

Scalable Event Matching for Overlapping Subscriptions in Pub/Sub Systems

Zhen Liu, Srinivasan Parthasarthy, Anand Ranganathan, Hao Yang

IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532

{zhenl,spartha,arangana,haoyang}@us.ibm.com

ABSTRACT

Content-based publish/subscribe systems allow matching the content of events with predicates in the subscriptions. However, most existing systems only allow a limited set of operators, such as comparison on primitive data types (string, integer, etc). In this paper, we consider a publish/subscribe system that supports more flexible events/subscriptions with the use of advanced, yet potentially expensive, matching operators. Examples of such operators are pattern recognizers on multimedia data and spatial operators on location data. We study a critical problem in these publish/subscribe systems, namely how to optimize the matching process for a large number of subscriptions. This is achieved by exploiting the overlap in the subscriptions and sharing the operator evaluation results whenever possible. We formulate the optimal subscription evaluation problem and show that it is NP-Hard. We propose an efficient d -approximation algorithm, where d is the maximum number of operators in one subscription, as well as a heuristic algorithm that can further improve the system performance in practice. Our experiment results show that the proposed algorithms can reduce the matching cost by up to 80%, as compared to a naive strategy that evaluates the subscriptions independently.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed Applications*

Keywords

Publish/subscribe, Semantic, Event matching, Optimization

1. INTRODUCTION

Publish/subscribe systems have become increasingly popular in event-based systems by interconnecting information providers and consumers in a distributed environment. In these systems, the information providers publish information in the form of events; the information consumers sub-

scribe to events that satisfy certain conditions; and the system ensures timely delivery of published events to all interested subscribers. Publish/subscribe systems are often classified as topic-based (or group-based) and content-based. In topic-based systems, publishers label each event with a topic name, and consumers subscribe to all events in a particular topic. In content-based systems, subscribers can choose filtering criteria along multiple dimensions. These filtering criteria are often expressed using attribute-value pairs (e.g. [6], [1]), XML-based tree structures (e.g. [2], [7]), or graph structures ([21]). While these systems each has its own merits, they allow only a limited set of operators in the matching process, e.g., comparisons on primitive data types like string, integer and timestamp.

In this work, we consider a general content-based publish/subscribe system in which the subscriptions may need to be evaluated using advanced and potentially expensive operators. Such systems are very useful to support emerging applications that have rich semantics, such as detecting patterns in multimedia data, performing logical reasoning, and evaluating various spatial or temporal relationship.

As an illustrative example, consider the case where the law enforcement department monitors a city through a number of cameras mounted on UAVs or unmanned aerial vehicles (e.g., balloons or drones). These cameras continuously publish events, each of which contains a snapshot image, the GPS location, speed and altitude of the UAV, and a timestamp. Depending on their roles, the users subscribe to events that satisfy different kinds of conditions. For example, one user may subscribe to images that are captured in the Bronx area of NYC and indicate a police car chasing another car on a highway. Another subscription may be for images in Bronx that show a mob forming. Evaluating such subscriptions requires the use of spatial operators, which can determine whether the GPS coordinates of an UAV fall within the Bronx area, and image pattern recognition algorithms for detecting car chasing or mobs.

While the support for such general operators is desirable for many applications, it also has negative impact on the performance of the publish/subscribe system, because these operators are generally much more expensive than the primitive ones (e.g., integer comparison, string matching). As such, it is critical to design an efficient matching algorithm that can minimize the evaluation overhead of expensive operators, especially when there are a large number of subscriptions in the system.

In this paper, we optimize the subscription evaluation process, in the presence of expensive filters, based on two

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DEBS'07, June 20–22, 2007, Toronto, Ontario, Canada.
Copyright 2007 ACM 978-1-59593-665-3/07/03 ...\$5.00.

observations. First, the subscriptions often share common predicates. For example, the two subscriptions described above share the same predicate that the location of the UAV should be in Bronx. We can exploit such overlap between subscriptions and reuse the same operators to evaluate multiple subscriptions. Second, an event satisfies a subscription only if it satisfies all predicates in the subscription. As such, when any predicate in a subscription is evaluated to *false*, we do not need to evaluate the remaining predicates. Furthermore, if this predicate is shared by many subscriptions, all remaining predicates in these subscriptions can be ignored, which can significantly reduce the evaluation overhead.

Given that the evaluation of one predicate may result in the elimination of some other predicates, the evaluation overhead is largely decided by the order in which the predicates are evaluated. Several factors come into play when deciding the optimal evaluation order. One is the *selectivity* of the predicates, i.e., the probability that an event satisfies a given predicate. Intuitively, it is beneficial to first evaluate the predicates with low selectivity, because they have large chances to eliminate other predicates. However, the optimal order also depends on another factor, the *popularity* of the predicates, which is the number of subscriptions containing a given predicate. It is also beneficial to evaluate the popular predicates early, because if they happen to be *false* (even with some probability), a large number of subscriptions can immediately be resolved.

We formulate the optimal subscription evaluation problem as finding the optimal order for the predicates, such that the expected cost of predicate evaluation in runtime is minimized. We show that this problem is NP-hard, based on a reduction to the classic set cover problem. We then propose an efficient d -approximation algorithm for this problem, where d is the maximum number of predicates in a subscription. While this algorithm provides worst-case performance guarantee (its evaluation cost is at most d times the optimal one), we further propose a heuristic to improve the practical average-case performance. The heuristic algorithm uses a novel linked-chains data structure to manage overlapping subscriptions, and provides an analytical model for the cost differences resulting from different evaluation orders. These two algorithms can effectively exploit the overlap in the subscriptions and the selectivity of different predicates, and they significantly reduce the expected evaluation cost when an event arrives.

We have implemented a prototype system and evaluated the performance of our proposed algorithms using synthesized workloads. The experimental results show that our system can match the events to the subscriptions very efficiently, with an average evaluation cost sublinear to the number of subscriptions. Compared to a naive strategy that evaluates the subscriptions independently, our algorithms can reduce the evaluation cost by a factor of up to 80%.

In summary, the key contributions of this work are:

- We propose a generalized model for content-based publish/subscribe systems using potentially expensive operators.
- We formulate the optimal subscription evaluation problem and prove its NP-Hardness.
- We propose an d -approximation algorithm for evaluating the subscriptions, where d is the maximum number of predicates in a subscription.

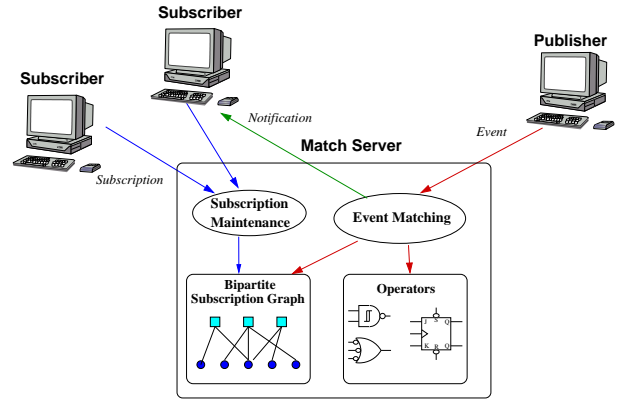


Figure 1: Architecture of a publish/subscribe system with generic operators.

- We propose a heuristic algorithm that improves upon the approximation algorithm with an analytical model of evaluation cost.

The rest of this paper is organized as follows. Section 2 formally defines the the publish/subscribe model that uses expensive operators. Section 3 formulates the optimal subscription evaluation problem and proves its NP-Hardness. Section 4 presents our two proposed matching algorithms in detail. Section 5 describes our system implementation and evaluates its performance using extensive experiments, and Section 6 compares to the related work. Finally, Section 7 concludes the paper with future work.

2. ARCHITECTURE AND MODELS

The architecture of our publish/subscribe system is shown in Figure 1. The system consists of multiple subscribers and publishers, as well as an event match server. The server contains all the existing subscriptions and matches the incoming events against them. The subscriptions can be represented internally by a bipartite graph that connects different subscriptions to the predicates they contain. The match server also has a set of operators that it can call for evaluating predicates in the subscriptions. These operators are in the form of libraries with well-defined interfaces. In what follows, we first describe our model for events and subscriptions, as well as its practical applications, and then define the matching process under this model.

2.1 Event and Subscription Model

We use a generic model for representing subscriptions and events in a content-based publish/subscribe system, which is useful for a variety of application domains. In this model, the events are simply application-specific objects that flow into the system. A subscription, on the other hand, consists of a set of predicates on these objects. Each predicate can have zero or more arguments. A predicate is, in general, of the form $p(a_1, a_2, \dots, a_l)$, where any subset of these arguments may be variables. In addition, a subscription can include a set of value constraints, which are boolean-valued functions on variables.

When a new event is published, a predicate in a subscription is evaluated by applying an operator with its arguments on the object contained in the event. The operator returns

either *true* or *false*, indicating whether the event satisfies the predicate. If any of the arguments of the predicate are variables, the operator also returns a set of feasible bindings of the variables, if the predicate evaluates to *true*.

Again, consider the scenario of monitoring a metropolitan area using unmanned aerial vehicles. In this case, the events are the images and auxiliary information (e.g., GPS location) received from these UAVs. Below are some exemplary subscriptions that can be represented in our model:

```
S1={locIn(Bronx), chasePattern(policeCar,anyCar)}
S2={locIn(Bronx), mobPattern(?people), (?people>50)}
S3={locWithin(TimesSquare,5000m),
    trafficPattern(congestion),
    pedestrianCount(?ped), ?ped>250}
S4={locIn(Bronx), trafficPattern(congestion),
    trafficPattern(accident), policeCarCount(?pol),
    (?pol<2)}
S5={distanceFrom(StatueOfLiberty,?x), (?x<1000m),
    tourBoatCount(?y), coastGuardBoatCount(?z),
    (?y>2*?z)}
```

The predicates in the subscription (e.g., *locIn*, *distanceFrom*, *chasePattern*, *trafficPattern*) are each associated with certain operators and evaluated by applying these operators on an incoming event. The value constraints (e.g., *?people > 50*) are evaluated after receiving variable bindings from the operators for any given event. We assume the cost of evaluating the value constraints to be negligible as compared to the expensive operators. Note that it is possible for variables to be shared across different predicates in a subscription.

We define that an event matches a subscription if and only if all the predicates and value constraints can be satisfied by a joint substitution of all variables used in the predicates. Formally, a subscription, S , is the union of a set of predicates and a set of value constraints, denoted by $S(P, VC)$ where P is the predicate set and VC is the value constraint set. An event, E , matches a subscription, $S(P, VC)$, if and only if there exists a variable substitution function, θ , that maps all variables used in P to specific values, such that the following conditions hold:

- For every predicate $t \in P$, $\theta(t)$ is *true*, where $\theta(t)$ is the predicate obtained after substituting all the variable arguments in t . If t has no variables, then $\theta(t) = t$. The truth value of $\theta(t)$ is obtained by calling an operator associated with t with the arguments of $\theta(t)$.
- For every value constraint $c \in VC$, $\theta(c)$ is *true*, where $\theta(c)$ is the boolean-valued expression obtained by substitution of the variables in c .

Note that an evaluation process need not proceed by finding all possible substitutions of the variable arguments of a predicate and evaluating their truth value. We assume that the operators can return all possible satisfying variable bindings when it is given a predicate with variable arguments.

Note that our subscription model bears a close resemblance to queries in logic programming languages, e.g., Prolog. However, one difference from pure logic programming models is that the predicates are not necessarily evaluated using a logical derivation process, but may be evaluated by invoking arbitrary operators written in any programming language. This model of subscription is very flexible and

can be easily used in a variety of scenarios. We now describe three example applications where our model of subscriptions and events can be readily employed.

2.1.1 Predicates Involving Pattern Recognition

Our event and subscription models are well-suited for operators that process multimedia data to determine if they satisfy certain conditions. We have already seen an example where predicates express conditions on images. Many other application scenarios can use similar forms of subscriptions, such as monitoring surveillance video cameras, monitoring voice calls, examining real-time text data generated as logs or traces by different systems, monitoring new web pages or blogs, etc. In all these cases, the actual processing of the multimedia content (image, text, video, audio) is expensive. Some of the operators for processing multimedia data return either *true* or *false*, depending on whether a specific pattern exists in the incoming data. Other operators can return values such as the number of people detected in an image. Both kinds of operators can be supported by our framework.

2.1.2 Predicates Involving Complex Operators

Another application domain for our subscription model is in cases where the subscription evaluation involves complex operations, with possible queries or lookups to a database. An example of such complex operators is spatial operator, such as determining whether a point is located within a region or calculating the distance between a point and a region. These spatial operators often require expensive processing for looking up a database to get the coordinates of a symbolic location like a street name, or for converting between locations expressed in different coordinate systems.

2.1.3 Predicates Involving Logical Reasoning

Our predicate-based model of subscriptions can be easily employed in cases where the truth value of a predicate is determined by posing a query to a logical reasoner. For example, a publish/subscribe system may use a rule-based reasoner like Datalog or Prolog in the case where policies or other such knowledge are represented as rules. Alternatively, it may employ a Description Logic [4] reasoner such as Pellet in the case of systems that describe data using Semantic Web representation languages like OWL.

It is worth noting some related work in the area of semantic publish/subscribe systems. Several existing systems, such as G-TOPSS [21] and OPS [22] use a semantic model of subscriptions based on RDF [5]. RDF can be viewed as describing data using binary predicates, i.e., predicates with two arguments. At a high level, the subscriptions model in these systems is similar to ours. However, a key difference is that in these existing systems, subscriptions expressed in RDF are matched against events that are also expressed in RDF. The matching process neither relies on the use of expensive operators nor uses complex description logic reasoning beyond subclass inferences. In our system, predicates are associated with operators, which may be in the form of queries to a Description Logic reasoner. Therefore, the evaluation of these operators may be very expensive, and our system seeks to reduce the subscription evaluation overhead through optimal ordering of operator evaluations.

2.2 Matching One Event to One Subscription

With the previously described event and subscription model,

the process of matching one event to one subscription is more complicated than simple value comparison or string lookup. It involves invoking the necessary operators and finding one assignment for variables in the subscription such that all predicates and value constraints are satisfied. This is done by first finding possible variable bindings for each predicate, and then joining the variable bindings obtained from different predicates. At the end, if there is at least one set of variable bindings that satisfy all value constraints, then the event matches the subscription. We assume the number of possible variable bindings for a given event is small. Hence the cost of performing joins and value constraint checks is small. The major cost comes from evaluating the operators for determining the truth values (i.e., *true* or *false*) and variable bindings of predicates.

Specifically, let the set of predicates in the subscription be $\{t_i, i = 1 \dots l\}$. For each predicate t_i , we invoke an operator to determine its truth value, and to find a candidate solution set (i.e., the feasible variable substitutions), denoted by $\{\theta_{i,j}\}$. Each solution $\theta_{i,j}$ maps the variables in t_i to values. If any predicate evaluates to *false*, then we can immediately stop the evaluation.

We obtain up to l sets of candidate solutions, one for each predicate (note that predicates with no variables will not contribute any candidate solutions). Next we construct a set of combined candidate solutions by taking the join of individual candidate solutions. Each combined solution maps all the variables in the subscription and satisfies every predicate in the subscription. Finally, we test the combined candidate solutions against the value constraints in the subscription. If there exists one combined candidate solution that satisfies all the value constraints, then it becomes the final solution, and the original event matches the subscription. Otherwise, the event does not match the subscription.

The above process describes how we can match one event to one subscription. When there are multiple subscriptions in the system, a straightforward approach is to evaluate the subscriptions one by one. However, it is inefficient when the number of subscriptions is high. Therefore, we are motivated to tackle one critical issue in such systems, that is, *how can we efficiently match one event against a large number of subscriptions*. In the following sections, we formulate this optimization problem and propose algorithms for solving it.

3. PROBLEM FORMULATION AND PROOF OF NP-HARDNESS

In order to optimize the matching process for many concurrent subscriptions, we make two important observations as follows. First, the users often share common interests, thus their subscriptions tend to share some predicates. In such cases, we only need to evaluate these common predicates once, and reuse the evaluation results (i.e. the truth value and bindings of variables, if any) across different subscriptions. Note that two predicates in different subscriptions are considered to be the same if they have the same non-variable arguments and all their variable arguments can be unified. For example, $p(?x1, ?y1)$ is the same as $p(?x3, ?y4)$, but it is not equivalent to $p(?x7, ?x7)$ since the last predicate requires two arguments to be same.

Our second observation is that a subscription does not match the event as long as *any* of its predicates cannot be satisfied. Thus, whenever we find one unsatisfied predicate,

we can immediately discard all associated subscriptions, and their remaining predicates do not need to be evaluated. In other words, for a given event, we may only need to evaluate a subset of the predicates.

Clearly, the number of predicates that are *actually* evaluated depends on *the order in which the predicates are evaluated*. The optimal schedule should consider both the subscription structure and the predicate selectivity. Intuitively, a good strategy is to first evaluate those predicates that are shared by many subscriptions or very unlikely to be satisfied, so that we can discard as many predicates as possible. Here we assume the selectivity of predicates is known or can be learned online through the past matching history.

We now formulate the optimal subscription evaluation problem and prove its NP-Hardness. To simplify the discussions, we consider the cases where all operators for evaluating predicates have the same cost. Later, we shall show how we can extend our model and algorithms for the cases where operators have different costs.

Subscription Evaluation Problem: Let Q be the set of subscriptions on a server, and T be the set of *unique* predicates in Q . Each subscription $q \in Q$ subscribes to a set of predicates, denoted by $T_q \subseteq T$. Each predicate $t \in T$ has a selectivity of $p_t \in [0, 1]$, which is the probability with which an event satisfies t . An event matches a subscription q if it satisfies every predicate in T_q .¹ Given an event, we need to evaluate the predicates in certain order, until all subscriptions are classified as either “*relevant*” (i.e., matching the event) or “*irrelevant*” (i.e., not matching the event). At this time, the set of predicates that have been evaluated is denoted by U . To ensure the correctness, (i) for every relevant subscription q , V must cover all predicates in T_q ; and (ii) for every irrelevant subscription q' , V must cover at least one predicate in $T_{q'}$ that is *not* satisfied. The goal is to find an optimal evaluation schedule that minimizes the expected size of V , i.e., the predicates that are actually evaluated.

Some remarks on the above problem are in order. First, observe that our model specifies a probability distribution over the set of predicates satisfied by an event (and hence, the set of subscriptions relevant to it). As such, the optimal number of predicates that need to be evaluated to satisfy (i) and (ii) is not fixed but also follows some probability distribution. Therefore, it makes sense to minimize the expected number of predicates evaluated for each event. Second, note that we can easily extend the problem formulation for the cases where predicates (or operators) have different costs by minimizing the sum of the costs of predicates in the set V . Third, we assume that the predicates are uncorrelated, i.e., whether an event satisfies one predicate is independent of the other predicates.

THEOREM 1. *The subscription evaluation problem is NP-Hard. Moreover, it has no polynomial time approximation algorithm with an approximation ratio better than $\Omega(\log n)$, where n is the number of subscriptions, unless $P = NP$.*

¹More exactly, the event should also provide at least one common variable substitution for T_q and also satisfy the different value constraints (Section 2). Here we simplify the description and ignore the join of different variable bindings obtained from different predicates and checking the value constraints, which is orthogonal to the evaluation schedule. In addition, the join and the value constraint check operations are much less expensive than the predicate evaluation.

PROOF. We prove the theorem by showing that the classical set-cover problem is a special case of the subscription evaluation problem.

Consider a set cover instance which consists of n elements in a *ground* set G (i.e., these are the set of elements which need to be covered), and a collection L of m subsets of G (these are the sets we will use to cover the elements in the ground set). Recall that the goal in the set cover problem is to choose as few subsets in L as possible in order to cover all the elements in G . This problem can be cast as an instance of the subscription evaluation problem as follows. We create n subscriptions q_1, q_2, \dots, q_n corresponding to each element e_1, e_2, \dots, e_n in G ; we also create m predicates t_1, t_2, \dots, t_m corresponding to each subset s_1, s_2, \dots, s_m in L . The subscription q_i contains each predicate t_j such that the set s_j includes the element e_i . Further, we set the satisfaction probability of every predicate to zero. Thus, it is enough to evaluate a single predicate in each subscription q_i in order to decide whether q_i is relevant to an event. It is now easy to verify that we can convert a feasible solution for the subscription evaluation problem where we evaluate c predicates into a feasible solution for the set-cover problem where the set cover has exactly c sets, and vice-versa.

It is well-known that the set-cover problem is NP-Hard, and no polynomial time approximation algorithm exists for it with an approximation ratio better than $\Omega(\log n)$, unless $P = NP$ [3]. This completes our proof. \square

4. OPTIMAL SUBSCRIPTION EVALUATION

In this section, we present two efficient algorithms for the subscription evaluation problem. The first algorithm sequentially evaluates different subscriptions and has an approximation ratio of d , where d is the maximum number of predicates in one subscription. The second algorithm reduces the expected number of predicate evaluations by analyzing the expected cost differences when subscriptions are evaluated in different orders.

4.1 Sequential Evaluation Algorithm

We now describe the sequential subscription evaluation algorithm. In this algorithm, we maintain a set U of undecided subscriptions and a set E of predicates that have currently been evaluated. Initially the set $U = Q$ and includes all subscriptions, and the set E is empty. While there are undecided subscriptions (i.e., U is not empty), we *arbitrarily* pick an undecided subscription q and *mark* this subscription. Recall that T_q is the set of predicates in q ; some of these predicates have already been evaluated at this point, while the rest of the predicates are yet-to-be evaluated. Let Y_q denote the latter set of predicates. We consider the predicates in Y_q in an arbitrary order; the predicate under consideration is evaluated if there exists some undecided subscription which subscribes to this predicate. As soon as a predicate is evaluated, it might result in a new set of subscriptions being decided; we remove such subscriptions immediately from the set U . An example of the evaluation is shown in Figure 2.

At first glance, this algorithm seems very simple and does not employ any optimization. However, for the special case where every subscription has at most a fixed number of predicates, we prove the surprising fact in Theorem 2 that the *sequential* algorithm can achieve an d -approximation guarantee, i.e., *constant-factor worst-case performance guarantee* w.r.t. the optimal algorithm.

Algorithm 1 Sequential Evaluation Algorithm

```

1: procedure SEQUENTIAL( $T, Q$ ) ▷
   The subset of predicates in  $T$  which must be evaluated
   to decide subscriptions  $Q$ 
2:    $U \leftarrow Q$  ▷ U is the current set of undecided
   subscriptions
3:    $E \leftarrow \Phi$  ▷  $E$  is the set of predicates evaluated so far
4:   while  $U \neq \Phi$  do ▷ some subscriptions are still
   undecided
5:     Select any subscription  $q \in U$ 
6:      $S \leftarrow$  the set of predicates subscribed by  $q$  which
   have currently not been evaluated
7:     while  $S \neq \Phi$  do
8:        $t \leftarrow$  any predicate in  $S$ 
9:       if  $t$  is subscribed by some subscription in  $U$ 
   then
10:         evaluate  $t$ 
11:         for all subscription  $q \in U$  which is now
   decided do
12:            $U = U \setminus \{q\}$ 
13:         end for
14:       end if
15:        $E = E \cup \{t\}$ 
16:        $S = S \setminus \{t\}$ 
17:     end while
18:   end while
19:   return  $E$  ▷ the set of evaluated predicates
20: end procedure

```

THEOREM 2. Let I be an instance of the subscription evaluation problem such that every subscription is subscribed to at most d predicates. Let $OPT(I)$ denote the optimal number of predicates which need to be evaluated for deciding all subscriptions in instance I . The sequential algorithm evaluates at most $d \cdot OPT(I)$ predicates.

PROOF. Let A denote the set of subscriptions that are marked by the *sequential* algorithm. We make the following three claims.

- *Claim (i)*: for any two distinct subscriptions q_1 and q_2 in A , the sets Y_{q_1} and Y_{q_2} are disjoint from each other. This is a simple consequence of the definition of the set Y_q . By definition, Y_q is the set of *yet-to-be evaluated predicates* which are subscribed by subscription q at the point when subscription q was marked by the algorithm. Since one of the two subscriptions q_1 and q_2 was marked prior to the other, one of the sets of predicates Y_{q_1} or Y_{q_2} was evaluated prior to the other and the claim follows.
- *Claim (ii)*: for any subscription q in A , the set Y_q contains at most d predicates. This claim simply follows due to the fact that each subscription subscribes to at most d predicates, as assumed by the theorem.
- *Claim (iii)*: consider any subscription $q \in A$; any subscription evaluation algorithm must evaluate at least one predicate in the set Y_q . To see why this claim holds, we first observe that $T_q \setminus Y_q$ is the set of predicates which are subscribed by q and which have already been evaluated at the point when q was marked. Crucially, each predicate in the set $T_q \setminus S_q$ must have

- Calculate the join of the variable bindings obtained from all the predicates and check the value constraints. If there exists at least one combined variable solution that satisfies all predicates and value constraints, then the event matches the subscription.

Note that the correctness of the above matching algorithm does not depend on how the subscriptions are ordered in the linked-chains structure. However, the expected evaluation cost is influenced by the subscription ordering. To determine an order that reduces the expected cost, in what follows, we analyze the difference in evaluated costs when subscriptions are evaluated in different orders.

We first derive an expression for evaluating a single chain. Next we consider two chains (say A and B) and evaluate the difference in costs between the two possible orders (i.e., A before B , or B before A). Note that these two orders have a cost difference only when there are equivalence links across these two chains, i.e., when they share common predicates. In such cases, these equivalence links allow us to reduce evaluation cost since we can reuse the results. In addition, if a common predicate is evaluated to *false*, we can eliminate the other subscription if it has not been evaluated. Finally, we combine this pair-wise comparison of cost differences and calculate an order for all subscription chains.

4.2.3 Cost of evaluating a single chain

Given a single subscription chain, let C be the expected cost of evaluating this chain. There are n nodes in the chain, and p_i denotes the probability that an event satisfies the i -th predicate node.

The first node in the chain must be evaluated in any case. The probability that this node is satisfied is p_1 , thus the probability of the second node being evaluated is p_1 . Similarly, the probability of the third node being evaluated is $p_1 \times p_2$. The cost of evaluating a single predicate is constant (using the hash table based data structure defined earlier). Without loss of generality, we can assign a normalized cost of 1 for evaluating each predicate. Hence the total cost of evaluating a chain is:

$$\begin{aligned} C &= 1 + p_1 + p_1 p_2 + \dots + \prod_{i=1}^{n-1} p_i \\ &= 1 + \sum_{i=1}^{n-1} \prod_{j=1}^i p_j \end{aligned}$$

4.2.4 Cost of evaluating two chains

Let us now consider the expected cost of evaluating two subscriptions, s_1 and s_2 . Let $p_{1,i}$ and $p_{2,j}$ be the selectivity of node i in s_1 and node j in s_2 respectively. If there are no equivalence links between nodes in the subscription, then the cost of evaluating the two subscriptions is simply

$$C = 1 + \sum_{i=1}^{n-1} \prod_{j=1}^i p_{1,j} + 1 + \sum_{i=1}^{m-1} \prod_{j=1}^i p_{2,j}$$

Let us now assume that there is one equivalence link between the chains. This link is between node $n_{1,a}$ and $n_{2,b}$ where $1 \leq a \leq n$ and $1 \leq b \leq m$. The order of evaluation of the chains is important now. Consider the case where s_1 is evaluated first. We can reuse the solutions obtained from equivalent node only if all the nodes above the equivalent node ($n_{1,1}, n_{1,2}, \dots, n_{1,a-1}$) have solutions. If this happens, then node $n_{1,a}$ will be evaluated and if there is at least one possible assignment of variables, the solutions can be sent

node $n_{2,b}$ (after appropriate variable renaming). If there is no possible assignment of variables, then s_2 does not have to be evaluated at all. Then, the expected cost of evaluating both subscriptions is:

$$\begin{aligned} C_1 &= (\text{Cost of evaluating all nodes in } s_1) \\ &+ (\text{Probability of nodes } n_{1,1}, n_{1,2}, \dots, n_{1,a-1} \text{ all having solutions}) \times [(\text{Probability of node } n_{1,a} \text{ having solutions}) \times (\text{Cost of evaluating } s_2 \text{ without having to evaluate node } n_{2,b}) \\ &+ (\text{Probability of node } n_{1,a} \text{ not having solutions}) \times (0)] \\ &+ (\text{Probability of at least one of the nodes } n_{1,1}, n_{1,2}, \dots, n_{1,a-1} \text{ not having a solution}) \times (\text{Cost of evaluating } s_2 \text{ including the evaluation of node } n_{2,b}) \\ &= \left[1 + \sum_{i=1}^{n-1} \prod_{j=1}^i p_{1,j} \right] + \prod_{i=1}^a p_{1,i} \times \left[1 + \sum_{i=1, i \neq b-1}^{m-1} \prod_{j=1, j \neq b}^i p_{2,j} \right] \\ &+ \left(1 - \prod_{i=1}^{a-1} p_{1,i} \right) \times \left[1 + \sum_{i=1}^{m-1} \prod_{j=1}^i p_{2,j} \right] \end{aligned}$$

Similarly, the expected cost of evaluating s_2 first and then s_1 is:

$$\begin{aligned} C_2 &= \left[1 + \sum_{i=1}^{m-1} \prod_{j=1}^i p_{2,j} \right] + \prod_{i=1}^b p_{2,i} \times \left[1 + \sum_{i=1, i \neq a-1}^{n-1} \prod_{j=1, j \neq a}^i p_{1,j} \right] \\ &+ \left(1 - \prod_{i=1}^{b-1} p_{2,i} \right) \times \left[1 + \sum_{i=1}^{n-1} \prod_{j=1}^i p_{1,j} \right] \end{aligned}$$

Taking into account that $p_{1,a} = p_{2,b}$ (since the nodes are equivalent), we have:

$$\begin{aligned} C_1 - C_2 &= (1 - p_{1,a}) \times \prod_{i=1}^{b-1} p_{2,i} \times \left(1 + \sum_{i=1}^{a-2} \prod_{j=1}^i p_{1,j} \right) \\ &- (1 - p_{1,a}) \times \prod_{i=1}^{a-1} p_{1,i} \times \left(1 + \sum_{i=1}^{b-2} \prod_{j=1}^i p_{2,j} \right) \\ &= (1 - p_{1,a}) \times (P_{2,b} \times C_{1,a} - P_{1,a} \times C_{2,b}) \end{aligned}$$

It is not difficult to see that if $a = 1, b = 1$, then there is no difference between C_1 and C_2 ; otherwise, $C_1 > C_2$ if and only if $\frac{P_{2,b}}{C_{2,b}} > \frac{P_{1,a}}{C_{1,a}}$. In this case, we evaluate s_2 before s_1 .

In general, the higher the P/C ratio of a chain, the earlier it should be evaluated. Intuitively this means that the algorithm should try to get to evaluating the equivalent node with high probability and low cost. This is true because the cost savings are achieved only when we evaluate the equivalent node.

Now we proceed to the general case where there are k equivalence links between the two subscriptions s_1 and s_2 . Similarly, we shall derive an expression for the difference in cost between evaluating s_1 first and evaluating s_2 first.

One key observation is that the chains will never cross in the linked-chains structure. In other words, if there are two links $(n_{1,a}, n_{2,b})$ and $(n_{1,c}, n_{2,d})$, then $a < c$ implies $b < d$. The proof arises from that fact that the links are arranged in decreasing order of selectivity. Therefore, if $a < c$, then node $n_{1,a}$ is above $n_{1,c}$ and hence $p_{1,a} < p_{1,c}$. Moreover, since $n_{1,a}$ is equivalent to $n_{2,b}$, we have $p_{1,a} = p_{2,b}$. Similarly $p_{1,c} = p_{2,d}$. Thus we have $p_{2,b} < p_{2,d}$, which implies that $n_{2,b}$ is above $n_{2,d}$, or $b < d$. As such, the equivalence links never cross each other.

Using a similar approach as before, we can derive an expression for $C_1 - C_2 =$

$$\sum_{l=1}^k (1 - p_{1,eq_{1,l}}) [P_{2,eq_{2,l}} \times C'_{1,eq_{1,l}} - P_{1,eq_{1,l}} \times C'_{2,eq_{2,l}}]$$

where $C'_{i,eq_{i,l}}$ is the cost of evaluating nodes in s_i above the l' th equivalent node from the top, $eq_{i,l}$, without considering any of the previous equivalent nodes $eq_{i,l'}, l' = 1..l - 1$.

The above derivation assumes that all predicates are independent, that is, the selectivity of one predicate is conditionally independent of the selectivity of any other predicate. The formula may be extended for the case when the predicates are conditionally dependent. However, such extensions are beyond the scope of this paper.

4.2.5 Ordering multiple chains

The previous derivations have provided an analytical model for the cost difference of evaluating one chain before another. Consider the general case where there are m subscriptions. We can analyze the cost differences for each pair of subscriptions using the above model. The next problem is to determine the optimal ordering of all the chains given the pair-wise cost differences, such that the overall cost is minimized.

This problem can be reduced to the Traveling Salesman Problem (TSP), which is a NP-complete problem. For the general TSP problem where the distance between nodes is not in a metric space (i.e., the distance does not satisfy symmetry and triangle inequality), it has been shown that any k-approximation for this problem is NP-hard. Therefore, we use a simple greedy approach based on the Kruskal's algorithm, which is known to produce good results in practice. Specifically, we construct a complete graph (V, E) and a weight function $W : E \rightarrow R$ as follows. Each vertex in V corresponds to a subscription chain, and the weight of an edge (u, v) is the cost savings of evaluating u before v . After the graph is constructed, we traverse the nodes and hence decide their order using the following heuristic algorithm:

1. Sort the edges in an increasing order of their weights.
2. Select with the least-weight edge. Check the remaining edges one by one and select an additional edge only if the edge, together with the already selected edges,
 - (a) does not cause a vertex to have a degree of three or more;
 - (b) does not form a cycle, unless the number of selected edges equals the number of vertices in the graph.
3. Repeat step 2 until the selected edges cover all vertices.

Together with the matching procedure over linked chains as described earlier (Section 4.2.2), the above subscription ordering algorithm completes the description of the *Sub-Order* subscription evaluation algorithm.

4.3 Handling Heterogeneous Evaluation Costs

So far, our analysis and algorithms assume that all predicates have the same evaluation cost. However, they can be easily extended to handle the case where different predicates have different evaluation results. The *Sequential* algorithm can remain the same; however the approximation guarantee increases from d to $\frac{d \times C_{max}}{C_{min}}$, where C_{max} and C_{min} are the maximum and minimum costs among the predicates, respectively. The costs of the predicate evaluation can also be included in the cost analysis for the *Sub-Order* algorithm. The only difference is that the probability, $p_{i,j}$ of each node

in a chain is now multiplied by the cost of the predicate in the cost difference expressions. The rest of the analysis and the algorithm remains the same.

5. IMPLEMENTATION AND EVALUATION

We have implemented a prototype system based on our proposed algorithms and evaluated its performance using extensive experiments. The results show that our system can efficiently match an event with a large number of subscriptions using expensive operators. In this section, we first describe our implementation and evaluation methodology, then present the experimental results in detail.

5.1 Prototype Implementation

Our prototype publish/subscribe system is based on Java and includes several modules for subscribers, publishers and servers respectively. The API that the system provides to a subscriber has three functions: *subscribe*, *unsubscribe*, and *renew*. Each subscription has a validity period, after which it is removed from the system. The validity period is specified by the subscriber (the default value is one day). However, a subscriber can renew or cancel her previous subscriptions at any time. For security reasons, the renew or cancellation requests must be authenticated, which is enforced by a ticket mechanism. Specifically, when the server receives a subscription, it generates a ticket which is the MD5 digest of the subscription content together with a random number. This ticket is returned to the subscriber in the acknowledgment message, and serves as the proof of the subscriber's identity. To be accepted, the renew or cancellation requests must carry the original ticket generated by the server. The API that the system provides to a publisher is fairly simple, which has only one function of *publish*, thus the events can be published in an anonymous manner.

At the core of the system is a Match Server that stores the subscriptions and evaluates them against a newly published event. The operators used in such evaluation are application-specific and can be easily plugged into our system. To expedite the matching process, we store the subscription graph internally using two hash tables. The first hash table keeps track of the unresolved subscriptions. It is indexed by the subscription id and stores a list of *remaining* predicates in each subscription. The second hash table maintains a queue of predicates that need to be evaluated. It is indexed by the predicate id and stores a list of *unresolved* subscriptions for each predicate. As such, the subscription graph is stored twice, one in each table. The benefit of such redundancy is that the primitive operations of subscription and predicate lookup/update can both be achieved in constant time.

5.2 Evaluation Methodology

We evaluate the performance of our system on a Linux machine with Xeon 3.4GHz CPU and 2 GB memory. The metric of interest is subscription evaluation cost (in short, *evaluation cost*), which is the average number of predicates that the system evaluates in matching an event to the subscriptions. Note that this is only a rough estimation of the absolute time that the matching process may take, because different evaluation operators may have different complexity and even the same operator may take different time when invoked with different parameters. However, in a long-term average sense, we believe the number of evaluated predicates

| parameter | description | value |
|-----------------|--------------------------------------------------------|-----------------|
| N_s | number of subscriptions | 2K ~ 80K |
| N_p | number of predicates | 1K ~ 40K |
| $model$ | popularity distribution of the predicate | Uniform or Zipf |
| min_{sl} | minimum number of predicates in a subscription | 3 |
| max_{sl} | maximum number of predicates in a subscription | 5 ~ 11 |
| N_e | number of events | 1000 |
| $ratio_{match}$ | ratio of matched subscriptions among all subscriptions | 0.2% |

Table 1: Workload parameters in the experiments.

can well reflect the efficiency of the evaluation process.

We conduct the experiments using a synthetic workload (i.e., the set of subscriptions and events) so that we can carefully examine various aspects of our system. The parameters used in synthesizing the workload are listed in Table. In each experiment, we first generate a set of N_p unique predicates and associate with each of them a popularity weight based on a given popularity model (either uniformly distributed in $[0, 1]$ or following the Zipf distribution). We then generate a set of N_s subscriptions as follows. For each subscription S_i , the number of predicates in it, denoted by l_i , is uniformly distributed between $[min_{sl}, max_{sl}]$, and l_i predicates are chosen from the entire set of N_p predicates with a probability proportional to their popularity weights. Finally we generate N_e random events and on average each event matches to $ratio_{match} * N_s$ subscriptions. We feed these events into the system and evaluate the performance of different algorithms. For each parameter setting, we repeat the experiments for multiple times and the results reported in the paper are based on the average over these runs.

To better gauge how well our proposed algorithms perform, we also implement a baseline algorithm, called *naive*, that evaluates the subscriptions independently. Specifically, the *naive* algorithm evaluates the subscriptions one by one and, for each subscription, evaluates its predicates until either one of them becomes *false* or all of them are evaluated. The *naive* algorithm also keeps the history of previously evaluated predicates and directly re-uses the old evaluation result if the current predicate has been evaluated before.

5.3 Experimental Results

Now we present our experimental results and compare our proposed algorithm to the *naive* one. We start with their performance as the numbers of subscriptions and predicates increase, and then study the impact of various parameters, such as the predicate distribution model and the number of predicates in each subscription.

5.3.1 Efficiency and Scalability

Our first set of experiments are conducted to understand how efficient our proposed *Sequential* and *Sub-Order* algorithms are and how their performance change as the number of subscriptions increases. During these experiments, we fix the number of predicates as 40K and vary the number of subscriptions, N_s , from 2K to 80K. The predicate popularity is uniformly distributed, and each subscription has at most 5 predicates (i.e., $sl_{max} = 5$).

The experimental results of evaluation cost for different algorithms are plotted in Figure 4. We can make two observations from this figure. First, both *Sequential* and *Sub-Order* algorithms perform much better than the *naive* algorithm in

all scenarios. For example, with 80K subscriptions, the *Sequential* and *Sub-Order* algorithms evaluate only 7761 and 7037 predicates respectively, thus reducing the evaluation cost by almost 80% as compared to *naive* algorithm. Secondly, the evaluation cost of *naive* algorithm increases linearly as the number of subscriptions grows, while the evaluation cost of *Sequential* and *Sub-Order* algorithms increases much slower. These results show that the common feature in these two algorithms, i.e., actively sharing the predicate evaluation across subscriptions, is very effective in reducing the overall evaluation cost.

While the *Sequential* algorithm has a nice d -approximation guarantee, we can see from Figure 4 that the *Sub-Order* algorithms perform consistently better in practice. This is because at each step, the *Sequential* algorithm blindly chooses one resolved subscription to evaluate, while the *Sub-Order* algorithms takes into account the cost differences due to the order in which the subscriptions are evaluated.

Next we repeat the above experiments with the same parameter settings, except that the predicate popularity follows the Zipf distribution (with exponent $\alpha = 1$) rather than the uniform distribution. The experiment results are plotted in Figure 5, which exhibits exactly the same trend as in Figure 4. However, by comparing these two figures, we can notice that the *Sequential* and *Sub-Order* algorithms perform even better in the Zipf-distribution cases, as compared to the uniform-distribution case. We will revisit this issue in Section 5.3.4.

5.3.2 Impact of Number of Predicates

Note that the size of the subscription graph is determined by not only the number of subscriptions, but also the number of predicates. Thus, we also conduct experiments to evaluate the the system performance as the number of predicates increases. During these experiments, we fix the number of subscriptions as 20K and vary the number of predicates, N_p , from 2K to 30K. Each subscription has at most 5 predicates (i.e., $sl_{max} = 5$).

Figure 6 shows the evaluation cost of different algorithms in these experiments, where the predicate popularity follows the Zipf distribution ($\alpha = 1$). We also experimented with the uniform distribution and the results are similar. Figure 6 clearly shows that as more predicates appear in the system, the evaluation cost of our *Sequential* and *Sub-Order* algorithms increases much slower than the *naive* algorithm. For example, with 30K predicates in the subscriptions, on average the *naive* algorithm needs to evaluate 16251 predicates for each event, while the *Sequential* and *Sub-Order* algorithms evaluate only 3267 and 1977 predicates respectively. Again, these result confirm the scalability of our proposed algorithms.

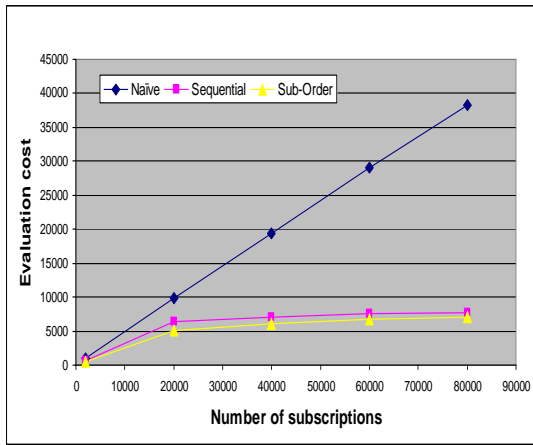


Figure 4: Evaluation cost vs. Number of subscriptions (Uniform predicate popularity)

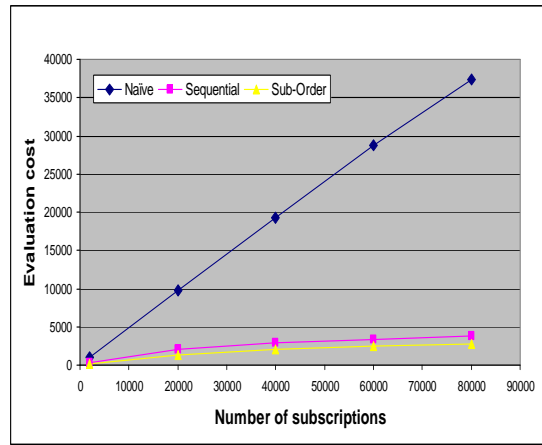


Figure 5: Evaluation cost vs. Number of subscriptions (Zipf predicate popularity)

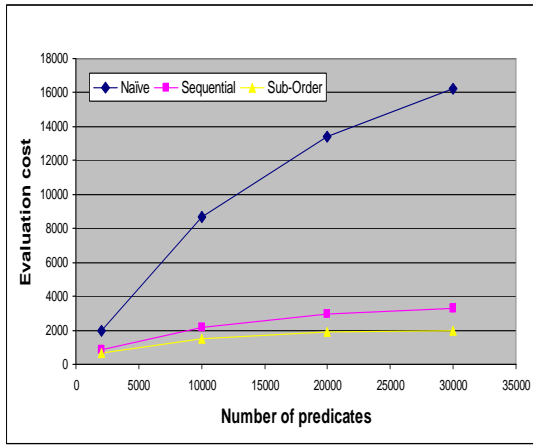


Figure 6: Evaluation cost vs. Number of predicates

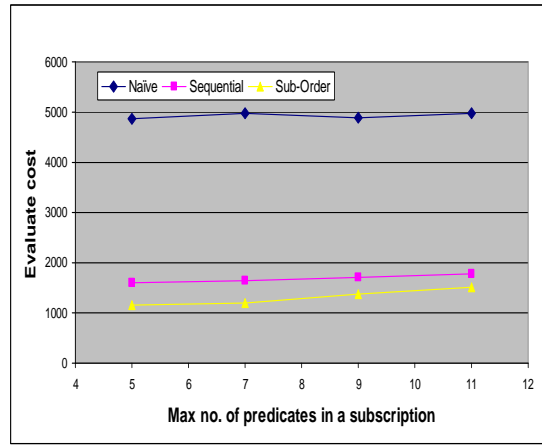


Figure 7: Evaluation cost vs. Maximum query length

5.3.3 Impact of Subscription Length

Given that the *Sequential* algorithm is a d -approximation algorithm (recall that d is the maximum number of predicates in a subscription), one may think that its performance degrades rapidly as the subscriptions become longer. In order to verify whether this is the case, we conduct several experiments and study the impact of subscription length. During these experiments, we fix 20K subscriptions and 5K unique predicates, while varying the maximum subscription length (i.e., d) between 5 predicates and 11 predicates. The minimum subscription length is fixed as 3 predicates, and the predicate popularity follows the Zipf distribution ($\alpha = 1$).

The evaluation costs with subscriptions of different lengths are plotted in Figure 6. Surprisingly, the performance of *Sequential* is not sensitive to the subscription length. For example, when d increases from 5 to 11, the evaluation cost of *Sequential* increases only from 1593 to 1784, and such an increase is mainly due to the fact that more edges exist in the subscription graph. These results, by no means, contradict with the d -approximation guarantee, which sheds insights on the algorithm’s worst-case performance; however, they indeed show that in practice, the *Sequential* algorithm can

perform much better than its theoretical guarantee.

In Figure 6, the performance of *naive* also seems insensitive to the subscription length. However, this is because it almost evaluates all unique predicates (i.e., the cost is close to $N_p = 5000$) in all cases. In other words, it consistently performs poorly in these experiments. It is also interesting to note that the performance gap between *Sequential* and *Sub-Order* shrinks as the subscriptions become longer. The reason is because *Sub-Order* only considers pairwise cost comparison between subscriptions. However, the order of two subscriptions also depends on other subscriptions that share at least one predicate with both of them. Therefore, with longer subscriptions, the pairwise cost comparison become less accurate.

5.3.4 Impact of Predicate Popularity Distribution

Finally we study the impact of the predicate popularity distribution, which implicitly controls the degree of predicate sharing across subscriptions. For this purpose, we experiment with both uniform distribution and Zipf distribution, which is well known as a good fit for keyword popularity in text-based searches. For Zipf distribution, we also experiment with different exponent parameters with $\alpha = 0.8$,

1, 1.2, respectively. During these experiments, we fix the number of subscriptions and predicates as $20K$ and $5K$ respectively, and the maximum subscription length is 5 predicates.

The results of these experiments are shown Figure 8. We can see that our *Sequential* and *Sub-Order* algorithms performs much better in the Zipf-distribution cases than in the uniform-distribution case. This is because with Zipf distribution, there are a few predicates that are extremely popular. In our algorithms, it is very likely that these predicates are evaluated early and, if they becomes *false*, eliminate many subscriptions early. However, the *naive* algorithm cannot benefit from such opportunities because it does not take into account the predicate sharing. Not surprisingly, with an increasing α in the Zipf distribution, the predicate popularity becomes more unbalanced, thus the the performance of *Sequential* and *Sub-Order* continues to improve.

6. RELATED WORK

Many content-based publish/subscribe systems have been built in the past few years. However, most of them do not consider the use of generic, expensive operators in evaluating subscriptions. For example, SIENA [6], Gryphon [1] and JEDI [13] represent events using attribute-value pairs, and subscriptions using conjunctions of predicates. These predicates can be evaluated relatively quickly by examining the event, which typically contains structured data and is small in size. Some systems, like XFilter[2], XTrie[7] and Web-Filter[19], represent events using XML documents and subscriptions using XPath expressions or its variations. Again, these XPath expressions can be evaluated quickly by examining the event. Since the predicates (or XPath expressions) in these systems are not expensive to evaluate, the order of predicate evaluation is less important than in our model.

Several publish/subscribe systems that use semantic matching have recently been proposed. The CREAM system [12] annotates events with a concept and semantic context. The Ontology-based publish/subscribe system (OPS) [22] uses RDF graphs and graph patterns to represent subscriptions and events. Chirita et al. also present a publish-subscribe system [11] based on RDF. However, these systems do not seek to optimize the matching process for a large number of subscriptions. Moreover, the S-ToPSS system [20] uses synonyms and concept hierarchies to improve matching. The G-ToPSS system [21] uses RDF to describe events and subscriptions and proposes a scalable matching algorithm for dealing with large number of subscriptions. It also uses RDFS class taxonomies to aid matching. While many of these systems use a predicate-based model (more precisely, binary predicates in RDF triples), their matching process do not require the use of expensive operators, because RDF predicates can be matched by directly comparing the arguments (i.e. subject and object) with the event. In contrary, our system focuses on the cases where predicates are expensive to evaluate and hence it is critical to minimize the number of predicate evaluation.

Our work is also related to the area of continuous queries on data streams. Continuously Adaptive Continuous Queries (CACQ [17]) seeks to optimize the evaluation of continuous queries by sharing relational operators (selections and join state) across queries. It also adapts to the changes in operator costs and selectivities over time. NiagaraCQ and it's extensions [10, 9] propose different grouping mechanisms for

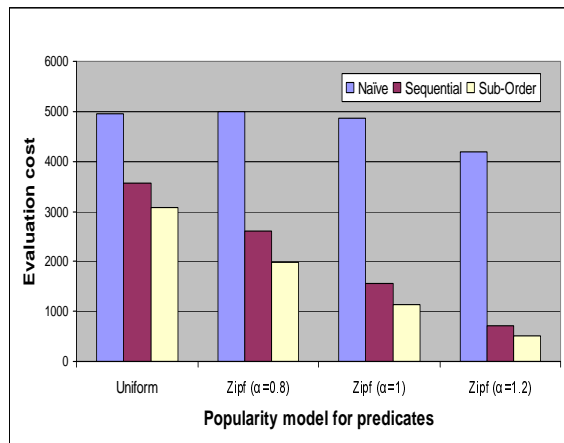


Figure 8: Evaluation cost under different predicate popularity distributions

continuous queries in order optimize a large number of continuous queries in the Internet. Our work differs from these designs in that we consider arbitrary operators in subscriptions, as opposed to a limited set of relational operators. As a result, many of these techniques proposed for optimizing continuous relational queries are not applicable in our scenario.

The problem of multi-query optimization has been a subject of study in relational databases for a long time [18, 14]. Again, these works focus on relational operators, while we consider a more general case with arbitrary expensive operators. Optimization of a single query with expensive filters has been considered in [8, 16]. Etzioni et al [15] consider a similar problem of finding an optimal schedule of querying different information sources to answer a given query. All these works consider only the optimization for a single query, while our work optimizes the evaluation of multiple queries by exploiting their overlaps.

7. CONCLUSION

In this paper, we have presented a publish-subscribe system that supports flexible subscription and event description through the use of generic, expensive filters. In order to support a large number of concurrent subscriptions, a key issue in such systems is the efficiency of the matching process. To this end, we have formulated the optimal subscription evaluation problem, proved its NP-Hardness, and proposed an d -approximation algorithm as well as a heuristic algorithm for this problem. Both algorithms can exploit the overlaps between the subscriptions to enable high degree of reuse for the predicate evaluation, thus reducing the processing overhead for event matching. We have implemented a prototype system and demonstrated its scalability and efficiency through extensive experiments.

Our current system is based on a centralized matching server that stores all subscriptions. In the future, we would like to extend our system to a distributed environment with multiple event brokers, where subscriptions and/or predicates can be evaluated in parallel. In such an environment, how to split the subscriptions among the available brokers, preferably with load balancing constraints, and how to efficiently route the events become very challenging problems.

Our proposed subscription evaluation algorithms should be adapted to take into account these new system models and requirements.

8. 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 *PODC '99: Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, pages 53–61, Atlanta, Georgia, USA, 1999.
- [2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 53–64, Cairo, Egypt, 2000.
- [3] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *STOC '97: Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, USA, 1997.
- [4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [5] D. Beckett. RDF/XML syntax specification. <http://www.w3.org/TR/rdf-syntax-grammar>.
- [6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
- [7] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *The VLDB Journal*, 11(4):354–379, 2002.
- [8] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Transactions on Database Systems*, 24(2):177–228, 1999.
- [9] J. Chen, D. DeWitt, and J. Naughton. Design and evaluation of alternative selection placement strategies in optimizing continuous queries. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 345–356, San Jose, California, USA, 2002.
- [10] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD '00: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 379–390, Dallas, Texas, USA, 2000.
- [11] P. A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/subscribe for RDF-based P2P networks. In *ESWS '04: Proceedings of the 1st European Semantic Web Symposium*, pages 182–197, Heraklion, Greece, 2004.
- [12] M. Cilia, C. Bornhoevd, and A. P. Buchmann. CREAM: An infrastructure for distributed, heterogeneous event-based applications. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 482–502, 2003.
- [13] G. Cugola, E. D. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *ICSE '98: Proceedings of the 20th International Conference on Software Engineering*, pages 261–270, Kyoto, Japan, 1998.
- [14] N. Dalvi, S. Sanghai, P. Roy, and S. Sudarshan. Pipelining in multi-query optimization. In *PODS '01: Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 59–70, Santa Barbara, California, USA, 2001.
- [15] O. Etzioni, S. Hanks, T. Jiangx, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the Internet. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 234–243, Burlington, Vermont, USA, 1996.
- [16] J. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *SIGMOD '93: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 267–276, Washington, D.C, USA, 1993.
- [17] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *SIGMOD '02: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, Madison, Wisconsin, USA, 2002.
- [18] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD '01: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 307–318, Santa Barbara, California, USA, 2001.
- [19] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Liribat, and D. Shasha. WebFilter: A high-throughput XML-based publish and subscribe system. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 723–724, Roma, Italy, 2001.
- [20] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS: Semantic Toronto publish/subscribe system. In *VLDB '03: Proceedings of 29th International Conference on Very Large Data Bases*, pages 1101–1104, Berlin, Germany, 2003.
- [21] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS: Fast filtering of graph-based metadata. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 539–547, Chiba, Japan, 2005.
- [22] J. Wang, B. Jin, and J. Li. An ontology-based publish/subscribe system. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 232–253, Toronto, Canada, 2004.