# Preference-Aware Publish/Subscribe Delivery with Diversity

Marina Drosou
Dept. of Computer Science
University of Ioannina, Greece
mdrosou@cs.uoi.gr

Kostas Stefanidis
Dept. of Computer Science
University of Ioannina, Greece
kstef@cs.uoi.gr

Evaggelia Pitoura
Dept. of Computer Science
University of Ioannina, Greece
pitoura@cs.uoi.gr

## ABSTRACT

In publish/subscribe systems, users describe their interests via subscriptions and are notified whenever new interesting events become available. Typically, in such systems, all subscriptions are considered equally important. However, due to the abundance of information, users may receive overwhelming amounts of events. In this paper, we propose using a ranking mechanism based on user preferences, so that only top-ranked events are delivered to each user. Since many times top-ranked events are similar to each other, we also propose increasing the diversity of delivered events. Furthermore, we examine a number of different delivering policies for forwarding ranked events to users, namely a periodic, a sliding-window and a history-based one. We have fully implemented our approach in SIENA, a popular publish/subscribe middleware system, and report experimental results of its deployment.

## 1. INTRODUCTION

With the explosion of the amount of information that is currently available online, publish/subscribe systems offer an attractive alternative to searching by providing a proactive model of information supply. In such systems, users express their interest in specific pieces of data (or *events*) via *subscriptions*. Then, they are *notified* whenever some other user generates (or *publishes*) an event that *matches* one of their subscriptions. Typically, all subscriptions are considered equally important and users are notified whenever a published event matches any of their subscriptions.

However, getting notified about all matching events may lead to overwhelming the users with huge amounts of notifications, thus hurting the acceptability of publish/subscribe systems. To control the rate of notifications received by the subscribers, it would be useful to allow them to rank the importance or relevance of events. Then, they would only receive notifications for the most important or relevant among them. For example, take a user Addison that generally likes drama movies but prefers drama movies directed by T. Burton to drama movies directed by S. Spielberg. Ideally, Addison would like to receive notifications about S. Spielberg drama movies only if there are no, or not enough, notifications about T. Burton drama movies.

In this paper, we propose extending subscriptions to allow users express the fact that some events are more important or relevant to them than others. To indicate priorities among subscriptions, we introduce *preferential subscriptions*. In general, there are two basic approaches to specifying preferences among items: the quantitative and the qualitative approach. In the *quantitative approach* (e.g. [5, 15, 19]), users employ scoring functions that associate a numeric score with specific data items to indicate their interest in them. In the *qualitative approach* (e.g. [7, 13, 14]), preferences between two data items are specified directly, typically using binary preference relations. We show how to formulate preferences among subscriptions using each one of these approaches. Events are ranked so that an event that matches a highly preferred subscription is ranked higher than an event that matches a subscription with a lower preference.

Based on preferential subscriptions, we introduce a top-$k$ variation of the publish/subscribe paradigm in which users receive only the matching events having the $k$ highest ranks as opposed to all events matching their subscriptions. Since the generation of events is continuous, we also introduce a number of delivering policies that determine the range of events over which the top-$k$ computation is performed.

However, the top-$k$ events are often very similar to each other. Besides pure accuracy achieved by matching the criteria set by the users, diversification, i.e. recommending items that differ from each other, has been shown to increase user satisfaction [23]. For instance, our user Addison would probably like to receive information about different drama movies by T. Burton as well as a couple of S. Spielberg's drama movies once in a while. To this end, we adjust the top-$k$ computation to take also into account the *diversity* of the delivered events. To achieve this, we consider both the importance of each event as specified by the user preferences as well as its diversity from other top-ranked events.

As a proof-of-concept, we have implemented a prototype, termed PrefSIENA [3]. PrefSIENA extends SIENA [4], a popular publish/subscribe middleware system, with preferential subscriptions, delivering policies and diversity towards achieving top-$k$ event delivery. We present a number of experimental results to assess the number of events delivered by PrefSIENA with respect to the original SIENA system, as well as their rank and diversity. We also report on the overheads of supporting diversity-aware top-$k$ delivery.

The rest of the paper is structured as follows. Section 2 presents publish/subscribe preliminaries. Section 3 introduces preferential subscriptions and event ranks. In Section 4, we focus on how to diversify the top-ranked events, while in Section 5, we examine a number of different delivering policies for forwarding events. In Section 6, we introduce an algorithm for computing the top-ranked events based on preferential subscriptions and in Section 7, we present our evaluation results. Section 8 describes related work and finally, Section 9 concludes the paper.

## 2. PUBLISH/SUBSCRIBE PRELIMINARIES

In general, a publish/subscribe system consists of three parts: (i) the publishers that provide events to the system, (ii) the subscribers that enter subscriptions and consume events and (iii) an event-notification service that stores the various subscriptions, matches the incoming events against them and delivers the matching events to the appropriate subscribers ([11]). Publishers can publish events at any time and these events will be delivered to all interested subscribers at some point in the future.

We use a generic way to form events, similar to the one used in [6, 12]. In particular, events are sets of typed attributes. Each event consists of an arbitrary number of attributes and each attribute has a type, a name and a value. Attribute types belong to a predefined set of primitive types, such as “integer” or “string”. Attribute names are character strings that take values according to their type. An example event about a movie is shown in Figure 1a. Formally:

An *event* $e$ is a set of typed attributes $\{a_1, \dots, a_p\}$, where each $a_i$, $1 \leq i \leq p$, is of the form $(a_i.type\ a_i.name = a_i.value)$.

Subscriptions are used to specify the kind of events users are interested in. Each subscription consists of a set of constraints on the values of specific attributes. Each attribute constraint has a type, a name, a binary operator and a value. Types, names and values have the same form as in events. Binary operators include common operators, such as, $=$, $\neq$, $<$, $>$ and *substring*. An example subscription is depicted in Figure 1b. Formally:

A *subscription* $s$ is a set of attribute constraints $\{b_1, \dots, b_q\}$, where each $b_i$, $1 \leq i \leq q$, is of the form $(b_i.type\ b_i.name\ \theta_{b_i}\ b_i.value)$, $\theta_{b_i} \in \{=, <, >, \leq, \geq, \neq, substring, prefix, suffix\}$.

Intuitively, we can say that an event $e$ *matches* a subscription $s$, or alternatively $s$ *covers* $e$, if and only if, every attribute constraint of $s$ is satisfied by some attribute of $e$. Formally:

Definition 1 (Cover Relation). *Given an event* $e = \{a_1, \dots, a_p\}$ *and a subscription* $s = \{b_1, \dots, b_q\}$, *$s$ covers $e$ ($s \succ e$), if and only if,* $\forall\ b_j \in s$, $\exists\ a_i \in e$, *such that,* $a_i.type = b_j.type$, $a_i.name = b_j.name$ *and* $((a_i.value)\ \theta_{b_j}\ (b_j.value))$ *holds,* $1 \leq i \leq p$, $1 \leq j \leq q$.

An event $e$ is delivered to a user, if and only if, the user has submitted at least one subscription $s$, such that $s$ covers $e$. For example, the subscription of Figure 1b covers the event of Figure 1a, and therefore, this event will be delivered to all users who have submitted this subscription.

| | | | |
|---|---|---|---|
| string | title | = | Big Fish |
| string | director | = | T. Burton |
| time | release_date | = | 13 Feb 2004 |
| string | genre | = | drama |
| integer | oscars | = | 0 |

(a)

| | | | |
|---|---|---|---|
| string | director | = | T. Burton |
| time | release_date | ≥ | 1 Jan 2003 |

(b)

**Figure 1: (a) Event and (b) subscription examples.**

| | | | |
|---|---|---|---|
| string | director | = | T. Burton |
| string | genre | = | drama |

$\succ$

| | | | |
|---|---|---|---|
| string | director | = | S. Spielberg |
| string | genre | = | drama |

**Figure 2: Qualitative preference example.**

## 3. PREFERENCE MODEL

In this section, we first extend subscriptions to include preferences. Then, we examine how to compute the importance of published events for the users.

### 3.1 Preferential Subscriptions

Our goal is for each subscriber, instead of receiving all matching events, to receive only the most interesting among them. To achieve this, we allow users to express preferences along with their subscriptions. In general, preferences can be expressed using either a quantitative or a qualitative approach. Following a *quantitative* preference model, users explicitly provide numeric scores to indicate their degree of interest (e.g. [5, 15, 19]). Following a *qualitative* model, users employ binary relations to directly define preferences between data items (e.g. [7, 13, 14]). We first use a qualitative preference model [7], since this model is more general than the quantitative one and also closer to the user’s intuition. Specifically:

Definition 2 (Preferential Subscription Model). *Let $S^X$ be the set of subscriptions of user $X$. Along with $S^X$, $X$ specifies a binary preference relation $C^X$ on $S^X$,* $C^X = \{(s_i \succ s_j) \mid s_i, s_j \in S^X\}$, *where $s_i \succ s_j$ denotes that $X$ prefers $s_i$ over $s_j$ or considers $s_i$ more interesting than $s_j$.*

An example is shown in Figure 2. Given $C^X$, we would like to rank subscriptions based on interest. To this end, we use the winnow operator [7]. The intuition is to assign the highest rank to the most preferred subscriptions, that is, to those subscriptions for which there is no other subscription in $S^X$ that is preferable over them. Formally, winnow at level 1, $win^X(1)$, is the set of subscriptions $s_i \in S^X$ for which, $\nexists\ s_j \in S^X$ with $(s_j \succ s_i) \in C^X$. An additional application of winnow, $win^X(2)$, returns the next most preferred subscriptions, that is, $s_i \in win^X(2)$, if and only if, $\nexists\ s_j \in (S^X - win^X(1))$ with $(s_j \succ s_i) \in C^X$. Generalizing, the winnow operator at level $l$, $l > 1$, returns a set of subscriptions, $win^X(l)$, consisting of the subscriptions $s_i \in (S^X - \cup_{q=1}^{l-1} win^X(q))$ such that $\forall\ s_i \in win^X(l)$, $\nexists s_j \in (S^X - \cup_{q=1}^{l-1} win^X(q))$ with $(s_j \succ s_i) \in C^X$. Repeated applications of winnow result in ranking all subscriptions in $S^X$.

The straightforward way to compute $win$ is to iterate through all subscriptions in $S^X$. Instead, if the preference relation is acyclic, we can organize subscriptions in a directed preference graph, where there is one node in the graph for each subscription in $S^X$ and an edge from a node representing subscription $s_i$ to a node representing subscription $s_j$, if and only if, $(s_i \succ s_j) \in C^X$. Now, a topological sort of

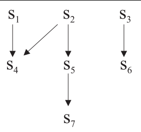
| Preference Relation | Graph | Preference Ranks | |
|---|---|---|---|
| $s_1 \succ s_4$ |  | $s_1, s_2, s_3$: | 1 |
| $s_2 \succ s_4$ | | $s_4, s_5, s_6$: | 2/3 |
| $s_2 \succ s_5$ | | $s_7$: | 1/3 |
| $s_3 \succ s_6$ | | | |
| $s_5 \succ s_7$ | | | |

**Figure 3: Extracting preference ranks.**

| | | | | |
|---|---|---|---|---|
| string | director | = | T. Burton | 0.8 |
| string | genre | = | drama | |

| | | | | |
|---|---|---|---|---|
| string | director | = | S. Spielberg | 0.6 |
| string | genre | = | drama | |

**Figure 4: Quantitative preferences examples.**

this graph can be used to compute *win*. In the first iteration, we output all nodes with no incoming edges. These nodes correspond to all subscriptions $s_i \in win^X(1)$, since for these, there is no $s_j \in S^X$ with $(s_j \succ s_i) \in C^X$. In the next iteration of the algorithm, these nodes are removed from the graph, along with their outgoing edges, and the nodes without incoming edges are output. Clearly, these nodes correspond to all subscriptions $s_i \in win^X(2)$. The algorithm stops when all nodes in the preference graph have been processed.

We associate a preference rank, $prefrank_i^X$ with each subscription $s_i$ based on the winnow level that the subscription belongs to. Since subscriptions retrieved earlier are of higher interest to the users, subscriptions returned at level $l$ of the winnow operator are assigned a preference rank equal to $\mathcal{G}(l)$, where $\mathcal{G}$ is a strictly monotonically decreasing function for which $\mathcal{G}(l) \mapsto [0, 1]$. Thus, for each user, we get pairs of subscriptions and preference ranks.

Definition 3 (Preferential Subscription). *A preferential subscription $ps_i^X$ of user $X$ is a pair of the form $ps_i^X = (s_i, prefrank_i^X)$, where $s_i$ is a subscription and $prefrank_i^X$ is a real number in $[0, 1]$ that expresses the degree of interest of $X$ for $s_i$.*

For instance, Figure 3 depicts the preference graph for an example preference relation and the extracted preference ranks when $\mathcal{G}(l) = (D + 1 - (l - 1))/(D + 1)$, where $D$ is the diameter of the preference graph. The cover relation (Definition 1) is extended to preferential subscriptions as follows: Given an event $e$ and a preferential subscription $ps_i^X = (s_i, prefrank_i^X)$, $ps_i^X \succ e$, if and only if, $s_i \succ e$.

Alternatively, instead of providing $C^X$, users could explicitly provide an interest score $prefrank_i^X$ for each of their subscriptions. This would correspond to a quantitative approach. A higher preference rank indicates a more important subscription. Examples are shown in Figure 4.

## 3.2 Computing Event Ranks

Let $P^X$ be the set of preferential subscriptions of user $X$. We use these preferential subscriptions to rank the published events and deliver to the user only the highest ranked ones. We define the *rank* of an event to be a function $\mathcal{F}$ of the preference ranks of the subscriptions that cover it.

Instead of using the preference ranks of all covering subscriptions, we use only the preference ranks of the *most specific* ones. A subscription $s$ is a most specific one if no other subscription in $P^X$ is covered by it, where:

Definition 4 (Cover between Subscriptions). *Given two subscriptions $s_i$ and $s_j$, $s_i$ covers $s_j$, if and only if, for each event $e$ such that $s_j \succ e$, it holds that $s_i \succ e$.*

For example, assume the event of Figure 1a and the preferential subscriptions ({*genre* = *drama*}, 0.7) and ({*genre* = *drama*, *director* = *T. Burton*}, 0.9) by Addison and ({*genre* = *drama*}, 0.7) and ({*genre* = *drama*, *director* = *T. Burton*}, 0.5) by another user Carson (for ease of presentation, we omit the type of each attribute). Both subscriptions of each user cover the event. Between the two, for each user, the latter subscription is more specific than the former one, in the sense that in the latter subscription the user imposes an additional, more specific requirement to movies (Addison prefers T. Burton's dramas over the rest, while Carson thinks that T. Burton's dramas are worse than other dramas). Thus, intuitively, the preference rank of the latter subscription should superimpose that of the former one, whenever an event matches both of them.

The event rank is formally defined as follows:

Definition 5 (Event Rank). *Given an event $e$, a user $X$, the set $P^X$ of the user's preferential subscriptions and the set $P_e^X = \{(s_1, prefrank_1^X), \ldots, (s_m, prefrank_m^X)\}$, $P_e^X \subseteq P^X$, such that $s_i \succ e$, $1 \leq i \leq m$, of the most specific subscriptions that cover $e$, the event rank of $e$ for $X$ is equal to $rank(e, X) = \mathcal{F}(prefrank_1^X, \ldots, prefrank_m^X)$, where $\mathcal{F}$ is a monotonically increasing function.*

User $X$ prefers an event $e_i$ over the event $e_j$, if and only if, $rank(e_i, X) > rank(e_j, X)$. As the aggregation function $\mathcal{F}$ for computing the rank of an event, we may use the maximum, mean, minimum or a weighted sum of the preference ranks of its covering subscriptions.

Now, we can formally define preferential top-$k$ delivery:

Definition 6 (Preferential Top-$k$ Delivery). *Given a set $M$ of $n$ matching events for a user $X$, deliver a subset $L$, $L \subseteq M$, with cardinality $k$, such that, $rank(e_i, X) \geq rank(e_j, X)$, $\forall\ e_i \in L$, $e_j \in M \backslash L$.*

## 4. EVENT DIVERSITY

Many times, the events that eventually reach the user are very similar to each other. However, it is often desirable that these events exhibit some diversity. In this section, we examine how to reduce the similarity of the matching events forwarded to the users. First, we introduce diversity-aware delivery and then describe how to integrate preferences and diversity towards improving the information quality of the delivered events.

## 4.1 Diversity-Aware Matching

Instead of overwhelming users with matching events that are all very similar to each other, we opt to select a representative set of events according to their diversity. To measure the *diversity* of events, i.e. how different they are, we first define the distance between two events. Without loss of generality, we assume that the events have the same number of attributes. Otherwise, we can simply append a sufficient number of "dummy" attributes to the event having the smaller number of attributes.

**Ranking of events**: $e_1(comedy), e_2(drama), e_3(drama), e_4(drama), e_5(horror), e_6(sci\text{-}fi)$

**Top-*4* events based on their ranks:** $e_1, e_2, e_3, e_4$

**Diverse Top-*4* events**:

| $divrank(e_1, X)$ | $divrank(e_2, X)$ | $divrank(e_3, X)$ | $divrank(e_4, X)$ | $divrank(e_5, X)$ | $divrank(e_5, X)$ | $L^X = \emptyset$ |
|---|---|---|---|---|---|---|
| - | - | - | - | - | - | $L^X = (e_1, e_4)$ |
| - | - | 0.4 | 0.4 | 0.85 | 0.8 | $L^X = (e_1, e_4, e_5)$ |
| - | - | 0.4 | 0.4 | - | 0.8 | $L^X = (e_1, e_4, e_5, e_6)$ |

**Figure 5: Computing top-4 diverse events.**

Definition 7 (Event Distance). *Given two events* $e_1 = \{a_1, \ldots, a_p\}$ *and* $e_2 = \{a'_1, \ldots, a'_p\}$, *the distance between* $e_1$ *and* $e_2$ *is defined as:*

$$d(e_1, e_2) = 1 - \frac{\sum_{i=1}^{p} \delta_i w_i}{\sum_{i=1}^{p} w_i}, \text{ where } \delta_i = \begin{cases} 1 & \text{if } a_i = a'_i \\ 0 & \text{otherwise} \end{cases}$$

*and each* $w_i$ *is an attribute specific weight,* $1 \leq i \leq p$.

Based on the above definition, the distance of any two events decreases as the number of their common attributes increases. Weights express the importance of an attribute for a specific application or user. In lack of such application-dependent information, we can assign equal weights to all attributes.

A number of different definitions of set diversity have been proposed in the context of recommender systems; here we model diversity as the aggregate or, equivalently, average distance of all pairs of events in the set [22]. We use the term "set" loosely to denote a set with *bag semantics* or a *multi-set*, where the same event may appear more than once in the set.

Definition 8 (Set Diversity). *Given a set of m events* $L = \{e_1, \ldots, e_m\}$, *the set diversity of L is:*

$$div(L) = \frac{\sum_{i=1}^{m} \sum_{j>i}^{m} d(e_i, e_j)}{(m-1)m/2}.$$

Now, the diversity-aware delivery problem can be defined as follows:

Definition 9 (Diverse Top-k Delivery). *Given a set M of n matching events,* $|M| = n$, *deliver a subset L,* $L \subseteq M$, *with cardinality k, such that,*

$$div(L) = \max_{L' \subseteq M, |L'| = k} \{div(L')\}.$$

The problem of selecting the $k$ items having the maximum average pair-wise distance out of $n$ items is similar to the *p-dispersion-sum* problem. This problem as well as other variations of the general $p$-dispersion problem (i.e. select $p$ out of $n$ points so that the minimum distance between any two pairs is maximized) have been extensively studied in operations research and are in general known to be NP-hard [9, 10].

A brute-force method to identify the $k$ most diverse events in $M$ is to first produce all $\binom{n}{k}$ possible combinations of $k$ events, and then pick the one with the maximum set diversity. The complexity of this process in terms of the required event distance computations is equal to $\frac{n!}{k! \cdot (n-k)!} \cdot \frac{n \cdot (n-1)}{2}$ and therefore, the computational cost is too high even for relatively small values of $n$ and $k$.

Instead, we use the following intuitive heuristic. We incrementally construct a diverse subset of events by selecting at each step an event $e$ that is furthest apart from the set of events already selected. The distance of an event $e$ from a set of events $L = \{e_1, \ldots, e_m\}$ is defined as:

$$dis(e, L) = \min_{1 \leq i \leq m} d(e, e_i).$$

In particular, let $M = \{e_1, \ldots, e_n\}$ be the input set of $n$ matching events and $L$ be the set we want to construct. Initially, $L$ is empty. We first add to $L$ the two furthest apart elements of $M$. Then, we compute the distances $dis(e_i, L)$, $\forall e_i$, such that $e_i \in M \backslash L$ and add to $L$ the event with the maximum corresponding distance. This process is repeated until $k$ events have been added to $L$. With this method, the required number of event distance operations are equal to $\frac{n \cdot (n-1)}{2} + [2 \cdot (n-2) + \ldots + (k-1) \cdot (n-k+1)]$. We can further reduce the number of performed operations based on the observation that after the insertion of an event $e$ to $L$, the distances of all other events that have not yet entered the diverse events from $L'$, $L' = L \cup \{e\}$, are affected only by the presence of $e$.

Proposition 1. *Given an event e and two sets L and* $L' = L \cup \{e\}$, *the distance of an event* $e'$ *from* $L'$ *is:*

$$dis(e', L') = \min\{dis(e', L), d(e', e)\}.$$

Using Proposition 1, the required event distance operations are equal to $\frac{n \cdot (n-1)}{2} + 2 \cdot (n-2) + (n-3) + \ldots + (n-k+1)$. Algorithm 1 summarizes the above procedure.

**Algorithm 1** Diverse Events Algorithm

**Input:** A set $M$ of matching events for user $X$.
**Output:** A subset $L$ of $k$ diverse events.

```
1: begin
2: $L \leftarrow \emptyset$;
3: find the events $e_1, e_2 \in M$ s.t.
   $d(e_1, e_2) = \max\{d(e_i, e_j) | e_i, e_j \in M, i \neq j\}$;
4: $L \leftarrow L \cup \{e_1, e_2\}$;
5: for all $e_i \in M \backslash L$ do
6:     $dis_i \leftarrow dis(e_i, L)$;
7: end for
8: find the event $e_{add}$ s.t. $dis_{add} = \max\{dis_i | e_i \in M \backslash L\}$;
9: $L \leftarrow L \cup \{e_{add}\}$;
10: while $|L| < k$ do
11:     for all $e_i \in M \backslash L$ do
12:         $dis_i \leftarrow \min\{dis_i, d(e_i, e_{add})\}$;
13:     end for
14:     find the event $e_{add}$ s.t. $dis_{add} = \max\{dis_i | e_i \in M \backslash L\}$;
15:     $L \leftarrow L \cup \{e_{add}\}$;
16: end while
17: return $L$;
18: end
```

## 4.2 Diverse Top-*k* Preference Ranking

We would like to combine both diversity and preference ranking when selecting which events to forward, so that the delivered events are both highly preferred as well as diverse

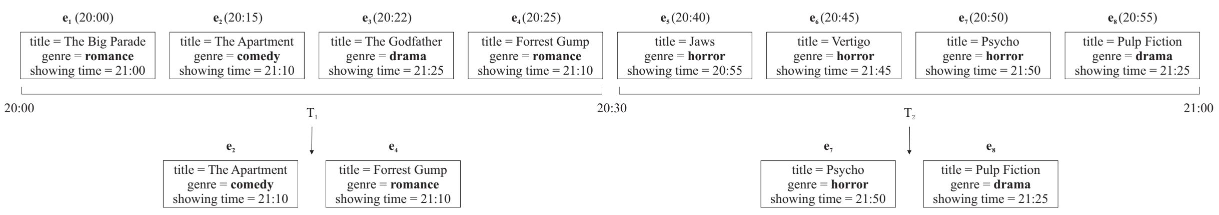


**Figure 6: Periodic top-2 events for Addison ($T$ = 30 min, $\sigma = 0.5$).**

with each other, i.e. we want to select $k$ out of $n$ events so that both the average of their preference ranks and their diversity are as good as possible. To this end, we combine the two measures to produce a combined ranking:

Definition 10 (Diversity-Aware Set Rank). *Let $X$ be a user. Given a set of $m$ events $L = \{e_1, \dots, e_m\}$, the diversity-aware rank of $L$ for $X$ is*

$$divrank(L, X) = \sigma \cdot \frac{\sum_{i=1}^{m} rank(e_i, X)}{m} + (1 - \sigma) \cdot div(L).$$

where $\sigma \in [0, 1]$. When $\sigma = 0$ (resp. $\sigma = 1$), events are chosen based only on diversity (resp. preference rank).

Now the problem becomes:

Definition 11. (Top-$k$ Preferred Diversity-Aware Delivery) *Given a set $M$ of $n$ matching events for a user $X$, deliver a subset $L$, $L \subseteq M$, with cardinality $k$, such that,*

$$divrank(L, X) = \max_{L' \subseteq M, |L'| = k} \{divrank(L', X)\}.$$

To locate the $k$ events with the maximum *divrank*, we use Algorithm 1, where we replace the $d(e_1, e_2)$ and $dis_i$ functions with the corresponding *divrank* versions.

In the following example, we apply Algorithm 1 to six events $e_1$, $e_2$ ..., $e_6$. To simplify our example, we assume that all events have only the attribute *genre* with value equal to *comedy*, *drama*, *drama*, *drama*, *horror*, *sci-fi* and event ranks 0.9, 0.8, 0.8, 0.8, 0.7, 0.6 respectively for a given user $X$. The distance between two events with the same genre is 0, while the distance between two events with different genres is 1. Figure 5 shows the trace of the heuristic applied on our example when $k = 4$ and $\sigma = 0.5$. We resolve ties in the case of events with the same *divrank* values by selecting the most recently published events.

## 5. DELIVERY MODES

Publish/subscribe systems offer an asynchronous mode of communication between publishers and subscribers by decoupling event publication from event delivery. In general, each event $e$ is associated with a number of time instants:

1. The time $e$ is published ($tpub_e$)
2. The time $e$ reaches the event-notification service ($tserv_e$)
3. The time $e$ is matched against subscriptions ($tmatch_e$)
4. The time $e$ is forwarded to the user ($tforw_e$) and
5. The time $e$ is actually received by the user ($trecv_e$)

Since events are continuously published and matched, we need to define over which sets of this stream of matching events we apply preference ranking and diversification. In the following, we use $tpub_e$ as the time instant associated with each event, since this is the time that characterizes best its freshness. However, note that, in general, events may reach the event-notification service and be matched in an order different from their publication order. Although out-of-order delivery does not invalidate our definitions, it may, however, complicate their implementation. Note that, alternatively, one could replace $tpub_e$ with $tmatch_e$ in all our definitions. This makes their implementation easier, but complicates semantics especially in the case of a distributed notification service.

We consider three fundamental modes of forwarding events, namely: (i) periodic, (ii) sliding-window and (iii) history-based filtering delivery. With *periodic delivery*, the top-$k$ events are computed over disjoint periods of length $T$ and forwarded to the subscribers once at the end of each period. With *sliding-window* delivery, the top-$k$ events are computed over sliding windows of length $w$, so that an event is forwarded, if and only if, it belongs to the top-$k$ events in the current window. Finally, *history-based filtering* continuously forwards new events as they are matched, if and only if, they are better than the top-$k$ events recently delivered.

The lengths $T$ and $w$ can be defined either in time units (e.g. as "the top-10 events matched per hour" and respectively, "the top-10 events matched in the last hour") or in number of events (e.g. as "the top-10 events per 100 matched ones" and respectively, "the top-10 events among the 100 most recently matched ones"). For clarity, in the following, we define $T$ in terms of time units and $w$ in terms of events, since this seems to fit better with the corresponding delivery modes. Next, we describe the three delivery modes in detail.

### 5.1 Periodic Delivery

Periodic delivery is appropriate for subscribers who wish to receive a list of important events regularly, for example, every morning when they reach their office or once in an hour. In this case, time is divided into disjoint periods of duration $T$ and top-ranked events are computed within each period. Whenever a period ends, the $k$ highest ranked matching events published within this period are forwarded to the users. To improve the freshness of events, ties are resolved by choosing to forward the most recent among the tied events. Formally:

Definition 12 (Periodic Top-$k$). *Let $X$ be a user and $M$ be the set of matching events published during a period starting at time instant $t$, i.e. an event $e_i \in M$, if and only if, $t \leq tpub_{e_i} < t + T$. Let $L$ be a subset of $M$ with $k$ events that has the maximum $divrank(L, X)$ among all subsets of $M$ with the same cardinality. If there are more than one such subsets, let $L'$ be one with the maximum $\sum_{e_i \in L'} tpub_{e_i}$ among them. If again, there are more such sets, we randomly select one of them, say $L_P$. An event $e$ with publication time $tpub_e$, $t \leq tpub_e < t + T$, is forwarded, if and only if, $e \in L_P$.*

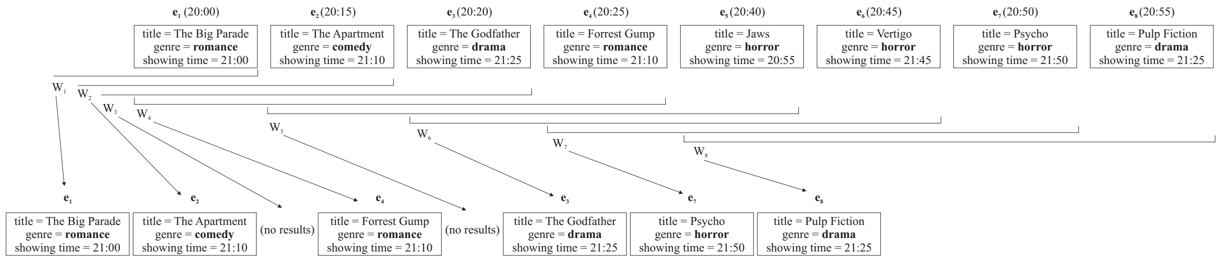

e1 (20:00)
title = The Big Parade genre = romance showing time = 21:00
e2(20:15)
title = The Apartment genre = comedy showing time = 21:10
e3(20:20)
title = The Godfather genre = drama showing time = 21:25
e4(20:25)
title = Forrest Gump genre = romance showing time = 21:10
e5(20:40)
title = Jaws genre = horror showing time = 20:55
e6(20:45)
title = Vertigo genre = horror showing time = 21:45
e7(20:50)
title = Psycho genre = horror showing time = 21:50
e8(20:55)
title = Pulp Fiction genre = drama showing time = 21:25
W1
W2
W3
W4
W5
W6
W7
W8
(no results)
(no results)

**Figure 7: Sliding-window top-2 events for Addison ($w = 4$, $\sigma = 0.5$).**

The number of events forwarded using periodic delivery is fixed and depends only on $k$ and $T$. Thus, we achieve a constant event delivery rate of $k \cdot \lfloor c/T \rfloor$ events in every time interval of duration $c$.

As an example, assume a single user, say Addison, who is interested in receiving events about movies showing in theaters. Addison has defined the following preferential subscriptions for movies: ($\{genre = comedy\}$, 0.9), ($\{genre = romance\}$, 0.9), ($\{genre = drama\}$, 0.8), ($\{genre = horror\}$, 0.6). She has also expressed her interest in receiving the top-2 events per period and that each period lasts 30 minutes. Assume further that the movie theaters which use the service publish the events $e_1$, $e_2$, ..., $e_8$ of Figure 6 in that order, at the time shown on top of each event. Figure 6 also shows the events that will be delivered to Addison for $\sigma = 0.5$. For the time period that begins at 20:00 and ends at 20:30, the top-2 results are the events $e_2$ and $e_4$ because comedies and romances are ranked higher that drama movies and $e_1$ is older than $e_4$, while from 20:30 to 21.00 the top-2 results are the events $e_7$ and $e_8$ because $e_5$ and $e_6$ are older than $e_7$.

## 5.2 Sliding-Window Delivery

With periodic delivery, top-$k$ computation starts anew at the beginning of each period. In contrast, with sliding window, top-$k$ computation starts anew, each time a new event is published. In particular, we call *window* of length $w$ an ordered list of $w$ events, denoted $W = (e_1, e_2, \dots, e_w)$ where $e_i$ precedes $e_{i+1}$ in the window, if and only if, no other event was published between them, that is, $\nexists\, e$, such that $tpub_{e_i} < tpub_e < tpub_{e_{i+1}}$. Let $W_f$ be the window that includes the first $w$ events published. If $W_i = (e_{i_1}, e_{i_2}, \dots, e_{i_w})$, $i \geq f$, then $W_{i+1} = (e_{i_2}, \dots, e_{i_w}, e)$, where $e$ is the first event published after $e_{i_w}$. As a special case, before the first $w$ events are published, the corresponding $w$-1 windows include only the events published so far and have length shorter than $w$. At the end of each window $W_i$, the $k$ highest ranked matching events published within this window are forwarded to the users. Formally:

Definition 13 (Sliding-Window Top-$k$). *Let $X$ be a user. Let $WS_e = \{W_m \mid e \in W_m\}$ be the set of $w$ windows an event $e$ belongs to. For each window $W_m$, let $WS_m$ be the set of events in $W_m$. Let $L_{W_m}$ be a subset of $WS_m$ with $k$ events that has the maximum $divrank(L_{W_m}, X)$ among all subsets of $W_m$ with the same cardinality. If there are more than one such subsets, let $L'_{W_m}$ be one with the maximum $\sum_{e_i \in L'_{W_m}} tpub_{e_i}$ among them. If again, there are more than one such sets, we randomly select one of them. Event $e$ is forwarded, if and only if, $e \in L'_{W_m}$ for some $W_m \in WS_e$.*

In our example, assume a window of length $w = 4$ and the published events of Figure 7. As shown in the figure, if Addison is again interested in the top-2 results with $\sigma = 0.5$, the first window $W_1$ returns its single event, i.e. $e_1$. The top-2 events of $W_2$ are $e_1$ and $e_2$ and since $e_1$ has already been sent to Addison, the only new result is $e_2$. $W_3$ contains no new results because dramas are less preferred than comedies and romances. The top-2 events of $W_4$ are $e_2$ and $e_4$, so $e_4$ is sent to Addison and so on.

In contrast to periodic delivery, the delivery rate is not constant, but depends on the relative order of the published events. When top-ranked events are computed based only on user preferences ($\sigma = 1.0$), we deliver at most one new event at each new window, as shown next.

Proposition 2. *When diversity is not used, between two consequent event-windows, at most one new event enters the top-$k$ results.*

Proof. Assume a window $W_q$ and its following window $W_{q+1}$, both of length $w$, and the two sets $L_{W_q}$, $L_{W_{q+1}}$ with the top-$k$ events for $W_q$ and $W_{q+1}$ respectively. Since $W_q$ and $W_{q+1}$ have $(w - 1)$ common events, let $W_q = (e_1, e_2, \dots, e_w)$ and $W_{q+1} = (e_2, e_3, \dots, e_{w+1})$. When $e_{w+1}$ is published, $e_1$ leaves the window and one of the following holds:

- $e_1 \in L_{W_q}$, then $L_{W_{q+1}} = (L_{W_q} - \{e_1\}) \cup \{e'\}$, where $e'$ is either $e_{w+1}$ or $e'$ was published in $W_q$ and $e' \notin L_{W_q}$, or
- $e_1 \notin L_{W_q}$, then $L_{W_{q+1}} = L_{W_q}$ or $L_{W_{q+1}} = L_{W_q} - \{e'\}) \cup \{e_{w+1}\}$, where $e'$ was published in $W_q$.

In any case, at most one event enters the set $L_{W_{q+1}}$. $\square$

However, when diversifying events, the top-$k$ events over $W_{q+1}$ are not necessarily related with the top-$k$ over $W_q$, since *divranks* are computed based not only on the (fixed) user preferences but also on the distances among the various candidate events. For example, a new highly preferable event may now disqualify more than one top-$k$ events because it is very similar to them. This observation leads us to the following property:

Proposition 3. *When diversity is used, between two consequent event-windows, more than one new event can enter the top-$k$ results.*

Proof. To illustrate this, let $e_1, \dots, e_5$ be a series of events. $e_1$ is a comedy directed by W. Allen with rank 0.9, $e_2$ is a thriller directed by T. Burton with rank 0.9, $e_3$ is an A. Hitscock's thriller with rank 0.8, $e_4$ is a S. Spielberg's drama with rank 0.85 and finally, $e_5$ is a Q. Tarantino's

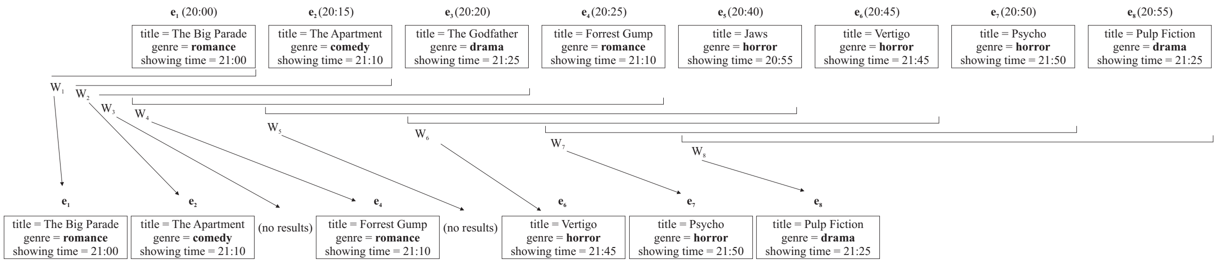


**Figure 8: History-based top-2 events for Addison ($w = 4$, $\sigma = 0.5$).**

drama with rank 0.9. Assume a window length $w = 3$, then $W_1 = (e_1)$, $W_2 = (e_1, e_2)$, $W_3 = (e_1, e_2, e_3)$, $W_4 = (e_2, e_3, e_4)$ and $W_5 = (e_3, e_4, e_5)$. Let $k = 2$ and $\sigma = 0.5$. $W_1$ will return $e_1$, $W_2$ will return $e_2$, $W_3$ will return no event, $W_4$ will return $e_4$ and $W_5$ will return both $e_3$ and $e_5$. $\square$

An event may remain in the window and be delivered after as many as $w$ other more recent events have been delivered. Thus, with sliding window, events may enter the top-$k$ list in an order different from their publication order (see for example $e_3$ in Figure 7).

## 5.3 History-based Filtering

Delivery using history-based filtering considers each new event as it arrives and decides whether to deliver it or not, based on history, i.e. the last top-$k$ events seen by the user. To refresh the events delivered, we assume that an event can remain in the top-$k$ list for up to a window of $w$ events.

DEFINITION 14 (HISTORY-BASED TOP-$k$). *Let $X$ be a user. Let $e$ be an event published at time instant $tpub_e$, $H$ be the set of the last top-$k$ events delivered to the user and $e'$ be the event published $w$ events prior to $e$ (the event that expires when $e$ is published). Event $e$ is delivered, if and only if, one of the following holds: (i) $e' \in H$, or (ii) $divrank((H - \{e_i\}) \cup \{e\}, X) \geq divrank(H, X)$ for some $e_i \in H$.*

In our example, when Addison selects this policy, her top-2 events will be the ones shown in Figure 8 (we again assume a window of length $w = 4$ and $\sigma = 0.5$).

As with sliding-window, the total number of delivered events is not bounded by $k$. However, since only newly published events can be delivered to the users, at most one new event can enter the top-$k$ results at each window, even with diversity.

## 6. THE EVENT-NOTIFICATION SERVICE

In this section, we outline a method for matching events with subscriptions and computing event ranks. To this end, we introduce a *preferential subscription graph* for organizing our preferential subscriptions. We also show how to compute the top-$k$ results for each delivery policy.

## 6.1 Event Matching

To reduce the complexity of the matching process between events and subscriptions, we organize the subscriptions using a graph similar to the *filters poset* data structure [6]. All subscriptions are organized in a directed acyclic graph, called *preferential subscription graph*, or $PSG$, whose nodes

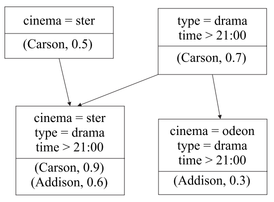


**Figure 9: Preferential subscription graph example.**

correspond to preferential subscriptions and edges to cover relations between them. Preferential subscriptions issued by different users which contain the same subscription are grouped together in a single graph node.

In particular, let $P$ be the set of all preferential subscriptions, i.e. the preferential subscriptions defined by all users, and $P_S$ be the set of all subscriptions in $P$. For each subscription $s_i \in P_S$, we maintain a set of pairs, called *PrefRank Set*, of the form ($X$, $prefrank_i^X$), where $X$ is a user and $prefrank_i^X$ is the preference rank of $X$ for $s_i$. A subscription $s_i$ is associated with the pair ($X$, $prefrank_i^X$), if and only if, a preferential subscription $ps_i^X = (s_i, prefrank_i^X)$ exists in $P$. For each $s_i \in P_S$, we define the *PrefRank Set* as the set $PR_i = \{(X, prefrank_i^X) \mid (s_i, prefrank_i^X) \in P\}$. Formally:

DEFINITION 15. (PREFERENTIAL SUBSCRIPTION GRAPH). *Let $P$ be a set of preferential subscriptions and $P_S$ the set of all subscriptions in $P$. A Preferential Subscription Graph $PSG_P(V_P, E_P)$ is a directed acyclic graph, where for each different $s_i \in P_S$, there exists a node $v_i$, $v_i \in V_P$, of the form $(s_i, PR_i)$, where $PR_i$ is the PrefRank Set of $s_i$. Given two nodes $v_i$, $v_j$, there exists an edge from $v_i$ to $v_j$, $(v_i, v_j) \in E_P$, if and only if, $s_i$ covers $s_j$ and there is no node $v'_j$ such that $s_i$ covers $s'_j$ and $s'_j$ covers $s_j$.*

For example, Figure 9 depicts the $PSG$ of the preferential subscriptions of two users, Carson and Addison. Carson has specified subscription $s_1 = \{cinema = ster, genre = drama, time > 21{:}00\}$ with preference rank 0.9, $s_2 = \{genre = drama, time > 21{:}00\}$ with 0.7 and $s_3 = \{cinema = ster\}$ with 0.5. Addison's subscriptions are $s_1$ with preference rank 0.6 and $s_4 = \{cinema = odeon, genre = drama, time > 21{:}00\}$ with 0.3.

When a new event $e$ arrives to the event-notification service, we traverse the $PSG$ to locate all matching subscriptions in a breadth-first manner starting from the root nodes. In some cases, it is not necessary to walk through all nodes of the graph. We may safely ignore a node $v$ with subscription $s$ for which there is no other node $v'$ with subscription

$s'$, such that $s'$ covers $s$ and $s' \succ e$. This means that whenever $e$ does not match a specific node of the $PSG$, its whole sub-tree can be ignored. This way, entire paths of the graph can be pruned. For example, in Figure 9, if an incoming event is not covered by $\{cinema = ster\}$, then it is certainly not covered by $\{cinema = ster,\ genre = drama\ \ time > 21:00\}$ and this subscription does not have to be checked against the event.

However, if the incoming event is covered by a node $v$ of the $PSG$, we have to check the event against other nodes in $v$'s sub-tree, to retrieve their preference rank, since it is possible that some of these nodes may be more specific to the event than $v$. In our example, for an event $e = \{cinema = ster,\ genre = drama,\ time = 21{:}30\}$, $s_3$ is more specific than $s_1$ for Carson. Therefore, even if $s_1 \succ e$, we have to continue traversing the $PSG$ (note that in a traditional publish/subscribe system that would not be necessary). To avoid unnecessary traversals, we associate each entry of $v$'s *PrefRank Set* with a status bit. This bit is set to 1, if the subscriber of the entry can also be found in some other node $v'$ covered by $v$ in the $PSG$ and to 0 otherwise.

For each subscriber $X$ associated with at least one subscription covering the event $e$, we compute the rank of the event. In our current work, we assume that preference ranks associated with subscriptions are indicators of positive interest, thus, we use as the aggregation function $\mathcal{F}$ the maximum value of the preference ranks of the covering subscriptions. Given that an event $e$ is covered by $m$ subscriptions $s_1, s_2, \ldots, s_m$ of user $X$, $rank(e, X) = \max\ \{prefrank_1^X, prefrank_2^X, \ldots, prefrank_m^X\}$. After this matching process, some information about the event (like the computed rank or the content) are stored according to the delivery policy used, as described next.

## 6.2 Event Delivery

Typically, publish/subscribe systems are stateless, in that, they do not maintain any information about previously delivered events. However, to provide users with the current top-ranked matching events, depending on the delivery policy, we may need to maintain some information about previously delivered events as well as buffer some published events prior to their final delivery or dismissal.

### *Periodic and Sliding-Window Delivery.*

In the case of periodic delivery, the server needs to buffer all events that match at least one subscription in its $PSG$ until the end of the period in which they were published and their corresponding ranks. At the end of the period, the top-$k$ results for all users are computed using Algorithm 1, having as input all the events matched during the period. Analogously, in sliding-window delivery, we buffer the content and ranks of the $w$ most recently published events, since an event may be forwarded to a user at some point after its publication time. In this case, the input set to Algorithm 1 is the set of events in the current window.

### *History-Based Filtering.*

With history-based filtering, we need to maintain some information about previously sent top-ranked events. If we do not consider diversity, we do not need to maintain the actual content of the matched events, it suffices to buffer just the preference rank of the current top-$k$ events. We also need to store an expiration counter along with each buffered event rank to determine when the event expires and disregard it. The counter is initialized to $w$ and is decreased by one every time a new event is published. An element is removed from the buffer, when its counter becomes 0. A newly published matching event $e$ is delivered, if and only if, (i) an event in the current top-$k$ expires when $e$ arrives or (ii) $e$ is better than the worst event in $X$'s current top-$k$. In this case, the worst event is disregarded and replaced by $e$.

If we consider diversity, we also need to store the content of the events in the buffer for computing their distances with any new event. As before, if an event in the current top-$k$ expires when a new event $e$ arrives, $e$ is forwarded to $X$. Otherwise, we swap each event $e_i$ in the buffer with the new event $e$ to produce a number of $k$ candidate sets. Then, we compute the new diversity-aware set rank (*divrank*) for each candidate set. If some of these new candidate sets have larger *divrank* than the current top-$k$ results, then we forward $e$ to the user and insert $e$ in the buffer in the place of the event $e_i$ that corresponds to the candidate set with the maximum *divrank*. The process described above is summarized in Algorithm 2.

**Algorithm 2** History-Based Filtering Event Delivery

**Input:** A new event $e$, a buffer $b$ of previously published matching events, a preferential subscription graph $PSG$, a subscriber $X$ and the number $k$ of desired top-results (and possibly a diversification factor $\sigma$).
**Output:** Whether $e$ should be forwarded to $X$ or not.

```
1: begin
2: $result \leftarrow false$;
3: $b \leftarrow b \cup \{e\}$;
4: $TOP_{init} \leftarrow$ the current top-$k$ results;
5: $TOP \leftarrow TOP_{init} -$ the event that has just left the window (if any);
6: if $|TOP| < k$ then
7:     $result = true$;
8: else
9:     if not diversify then
10:        find the lowest rank $r_{low}$ in $TOP$;
11:        if $rank(e, X) > r_{low}$ then
12:            $result = true$;
13:        end if
14:    else
15:        $DIVR_{init} = divrank(TOP_{init}, X)$;
16:        for all $e_i \in TOP$ do
17:            $TOP_i \leftarrow \{TOP - e_i\} \cup \{e\}$;
18:            $DIVR_i = divrank(TOP_i, X)$;
19:        end for
20:        $DIVR_{final} = \max\{DIVR_1, \ldots, DIVR_{|TOP|}\}$;
21:        if $DIVR_{final} > DIVR_{init}$ then
22:            $result = true$;
23:            $TOP \leftarrow \{TOP - e_{final}\} \cup \{e\}$;
24:        end if
25:    end if
26: end if
27: return $result$;
28: end
```

## 7. EVALUATION

To evaluate our approach, we have extended the SIENA event notification service [4], a multi-threaded publish/subscribe system, to include preferential subscriptions with diversity and ranked event delivery. We refer to our prototype as PrefSIENA. PrefSIENA is available for download [3].

**Table 1: Heuristic vs Brute-force performance.**

| $n$ | $k$ | Heuristic | | Brute-force | |
|---|---|---|---|---|---|
| | | Diversity | Time | Diversity | Time |
| 10 | 4 | 0.873 | 16 | 0.917 | 41 |
| | 8 | 0.846 | 26 | 0.851 | 42 |
| 20 | 4 | 0.905 | 29 | 0.917 | 384 |
| | 8 | 0.850 | 32 | 0.866 | 43065 |
| | 12 | 0.820 | 37 | 0.832 | 99608 |
| | 16 | 0.811 | 58 | 0.814 | 6784 |
| 30 | 4 | 0.929 | 31 | 1.000 | 1987 |
| | 8 | 0.881 | 38 | 0.923 | 1967339 |
| | 12 | 0.870 | 46 | 0.889 | 51652649 |
| | 16 | 0.859 | 57 | 0.874 | 162827625 |
| | 20 | 0.846 | 69 | 0.857 | 50641750 |

## 7.1 System Description

To evaluate the performance of our model, we use a real movie-dataset [2], which consists of data derived from the Internet Movie Database (IMDB) [1]. The dataset contains information about 58788 movies. For each movie, the following information is available: title, year, budget, length, user rating (rating), MPAA rating (mpaa) and genre(s).

Publishers generate publications by randomly selecting $m_P$ movies and creating a new event for each of them consisting of the title, year, length, rating, MPAA and genre(s) of the movie. Publications are produced at a constant rate. Each subscriber generates $m_S$ subscriptions, each of which is generated independently from the others. We select a number of the available attributes to appear in a subscription based on a zipf distribution, i.e. some attributes are more popular than others. The value of each attribute is also generated using a zipf distribution, so that some values are more common. Preferences are generated by associating preference ranks in $[0, 1]$ with the generated subscriptions. Those ranks have an average value around 0.5. In general, most specific publications get higher ranks.

Event delivery is performed following either one of our three delivering policies, i.e. the periodic, the sliding-window and the history-based filtering ones.

## 7.2 Experiments

We perform a number of different experiments. First, we evaluate the performance of our diversity heuristic. Then, we evaluate the number and quality of the events delivered to the users using PrefSIENA and SIENA. We also evaluate the overheads introduced by ranking and diversifying.

### *7.2.1 Heuristic Performance*

To evaluate the performance of our diversity heuristic, we compare it against the brute-force method that finds optimal solutions. We compare these methods both in terms of the quality of produced results as well as the required time to produce them. The complexity of both methods depends on the number $n$ of candidate events to choose from and on the required number $k$ of events. We experiment with a number of different values for $n$ and $k$. However, the high complexity of the brute-force algorithm prevents us from using large values for these two parameters. Therefore, we limit our study to $n = 10, 20, 30$ and $k = 4, 8, 12, 16, 20$. The results of our experiment are summarized in Table 1 (time is measured in milliseconds).

The complexity of the brute-force method is so high that even with the relatively small values of $n = 30$ and $k = 8$, the required time climbs up to 1967339 ms ($\approx 0.5$ hour). Our heuristic required only 38 ms in this case. This reduction in time complexity comes at the cost of decreased diversity of the results. However, this reduction is only marginal, as the set diversity of the results produced by the heuristic is decreased by less than 1% in all cases.

### *7.2.2 Number and quality of delivered events*

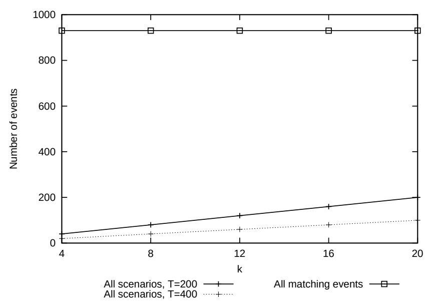
One of the reasons that motivated ranked delivery was the need to reduce the large amount of events delivered to users in a traditional publish/subscribe system. Therefore, in this set of experiments, we first measure the total number of delivered events and then evaluate their quality in terms of average rank and diversity.

Since the number and quality of events depend on the order of publications with regard to their ranks, we consider a number of different event-scenarios. In particular, in the "Best-First" scenario, the highest-ranked events are published first, while in the "Best-Last" scenario, these events are published after the lower-ranked ones. In the "Burst" scenario, we consider that there exist bursts of highly-ranked events at specific moments in time and finally, in the "Random" scenario, high and low ranked events are interleaved. For comparison, besides top-$k$ delivery, we also consider the case in which all matching events are delivered to the users, as in the case of a traditional publish/subscribe system. All scenarios consist of 2000 events out of which 930 match the subscriptions.

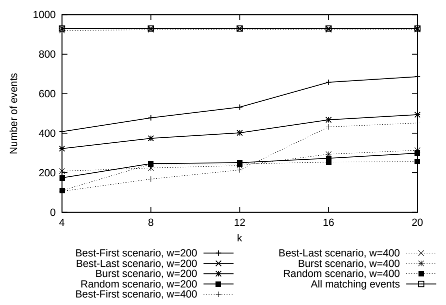
NUMBER OF DELIVERED EVENTS. We measure the number of events delivered to a specific subscriber using PrefSIENA as a function of the number $k$ of the top results the subscriber is interested in. We first consider the case where events are selected based solely on event ranks (i.e. $\sigma = 1.0$). In Figure 10a, we show the number of delivered events for the periodic delivery policy. For comparison reasons, we consider a constant rate of publications and run this experiment for periods with $T = 200$ and $T = 400$ events. We see that the number of delivered events does not depend on the used scenario, since at each period this number is bounded by $k$. Therefore, we achieve a constant rate of event delivery. On average, the number of events delivered by PrefSIENA ranges from 2.2% to 21.5% of all matching events for the various values of $k$ and $T$.

In Figures 10b and 10c, we present the total number of delivered events for the sliding-window and history-based filtering policies. We run the experiment for all of the above scenarios and use window lengths of $w = 200$ and $w = 400$ events. We see that when those policies are used, the number of delivered events depends not only on $k$ and $w$ but also on the input. For the sliding-window policy, we observe that a larger window size leads to the delivery of fewer events. The best pruning is achieved when the "Random" scenario is used. This happens because in this case, there are always some highly preferable events in the current window to filter out less preferable ones. The worse pruning, as expected, is observed in the case of the "Best-Last" scenario, since in this case every new event is better than the old ones and is thus forwarded to the user. In the case of the history-based filtering, the "Best-Last" scenario also has the worse pruning while the "Random" one achieves the greatest pruning. Once again, a larger window length results in greater reduction of delivered events.

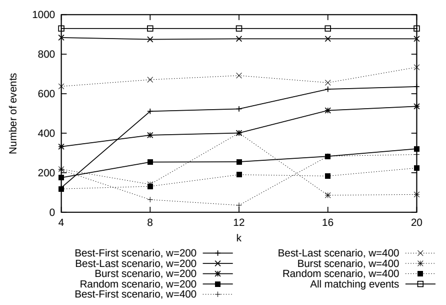
When diversity is also a factor for choosing which events



(a) Periodic delivery policy

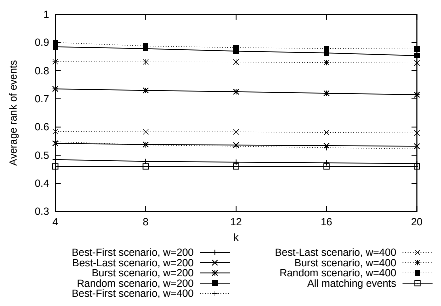


(b) Sliding-window delivery policy

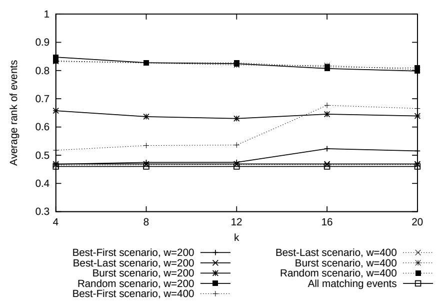


(c) History-based delivery policy

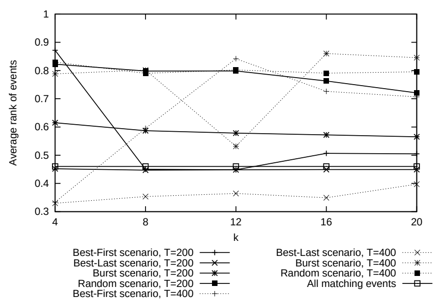
**Figure 10: Total number of delivered events ($\sigma = 1.0$ - no diversity).**



(a) Periodic delivery policy

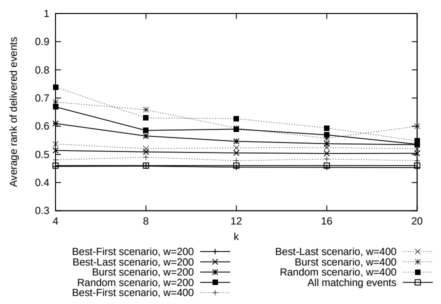


(b) Sliding-window delivery policy

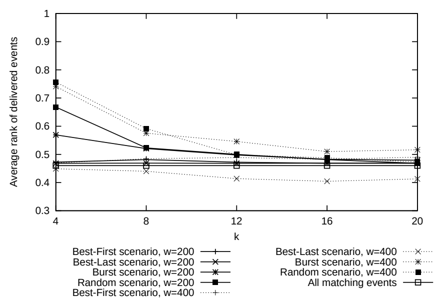


(c) History-based delivery policy

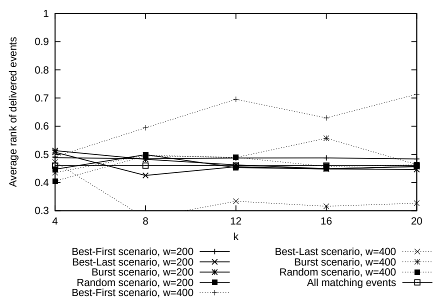
**Figure 11: Average rank of delivered events ($\sigma = 1.0$ - no diversity).**



(a) Periodic delivery policy

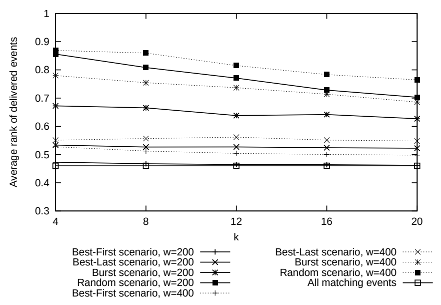


(b) Sliding-window delivery policy

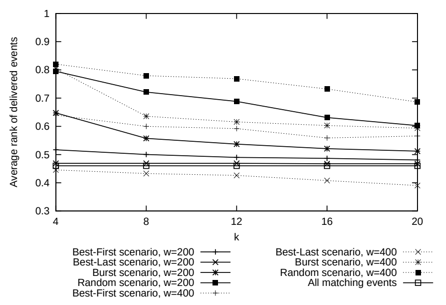


(c) History-based delivery policy

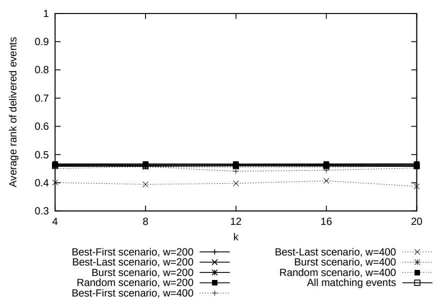
**Figure 12: Average rank of delivered events ($\sigma = 0.0$ - no ranking).**



(a) Periodic delivery policy

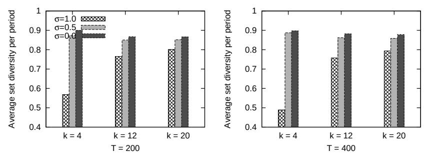


(b) Sliding-window delivery policy

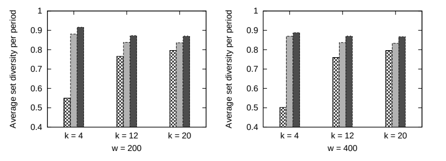


(c) History-based delivery policy

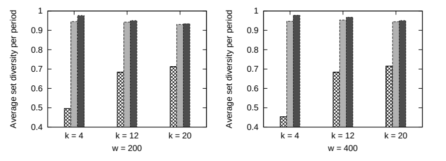
**Figure 13: Average rank of delivered events ($\sigma = 0.5$).**



(a) Periodic delivery policy



(b) Sliding window delivery policy



(c) History-based delivery policy

**Figure 14: Average diversity - random scenario.**

to deliver, only the sliding-window and history-based filtering policies are affected. In the periodic policy, while the delivered events may actually be different than in the no-diversity case, their total number remains the same. In the sliding-window case, the total number of delivered results is slightly larger due to Proposition 3, while in the history-based case the number depends on the input (we omit the relative figures).

Quality of delivered events. We also run a set of experiments to evaluate the quality of the delivered events. We characterize quality based on two factors: (i) the average rank of delivered events and (ii) their diversity.

Figure 11 depicts the average rank of all the delivered events for the various timing policies and scenarios when $\sigma = 1.0$ (no diversity case). Average rank is computed over all events delivered in each delivery policy (as opposed to over the same number of top-ranked events). Generally, we observe that the average rank depends on the input. The average rank of all matching events is 0.46. In PrefSIENA, even though in the presence of many high-ranked events some of them may fail to appear in the top-$k$ results, the average rank is larger than that in all cases. When diversifying the events, there is a decrease of the average rank, since diverse events may have lower ranks, as expected (Figures 12 and 13). The average rank of events decreases along with $k$, since for large values of $k$ more events are delivered to the users. This decrease is more evident when diversity is used.

In Figure 14, we measure the average diversity of the events that are forwarded to a user for the "Random" scenario. Average diversity is computed over the events delivered in each period or window. We run this experiment for different period and window lengths using our diversification methods with $\sigma = 1.0$ (no diversity case), $\sigma = 0.5$ and $\sigma = 0.0$ (no ranking case). We see that the produced results do indeed exhibit a higher diversity when they are chosen based not only on their ranks but also on their distance from each other. This increase is larger for smaller values of $k$. Similar behavior can be observed for the other scenarios as well. Due to space limitations, we omit the related figures.

### 7.2.3 *Performance*

Finally, we perform a number of experiments to evaluate the performance of PrefSIENA. There are two sources of extra overhead for implementing ranked delivery of events. First, to compute the importance of a new event, we have to locate all matching subscriptions, while in traditional publish/subscribe systems it suffices to locate the most general one. Second, there is also the overhead of maintaining state for previously forwarded events and performing computations to decide whether a new event belongs in the diverse top-ranked results or not.

The matching overhead depends on the relations among the various user subscriptions. More specifically, the overhead is more evident when users issue many subscriptions that cover each other, i.e. users refine their previously made subscriptions. To compute this overhead, we perform the following experiment: we construct a number of profiles in which a percentage $c$ of user subscriptions are covered by some other subscription. We also construct a number of scenarios in which a percentage $m$ of the published events match the user subscriptions. In Table 2, we see the number of $PSG$ nodes checked during the execution of each scenario for each of the user profiles in SIENA and PrefSIENA, with $c = 0\%, 10\%$, 30%, 50% and $m = 0\%$, 10%, 20%.

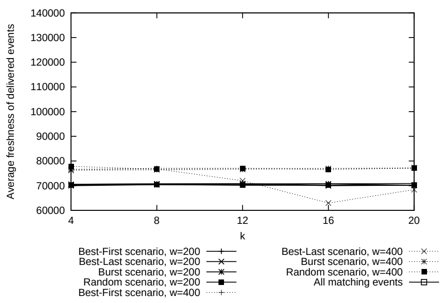
**Table 2: Matching overhead.**

| $m$ | $c$ | SIENA | PrefSIENA |
|---|---|---|---|
| 0% | 0% | 10000 | 10000 |
| | 10% | 9000 | 9000 |
| | 30% | 7000 | 7000 |
| | 50% | 5000 | 5000 |
| 10% | 0% | 10000 | 10000 |
| | 10% | 9000 | 9010 |
| | 30% | 7000 | 7010 |
| | 50% | 5000 | 5010 |
| 20% | 0% | 10000 | 10000 |
| | 10% | 9000 | 9020 |
| | 30% | 7000 | 7020 |
| | 50% | 5000 | 5010 |

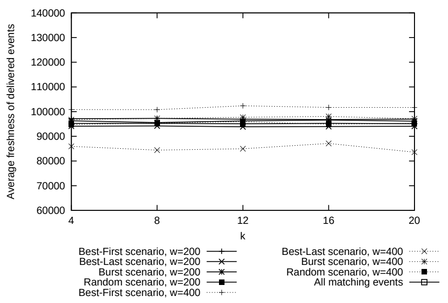
To measure the time overhead introduced by the ranking and diversifying algorithms, we measure the time between the publication and the delivery of each event (Figure 15). In the periodic policy, the sequence of the published events influences the freshness of the delivered ones. For example, if high-ranked events are published towards the end of a period, they will reach the user earlier than if they are published at the beginning. As expected, a larger period length results in larger delays between publication and delivery. In the sliding-window delivery policy, a larger window length increases the average delivery time. This happens (i) because an event remains in the window for longer and therefore, it has more opportunities to enter the top-$k$ results and (ii) because the complexities of the ranking and diversifying algorithms depend on the window size. In the history-based filtering delivery policy, the freshness of data does not depend much on the scenario, but is rather more influenced by the size of the window.

## 8. RELATED WORK

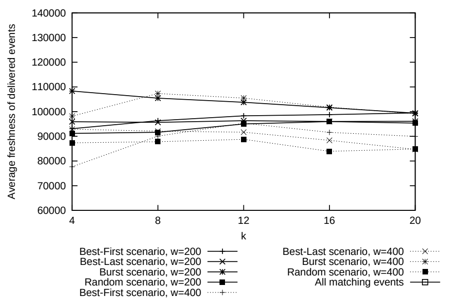
Although there has been a lot of work on developing a variety of publish/subscribe systems, there has been only little work on the integration of ranking issues into publish/subscribe. Recently, in [17], the problem of ranked publish/subscribe systems is also considered. However, the problem is viewed in a different way. In a sense, the authors consider the "reverse" or "dual" problem. Instead of locating the most relevant events to each subscription, the authors aim at recovering the most relevant matching subscriptions to a published event. Subscriptions are modeled as sets of interval ranges in some dimensions and events as points that match all the intervals that they stab. Another work that also deals with the problem of ranked publish/subscribe is [18]. In the proposed model, a subscriber receives the $k$ most relevant events per subscription within a window $w$ which can be either time-based or event-based. For each user subscription, a queue is maintained. This queue buffers those events that are relevant to the subscription and have a high probability to enter the top-$k$ result at some point in the future. The focus is on efficiently maintaining this buffer queue. Here, we aim at specifying and computing event ranks. [24] considers the case where only a subset of top-ranked publishers provide notifications for a specific query. These publishers are ranked according to the similarity of their past publications to the query. Similarity is



(a) Periodic delivery policy



(b) History-based delivery policy



(c) Sliding window delivery policy

**Figure 15: Average freshness of delivered events ($\sigma = 0.5$).**

computed via IR techniques. [21] suggests using an extensive preference model to enhance expressiveness of subscription matching, while [16] proposes an approximate matching mechanism so that relevant events are delivered even if they do not match exactly the users' subscriptions.

In terms of diversity, in [23], a method for topic diversification is proposed for recommendations. The intra-list similarity metric is introduced to assess the topical diversity of a given recommendation list. An algorithm that considers the candidate items' scores is provided for creating lists with small intra-list similarity. The notion of diversity is also explored in [20]. Motivated by the fact that some database relation attributes are more important to the user, a method is proposed where a recommendation list consisting of database tuples is diversified by first varying the values of higher priority attributes before varying the values of lower priority ones. In case the tuples are associated with scores, a scored variation of diversity always picks tuples with higher scores first.

A preliminary version of our preference model (Section 3) was presented in a workshop [8].

## 9. CONCLUSIONS

Our overall goal in this paper has been to increase the quality of events received by the users of publish/subscribe systems in terms of their importance or relevance and diversity. Ranking events by importance is achieved by letting users express preferences along with their subscriptions. Events that match more preferable subscriptions are ranked higher than events that match less preferable ones. For ranking an event, we also take into account how different the event is from the other top-ranked ones so that the overall diversity among the event notifications is increased. We have examined a number of policies with regards to the time range over which the top-$k$ events are computed, namely a periodic, a sliding-window and a history-based one.

Our overall focus has been on increasing the value of the events received by each user. There are many issues for future work, mainly regarding performance and scalability. We are currently working on indexing structures towards making matching of events and ranking more efficient.

## 10. REFERENCES

[1] *The Internet Movie Database*. http://www.imdb.com.

[2] *Movies dataset*. http://had.co.nz/data/movies.

[3] *PrefSIENA*. http://www.cs.uoi.gr/∼mdrosou/PrefSIENA.

[4] *SIENA*. http://serl.cs.colorado.edu/∼serl/dot/siena.html.

[5] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD Conference*, pages 297–306, 2000.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. on Computer Syst.*, 19:332–383, 2001.

[7] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[8] M. Drosou, E. Pitoura, and K. Stefanidis. Preferential publish/subscribe. In *PersDB*, pages 9–16, 2008.

[9] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48 – 60, 1990.

[10] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of $p$-dispersion heuristics. *Computers & OR*, 21(10):1103–1113, 1994.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[12] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *SIGMOD Conference*, pages 115–126, 2001.

[13] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyratos. Efficient rewriting algorithms for preference queries. In *ICDE*, pages 1101–1110, 2008.

[14] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[15] G. Koutrika and Y. E. Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, pages 841–852, 2005.

[16] H. Liu and H.-A. Jacobsen. Modeling uncertainties in publish/subscribe systems. In *ICDE*, pages 510–522, 2004.

[17] A. Machanavajjhala, E. Vee, M. N. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. *PVLDB*, 1(1):451–462, 2008.

[18] K. Pripuzic, I. P. Zarko, and K. Aberer. Top-k/w publish/subscribe: finding k most relevant publications in sliding time window w. In *DEBS*, pages 127–138, 2008.

[19] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *ICDE*, pages 846–855, 2007.

[20] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008.

[21] Q. Wang, W.-T. Balke, W. Kießling, and A. Huhn. P-news: Deeply personalized news dissemination for mpeg-7 based digital libraries. In *ECDL*, pages 256–268, 2004.

[22] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130, 2008.

[23] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

[24] C. Zimmer, C. Tryfonopoulos, K. Berberich, M. Koubarakis, and G. Weikum. Node behavior prediction for large-scale approximate information filtering. In *LSDS-IR*, 2007.