Analysis and Algorithms for Content-based Event Matching

Satyen Kale* Elad Hazan* Fengyun Cao Jaswinder Pal Singh
Computer Science Department, Princeton University, Princeton, NJ 08544, USA
{satyen, ehazan, fcao, jps}@cs.princeton.edu

Abstract

Content-based event matching is an important problem in large-scale event-based publish/subscribe systems. However, open questions remain in analysis of its difficulty and evaluation of its solutions. This paper makes a few contributions toward analysis, evaluation and development of matching algorithms. First, based on a simplified yet generic model, we give a formal proof of hardness of matching problem by showing its equivalence to the notoriously hard Partial Match problem. Second, we compare two major existing matching approaches and show that counting-based algorithms are likely to be more computationally expensive than tree-based algorithms in this model. Third, we observe an important, prevalent characteristic of real-world publish/subscribe events, and develop a new matching algorithm called RAPIDMatch to exploit it. Finally, we propose a new metric for evaluation of matching algorithms. We analyze and evaluate RAPIDMatch using both the traditional and new metrics proposed. Results show that RAPIDMatch achieves large performance improvement over the tree-based algorithm under certain publish/subscribe scenarios.

1. Introduction

Publish/subscribe (pub/sub) systems aim at providing customized information delivery, often in real time. Users of the system, called *subscribers*, make their preferences known to the system in the form of *subscriptions*. Providers of information *publish* events into the system. It is the system's responsibility to transmit events to the subscribers who are interested in knowing about them.

This paper focuses on the design and analysis of an important algorithmic problem in pub/sub systems: the matching of events with subscriptions to identify interested users. The performance of the matching algorithm directly affects user-perceived event delivery latency and throughput of the pub/sub system. A large volume of work has been devoted to the matching problem [1][3][8][9][12]. However, despite the importance of the problem, analysis and evaluation of the problem's difficulty and its solutions are not yet well developed. For example, no theoretical hardness of the problem has been established; almost no matching algorithm described so far has been proven to obtain non-trivial performance guarantee (one exception is [1], following [11], in which a non-trivial guarantee on the average running time of the algorithm under a uniform distribution of inputs is shown); it is also unclear how the existing matching algorithms compare with one another, and how a pub/sub system should choose which algorithm to implement. In this paper, we address some of these issues, in the hope of providing useful insights and guidelines for matching algorithm design and evaluation in the future.

The rest of the paper is organized as follows:

In Section 2, we establish a simplified pub/sub model, and based on this model, present a formal analysis of the difficulty of the matching problem by reducing it to the well-studied Partial Match Problem.

In Section 3, we compare the basic idea in two major existing matching approaches, the counting-based algorithms and the tree-based algorithms. Analysis shows that counting-based algorithms are likely to be more computationally expensive than tree-based algorithms.

Based on the results from Section 2 and 3, we focus on developing matching algorithms that can benefit from specific pub/sub event properties. We observe that in real-world pub/sub systems, many events only have a few "relevant" properties by which they should be matched. To exploit this, we develop a new matching algorithm called *RAPIDMatch*, for *R*eordering *A*nd *P*artitioning *I*nput *D*ata – Match. RAPIDMatch preprocesses subscriptions by partitioning them by the

* Supported by Prof. Sanjeev Arora's David and Lucile Packard Fellowship, NSF grants CCR 0205594 and CCR 0098180

predicates corresponding to the "relevant" properties of events, so that for a given event, it can quickly eliminate large amounts of non-matching subscriptions and focus on a small subset of possibly matching subscriptions.

Matching algorithms have been evaluated by their running time as a function of the size of the input subscription set. We believe that the work done by an algorithm per matched subscription is also an important factor of consideration. We propose a new evaluation metric to include this factor in Section 5.

In Section 6, we evaluate RAPIDMatch and compare it with a tree-based algorithm, using both the traditional and new metrics proposed. Experimental results show that RAPIDMatch effectively improves performance over tree-based algorithm under certain publish/subscribe scenarios, and achieves similar performance for other cases.

2. The Matching Problem

Our mathematical model is derived from the work of Aguilera *et al* [1] and seems to be the most widely used one in the literature. To motivate the model, consider the example of an online newspaper. The newspaper publishes articles periodically and wishes to notify subscribers when articles of their interest are published. Users specify their preferences in terms of the topics they are interested in, such as current affairs, sports, arts, etc. They may specify if they are interested, not interested, or neutral about each topic. For example, a subscription specifying the subscriber's interest in sports, a *disinterest* in politics and indifference about the rest may have a subscription like
(sports = 1) and (politics = 0) and (arts = *) and …
Here, 1 indicates interest, 0 indicates disinterest, and * indicates "don't care".

An event, such as an article concerning politics and arts but not any other topic will be specified by
(sports = 0) and (politics = 1) and (arts = 1) and …

Formally, subscriptions consist of a conjunction of k *predicates* $(p_1 = v_1)$ and $(p_1 = v_2)$ and ... $(p_k = v_k)$ where the p_i are called *properties* and the v_i take values in {0, 1, *}. Events are represented in a similar fashion except that the v_i take values in {0, 1} (note that events specify *all properties*). An event is said to *match* a subscription if its property values satisfy all predicates in the subscription, i.e., 0 and 1 in the subscription match 0 and 1 respectively in the event and * matches either. For conciseness, we will omit the property names and only specify the values.

The model may seem to be an oversimplification, but it allows us to present the analysis and the RAPIDMatch data structure in a concise form. An exposition in the full-blown generality (such as ranges for the values, comparison operators instead of equality tests, etc.) would be complicated and obscure the underlying ideas, which are simple and intuitive. For the purpose of this paper, we only assert that the extension to the more general model is possible but requires more work. We now formally define the pub/sub matching problem:

Definition 1 (Matching Problem).
Given a set of subscriptions $S \subseteq \{0,1,*\}^k$ over k predicates $p_1, p_2, ..., p_k$, and an event $e \in \{0,1\}^k$. The ***Matching Problem*** *is to identify all subscriptions in S that e matches.*

2.1. Lower bounds

In this section we show that the pub/sub matching problem is of the same computational complexity as the *Partial Match problem* [10] within constant factors. The Partial Match problem is a classical problem of database querying: given N tuples of data in $\{0, 1\}^k$, find all data that satisfy a query, which is a partially specified tuple $\{0, 1, *\}^k$.

The reduction is analogous to the one given in [6]. Given an instance of Partial Match, we can reduce it to an instance of Pub/Sub Matching as follows:

Convert data tuple $e \in \{0,1\}^k$ to subscription $s_e \in \{0,1\}^{2k}$:

$$(s_e)_i = \begin{cases} 0 \rightarrow 0* \\ 1 \rightarrow *0 \end{cases}$$

Convert query $q \in \{0,1,*\}^k$ to event $e_q \in \{0,1,*\}^{2k}$:

$$(e_q)_i = \begin{cases} 0 \rightarrow 01 \\ 1 \rightarrow 10 \\ * \rightarrow 00 \end{cases}$$

In this way, given a query to the database, the tuples that match the query are exactly the ones corresponding to the subscriptions that match the event corresponding to the query. An analogous reduction works in the other direction, using the very same transformations of 0, 1, * in subscriptions and events. Therefore, the problems are equivalent computationally within a constant factor.

The Partial Match problem has been studied for over thirty years, and strong lower bounds are known for it [10]. From the reduction, we know that the pub/sub Matching Problem is equally hard, and it is unlikely that algorithms can be developed to achieve low time complexity and space complexity at the same time. Therefore, we suggest development of matching algorithms may focus on exploitation of specific pub/sub application characteristics.

3. Existing Approaches

In this section, we review the central idea of two major categories of state-of-the-art matching algorithms: *counting-based* algorithms [3][12] and *tree-based* algorithms [1][8][9]. In this section we will compare the algorithms in their most basic form to elucidate their features and differences.

A counting-based algorithm maintains a counter for each subscription that records the number of its predicates satisfied by the current event. Given an event, the algorithm iterates over the event's properties. For each property, it finds all subscriptions whose corresponding predicate is satisfied by the property value of that event, and increments the counters of these subscriptions. After going over all properties, the algorithm returns subscriptions whose counter values equal their number of predicates.

A tree-based algorithm organizes subscriptions into a rooted *search tree*. Each node at level i of the tree represents a property p_i, and has at most 3 successors, corresponding to values $p_i \in \{0,1,*\}$. Each tree leaf contains a list of all subscriptions with predicate values specified by the path from the root to the leaf. Given an event $e \in \{0,1\}^k$, the search tree is traversed from the root down. At every node, the value of the property for the event is tested, and the satisfied branches are followed. All the matching subscriptions are obtained at the leaves of the tree that the algorithm hits.

In the specific event model we are considering (viz. events specify *all* properties) we can show that the counting algorithm as described above performs worse than the tree based one if we just compare the number of operations (without considering the efficiency of their low-level implementation). For comparison, let's assume that both algorithms iterate through event properties in the same order, and there are x subscriptions whose predicates for property p_i are satisfied by the current event. After evaluation of p_i, the counting-based algorithm updates x counters for the subscriptions that satisfied it. The tree-based algorithm is different in two ways: first, after evaluation of p_i, it only considers subscriptions whose predicates 1, 2, … i have all been satisfied. The number of such subscriptions, y, is clearly at most x. Second, the number of links, z, followed by the tree-based algorithm at level i is also no more than y, and is usually smaller. In other words, the tree-based algorithm implicitly increments a *single* counter for all subscriptions in the same subtree below one satisfied link. Because $z \leq y \leq x$, the comparison and update operations performed in tree-based algorithm is expected to be less than in counting-based algorithm.

A simple example goes as follows: consider a set of N subscriptions with k predicates ($N >> k$), all with * in the first k-1 predicates and 1 in the k'th predicate. For an event with all property values being 0, the counting algorithm updates counters kN times, while the trivial upper bound for the number of nodes a tree-based algorithm checks is the size of the whole tree, which is less than $2N$.

It must be mentioned here that the view of the tree based and counting algorithms as given here is overly simplistic; there are much more optimized versions in the literature which make use of heuristics to perform much better. For instance, with under-specified events, counting algorithms can be made to output matches as soon as all the properties in the event are matched.

4. RAPIDMatch

We have shown that the Matching Problem is hard, and in our model, tree-based algorithms perform better than counting-based algorithm. Therefore, we focus on developing matching techniques to benefit from specific event properties, and choose to base our design on the tree-based approach.

We observe that in real-world pub/sub applications, many events only have a few "relevant" properties by which they will be matched. For example, in the newspaper example given earlier, an article is expected to concern only a few topics. Expressing this property in our model, an event is likely to have only a few 1's and many 0's in its properties.

Definition 2 (c-light and exactly c-light events). *An event $e \in \{0,1\}^k$ over k properties is called **c-light** if it has at most c 1's in it. An event is called **exactly-c-light** if it has exactly c 1's in it.*

We are interested in the case $c << k$. The following Light Query Model assumes that the number of 1's in event properties follow the Zipf distribution [3], which implies that a few events have many 1-value properties, while most events only have a small number of them.

Definition 3 (Light Query Model).

*In the **Light Query Model** with parameters (c, α), for all $t < c$ the probability that an event is exactly-t-light event is proportional to $t^{-\alpha}$ for some constant $\alpha \geq 1$. For any t, all exactly-t-light events are equally likely to appear.*

Here we state the insight of the Light Query Model and emphasize its applicability to the general pub/sub data model. A general pub/sub model can be converted into our simplified model with binary properties, by partitioning the content space and taking each partition as a binary property. An event has value 1 for a

property if it falls into the corresponding partition and 0 otherwise. Obviously, if the partitions do not overlap, an event can have value 1 for only one property. Subscriptions can also be partitioned, based on their interest in the content space partitions. In this way, for a given event, the algorithm can quickly identify the partition(s) it falls into, and the subset of subscriptions interested in that partition(s). Large amounts of subscriptions that do not cover the partition(s) are excluded from consideration, as they cannot possibly match the event.

Next, we present our matching algorithm, RAPIDMatch, that expedites event matching under the Light Query Model.

4.1. Data structure

RAPIDMatch is based on the tree-based matching approach. Usually, search trees make the following restriction:

Definition 4 (restricted search trees).
A (p_1,..., p_k)-restricted search tree, denoted T(p_1,..., p_k), is a search tree of depth exactly k, such that all nodes in depth i correspond to property p_i.

4.1.1. Subscription partitioning

We need the following definition:

Definition 5 (n_1, n_{1*}).
*Consider a subscription $s \in S \subseteq \{0,1,*\}^k$. We define $n_1(s)$ as the number of predicates with value 1 in s, and $n_{1*}(s)$ as the total number of predicates with values either 1 or *.*

RAPIDMatch partitions subscriptions into c+2 (not necessarily disjoint) *RAPIDMatchSets* S_0, S_1,… S_{c+1}. The sets are defined as follows:

$\forall \ 0 \le i \le c$, $S_i = \{s \in S \mid n_i(s) \le i \text{ and } n_{i*}(s) \ge i\}$

$S_{c+1} = \{s \in S \mid n_{i*}(s) \ge c\}$

For $0 \le i \le c$, S_i consists of subscriptions with at most i 1's and at least i 1's and *'s, and will be used to match events with exactly i 1's. S_{c+1} consists of subscriptions with at least c+1 1's and *'s, and will be used to match events with more than c 1's. This is shown in the following proposition:

Proposition 1. *For an exactly t-light ($t \le c$) event $e \in \{0,1\}^k$, the set S_t contains all subscriptions matching e.*

Proof. An exactly-t-light event e has exactly t 1's, and exactly k-t 0's. A subscription matching e cannot have more than t predicates with value 1, because they must match a 1 in e. Also, it must have at least t predicates with values 1 or *, because only these predicates can match the t 1's in e. All such subscriptions are included in S_t. □

4.1.2. Further subscription partition

We first divide properties p_1, p_2,…, p_k, into t+1 blocks of $k/(t+1)$ consecutive properties each: let $r = k/(t+1)$, and the partition is

$$| p_1p_2\ldots p_r || p_{r+1}p_{r+2}\ldots p_{2r} | \ldots | p_{tr+1}p_{tr+2}\ldots p_{(t+1)r} |$$

For clarity, we denote the blocks as P^t_1, P^t_2,…P^t_{t+1}.

Using this partition, we now define the secondary partition within RAPIDMatchSets as follows: S_t is partitioned into a disjoint and exhaustive collection of t+1 sets T^t_1, T^t_2, …T^t_{t+1}. The set T^t_j contains subscriptions which have either a 0 or a * in P^t_j. Note that since S_t contains subscriptions with at most t 1's, and since there are t+1 blocks of predicates, at least one block of predicates must be totally devoid of 1's, and so each subscription can be placed in at least one T^t_j. It is possible that a subscription may fit (and be put into) more than such sets. The partitions are illustrated in Figure 1.

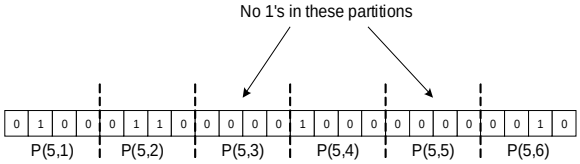


Figure 1. Partitioning the properties of an exactly-5-light event. P(i,j) indicates P^i_j. Partitions P^5_3, P^5_5 have no 1's, so all matching subscriptions will be in either T^3_5 or T^5_5.

Proposition 2. *For every c-light event $e \in \{0,1\}^k$, there exists a set T^t_j that contains all subscriptions matching e.*

Proof. First, we know that all subscriptions matching an exactly-t-light event e are in S_t. Because e has exactly t 1's, it must have only 0's in at least one of the t+1 block of properties. We denote this block as P^t_j. Any subscription s that matches the event must have only 0's or *'s in P^t_j as well. This implies that $s \in T^t_j$. □

4.1.3. RAPIDMatch data structures

Based on the partitions above, we build data structures in RAPIDMatch as follows:

1. Partition subscriptions into c+2 RAPIDMatchSets S_0, S_1,… S_{c+1}.
2. For $0 \le t \le c$, further partition S_t into t+1 sets T^t_1, T^t_2,…T^t_{t+1}. For each such set, build a search tree, called RAPIDMatchTree for subscriptions in the set.
3. Each RAPIDMatchTree T^t_j is a restricted search tree which handles all the properties in cyclic order starting from the property after P^t_j, namely p_{jr+1}, to p_k , then wrapping around to p_1 and ending at $p_{(j-1)r}$. Thus, the properties in block P^t_j are avoided.

4. For the subscriptions in S_{c+1}, build a standard (p_1, p_2,…, p_k)-restricted search tree (see Definition 4), to test all the k properties.

The data structures are illustrated in Figure 2.

4.2. Matching algorithm

The RAPIDMatch matching algorithm runs as follows:

1. Given an event $e \in \{0,1\}^k$, find t, the number of 1's in it.
2. If $0 \le t \le c$, iterate over the P^t_1, P^t_2,…,P^t_{t+1} blocks of e to find an index i such that e has only 0's in P^t_i. i is guaranteed to exist, as e has only t 1's and there are t+1 blocks. Search RAPIDMatchTree T^t_i, and return all matching subscriptions.
3. If $t > c$, search the tree of S_{c+1}, and return all matching subscriptions.

The correctness of the algorithm can be easily derived from Proposition 1 and Proposition 2.

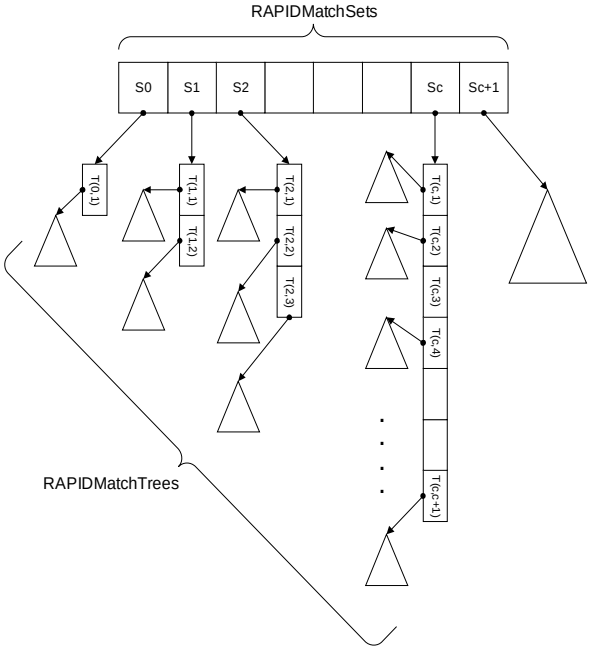


Figure 2. RAPIDMatch data structure. T(i,j) denotes T^i_j.

4.3. Performance analysis

Next, we derive worst-case guarantee of space and time complexity for RAPIDMatch.

Proposition 3 (RAPIDMatch space complexity). *The space of RAPIDMatch data structures is at most $O(c^2kN)$.*

Proof. In the worst case, every subscription appears in all the c^2 RAPIDMatchTrees, As each RAPIDMatchTree takes no more than $O(kN)$ space, total space complexity of RAPIDMatch is at most $O(c^2kN)$. □

Proposition 4 (RAPIDMatch time complexity). *For c-light events, RAPIDMatch returns all matching subscriptions within time at most $O(2^{k(1-1/(c+1))})$. For other events, RAPIDMatch has same performance as tree-based algorithm.*

Proof. For a c-light event with t 1's ($t \le c$), the running time of RAPIDMatch consists of the time it takes to find the appropriate search tree, which is $O(k)$, and the search time on the search tree, which is trivially bounded by $2^{k(1-1/(t+1))}$ as the depth of the tree is at most $k(1-1/(t+1))$. Since $t \le c$, the claim follows. For events with more than c 1's, RAPIDMatch searches the RAPIDMatchTree for S_{c+1} and becomes normal tree-based matching algorithm. □

We emphasize that though the worst case running time bound is exponential, it only applies when it is smaller than N. RAPIDMatch derives its performance gains by eliminating k/t property tests.

The equivalence of the Pub/Sub Matching problem to the Partial Match problem, as demonstrated in Section 2.2, shows that the Matching problem also inherently suffers from the *"Curse of Dimensionality"*: With increasing number of properties, the matching algorithm can improve running time only at the expense of space and vice-versa. RAPIDMatch uses a small constant factor - $O(c^2)$ - more space than the basic tree-based matching algorithm. However, it is able to significantly improve running time for c-light events, for which the worst-case tree-based algorithm running time can only be bounded by $O(2^k)$.

5. A new evaluation metric

Traditionally, matching algorithms have been evaluated by their running time as a function of input subscription database size. The output size, i.e. the number of subscriptions successfully matched, was not considered. However, we observe that such a metric may not best evaluate the characteristics of matching algorithms. For example, given N subscriptions, for an event matching all of them, even the best matching algorithm needs $O(N)$ time to check and return all subscriptions; while for an event that matches no subscription, an algorithm that returns nothing after $O(N)$ time would be unacceptable. Intuitively, a good algorithm should be able to quickly exclude non-matching subscriptions from consideration, and focus only on the relevant ones.

Therefore, we propose to evaluate matching algorithms by a new metric: running time normalized by output size. This metric measures the time spent per

matching subscription. Algorithms should be evaluated under various output size scenarios.

Another advantage of the metric is that it can also be used to evaluate *approximate* matching algorithms, which aim at quickly finding most but perhaps not all of the matches. For such an algorithm, its *time per match* value is likely to first increase rapidly with output size, and then decrease.

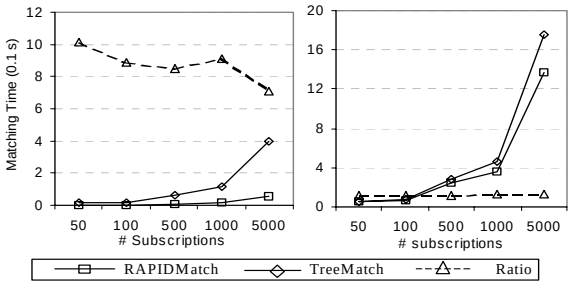
6. Experimental results

We implemented three matching algorithms for evaluation: CountMatch as in [5], TreeMatch as in [1], and RAPIDMatch in C++. All experiments were conducted on a Pentium 4 machine running at 2.0 Ghz with 512 MB RAM. No attempt was made at optimizing these algorithms, since we aim at a proof of concept at this stage. To make comparisons fair, the same search tree implementation was used in TreeMatch and RAPIDMatch. In all our experiments, CountMatch performed worse than TreeMatch., which confirms our observation from Section 3. For this reason, we focus only on comparing the performance of RAPIDMatch and TreeMatch, under both the traditional and the new metrics.

We generate subscriptions with 100 predicates and events with 100 properties. Each subscription predicate takes value 0, 1, or * independently with probability randomly drawn around 0.05, 0.05 and 0.9 respectively. Event property values are generated using the Light Query Model. We denote this as *Zipf* distribution. For completeness, we also experimented with an *Exponential* event distribution, which assigns an event property to 1 with probability p and 0 with probability (1-p). Performance results are shown as averages of 100 runs.

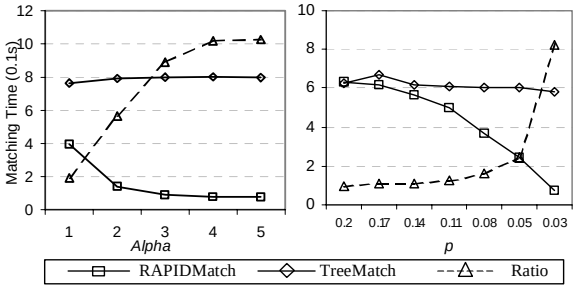
6.1. Running time versus input size

We first look at performance of matching algorithms for various sizes of the input subscription database. Figure 3 shows the average running time of RAPIDMatch and TreeMatch, and the ratio of the running time of TreeMatch to that of RAPIDMatch. Events were generated using Zipf distribution with α = 2, or exponential distribution with p = 0.1. Under both distributions, the ratio of the two algorithms is relatively stable, independent of input size. However, RAPIDMatch achieves more than 8 times performance improvement over TreeMatch under Zipf distribution, while only 1.2 times under exponential distribution. This is because with the parameter settings, the expected number of 1 properties in an event is about 3 (out of 100) in the Zipf distribution but 10 in the exponential distribution. As we expected, the small number of 1's in Zipf distribution offers RAPIDMatch with greater optimization opportunity. For other cases, the performance of RAPIDMatch is close to, and never worse than, that of TreeMatch.



(a) Zipf distribution with α = 2. (b) Exponential distribution with p = 0.1.

Figure 3. Matching time with different subscription sizes.



(a) Zipf distribution. (b) Exponential distribution.

Figure 4. Matching time with different event distribution parameter settings.

6.2. Dependence on data distribution

To further evaluate algorithms' ability to exploit characteristics of data distributions, in Figure **4**, we vary the parameters in Zipf and Exponential distribution. With increasing α or decreasing p, there are less 1's in the events, and RAPIDMatch's running time decreases as well. In contrast, the running time of TreeMatch stays constant for all parameter values, showing its inability to benefit from the skewed data distribution. The ratio between two algorithms flattens out with α higher than 3 in Zipf distribution, because the expected number of 1's in our events becomes close to 1.

6.3. Running time per match

Next, we evaluate the algorithms using the new metric proposed in Section 5. Figure 5 and Figure 6 present the average running time per matched subscription, against various number of matched subscriptions returned. When there are more matching subscriptions, both RAPIDMatch and TreeMatch spend less time to find each matched subscription. This is because there are more matching subscriptions in each of the subtrees the algorithms look into. However,

RAPIDMatch always performs better than TreeMatch, and its advantage is greater for smaller output sizes. This shows that with the help of multiple RAPIDMatchTrees, RAPIDMatch is able to quickly identify a small subset of relevant subscriptions, and exclude the large amount of unmatched subscriptions.

7. Conclusion and future work

We have shown that the event matching problem in content-based pub/sub systems is hard by reducing it to a known hard problem. We also showed that in our simplified model, tree-based algorithms are more efficient than counting-based algorithms. Following these observations, we developed a matching algorithm with the goal of exploiting common features of event and subscription data. Our algorithm, RAPIDMatch, partitions subscriptions offline based on their predicate characteristics, so that for a given event, it can quickly identify a small subset of relevant subscriptions and confine its search space. We also proposed a new metric to measure efficiency of event matching algorithms, and evaluated RAPIDMatch using both the traditional and new metrics.

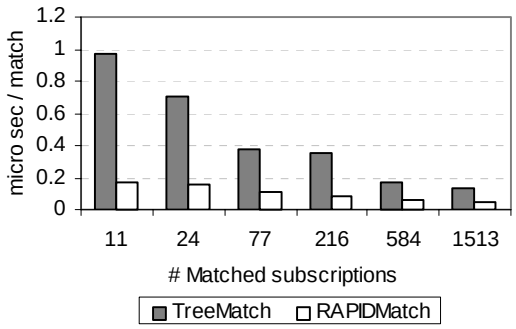


Figure 5. Matching performance measured as time/output, with Zipf distribution $\alpha = 2$.

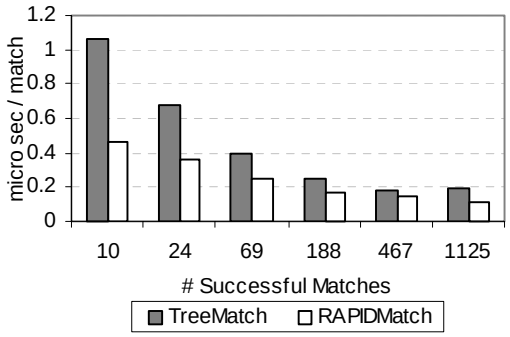


Figure 6. Matching performance measured as time/output, with Exponential distribution for p = 0.05.

Our goal is to stimulate discussions and further analysis of the matching problem from a theoretical point of view, as we believe such analysis can provide useful insights and guidelines for pub/sub algorithm and application developers. We have restricted ourselves so far to a simple data model. The next step is to develop deeper understanding and more comprehensive evaluation using realistic pub/sub data.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M.Astley, and T. D. Chandra, "Matching events in a content-based subscription system," In *Eighteenth ACM Symp on Principles of Distributed* Computing, 1999.

[2] A. *Bharambe*, *M. Agrawal, S. Seshan.* "Mercury: Supporting Scalable Multi-Attribute Range Queries". In *Proc. ACM SIGCOMM,* 2004

[3] L. Breaslau, P. Cao, et al. "Web Caching and Zipf-like Distributions: Evidence and Implications", In *Proc. IEEE INFOCOM*, 1999

[4] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. "Efficient Filtering in Publish-Subscribe Systems Using Binary Decision Diagrams," in *Proceedings of the 23rd International Conference on Software Engineering*, pp. 443-452, Toronto, Canada, May 2001

[5] A. Carzaniga, A.L. Wolf, "Forwarding in a Content-Based Network". In *Proceedings of ACM SIGCOMM 2003.*

[6] M. Charikar, P. Indyk, R. Panigrahy, "New Algorithms for Subset Query, Partial Match, Orthogonal Range Searching and Related Problems", *in Proceedings of the 29th International Colloquium on Automata Languages and Programming,* (2002)

[7] G. Cugola, E. Di Nitto, A. Fuggetta, "The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS", in *Proc. Of IEEE Trans. on Software Engineering*, 2001.

[8] F. Fabret, H. A. Jacobsen, et al. "Filtering algorithms and implementation for very fast publish/subscribe systems". In *Proc. ACM SIGMOD*, 2001.

[9] F. Fabret, F. Llirbat, et al. "Efficient matching for content-based publish/subscribe systems". Technical report, INRIA. http://www.caravel.infira.fr/pereira/matching.ps.

[10] T.S. Jayram, S. Khot, et al. "Cell-probe lower bounds for the partial match problem". In *Journal of Computer and System Sciences, Vol 69, Issue 3,* 2004.

[11] R. Rivest. "Partial match retrieval algorithms". SIAM Journal on Computing, 5:19-50, 1976.

[12] T. Yan and H. Garcia-Molina. "Index structures for selective dissemination of information under the Boolean model". In *ACM Trans. On Database Systems*, 19(2):334-364, 1994.