (PADA) to make performance adjustment plans for matching algorithm.

IV. Design of PSAM

In this section, we detail the design of PSAM, a dynamic and quantitative performance adjustment mechanism that adopts the predicate skipping strategy to realize a switch between exact matching and approximate matching.

A. Case Analysis

For the use case of stock event dissemination, there are 64 attributes in the Level-2 ticks and more than 1,000 attributes derived by security agencies, such as Wind [20]. These attributes can be used as references by subscribers for decision-making. In addition, there are millions of investors who subscribe to stock events. Mathematically, when something is shared among a sufficiently large set of participants, there must be a number k between 50 and 100 such that "$k\%$ is taken by $(100-k)\%$ of the participants" [21]. This phenomenon is called "Zipf distribution" which is found in many domains, such as web requests [22] and Page Rank [23]. Thus, it is reasonable to assume the attribute popularity of stock events follows the Zipf distribution.

B. Basic Idea

The adjustability of matching algorithms implies two requirements on the design of an adjustment mechanism: dynamics and quantification. First, performance adjustment can be made anytime, without interrupting the normal operation of matching algorithms. Actually, it is difficult to meet this requirement because matching algorithms usually rely on their underlying data structures which are time-consuming to adjust. Second, the mechanism should support quantitative performance adjustment. Since matching performance is algorithm specific, a test bench should be selected to realize quantification.

In existing works, some approximate algorithms are proposed to pursue a high matching performance, such as Mics [16] and Tama [6]. In terms of matching mode, matching algorithms can be classified into two categories: exact matching and approximate matching. The metrics that are used to measure matching precision are false positives and false negatives. When an unmatching subscription is reported as matching by an algorithm, a false positive occurs. When a matching subscription is not picked out by an algorithm, a false negative occurs. For exact matching algorithms, neither false

positives nor false negatives are allowed. Approximate matching is to trade off matching precision in favor of matching speed, usually allowing for a small false positive rate. When there exist false positives, subscribers may receive events that are not interesting to them.

The basic idea of designing a dynamic performance adjustment mechanism is to realize a smooth switch between exact matching and approximate matching in order to achieve instantaneous performance improvement. According to the frequency with which attributes appear in subscriptions, attributes can be categorized as either popular and infrequent. For attributes that rarely appear in subscriptions, skipping the predicates defined on these attributes is a feasible way to improve matching performance while keeping a reasonable false positive rate. The predicate skipping adjustment mechanism coincides with the idea of principal component analysis (PCA) [24]. As for the quantification requirements, the key is to quantify the relationship between the sacrificed false positive rate and the improved matching performance. We deduce two models characterizing this relationship.

C. Data Model

We follow the data model that is widely used in the existing literature [6] [7] [8] [16].

An **event** e is expressed as a conjunction of attribute-value pairs. The set of attributes appearing in e is defined as $\mathbb{A} = \{a_1, a_2, ..., a_m\}$. Each attribute appears only once in an event expression. An **interval predicate** is a condition defined on an attribute selected from $\mathbb{A}$, which is represented as a 3-tuple $\{a_i, v_1, v_2\}$ where a_i is an attribute in $\mathbb{A}$, v_1 and v_2 are bounded by the value domain of a_i, and v_1 is not larger than v_2. A **subscription** defines a user's interests in events, which is a conjunction of multiple interval predicates. Each subscription is identified by a unique identification called subID. A subscription matches an event if all the interval predicates contained in the subscription are satisfied when they are assigned the corresponding attribute values of the event.

The **matchability** of a subscription is the matching probability that it matches events. For example, when a subscription is matched with 100 events, if it matches 20 of them, then its matchability is 20%. The matchability of a predicate is the probability that it is satisfied by assigning the corresponding attribute value in an event.

D. Matching Precision Model

In this subsection, we deduce a model to quantify the relationship between the false positive rate and the number of attributes to be skipped.

Suppose there is a set of n subscriptions $\mathbb{S} = \{s_1, s_2, ..., s_n\}$. Let m be the number of attribute-value pairs in events, k be the number of predicates in each subscription, and w be the matchability of predicates. Let f_i be the appearance frequency of attribute a_i in the set of subscriptions $\mathbb{S}$ for $i = 1, 2, ..., m$. We refer to the size of a subscription s as the number of predicates contained in s, denoted by $|s| = k$.

Lemma 1: Suppose the attributes of subscriptions are uniformly selected from the set of event attributes. When X attributes are skipped in the matching process, the probability P_j that a subscription contains j predicates that are defined on the X skipped attributes is upper-bounded by $\frac{C_{m-X}^{k-j} \cdot C_X^j}{C_m^k}$ for $(0 \leq j \leq k)$.

Proof 1: Each event contains m attributes from which k attributes are selected to define the predicates contained in a subscription. This selection problem is equivalent to selecting k balls from m ones where $m - X$ balls are white and X balls are red. When balls are uniformly selected, the selecting probability of all balls, no matter white or red, is the same at the beginning. In the first case where the selected balls are returned, the selecting probability remains constant, which is $\frac{1}{m-X}$ and $\frac{1}{X}$ for white and red balls, respectively. However, in the second case where the selected balls are not returned, the selecting probability of balls decreases with each selection, and is less than when the selected balls are returned since each attribute appears only once in a subscription, which corresponds to the second case. Therefore, P_j is upper-bounded by

$$P_j \leq \frac{C_{m-X}^{k-j} \cdot C_X^j}{C_m^k} \quad (0 \leq j \leq k) \tag{1}$$

Lemma 2: Suppose the attributes of subscriptions are not uniformly selected from the set of event attributes. Given the X attributes to be skipped and their appearance frequency f_X in subscriptions, the probability P_j that a subscription contains j predicates defined on the X skipped attributes is upper-bounded by $C_k^j \cdot (\frac{f_X}{nk})^j \cdot (1 - \frac{f_X}{nk})^{k-j}$ for $(0 \leq j \leq k)$.

Proof 2: Since the frequency of the X skipped attributes is f_X, the probability of selecting a skipped attribute can be expressed by $\frac{f_X}{nk}$. Given this probability, as proved by Lemma 1, P_j is upper-bounded by

$$P_j \leq C_k^j \cdot (\frac{f_X}{nk})^j \cdot (1 - \frac{f_X}{nk})^{k-j} \quad (0 \leq j \leq k) \tag{2}$$

Lemma 3: Given P_j for $0 \leq j \leq k$ when X attributes are skipped in the event matching process, the expected size of subscriptions $\bar{k}$ is $\sum_{j=0}^{k} P_j \cdot (k - j)$.

Proof 3: When an attribute is skipped in the event matching process, if a subscription contains a predicate defined on the attribute, the real size of the subscriptions is reduced. For $0 \leq j \leq k$, the probability P_j that a subscription contains j predicates defined on the X sketched attributes is given in Equation (2) when the appearance frequency of the X skipped attributes is available. With the probability P_j $(0 \leq j \leq k)$, the expected size of subscriptions $\bar{k}$ is

$$\bar{k} = \sum_{j=0}^{k} P_j \cdot (k - j) \tag{3}$$

Lemma 4: Given $\bar{k}$, the expected false positive rate F is $\frac{w^{\bar{k}} - w^k}{w^k}$ when X attributes are skipped.

Algorithm 1 Selecting skipped attributes

Input: a false positive F and the attributes frequencies $fre[\,]$
Output: an array denoting the skipped attributes $skip[\,]$

```
Initialize skip to all False
Initialize total ← 0
for i = 1 → m do
    total ← total + fre[i]
end for
Sort fre in non-decreasing order
Initialize sum ← 0
for each attribute a in sorted order do
    sum ← sum + fre[a]
    if (total − sum)/n < k + log_w(F + 1) then
        Break
    end if
    skip[a] ← True
end for
```

Proof 4: For a subscription s_i that contains k predicates with matchability w, the matchability of s_i is w^k. When X attributes are skipped, all predicates defined on these attributes are not checked. For s_i, it may contain j $(0 \leq j \leq k)$ predicates that are defined on the skipped attributes. In other words, the size of s_i is shortened to $k-j$. Since the matchability of s_i decreases monotonically with the increase of the size of s_i, skipping the X attributes leads to increasing the matchability of s_i which is w^{k-j}. The false positive rate F_{s_i} arising from skipping the j predicates of s_i is

$$F_{s_i} = \frac{w^{k-j} - w^k}{w^k} \tag{4}$$

Given the expected size of subscriptions $\bar{k}$ as proved by Lemma 3, the expectation of false positive rate F is

$$F = \frac{w^{\bar{k}} - w^k}{w^k} \tag{5}$$

Theorem 1: Given a false positive rate F, the number of predicates that can be skipped in the event matching process is $Y = n(\log_w(F+1))$.

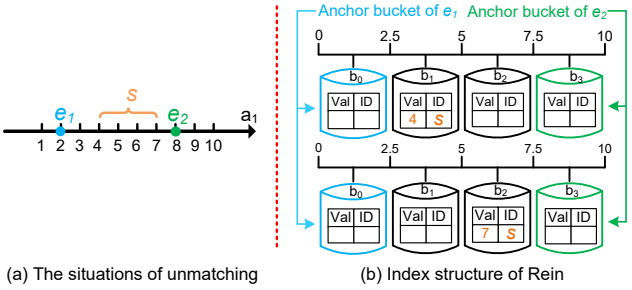
Proof 5: Substituting F into Equation (5) gets the expected subscription size $\bar{k}$ for the given false positive rate F.

$$\bar{k} = k + \log_w(F+1). \tag{6}$$

On average, $k - \bar{k}$ predicates are skipped in each subscription. The expected number of predicates that can be skipped without offending F for n subscriptions is

$$Y = n(\log_w(F+1)). \tag{7}$$

Given a false positive rate F, the number of predicates that can be skipped is computed according to Equation (7). Since the appearance frequency of attributes in the subscriptions is available, the set of X skipped attributes, $\mathbb{A}_s$ where $\mathbb{A}_s \subseteq \mathbb{A}$, can be greedily determined starting from the least frequent attributes. The pseudo code of selecting skipped attributes is shown in Algorithm 1 with time complexity $O(mlogm)$ where m is the number of attributes in events.



(a) The situations of unmatching (b) Index structure of Rein

Fig. 3. The searching strategy and index of Rein.

E. Test Bench

Given the matching precision model, the next step is to characterize the relationship between the number of skipped attributes and the improved matching performance. Since the performance measurement is specific to a matching algorithm, we select Rein [8] as a test bench to quantify this relationship for two reasons. First, the index structure of Rein is predicate-oriented, which well conforms to the spirit of our adjustment mechanism. Second, predicates defined on the same attribute are indexed together in Rein, independent from other attributes. This independence makes it possible to quantify the relationship between the matching time and matching precision, thus satisfying the quantification design requirement. To make this paper self-contained, a brief introduction to Rein is presented in this subsection.

1) Basic Idea of Rein: The basic idea behind Rein is to quickly search unmatching subscriptions to indirectly obtain the matching ones. For each attribute, situations where an interval predicate does not match an event are: i) the attribute value of the event is smaller than the low value of the predicate; ii) the attribute value of the event is larger than the high value of the predicate. These two situations are depicted in Fig. 3(a) for event $e_1 : \{a_1 = 2\}$ and $e_2 : \{a_1 = 8\}$ respectively for the predicate $s : \{4 \leq a_1 \leq 7\}$.

2) Index Structure of Rein: Two sets of buckets are constructed for each attribute by dividing the attribute's value domain into cells and mapping the cells to buckets. One set of buckets stores interval predicates with low values and the other stores interval predicates with high values, as shown in Fig. 3(b). In addition, a bit set is used to mark unmatching subscriptions.

3) Matching Procedure of Rein: The matching process of Rein can be divided into two stages: marking and checking. In the marking stage, for each attribute, the event value is mapped to the corresponding bucket (anchor bucket) in each of the two sets. A comparison operation is performed in the two anchor buckets, marking all unmatching subscriptions in the bit set. For the set of buckets storing the predicates with the low (high) values, all buckets on the right (left) side of the anchor bucket are quickly traversed to mark all unmatching subscriptions. In the checking stage, each bit in the bit set is checked, and the subscriptions represented by the unmarked bits are added to the results.

F. Matching Time Model

On the basis of the matching precision model, we further quantify the relationship between the number of skipped attributes and the rate of performance improvement based on the data structure and matching procedure of Rein.

Lemma 5: Suppose the predicate values defined on the attributes are uniformly distributed in a set of subscriptions. For an attribute a_i with frequency $f_i (0 \leq i \leq m)$, the bucket size, which denotes the average number of predicates in a bucket, is $b_i = \frac{f_i}{b}$.

Proof 6: In Rein, the number of buckets is b for each of the m event attributes. For an attribute a_i, when the predicate values defined on the attribute are uniformly distributed, these predicates are evenly stored in the b buckets, so the bucket size is $b_i = \frac{f_i}{b}$. For attributes with different frequencies, their bucket sizes are different.

Lemma 6: The matching cost of Rein is characterized by $\sum_{i=1}^{m} b_i(2\beta + b\gamma) + mb\delta + n\epsilon$.

Proof 7: The matching process of Rein consists of the marking stage and checking stage. In the marking stage, a comparison operation and traversing operation are performed. In the checking stage, each bit in the bit set is checked to determine the matching subscriptions.

Given the number of buckets b and the number of attributes m in events, let β be the unit time to compare a predicate in a bucket, γ be the unit time to traverse a predicate in a bucket, δ be the unit time to switch buckets when traversing, and ϵ be the unit time to check a bit. When matching events, for each attribute, a comparison operation is performed in two anchor buckets to find the unmatching subscriptions, thus the cost is $\sum_{i=1}^{m} b_i 2\beta$ where e_i is the bucket size of attribute a_i. After comparison, the predicates stored in the other buckets are quickly traversed, just marking the corresponding bit in the bit set of Rein. For the low values and high values of predicates stored in the buckets, the whole b buckets will be traversed, so the cost is $\sum_{i=1}^{m} b_i b\gamma$. In addition, the cost of switching the b buckets is $b\delta$ for each attribute, so the total cost is $mb\delta$. In the checking stage, each bit is checked, so the cost is $n\epsilon$. Therefore, the total matching cost of Rein can be denoted as:

$$T = \sum_{i=1}^{m} b_i(2\beta + b\gamma) + mb\delta + n\epsilon \tag{8}$$

Theorem 2: Given the set of X attributes $\mathbb{A}_s$, the matching performance improvement by skipping the X attributes in Rein is $R = \frac{\sum_{a_j \in \mathbb{A}_s} b_j(2\beta+b\gamma)+Xb\delta}{\sum_{i=1}^{m} b_i(2\beta+b\gamma)+mb\delta+n\epsilon}$.

Proof 8: Skipping X attributes in the matching process of Rein decreases the comparison cost, traversing cost and switching cost. The sum of these reduced costs is $\sum_{a_j \in \mathbb{A}_s} b_j(2\beta + b\gamma) + Xb\delta$. The total matching cost of Rein is given in Equation (8). Therefore, the matching performance improvement R obtained by skipping X attributes is

$$R = \frac{\sum_{a_j \in \mathbb{A}_s} b_j(2\beta + b\gamma) + Xb\delta}{\sum_{i=1}^{m} b_i(2\beta + b\gamma) + mb\delta + n\epsilon}. \tag{9}$$

Algorithm 2 Ada-Rein

Input: an event e and a false positive rate F
Output: $matchList$

```
Determine the skipped attributes skip by Algorithm 1
for i = 1 → m do
    if skip[i] == False then
        Perform matching like Rein
    end if
end for
for each subscription s in subscription list do
    if the bit of s not marked in the bit set then
        Add s to matchList
    end if
end for
```

TABLE I
PARAMETERS USED IN EXPERIMENTS

Note	Description	Default value
n	the number of subscriptions	1M
m	the number of events attributes	1000
α	the parameter of attributes' Zipf distribution	2
k	the number of predicates in subscriptions	5
w	the matchability of interval predicates	0.5
b	the number of buckets in Rein	500

G. Matching Procedure

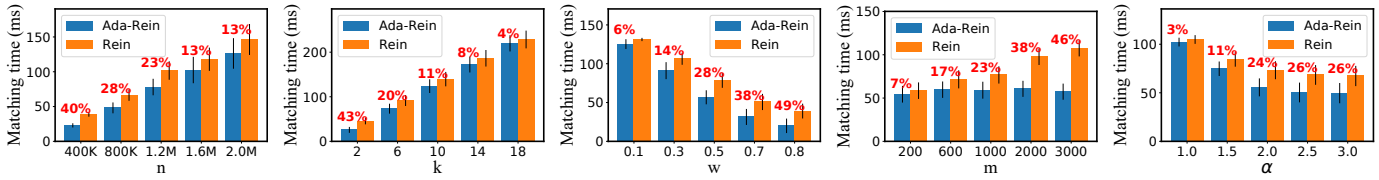
Integrating PSAM into Rein gives rise to Ada-Rein which is shown in Algorithm 2. Compared with Rein, an additional input parameter F is needed, which indicates the expected false positive rate of event matching. If F is set to 0, Ada-Rein runs as an exact matching algorithm. Otherwise, Ada-Rein works in the approximate matching mode with the specified false positive rate. The switch between exact matching and approximate matching is dynamic and seamless, without reconstructing the underlying index structure. Furthermore, different events can be matched at different false positive rates.

H. Effectiveness Evaluation

In this subsection, we evaluate the quantification effect of Ada-Rein.

1) Experiment Setup: The effectiveness of PSAM is verified on a Dell PowerEdge T710 server that has 8 2.4 GHz cores and 32 GB memory. All codes are written in the C++ language. For brevity, the value domain of attributes is normalized on [0, 1.0] from which the predicate values and event values are uniformly generated with the precision 10^{-6}. The parameters used in the experiments are listed in Table I. 1000 events are matched in each experiment and each experiment is repeated 10 times. The matching time is used to evaluate the performance of the matching algorithms. For our testing server, the value of β, γ, δ and ϵ is 205.43 ns, 119.73 ns, 145.85 ns and 6.58 ns, respectively.

2) Evaluation Results: In the experiments, Ada-Rein with different expected false positive rates is compared to Rein. For each expected false positive rate, the number of skipped predicates and attributes are counted, which are used to compute the expected rate of performance improvement based on the matching time model given in Equation (9). In addition,



(a) Number of subscriptions (b) Number of predicates (c) Matchability of predicates (d) Number of attributes (e) Attributes distribution

Fig. 4. Effect evaluation of different parameters.

TABLE II
EFFECTIVENESS VERIFICATION RESULTS

F_1	F_2	Y	X	R_1	R_2
0.01%	0.0096%	93	678	17.74%	19.69%
0.05%	0.0452%	473	816	21.33%	23.46%
0.1%	0.0984%	969	874	22.87%	25.13%
0.5%	0.481%	4,707	941	24.78%	27.74%
1%	0.91%	9,138	958	25.41%	29.18%
3%	2.67%	27,736	976	26.68%	30.31%
5%	4.53%	48,856	982	27.74%	32.91%

the number of matching subscriptions of Ada-Rein is counted in the experiment, which is compared with the one of Rein to measure the real false positive rate. The matching time of Rein is used as the baseline to measure the real rate of performance improvement of Ada-Rein. The results are listed in Table II, where $n = 1M$, $k = 4$, $m = 1000$, $w = 0.5$ and $\alpha = 2$. In the table, the six columns are expected false positive rate (F_1), measured false positive rate (F_2), number of skipped predicates (Y), number of skipped attributes (X), expected rate of performance improvement (R_1), and measured rate of performance improvement (R_2), respectively.

As shown in the table, the measured false positive rate and performance improvement are very close to the expected ones respectively, well demonstrating the effectiveness of PSAM. Specifically, the measured false positive rate is a little smaller than the expected one while the measured rate of performance improvement is not less than the expected one. The philosophy behind this is that the expected false positive rate should be an upper bound, limiting the costs; while the expected performance improvement should be a lower bound, ensuring the gains.

The performance improvement does not linearly grow with the false positive rate. This phenomenon is explainable. At the beginning, even with a small false positive rate, a certain number of infrequent attributes are skipped. Thus, the obtained performance improvement is considerable. For example, when the expected false positive rate is 0.01%, the measured rate of performance improvement is 19.69%. In this case, 678 infrequent attributes out of 1000 ones are skipped. However, increasing the false positive rate does not bring proportional performance improvement. For example, when the expected false positive rate is 0.1%, the measured rate of performance improvement is 25.13%. Although the false positive rate increases tenfold compared with 0.01%, the performance improvement only grows 27.6%. This is because by increasing the false positive rate 10 times, only 195 additional attributes are skipped without offending the expected false positive rate. Therefore, from the viewpoint of cost-effectiveness, the "Elbow Rule" can be used to determine the expected false positive rate [25]. In this experiment, Ada-Rein with a false positive rate less than 0.1% is more cost efficient. So, in the next experiments, we compare Ada-Rein of 0.1% false positive rate with Rein to evaluate the impact of different parameters.

a) Impact of the number of subscriptions: The performance improvement of Ada-Rein gradually diminishes with more subscriptions, as shown in Fig. 4(a). When $n = 400K$, the improvement is 40%; but when $n = 2M$, the improvement reduces to 13%. Fixing the number of event attributes and the number of predicates contained in the subscriptions, increasing the number of subscriptions leads to growing the frequencies of attributes, which reduces the number of infrequent attributes to be skipped while satisfying the expected false positive rate.

b) Impact of the number of predicates: With less predicates contained in the subscriptions, the benefit of Ada-rein is more apparent, as shown in Fig. 4(b). When $k = 2$, Ada-rein makes about a 43% performance improvement over Rein whereas the improvement reduces to only 4% when $k = 18$. The cause of this issue is due to the increased frequencies of attributes, as explained in the previous experiment. For such subscriptions, skipping even a small number of attributes will offend the expected false positive rate, which limits the adjustment space of Ada-Rein.

c) Impact of the matchability of predicates: Subscriptions with high matchability are beneficial for Ada-Rein, as shown in Fig. 4(c). When $w = 0.1$, Ada-Rein is faster than Rein by only 6%, however the performance improvement is up to 49% when $w = 0.8$. This behavior of Ada-Rein is related to the negative searching strategy employed by Rein which is favorable of subscriptions with high matchability.

d) Impact of the number of attributes: High-dimension space is beneficial for Ada-Rein, as depicted in Fig. 4(d). Given the total number of predicates, the number of infrequent attributes will increase with higher dimensions, providing sufficient candidates for Ada-Rein to select attributes to be skipped. When $m = 200$, Ada-Rein has trivial performance improvement over Rein; when $m = 3000$, the improvement increases to 46%.

e) Impact of attribute distribution: Ada-Rein prefers a skewed frequency distribution of attributes, as shown in Fig. 4(e). When the popular attributes are concentrated, more attributes belong to the infrequent category, which provides more candidates for Ada-Rein to skip without offending the expected false positive rate. In the case $\alpha = 1.0$, there is

only 3% performance improvement of Ada-Rein over Rein. However, when $\alpha = 3$, the improvement is up to 26%.

In summary, given the number of attributes, increasing the total number of predicates by raising either the number of subscriptions or the number of predicates contained in subscriptions will shrink the effectiveness of Ada-Rein. Given the total number of predicates, Ada-Rein has larger adjustment space by increasing the matchability of predicates, raising the number of attributes, or making the attributes more skewed.

V. Design of PADA

Given a performance adjustable matching algorithm Ada-Rein, the next thing is to design a performance adjustment decision algorithm (PADA), which is the focus of this section.

Instead of making the adjustment plan according to the congestion degree of events at brokers, we specify an upper bound on the false positive rate F_{max} by analyzing the cost-efficiency between false positives and performance improvement, for the following reasons. First, the position of our proposed solution is clear, as a complement to the broker provisioning techniques to deal with workload churn to a certain extent. Second, when the performance of the matching algorithm needs to be improved, the contention of network resources is also intense. Allowing a larger false positive rate will deteriorate the network contention.

Making adjustment plans is in essence a control problem and we borrow the spirit of PID [26] to design PADA which is detailed in Algorithm 3. The rationale of this algorithm is to make a trade-off between matching speed and matching precision, by adjusting the false positive rate F in a self-adaptive way. When the broker receives an event e (Line 2), it first calculates how long this event has been waiting in the input queue (Line 3), which reflects the congestion degree of events at the broker. If the latency is larger than the normal one $t\_threshold$ (Line 4), it can be inferred that events are congested at the broker at this moment. Furthermore, if the congestion degree of e is more serious than that of the last event, F is increased to further improve the matching performance (Line 6), otherwise keeping F constant (Line 8), where F is subjected to the upper bound F_{max} (Line 10). Similarly, when the congestion of events is alleviated, F is gradually decreased in order to achieve high matching precision (Line $11-17$). As can be seen, PADA will adaptively converge to the case where exactly no congestion occurs at the broker, and the false positive rate is minimized. The time complexity of PADA is $O(1)$ to determine a false positive rate for each event.

Algorithm 3 PADA

1: $F \leftarrow 0,\ F_{last} \leftarrow 0,\ t \leftarrow 0,\ t_{last} \leftarrow 0$;
2: **for** each incoming event e to the broker **do**
3: $\quad t \leftarrow$ the latency of e
4: $\quad$ **if** $t \geq t_{threshold}$ **then**
5: $\quad\quad$ **if** $t \geq t_{last}$ **then**
6: $\quad\quad\quad F \leftarrow F_{last} + \Delta_F$
7: $\quad\quad$ **else**
8: $\quad\quad\quad F \leftarrow F_{last}$
9: $\quad\quad$ **end if**
10: $\quad\quad F \leftarrow \min\{F, F_{max}\}$
11: $\quad$ **else**
12: $\quad\quad$ **if** $t < t_{last}$ **then**
13: $\quad\quad\quad F \leftarrow F_{last} - \Delta_F$
14: $\quad\quad$ **else**
15: $\quad\quad\quad F \leftarrow F_{last}$
16: $\quad\quad$ **end if**
17: $\quad\quad F \leftarrow \max\{F, 0\}$
18: $\quad$ **end if**
19: $\quad e$ is matched by Ada-Rein in Algorithm 2 with F
20: $\quad F_{last} \leftarrow F,\ t_{last} \leftarrow t$
21: **end for**

VI. Experiment Results

A. Experiment Setup

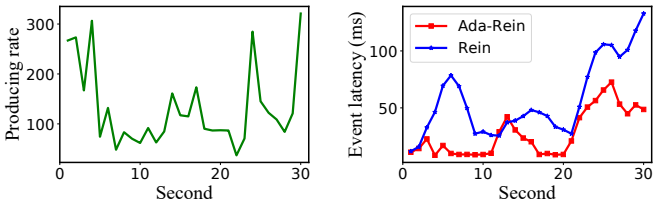
A test bed composed of three virtual machines (VMs) is set up to evaluate the adjustment effect of Ada-Rein in collaboration with PADA. Each VM has 4 vCPUs, 8GB RAM and 80GB hard drive. Two VMs are used to imitate a publisher that generates events at changing rates and to simulate a certain number of subscribers who submit subscriptions to the broker respectively. Ada-Rein and PADA are run at the third VM which acts as a broker. We modify the source code of Siena[1] to implement Ada-Rein and PADA in the pub/sub service.

B. Dataset

We collected Chinese stock market data generated on March 22, 2018 from the Wind Economic Database[2]. There are 616 attributes in the tick data from 10:00 A.M. to 11:30 A.M. which is used to generate the subscription set and event set. We set the event generation rate in our experiments to be proportional to the actual generation rate in the dataset. In this sense, the rate at which events are produced well simulates the workload fluctuations of the real-world stock exchanges.

The experiment procedure can be divided into two stages: submitting subscriptions and publishing events. In the first stage, $100,000$ subscriptions are synthesized and submitted to the broker. Each subscription contains five predicates and the matchability of predicates is set to 0.3. The selection of subscription attributes follows the Zipf distribution with $\alpha = 2$. In the second stage, the publisher generates events at a changing rate that is proportional to the real one in the stock dataset. Whenever the broker receives an event, it judges the congestion degree, determines the allowed false positive rate, matches the event with the subscription, and sends the events to the interested subscribers. We measure the overall latency of events from generation to the moment when each interested subscriber receives the events. The publish stage lasts 30 seconds and the average latency of events at each second is calculated. Since the system workload is fluctuating, we anticipate that the average latency at each second will vary with the workloads.

[1] http://www.inf.usi.ch/carzaniga/siena/
[2] http://www.wind.com.cn/en/



(a) Changing workloads (b) Event latency

Fig. 5. The changing workload and event latency.

C. Experiment Results

The changing workload is shown in Fig. 5(a). The average publishing rate is 131.17 events per second and the fluctuation degree is up to 767.57%. In the experiment, the PADA algorithm is used to determine the false positive rate for each event where $F_{max} = 0.1\%$, $\Delta_F = 0.01\%$, and $t_{threshold} = 5ms$.

The event latency calculated at each second is shown in Fig. 5(b). Overall, the average event latency of Ada-Rein is shortened by 2.09 times compared to Rein, reducing from 57.11 ms to 27.32 ms. The reason for this is that when encountering a workload spike, Rein does not have the adjustability to adapt to the changing workloads, thus events are congested at the broker, forming a cumulative effect on event latency. On the contrary, Ada-Rein has the ability to alleviate event congestion to some extent, stabilizing the latency of events. The standard deviation of the event latency of Ada-Rein is improved by almost 1.65 times compared to Rein, decreasing from 33.19 to 20.07.

VII. Conclusions

In this paper, we explore the idea of enhancing the adaptability of matching algorithms to deal with workload fluctuations. To do so, we propose a Predicate-Skipping Adjustment Mechanism (PSAM) and a Performance Adjustment Decision Algorithm (PADA). The integration of PSAM into Rein gives rise to Ada-Rein. In collaboration with PADA, Ada-Rein is able to deal with workload fluctuations to a certain extent. To evaluate the adjustment effectiveness of Ada-Rein and PADA, a series of experiments are conducted based on both synthetic data and real-world stock tick data. The experiment results show that, even after adjusting the performance of Ada-Rein at the price of a small false positive rate, less than 0.1%, event latency can be shortened by almost 2.1 times, which well demonstrates the feasibility of our exploratory idea.

Acknowledgement

This work was supported by National Key R&D Program of China (2017YFC0803700), the National Science Foundation of China (61772334, 61702151), the Joint Key Project of the National Natural Science Foundation of China (U1736207), and Shanghai Talent Development Fund, Shanghai Jiao Tong arts and science inter-project (15JCMY08)..

References

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

[2] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski, "Top-k publish-subscribe for social annotation of news," *Proceedings of the Vldb Endowment*, vol. 6, no. 6, pp. 385–396, 2013.

[3] R. Barazzutti, T. Heinze, A. Martin, E. Onica, P. Felber, C. Fetzer, Z. Jerzak, M. Pasin, and E. Rivière, "Elastic scaling of a high-throughput content-based publish/subscribe engine," in *IEEE ICDCS 2014*, pp. 567–576.

[4] X. Ma, Y. Wang, X. Pei, and F. Xu, "A cloud-assisted publish/subscribe service for time-critical dissemination of bulk content," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, pp. 1–15, 2017.

[5] Y. Wang and X. Ma, "A general scalable and elastic content-based publish/subscribe service," *IEEE Transactions on Parallel & Distributed Systems*, vol. 26, no. 8, pp. 2100–2113, 2015.

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

[7] 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.

[8] 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.

[9] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang, "H-tree: An efficient index structurefor event matching in content-basedpublish/subscribe systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1622–1632, 2015.

[10] 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.

[11] 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.

[12] 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.

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

[14] S. E. Whang, H. Garcia-Molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, and R. Yerneni, "Indexing boolean expressions," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 37–48, 2009.

[15] W. Fan, Y. A. Liu, and B. Tang, "Dexin: A fast content-based multi-attribute event matching algorithm using dynamic exclusive and inclusive methods," *Future Generation Computer Systems*, vol. 68, pp. 289–303, 2017.

[16] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "Mics: an efficient content space representation model for publish/subscribe systems," in *ACM DEBS 2009*, pp. 1–12.

[17] H. A. Jacobsen, H. A. Jacobsen, and H. A. Jacobsen, "Omen: overlay mending for topic-based publish/subscribe systems under churn," in *ACM DEBS 2016*, pp. 105–116.

[18] J. Gascon-Samson, F. P. Garcia, B. Kemme, and J. Kienzle, "Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud," in *IEEE ICDCS 2015*, pp. 486–496.

[19] A. K. Y. Cheung and H. A. Jacobsen, "Load balancing content-based publish/subscribe systems," *Acm Transactions on Computer Systems*, vol. 28, no. 4, pp. 1–55, 2010.

[20] "Wind.(2018). [online]." http://www.wind.com.cn/.

[21] G. K. Zipf, "Relative frequency as a determinant of phonetic change," *Harvard studies in classical philology*, vol. 40, pp. 1–95, 1929.

[22] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM 1999*, vol. 1, pp. 126–134.

[23] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *international world wide web conferences*, vol. 30, pp. 107–117, 1998.

[24] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989.

[25] R. L. Thorndike, "Who belongs in the family?" *Psychometrika*, vol. 18, no. 4, pp. 267–276, 1953.

[26] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.