Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

MATCH-LADDER: AN EFFICIENT EVENT MATCHING ALGORITHM IN LARGE-SCALE CONTENT-BASED PUBLISH/SUBSCRIBE SYSTEM

Menglu Xu
Pin Lv
Haibo Wang

Science and Technology on Integrated Information System Laboratory in Institute of Software Chinese Academy of Sciences, University Chinese of Academy Sciences
4# South Fourth Street, Zhong Guan Cun
Beijing, 100190, CHINA

ABSTRACT

To resolve high-performance content-based event matching problem for large-scale publish/subscribe systems, we have focused on how to use some priori knowledge to improve the efficiency. In this paper, by theoretically analyzing the inherent problem of the matching order of predicates, we propose a matching algorithm called Match-Ladder which based on the best matching order. The Match-Ladder can achieve better trade-off between time efficiency and the usage of memory space. It has been verified through both mathematical and simulation-based evaluation.

1 INTRODUCTION

Publish/subscribe is an appealing communication primitive for large-scale dynamic networks due to the loosely coupled interaction between the publishers and subscribers. In this paradigm, subscribers express their interests in data by registering subscriptions with the system, in order to be notified of any forthcoming events (issued by publishers) matching their subscriptions. The matching procedure is performed by brokers, which are also responsible for the event delivery. In this way, publishers and subscribers are completely desynchronized in time and space. There are two kinds of publish/subscribe system: topic-based and content-based. In the topic-based system, such as IBM's MQSeries (IBM 1997), each event was divided into a number of fixed topics. Each event belongs to a specific topic. A publisher must specify their domain topics in advance, while subscribers can subscribe their contents to the particular publishers. In the content-based system, due to the subscriber could provide a finer granularity to find interested events, the content-based publish/subscribe networks (CBNs) are especially useful for processing data which supplied by the publisher is constantly changing and updating a large number of subscribers quickly with the latest event data at the same time. And in recent years, CBNs are more and more extensively used in many applications, such as stock quotes, web advertisement (Fontoura et al. 2010), network monitoring system (Fawcett and Provost 1999), microblog and so on.CBNs are composed of several components, such as topological structure, event model, subscription scheme, event matching algorithm, forwarding engine and some other facilities. Among them, the highly efficient event matching is a critical component of CBNs.

So far, there are many event matching algorithms. We could divide them into two clusters according to the data structure used in the algorithms. The first one is tree-based algorithms (Aguilera et al. 1999; Gao, Li, and Zhao 2011; Gough and Smith 1995; Kale et al. 2005), the search-tree (Gough and Smith 1995) and parallel-search-tree (Aguilera et al. 1999) are the typical representatives of this algorithms. Although these algorithms may improve the matching efficiency, they are difficult to cancel the subscription in the tree structure and they have a combinatorial explosion in the number of times each

predicate is stored (Ma 2009). As a result, they are not suitable for large-scale systems. The second one is table-based (Chen et al. 2011; Xu, Lv, and Wang 2013; Xue and Jia 2013; Zhao and Wu 2011), its advantage is easy to maintain the subscriptions and the structure is very simple, but some of them only use the predicate's cover relationship to reduce the number of matching times to improve the efficiency, but ignoring other priori knowledge such as the popularity and the filter ability of the predicate which could influence the matching sequence of predicates. According to the filter ability of predicate, we have presented an event matching algorithm(PPEM) (Xu, Lv, and Wang 2013), the problem in PPEM is that it only considered the filter ability but ignored the popularity of predicate. In order to study the influence of those priori knowledge on the matching efficiency, firstly, we model the matching process and then find the method to get the best matching sequence to improve the efficiency; secondly, we propose a matching algorithm called Match-Ladder that lends itself to very efficient matching by using the best matching sequence. Our main contributions in this paper are:

- According to the filter ability and the frequent degree of one predicate, we have theoretically proved that different matching order of predicates in one subscription would affect the efficiency of the event matching process.
- A data structure which is designed to be simple while supporting the best predicate matching order and the most usual match on different kinds of events.
- An efficient matching algorithm for processing events in real time which can handle a large number of volatile subscriptions and supports high event rates (several millions per day).

The remaining sections of this paper are organized as follows. In section 2 we present the system model. Section 3 discusses the predicate matching sequence. A simple data structure and its corresponding algorithm are presented in Section 4. Some experimental results are shown in Section 5. Finally our conclusions are presented in Section 6.

2 THE PUBLISH/SUBSCRIBE MODEL

We first give some definitions used in the publish/subscribe system model.

Definition 1 (Event) An event is composed of a set of attributes $a_1, a_2, \mathrm{L}\ , a_n$, *where* a_i, *is a pair <name, value>.*

Definition 2 (Predicate) *A predicate is a triple <name, op, value>, where* $name$ *is an attribute name,* op *is an operator symbol which is used to limit the value.*

Definition 3 (Subscription) *A subscription is a conjunctive predicates, in form of* $p_1, p_2, \mathrm{L}\ , p_m$.

Definition 4 (predicate filter ability) *The filter ability PA of a predicate is associated with its operator type and value.*

The filter ability of predicate could be used to evaluate the matching cost of predicate.

Definition 5 (predicate popularity) *It is the frequent degree of a predicate that occurs in all the subscriptions (shortly for PP).*

Definition 6 (Predicate cover relation) *Predicate* p_1 *covers predicate* p_2*, if and only if for any event E matched by* p_1*, E must be matched by* p_2.

Although cover relation maintenance is rather a complex work, it's very useful for reducing the matching times. As long as we choose suitable data structure (like map in C++) to represent the cover relation, it will make more advantages than disadvantages.

Definition 7 (Matching Problem) *An event matches a subscription if all the predicates in the subscription are satisfied by the value of the corresponding attributes contained in the event.*

In the following section, we will study two problems: the first one is how the predicate filter ability and popularity could affect the matching order of predicates, the other one is whether the matching order of predicates could affect the efficiency of event matching algorithm or not.

3 THEORETICAL ANALYSIS OF MATCHING ORDER

In this section, we will discuss the matching order of predicates and analysis how to obtain a best matching efficiency by changing the matching order of predicates.

Considering a conjunctive subscription $S = \{p_1 \wedge p_2 \wedge \mathrm{L} \wedge p_n\}$, each predicate associates a corresponding PA_i and PP_i. Suppose the matching order is from left to right, we first evaluate p_1. If p_1 matches successfully, then, p_2 will be processed. We can get the matching cost of subscription S:

$$CS = pa_1 + pp_1 pp_2 pa_2 + \mathrm{L} + pa_k \prod_{i=1}^{k} pp_i + \mathrm{L} + pa_m \prod_{i=k+1}^{m} pp_i + \mathrm{L} + pa_n \prod_{i=1}^{n} pp_i$$

The matching cost uses the knowledge about the popularity and filter ability of predicates in the system. By switching the matching order of any two predicates, we have the following result:

Lemma 1 *Matching cost difference by switching the matching order of any two predicates p_k, p_m of a conjunctive subscription is*

$$(pa_k pp_k - pa_m pp_m) \prod_{i=1}^{k-1} pp_i + (pa_m \quad pa_k) \prod_{j=1}^{m} pp_j .$$

The proof of Lemma 1 is presented in the appendix. The result of Lemma 1 not only claims that the abstract math model of the matching process is independent of the concrete matching algorithm, but also indicates that the predicate matching order could affect the efficiency of any matching algorithm. Since predicate matching sequence is very important, how to obtain the best matching order? The following Theorem 1 gives the answer.

Theorem 1 *By sorting the matching order of predicates of a conjunctive subscription satisfying with increased $pa \bullet pp / (1 - pp)$, we can achieve the shortest matching time(The proof of Theorem 1 is shown in the appendix).*

In this section, we have presented that the predicate matching order could affect the matching efficiency and we also acquire the method to get the best matching order by using some priori knowledge like predicate's property. In the next section, we will present a matching algorithm which uses this matching order to improve efficiency.

4 EVENT MATCHING ALGORITHM

The Figure 1 presents the data structure used in the Match-Ladder algorithm, it mainly contains three components: Subscriptions ID list, Match ladder and Multi-index. We will introduce their functions respectively.

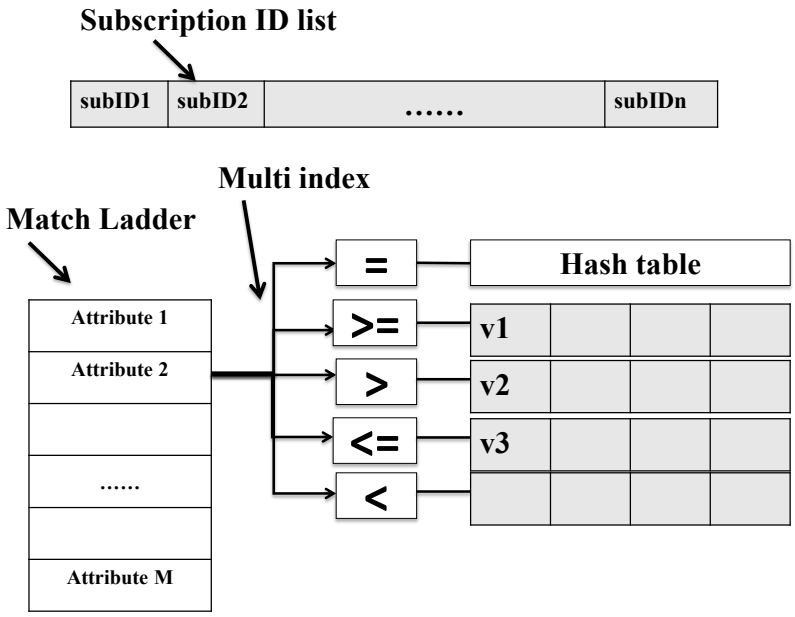

Figure 1: Data Structure of the Match-Ladder.

4.1 Data structure of Match-Ladder

4.1.1 Subscriptions ID list

The subscription IDs are stored in the list in sequence. The program creates a unique ID for each subscription automatically, which is used to identify the subscription. To maintain the subordination relation of the predicates and subscriptions, every predicate has a subID attribute to indicate that it belongs to which subscription.

What's more, due to the predicate match order, we know which predicate is the first or the last one in the subscription. This is very useful in the initial and matching phase, and we will describe this in detail in the back.

4.1.2 Match ladder

From up to down, the attribute names are increased by its match order. That is to say: $pa_1 pp_1 / (1 - pp_1) < pa_2 pp_2 / (1 - pp_2) < \mathrm{L} < pa_n pp_n / (1 \quad pp_n)$.

When the matching sequence was determined, the content in the match ladder will remain invariability. Each layer has a pointer which points to a multi-index structure to store the predicates of the same attribute name. The content in the multi-index will change all the time.

4.1.3 Multi-index structure

The predicates have the same attribute name are stored into multi-index structure according to their operator type (Xue and Jia 2013). If the comparison operator is "=", the value of the predicate will be stored into a hash table, and for other comparison operators, the predicates will be stored by the coverage relationship. This structure can guarantee that repetitive predicates are stored only once by adding a subID list that stores all the subscriptions identification that contains the predicate.

4.2 Preprocessing procedure

In order to organize the subscriptions into the data structure of the algorithm, we need a preprocessing before matching. In the beginning, all the structures are empty. The preprocessing procedure includes four steps to add each subscription into the subscriptions ID list, match ladder and the multi-index structure.

Step 1: Compute the predicate match order by its attribute's filter ability and frequent degree.

Step 2: Assign a unique ID for each subscription, and save the ID in the subscription ID list. Then, sort the predicates of the same subscription according to the order. This step assures that all the predicates of the same subscription will be matched in order and we could know which predicate is the last one in the subscription when matching with the event.

Step 3: Put all the attribute names into the match-ladder in order.

Step 4: Put the first predicate of every subscription into the multi-index structure.

Figure 2 shows the contents of the data structure during the preprocessing.

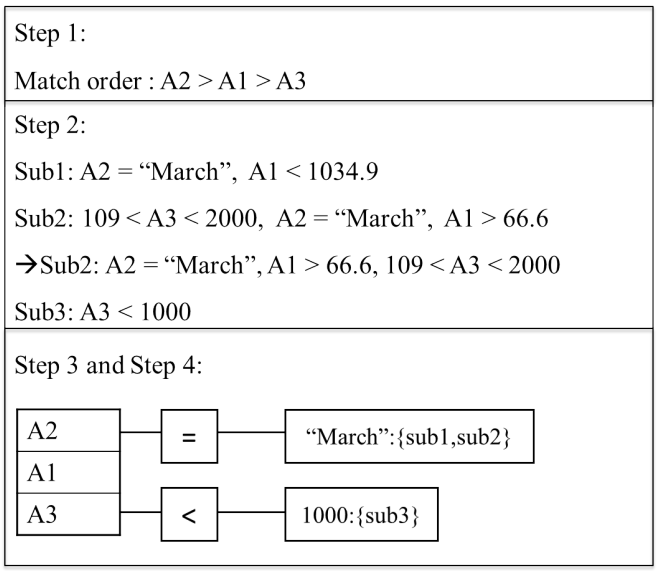

Figure 2: Example of initialize Match-Ladder.

4.3 The Match-Ladder Algorithm

After preprocessing, the first predicate of every subscription has been stored into match-ladder. For each layer in the match-ladder, we will get a multi-index structure. If the new event has the attribute name and the multi-index of this layer isn't empty, then we will follow these steps to find the subscriptions satisfying the new event:

If the operator type is equal, do the hash table lookup directly, and for other comparison operators, get the first match of the current predicate in the new event, thanks to the coverage relationship between predicates, its subsequent binding will also meet the match.

When we get the match predicates, we will judge whether the predicate is the last one in the subscription or not. If it is the last one, the subscription satisfies the new event; otherwise, we will add the next predicate of the current predicate into the corresponding layer of the match ladder according to its attribute name. The Pseudo code is showed in the following.

```
Matching algorithm based on Match-Ladder
Input: A new Event E
       Match ladder after being initialized CurMatchLadder
Output: matchSubs
```

```
matchSubs := {}
For each layer k of the Match ladder
  AttrName := the attribute name in layer k
  if E doesn't have AttrName OR the multi-index in layer k is empty then
      continue
  else
      Pe :=getPredicate(E,AttriName)
      operator op in layer k
      MatchPredicates :=MatchByOperator(predicates in layer k, op,Pe)
      For each predicate mp in MatchPredicates
          For each ID in mp.SubID list
              Subscription S :=getSub(ID)
              if mp is the last one in S then
                matchSubs.add(mp.SubID)
             else
                Predicate np :=getNextPredicate(S)
                CurMatchLadder.add(np)
          EndFor
      EndFor
EndFor
```

4.4 Performance analysis

Assuming the attribute number of an event or a subscription schema is c and the total number of subscriptions is S.

Space complexity: The space requirement of our algorithm is obvious, that is the storage for all the subscription. So the space complexity in our algorithm is $O(cS)$.

Time complexity: Because the attribute number is c (It's a very small constant), the match ladder structure has c layers. Supposing the *Match Predicates* has L predicates and the size of each matching predicate's subID list is K, the worst time complexity of our algorithm is $O(cLK)$.

5 EXPERIMENTATION

Our simulation is based on OverSim (OverSim 2014), a discrete event simulator written in C++ and a popular choice for simulating application-layer protocols in large-scale scenarios. We implemented the ferry (Zhu and Hu 2007) which is a p2p framework as the overlay of OverSim, and constructed the event-based system simulation based on it. Because the mainly task of the simulation is to evaluate the effectiveness of matching algorithms, we use other components in ferry and only modify the matching component.

5.1 Implementation and Methodology

To study the effectiveness of our matching algorithm, we have implemented three major matching algorithms in our simulation system.

1. The Match-Ladder algorithm.
2. PPEM algorithm (Xu, Lv, and Wang 2013).
3. The Match Bucket algorithm (Chen et al. 2011).

To fairly compare the performance of each algorithm, all experiments were conducted on the same computer with Intel(R) Core(TM) 2 Duo 2.93 GHz CPU, 2 GB RAM running Windows XP SP2. What's more, we choose the same network model and event scheme as ferry in our simulation. The network model is derived from the King Dataset, which includes the pair wise latencies of 1024 DNS servers in the Internet measured by King Method (Gummadi, Saroiu, and Gribble 2002). The event scheme is

derived from a stock quotes model proposed in Meghdoot (Gupta et al. 2004). The scheme is defined as follows:

$$S = \begin{Bmatrix} \{Date : String, 2/Jan/98, 31/Dec/2013\}, \\ \{Symbol : String, aaa, zzzz\}, \\ \{Open : Float, 0, 500\}, \\ \{High : Float, 0, 500\}, \\ \{Low : Float, 0, 500\}, \\ \{Close : Float, 0, 500\}, \\ \{Volume : Integer, 0, 310000000\} \end{Bmatrix}$$

The scope of each attribute is defined and we also endowed each attribute a frequent degree to represent the possibility that the attribute will appear in the subscriptions and events: {p(Date)=10%; p(Symbol)=96%; p(Open)=49%; p(High)=80%; p(Low)=78%; p(Close)=44%; p(Volume)=28%}. The subscriptions are generated based on the frequent degree of attributes and some predefined templates introduced in Meghdoot. Events are randomly generated according to the frequent degrees of attributes.

The simulation parameters are as follows unless otherwise noted: network size of 1000 nodes; subscription of 10000; events of 10000.

5.2 Results

Figure 3 shows the matching time per 10000 events publishing with subscription number increased from 1000 to 10000. Match Bucket consumes the most matching time and the matching algorithm PPEM in our previous paper is slightly better than the Match Bucket. The best one is that we proposed in this paper Match-Ladder. As shown in Figure 3, with the increase of subscription number, it grows much more slowly than the other tow algorithms.

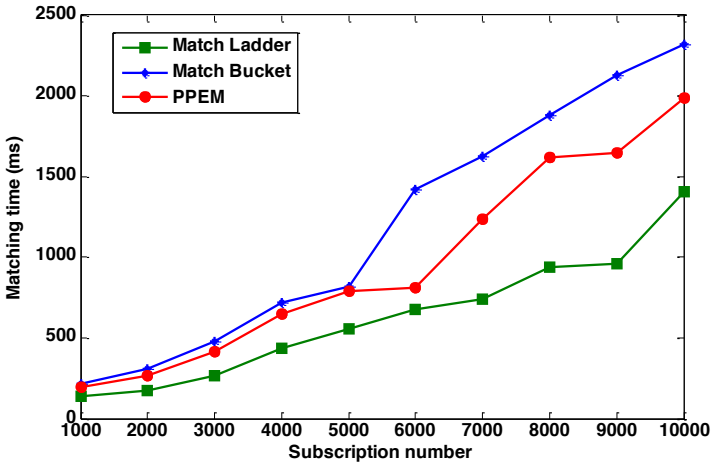

Figure 3: Event matching time of different subscription number.

The usage of memory space is shown in the Figure 4. By the data, it is easy to find that the memory space of the Match-Ladder algorithm is the lowest. So considering both the matching time and the usage of memory space, the conclusion is that the Match-Ladder performs better than the other two algorithms in both time efficiency and the usage of memory space.

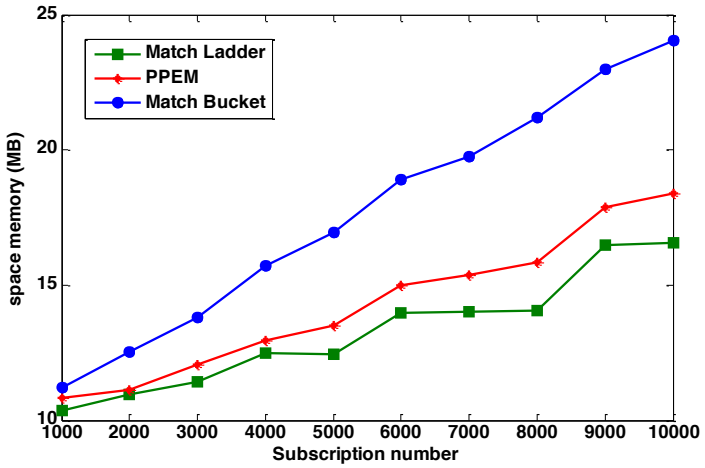

Figure 4: Memory space for different subscription number.

As previously documented in Theoretical analysis of Matching order section, the matching order of predicates is independent of the concrete matching algorithm, that is to say, the predicates' matching order could affect the efficiency of any matching algorithm, not just the Match-Ladder algorithm proposed in this paper. So in the next experiment, we will change the predicates' matching order for both Match-Ladder algorithm and Match Bucket algorithm.

In Figure 5 and Figure 6 (the event number is 1000), the different match order of predicates in one subscription does really affect the efficiency of matching algorithm. What's more, through the data in Table 1, it is not hard to find that the matching time of the worst match order is twice as long as the best match order. So all the results indicate that the match order is a significant factor for improving the efficiency of any matching algorithm.

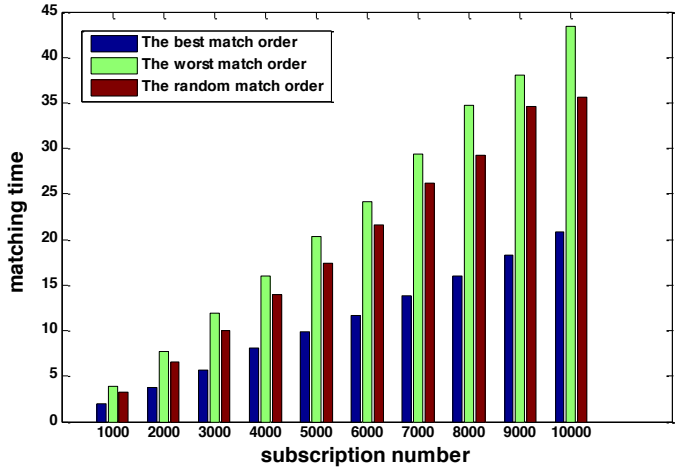

Figure 5: The matching time of different match order for Match-Ladder algorithm.

Table 1: The ratio of matching time between the worst and the best order.

Subscription number	1000	2000	3000	4000	5000
The worst/ The best	199.75%	203.32%	208.17%	197.36%	207.56%
Subscription number	6000	7000	8000	9000	10000
The worst/ The best	207.41%	212.80%	217.22%	208.35%	208.36%

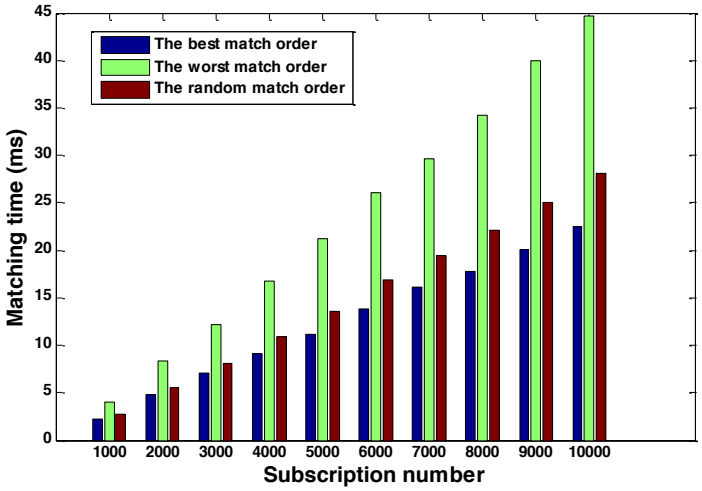

Figure 6: The matching time of different match order for Match Bucket algorithm.

6 CONCLUSIONS

In this paper, firstly, we have theoretically proved that the matching order of predicates affects the efficiency of the matching process, and we also acquire the method to get the best matching order by making use of the property of predicates; secondly, we have presented an efficient matching algorithm Match-Ladder based on the best matching order.

Our experimental results show that the Match-Ladder algorithm is much better than the PPEM and the Match Bucket algorithms in performance of large scale publish/subscribe systems and the Match-Ladder also requires low memory under different circumstances. In addition, the results also verify the Lemma 1, namely, the predicate matching order could affect the efficiency of any matching algorithm. But the weak point of the Match Ladder algorithm is that the calculation of the frequent degree of predicates in every subscription is time-consuming. In the future, we will speed up the calculation process by digging the priori knowledge. What's more, we will also research on how to apply our algorithm to approximate matching problem in the distributed systems.

ACKNOWLEDGMENTS

This paper is supported by the National High Technology Research and Development Program of China(863 Program) under Grant No. 2012AA011206.

A THE PROOF OF LEMMA 1

Suppose the original matching order of the subscription is $p_1 p_2 \mathrm{L}\ p_k \mathrm{L}\ p_m \mathrm{L}\ p_n$, so the original matching cost is:

$$CS = pa_1 + pp_1 pp_2 pa_2 + \mathrm{L} + pa_k \prod_{i=1}^{k} pp_i + \mathrm{L} + pa_m \prod_{i=k+1}^{m} pp_i + \mathrm{L} + pa_n \prod_{i=1}^{n} pp_i$$

By switching the matching order of any two predicates p_k, p_m. The new matching cost CS' is:

$$CS' = pa_1 + pp_1 pp_2 pa_2 + \cdots + pa_m pp_m \prod_{i=1}^{k-1} pp_i + \cdots + pa_k \prod_{j=1}^{m} pp_j + \cdots + pa_n \prod_{i=1}^{n} pp_i$$

The matching cost difference between the two matching order is:

$$CS - CS'$$

$$= (pa_k pp_k - pa_m pp_m) \prod_{i=1}^{k-1} pp_i - (pa_k pp_k - pa_m pp_m) \prod_{j=1}^{m-1} pp_j$$

$$= (pa_k pp_k - pa_m pp_m) \prod_{i=1}^{k-1} pp_i + (pa_m \quad pa_k) \prod_{j=1}^{m} pp_j$$

■

B THE PROOF OF THEOREM 1

Suppose there is a matching sequence $MS_1 : [p_1, p_2, \mathrm{L}\ p_k, p_{k+1}, \mathrm{L}\ , p_n]$ that can achieve the shortest matching time and there exists $pa_k pp_k / (1 - pp_k) > pa_{k+1} pp_{k+1} / (1 \quad pp_{k+1})$ between p_k and p_{k+1}.

In Lemma 1, suppose m = k+1, so

$$CS - CS'$$

$$= (pa_k pp_k - pa_{k+1} pp_{k+1}) \prod_{i=1}^{k-1} pp_i + (pa_{k+1} \quad pa_k) \prod_{i=1}^{k+1} pp_i$$

$$= [pa_k pp_k (1 - pp_{k+1}) - pa_{k+1} pp_{k+1} (1 - pp_k)] \prod_{i=1}^{k-1} pp_i$$

If we switch the matching order of p_k and p_{k+1}, we can get a more shorter matching time. This contradicts our assumption. Thus the matching order with increased $pa \bullet pp / (1 - pp)$ can achieve the shortest matching time. ■

REFERENCES

Aguilera, M. K., R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. 1999. "Matching Events in a Content-based Subscription System." In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing(PODC'99),* Atlanta, Georgia, USA, 53-61.

Chen, J. M., S. G. Ju, J. G. Pan, Z. W. Zu, and Z. Y. Gong. 2011. "Content-based effective event matching algorithm." *Journal on Communication* 32(6):78-85.

Fawcett, T., and F. Provost. 1999. "Activity Monitoring: Noticing Interesting Changes in Behavior." In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, California, USA, 53-62.

Fontoura, M., S. Sadanandan, J. Shanmugasundaram, S. Vassilvitski, E. Vee, S. Venkatesan, and J. Zien. 2010. "Efficiently Evaluating Complex Boolean Expressions." In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data,* Indianapolis, Indiana, USA, 3-14.

Gao, S., G. H. Li, and P. Zhao. 2011. "Marshmallow: A Content-Based Publish-Subscribe System over Structured P2P Networks." *Computational Intelligence and Security(CIS)*, Hainan, CHINA, 290-294.

Gough, J., and G. Smith. 1995. "Efficient Recognition of Events in a Distributed Systems." In *Proceedings of the 18th Australasian Computer Science Conference*, 55-65.

Gummadi, K.P., S. Saroiu, and S.D. Gribble. 2002. "King: Estimationg Latency Between Arbitrary Internet End Hosts." *Proceedings 2nd SIGCOMM Workshop on Internet Measurement,* 5-18.

Gupta, A., O. D. Sahin, D. Agrawal, and A. E. Abbadi. 2004. "Meghdoot: Content-based Publish/Subscribe over P2P Networks." In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware'04)*, Toronto, Canada, 254-273.

IBM. 1997. "Internet Application Development with MQSeries and Java." Palos Verdes: Vervante Corporate Publishing.

Kale, S., E. Hazan, F. Y. Cao, and J. P. Singh. 2005. "Analysis and Algorithms for Content-based Event Matching." *ICDCSW '05 Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS)*, Washington, DC, USA, 363-369.

Ma, J. G. 2009. "Research on Key Techniques for Large-Scale Data Dissemination Oriented Publish/Subscribe System." Ph.D. thesis, Institute of Software Chinese Academy of Sciences.[Accessed December 3, 2008].

OverSim. 2014. http://www.oversim.org/.

Xu, M. L., P. Lv, and H. B. Wang. 2013. "Predicate Priority Based Event Matching Algorithm in Publish/Subscribe System." *the Forth IEEE International Conference on Networking and Distributed Computing*, HongKong, HK, CHINA.[Accessed December 6, 2013].

Xue, T., and Q. Jia. 2013. "A Fast Matching Algorithm for Content-Based Publish/Subscribe Systems." In *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, Advances in Intelligent Systems and Computing 181:997-1001.

Zhao, Y., and J. Wu. 2011. "Towards Approximate Event Processing in a Large-Scale Content-Based Network." In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems,* Minneapolis, MN, USA, 790-799.

Zhu, Y., and Y. Hu. 2007. "Ferry: A p2p-based architecture for content-based publish/subscribe services." *IEEE Transaction on Parallel and Distributed Systems* 18(3): 672-685.

AUTHOR BIOGRAPHIES

MENGLU XU is currently pursuing her Master's degree at the Science and Technology on Integrated Information System Laboratory in Institute of Software Chinese Academy of Sciences, University Chinese of Academy Sciences. She holds a B.S. degree in Automation from the Xiamen University in Xiamen, China. Her research interests involve pattern recognition, network control, and distributed computing. Her email address is lumengxu@gmail.com.

PIN LV is an Associate Professor at the Science and Technology on Integrated Information System Laboratory in Institute of Software Chinese Academy of Sciences. He received a Ph.D. degree in Computer Application Technology from Institute of Software Chinese Academy of Sciences. His research deals with Modeling and Simulation. His email address is lvpin@iscas.ac.cn.

HAIBO WANG is a Senior Engineer at the Science and Technology on Integrated Information System Laboratory in Institute of Software Chinese Academy of Sciences. He received a M. S. degree in Computer Application Technology from Northwestern Poly technical University. His research deals with middleware. His email address is haibo@iscas.ac.cn.