Preferential Publish/Subscribe

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

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

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

ABSTRACT

In publish/subscribe systems, subscribers express their interests in specific events and get notified about all published events that match their interests. As the amount of information generated increases rapidly, to control the amount of data delivered to users, we propose enhancing publish/subscribe systems with a ranking mechanism, so that only the top-ranked matching events are delivered. Ranking is based on letting users express their preferences on events by ordering the associated subscriptions. To avoid the blocking of new notifications by top-ranked old ones, we associate with each notification an expiration time. We have fully implemented our approach in SIENA, a popular publish/subscribe middleware system.

1. INTRODUCTION

The publish/subscribe paradigm provides loosely coupled interaction among a large number of users of a large-scale distributed system. Users can express their interest in an event via a *subscription* and inject this subscription into the system. The system will then *notify* them whenever some other user generates (or *publishes*) an event that *matches* a previously made subscription. Users that generate such events are called *publishers*, while users that consume the published events are called *subscribers*. All published events that are relevant to at least one of a specific user's subscription will eventually be delivered to this user.

Typically, in publish/subscribe systems, all subscriptions are considered equally important. However, due to the abundance of information, users may receive overwhelming amounts of event notifications. In such cases, users would prefer to receive only a fraction of this information, namely the most interesting to them. For example, assume a user, say John, who is generally interested in drama movies. Specifically, John is more interested in drama movies directed by Tim Burton than drama movies directed by Steven Spielberg. Ideally, John would like to receive notifications about Steven Spielberg drama movies only in case there are no, or

not enough, notifications about Tim Burton drama movies.

In this paper, we advocate using some form of ranking among subscriptions, so that users can express the fact that some events are *more important* to them than others. To rank subscriptions, we use preferences. A variety of preference models have been proposed, most of which follow either a quantitative or a qualitative approach. In the *quantitative approach* (e.g. [5, 13]), users employ scoring functions that associate a numeric score with specific data to indicate their interest in it. In the *qualitative approach* (e.g. [8, 12, 11]), preferences between two data items are specified directly, typically using binary preference relations. To express priorities among subscriptions, we first introduce *preferential subscriptions*, that is, subscriptions enhanced with interest scores following the quantitative preference paradigm. Based on the subscription scores, we propagate to users only the notifications that are the most interesting to them. We extend this idea to encompass qualitative preferences.

Based on preferential subscriptions, we introduce a top-k variation of the publish/subscribe paradigm in which users receive only the k most interesting events as opposed to all events matching their subscriptions. Since the delivery of notifications is continuous, we introduce a timing dimension to the top-k problem, since without some notion of freshness, receiving top-ranked notifications would block for ever the delivery of any new, less interesting notifications. To this end, we associate an expiration time with each notification, so that, notifications for old events will eventually die away and let new ones be delivered to the users.

To locate the subscriptions that match a specific event notification efficiently, we adopt a graph-based representation of subscriptions, called *preferential subscription graph*. Subscriptions correspond to nodes in the graph and edges point from more general to more specific subscriptions.

Our prototype implementation, PrefSIENA [3], extends SIENA [6], a popular publish/subscribe middleware system, by including preferential subscriptions and top-k notification delivery. We report some preliminary experimental results that compare the number of notifications delivered by PrefSIENA with respect to the corresponding number in the case of the original SIENA system.

The rest of this paper is structured as follows. In Section 2, we introduce preferential subscriptions, that is, subscriptions augmented with interest scores. We also propose time-valid notifications by associating expiration times with notifications. In Section 3, we focus on how to compute the top-k notifications based on preferential subscriptions and time-valid notifications, while in Section 4, we extend pref-
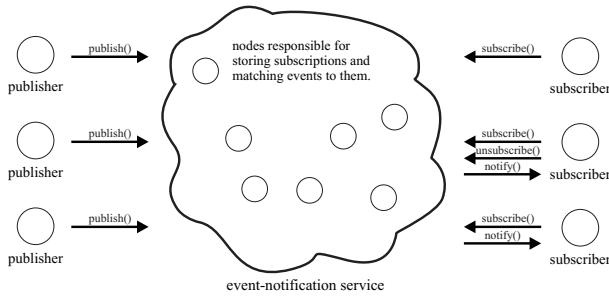
Figure 1: Basic publish/subscribe system.

erential subscriptions to encompass qualitative preferences. In Section 5, we present our evaluation results. Section 6 describes related work and finally, Section 7 concludes the paper with a summary of our contributions.

2. PREFERENTIAL MODEL

In this section, we first describe a typical form of notifications and subscriptions used in publish/subscribe systems. Then, we introduce an extended version of subscriptions that include the notion of preferences. Also, we extend the definition of notifications to include the notion of time validity.

2.1 Publish/Subscribe Preliminaries

A publish/subscribe system is an event-notification service designed to be used over large-scale networks, such as the Internet. Generators of events, called *publishers*, can publish event notifications to the service and consumers of such events, called *subscribers*, can subscribe to the service to receive a portion of the published notifications. Publishers can publish notifications at any time. The notifications will be delivered to all interested subscribers at some point in the future.

Architecture: In general, a publish/subscribe system [9] consists of three parts: (i) the publishers that provide events to the system, (ii) the subscribers that consume these events and (iii) an event-notification service that stores the various subscriptions, matches the incoming event notifications against them and delivers the notifications to the appropriate subscribers. As shown in Figure 1, the event-notification service provides a number of primitive operations to the users. The **publish**() operation is called by a publisher whenever it wishes to generate a new event. The **subscribe**() operation is called by a subscriber whenever it wishes to express a new interest. An **unsubscribe**() operation is usually also provided to cancel previous subscriptions. The event-notification service can use the **notify**() operation whenever it wants to deliver a notification to a subscriber. An event-notification service can be implemented using a centralized or a distributed architecture, that is, we may have one or a set of servers responsible for the process of matching notifications to subscriptions.

Notifications: We use a generic way to form notifications, similar to the one used in [6, 10]. In particular, notifications are sets of typed attributes. Each notification 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 accord-

string title	=	LOTR: The Return of the King
string director	=	P. Jackson
time release_date	=	1 Dec 2003
string genre	=	fantasy
integer oscars	=	11

Figure 2: Notification example.

| string director | = | P. Jackson |
| time release_date | > | 1 Jan 2003 |

Figure 3: Subscription example.

ing to their type. An example notification about a movie is shown in Figure 2.

DEFINITION 1 (NOTIFICATION). *A notification n is a set of typed attributes $\{a_1, \ldots, a_p\}$, where each a_i, $1 \le i \le p$, is of the form (a_i.type a_i.name $= a_i$.value).*

Subscriptions: Subscriptions are used to specify the kind of notifications 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 notifications. Binary operators may include common operators such as $=, \neq, <, >, \le, \ge$, $substring$, $prefix$ and $suffix$. An example subscription is depicted in Figure 3.

DEFINITION 2 (SUBSCRIPTION). *A subscription s is a set of attribute constraints $\{b_1, \ldots, b_q\}$, where each b_i, $1 \le i \le q$, is of the form (b_i.type b_i.name θ_{b_i} b_i.value), $\theta_{b_i} \in \{=, <, >, \le, \ge, \neq, substring, prefix, suffix\}$.*

Matching notifications to subscriptions: Intuitively, we can say that a notification n *matches* a subscription s, or alternatively a subscription s *covers* a notification n, if and only if every attribute constraint of s is satisfied by some attribute of n. Formally:

DEFINITION 3 (COVER RELATION). *Given a notification n of the form $\{a_1, \ldots, a_p\}$ and a subscription s of the form $\{b_1, \ldots, b_q\}$, s covers n if and only if $\forall b_i \in s$, $\exists a_j \in n$ such that b_i.type $= a_j$.type, b_i.name $= a_j$.name and it holds ($(a_j.value)$ θ_{b_i} $(b_i.value)$), $1 \le i \le p$, $1 \le j \le q$.*

A notification n is delivered to a user if and only if the user has submitted at least one subscription s, such that s covers n. For example, the subscription of Figure 3 covers the notification of Figure 2 and therefore, this notification will be delivered to all users who have submitted this subscription.

2.2 Preferential Subscriptions

In this paper, we extent the publish/subscribe paradigm to incorporate ranking capabilities. Assuming that each user has defined a set of preferences, then this user should receive a newly published notification, if and only if, the notification describes an event that is more preferable to the user than any previously received event. To express preferences along with subscriptions, we can follow either the quantitative or the qualitative approach. For simplicity reasons, we first consider the quantitative preference model, while in Section 4, we apply qualitative preferences.

A *preferential subscription* is a subscription enhanced with a numeric score. The higher the score, the more interested

| string director | = | P. Jackson | 0.7 |
| date release_date | > | 1 Jan 2003 | |

Figure 4: Preferential subscription example.

the user is in notifications covered by this specific subscription. These scores can have any real value. We assume here that they take values within the range $[0, 1]$. An example of a preferential subscription is shown in Figure 4.

DEFINITION 4 (PREFERENTIAL SUBSCRIPTION). *A preferential subscription ps_i^X, submitted by user X, is of the form $ps_i^X = (s_i, score_i^X)$, where s_i is a subscription and $score_i^X$ is a real number within the range $[0, 1]$.*

Assuming that a user X defines a set of preferential subscriptions P^X, we use the user's preferential subscriptions to rank the published notifications and deliver to the user only the top-k notifications, i.e. the k highest ranked ones (where k is a user-defined parameter). We define the score of a notification to be the largest among the scores of the subscriptions that cover it:

DEFINITION 5 (NOTIFICATION SCORE). *Assume a notification n, a user X and the set P^X of the user's preferential subscriptions. Assume further the set $P_n^X = \{(s_1, score_1^X), \ldots, (s_m, score_m^X)\}$, $P_n^X \subseteq P^X$, for which s_i covers n, $1 \leq i \leq m$. The notification score of n for X is equal to $sc(n, X) = max \{score_1^X, \ldots, score_m^X\}$.*

A newly published notification n is delivered to a user X, if and only if, it is covered by some subscription s previously issued by X and X has not already received k notifications more preferable to n. A notification n_1 is more preferable for user X to a notification n_2, if and only if, it has a higher notification score for X than n_2.

In general, we assume that scores are indicators of positive interest, thus, we use the maximum value of the corresponding subscriptions. One could argue for other ways of aggregating the scores, for instance using the mean, minimum or a weighted sum. Yet, an alternative way would be to use only the scores of a subset of P_n^X, namely the *most specific* subscriptions. A similar notion for preferences is introduced in [15]. For example, assume the notification of Figure 2 and the preferential subscriptions $(\{genre = fantasy\}, 0.7)$ and $(\{genre = fantasy, director = P. Jackson\}, 0.6)$ (for ease of presentation we omit the type of each attribute). The second subscription is more specific than the first one, in the sense that in the second subscription the user poses an additional, more specific requirement to movies than in the first one, and so, the score of the first one should be ignored. Formally, a subscription $s \in P_n^X$ is a most specific one if no other subscription in P_n^X covers it (see Definition 7).

We leave as future work a user study to evaluate the appropriateness of the different methods for assigning scores. In general, all such methods may increase the complexity of the process of matching notifications to subscriptions, since in traditional publish/subscribe, for matching to be completed successfully, it suffices to find just one subscription that covers the notification, whereas for computing the notification score, we may need to locate all covering subscriptions.

2.3 Time-Valid Notifications

In a publish/subscribe system, where new event notifications are constantly produced, the following problem may
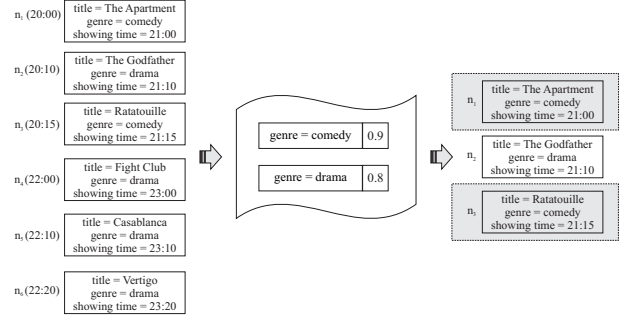

Figure 5: Top-2 notifications for a single user at 22:30 (no expiration time used).

arise: it is possible for very old but highly preferable notifications to prevent newer notifications from reaching the user. Specifically, after receiving k very highly preferable notifications, the user will receive no new ones unless they are ranked higher, something that may not be desirable.

For instance, consider the example of Figure 5. For simplicity, we assume a single user, say John, who has defined the following preferential subscriptions for movies: John has assigned score 0.9 to comedies and score 0.8 to dramas. Assume that a movie theater generates the notifications $n_1, n_2, \ldots, n_6$ of Figure 5 in that order and that John is interested in the top-2 results. n_1 will be delivered to John, since it is the first notification that is covered by his subscriptions. n_2 will also be delivered, since it is the second best result seen up to this moment. n_3 is equally preferred to n_1 and will therefore also be delivered to John (replacing n_2 in the current top-2 results). Since n_4, n_5 and n_6 are less preferable to the current top-2 results, none of them will be delivered to John. Assuming that notifications are published one hour prior to the showing time, if John checks his top-2 results at 22:30 he will only find movies that he can no longer watch (the top-2 results at 22:30 are marked with gray color), even though other interesting movies that start at 23:00 have been published.

To overcome this problem, we need to define the subset of notifications over which the top-k notifications for each user will be located. One solution would be to split time in periods of duration T and at each time instance, deliver a notification to the user, if the user has not already received, during the current period, k notifications with higher scores. However, since top-k computation starts anew in the beginning of each period, the rank of events received by the user may end up being rather arbitrary. For example, high-ranked notifications appearing in periods with many other high-ranked ones may not be delivered to the user, whereas low-ranked publications appearing in periods with a small number of high-ranked ones may be delivered.

A more general approach is to associate each published notification n with an expiration time $n.exp$. The notification is considered *valid* only while $n.exp$ has not expired. The top-k results for each user are defined over the subset of valid notifications. This way, older notifications which have expired do not prevent valid ones from reaching the user even if they are more preferable than those. Considering the previous example, assume that each notification expires at the showing time of the corresponding movie (see Figure 6). n_1, n_2 and n_3 will be delivered to the user as
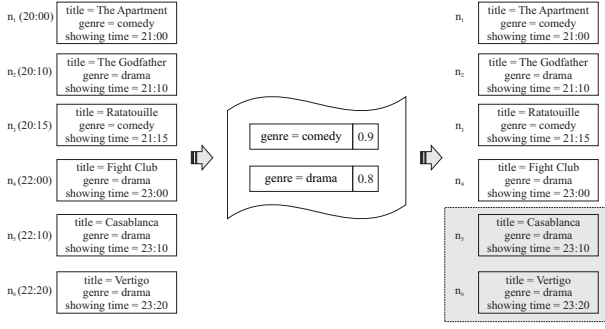
Figure 6: Top-2 notifications for a single user at 22:30 (with expiration time used).

Figure 7: Preferential subscription graph example.

before. By the time n_4 is published (22:00), n_1, n_2 and n_3 have expired and therefore, n_4 will also be delivered to the user. n_5 and n_6 will be delivered as well, since they are equally preferred as the notifications in the top-2 at the time of their publication. Notice that the periodic approach is a special case of the expiration-time one. By setting the expiration time of each notification equal to the ending time of the current period, we achieve the same result as with the periodic approach.

Based on the assumption that published notifications are valid only for a specific time period, next we define which ones belong to the top-k notifications for a user.

DEFINITION 6 (TOP-k NOTIFICATIONS). *Assume a user X and P^X the set of X's preferential subscriptions. A notification n published at time t belongs to the top-k notifications of X, if and only if, n is covered by at least one subscription s appearing in a preferential subscription $ps \in P^X$ and X has not already received k notifications $n_1, \ldots, n_k$ with $n_i.exp > t$ and $sc(i, X) > sc(n, X)$, $1 \leq i \leq k$.*

An alternative way to set the expiration time for a notification would be to let the user define a refresh time along with each subscription. This can be also expressed through defining appropriate values of the expiration time for notifications as follows. Assuming that a notification n is covered by a user subscription s associated with a refresh time r, then the expiration time of n could be set to $t + r$, where t is the time that n is sent to the user. Note that in this approach, a specific notification does not have a single expiration time but instead, it is associated with a different one for each user.

3. RANKING IN PUBLISH/SUBSCRIBE

In this section, we introduce a *preferential subscription graph* for organizing our preferential subscriptions. We also present an algorithm for computing the top-k results and discuss the server topology.

3.1 Preferential Subscription Graph

To reduce the complexity of the matching process between notifications and subscriptions, it is useful to organize the subscriptions using a graph. We use preferential subscriptions to construct a directed acyclic graph, called *preferential subscription graph*, or *PSG*. To form such graphs, we use the cover relation between subscriptions defined as follows.
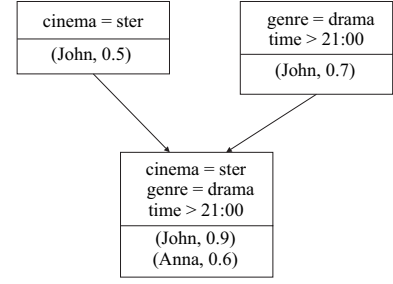
DEFINITION 7 (COVER BETWEEN SUBSCRIPTIONS). *Given two subscriptions s_i and s_j, s_i covers s_j, if and only if, for each notification n such that s_j covers n, it holds that s_i covers n.*

For example, the subscription $\{genre = fantasy\}$ covers the subscription $\{genre = fantasy, director = P. Jackson\}$.

In a preferential subscription graph, nodes correspond to subscriptions and edges to cover relations between subscriptions. Assume the set P of all preferential subscriptions, i.e. the preferential subscriptions defined by all users. For each subscription $s_i \in S_P$, where S_P is the set of all subscriptions in P, we maintain a set of pairs, called *score set*, of the form $(j, score_i^j)$, where j is a user and $score_i^j$ is the numeric score that j has assigned to s_i. A subscription s_i is associated with the pair $(j, score_i^j)$, if and only if, a preferential subscription $ps_i^j = (s_i, score_i^j)$ exists in P. Next, we define formally the score set of a subscription.

DEFINITION 8 (SCORE SET). *Assume a set of users U, a set of preferential subscriptions P, and S_P the set of all subscriptions in P. For each $s_i \in S_P$, the score set is the set $W_i = \{(j, score_i^j) \mid (s_i, score_i^j) \in P \}$.*

Having defined the score set of a specific subscription, we now define the preferential subscription graph.

DEFINITION 9. (PREFERENTIAL SUBSCRIPTION GRAPH). *Let P be a set of preferential subscriptions and S_P 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 $s_i \in S_P$, there exists a node v_i, $v_i \in V_P$, of the form (s_i, W_i), where W_i is the score 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, assume two users, John and Anna, who express the following preferential subscriptions: John gives to subscription $s_1 = \{cinema = ster, genre = drama, time > 21:00\}$ score 0.9, to $s_2 = \{genre = drama, time > 21:00\}$ score 0.7 and to $s_3 = \{cinema = ster\}$ score 0.5. Similarly, Anna assigns to s_1 score 0.6. For the above preferential subscriptions, the graph of Figure 7 is constructed.

The preferential subscription graph resembles the *filters poset* data structure proposed in [6]. Whereas the filters poset represents a partially ordered set of subscriptions, the preferential subscription graph is based on subscriptions enhanced with interest scores.

3.2 Forwarding Notifications

To show how the top-k results for each user are computed, we first assume a single server maintaining a preferential

subscription graph PSG. In the next section, we generalize our approach for more servers. This single server acts as an access point for all subscribers and publishers. Although publish/subscribe systems are typically stateless, in the sense that they do not maintain any information about previous notifications, here, we need to maintain some information about previously sent top-ranked notifications. The server maintains a list of k elements for each of the subscribers (users) that are connected to it. These lists contain elements of the form $(score, expiration)$ where $score$ is a numeric value and $expiration$ is a time field. The $score$ part of such a pair represents the score of a notification that has already been delivered to the corresponding user and expires at time $expiration$. Only the scores corresponding to the top-k most preferable valid notifications that have been already sent to the users appear in these lists.

All lists are initially empty. Whenever the server receives a notification n, it walks through its PSG to find all subscriptions that cover n. For each subscriber j associated with at least one of these subscriptions, a score $sc(n,j)$ is computed: assuming that m subscriptions $s_1, s_2, \ldots, s_m$ submitted by j cover n, then $sc(n,j) = max \{score_1^j, score_2^j, \ldots, score_m^j\}$. After that, the corresponding list, denoted $list^j$, is checked and all elements which have expired are removed. If $list^j$ contains less than k elements, n is forwarded to j and the pair $(sc(n,j), n.exp)$ is added to the list, where $n.exp$ is the expiration time of n. Otherwise, n is forwarded to j only if $sc(n,j)$ is greater or equal to the score of the element with the minimum score in the list. In this case, this element is replaced by $(sc(n,j), n.exp)$. Note that, a more recent notification equally important to an older one is forwarded to the user to favor fresh data over equally-ranked old ones. The process described above is summarized in the *Forward Notification Algorithm* shown in Algorithm 1.

Next, we prove the completeness and correctness of Algorithm 1. First, we will show that if a notification n belongs to the top-k results of user j, then it will be forwarded to j. Assume for the purpose of contradiction, that such a notification is not forwarded to j. Let $sc(n,j)$ be the score of n for j. Since n is not forwarded to j, there exist k valid notifications $n_1, \ldots, n_k$ with scores $sc(n_1, j), \ldots, sc(n_k, j)$ such that $sc(n_i, j) > sc(n,j)$, $1 \le i \le k$. This means that n does not belong to the top-k results of user j, which violates our assumption. Next, we proceed with showing that if a notification n is forwarded to j, then it belongs to the user's top-k results. For the purpose of contradiction, assume that n does not belong to the user's top-k results. This means that there exist k valid notifications $n_1, \ldots, n_k$ with scores $sc(n_1, j), \ldots, sc(n_k, j)$ such that $sc(n_i, j) > sc(n,j)$, $1 \le i \le k$. Therefore, according to Algorithm 1 (line 21), n will not be forwarded to j, which is a contradiction.

Note that it is not necessary to walk through all nodes of the preferential subscription graph to locate the subscriptions that cover a specific notification. 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 at the same time s' covers n. This way, entire paths of the graph can be pruned and not used in the matching process.

3.3 Hierarchical Topology of Servers

An event-notification service can be implemented over various architectures. At one extreme, a centralized approach can be followed, e.g. [10]. In this case, a single server

---

Algorithm 1 Forward Notification Algorithm

Input: A notification n and a preferential subscription graph PSG.
Output: The set of subscribers $ResSet$ n will be forwarded to.

1: **Begin**
2: $ResSet = \varnothing$;
3: $tmpW = \varnothing$; /* temporary score set */
4: **for all** nodes v_i in PSG **do**
5:   **if** s_i covers n **then**
6:     $tmpW = tmpW \cup W_i$;
7:   **end if**
8: **end for**
9: **for all** subscribers j that appear in $tmpW$ **do**
10:   $sc(n,j) = max\{score_1^j, \ldots, score_{m^j}^j\}$, where $(j, score_i^j)$ $\in tmpW$, $1 \le i \le m^j$;
11:   **for all** elements i in $list^j$ **do**
12:     **if** i has expired **then**
13:       remove i from $list^j$;
14:     **end if**
15:   **end for**
16:   **if** $list^j$ contains less than k elements **then**
17:     add $(sc(n,j), n.exp)$ to $list^j$;
18:     $ResSet = ResSet \cup j$;
19:   **else**
20:     find the element i of $list^j$ with the minimum score;
21:     **if** $sc(n,j) > i.score$ **then**
22:       remove i from $list^j$;
23:       add $(sc(n,j), n.exp)$ to $list^j$;
24:       $ResSet = ResSet \cup j$;
25:     **end if**
26:   **end if**
27: **end for**
28: **return** $ResSet$;
29: **End**

---

gathers all subscriptions and notifications and carries out the matching process. However, due to the nature of such systems, where participants are physically distributed across the globe, a distributed architecture is more scalable. When more than one server exists in the network, each server runs Algorithm 1 for its own preferential subscription graph. Notifications are propagated among servers based on the server topology. The servers of the system are responsible for collecting all the published notifications and carrying out the selection process, i.e. delivering each notification only to the subscribers that have declared their interest to it.

We consider a hierarchical topology, where the servers that implement the event-notification service are connected to each other to form a hierarchy. Each publisher and subscriber is connected to one of the servers in the hierarchy.

Furthermore, we wish to organize the participants of the network in an efficient way, i.e. in a way that will reduce the number of messages exchanged between the servers and the complexity of the maintained data structures. One way to achieve this is by placing subscribers with similar subscriptions nearby in the hierarchy, so that the notifications covered by those subscriptions need to be propagated only toward this part of the hierarchy.

While in most publish/subscribe systems, new subscribers randomly select a server to connect to, in our approach, when a new subscriber enters the network it probes a number of servers and chooses one of them according to a number of criteria:

- (*Criterion 1*) The number of new nodes added to the highest level of the server's preferential subscription graph. The smaller the number of such nodes, the
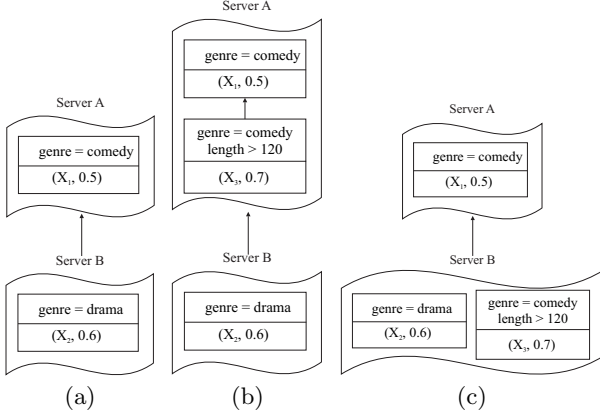
Figure 8: Clustering.

fewer the additional notifications that should be propagated to the server in the future.

- (*Criterion 2*) The number of nodes in the server's preferential subscription graph. The fewer the nodes in the graph, the lower the complexity of searching it.

A new subscriber X chooses a server to subscribe according to the above criteria. For instance, X may first use *Criterion 1*, and in case of a tie, *Criterion 2*. For example, consider the case of Figure 8a where there are two servers, Server A and Server B, both already storing some user subscriptions from subscribers X_1 and X_2. Assume that a new subscriber X_3 wishes to insert a new preferential subscription ($\{genre = comedy, length > 120\}, 0.7$) to the system. If X_3 chooses Server A to subscribe, the result will be the one shown in Figure 8b. If X_3 chooses Server B, the result will be the one shown in Figure 8c. Using the first criterion, X_3 will choose to join Server A because in this case no new nodes will be added to the highest level of the PSG of Server A and thus, no new message traffic will be generated (except from the messages sent from Server A to X_3).

4. SUBSCRIPTIONS USING A QUALITATIVE MODEL

Preferential subscriptions as defined in Section 2.2 exploit the notion of quantitative preferences. Here, we discuss how to express preferential subscriptions using a qualitative preference model.

Assume that a user X provides a set of subscriptions S_P^X. To define choices between subscriptions, X expresses priority conditions of the form $s_i \succ s_j$, $s_i, s_j \in S_P^X$, to denote that s_i is preferred to s_j for X. Let C^X be the set of priority conditions expressed by user X, i.e. $C^X = \{(s_i \succ s_j) \mid s_i, s_j \in S_P^X\}$. To extract the most preferable subscriptions C^X, we use the winnow operator [8]. In particular, the first application of the winnow operator returns the set $win_X(1)$ of subscriptions $s_i \in S_P^X$ such that $\forall s_i \in win_X(1)$ there is no $s_j \in S_P^X$ with $s_j \succ s_i$. If we would like to retrieve further the most preferable subscriptions after the ones included in $win_X(1)$, we apply the winnow a second time. $win_X(2)$ consists of the subscriptions $s_i \in (S_P^X - win_X(1))$ such that $\forall s_i \in win_X(2)$ there is no subscription $s_j \in (S_P^X - win_X(1))$ with $s_j \succ s_i$. The winnow operator may be applied until all

subscriptions are returned. To locate the subscriptions that belong to a specific winnow result set, we define the *multiple level winnow operator*.

DEFINITION 10. (MULTIPLE LEVEL WINNOW OPERATOR). *Assume a user X and let S_P^X be the set of X's subscriptions. Let C^X be the set of priority conditions of X, the multiple level winnow operator at level l, $l > 1$, returns a set of subscriptions, $win_X(l)$, consisting of the subscriptions $s_i \in S_P^X - \cup_{q=1}^{l-1} win_X(q)$ such that $\forall\ s_i \in win_X(l)\ \nexists s_j \in S_P^X - \cup_{q=1}^{l-1} win_X(q)$ with $(s_j \succ s_i) \in C^X$.*

To compute the top-k results for each user when priority conditions between subscriptions are specified, we modify the process described in Algorithm 1 as follows. Again, each server maintains a list of k elements for each user that is connected to it. These lists now contain elements of the form (*pos, expiration*) where *pos* represents a position value and *expiration* is a time field. The *pos* value denotes the winnow level that a subscription that covers a notification which has already been delivered to the user belongs to, and *expiration* the time instance the notification expires. Again, only the elements corresponding to the top-k most preferable valid notifications that have been already sent to the users appear in these lists. In this work, we assume that there are no conflicting priority conditions.

Whenever the server receives a notification n, it walks through its PSG to find all subscriptions that cover n. For each subscriber j associated with at least one of these subscriptions, a value $rank(n, j)$ is calculated: assuming that m subscriptions $s_1, s_2, \ldots, s_m$ submitted by j cover n, then $rank(n, j) = min\ \{level_1^j, level_2^j, \ldots, level_m^j\}$, where $level_i^j$ denotes the winnow level that the subscription s_i belongs to. In the following, the corresponding list, $list^j$, is checked and all expired elements are removed. If $list^j$ contains less than k elements, n is forwarded to j and the pair $(rank(n, j), n.exp)$ is added to the list, where $n.exp$ is the expiration time of n. Otherwise, n is forwarded to j only if $rank(n, j)$ is less or equal to the rank value of the element with the maximum rank in the list. In this case, this element is replaced by $(rank(n, j), n.exp)$.

5. EVALUATION

To evaluate our approach, we have extended the SIENA event notification service [4], a multi-threaded publish/subscribe implementation, to include preferential subscriptions. We refer to our implementation as PrefSIENA. Our source code is available for download at [3].

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, rating, MPAA and genre.

string title	=	LOTR: The Return of the King
integer year	=	2003
integer length	=	251
integer rating	=	9
string mpaa	=	PG-13
string genre	=	Action

Figure 9: Generated notification.

Each publisher randomly selects m_P numbers from 1 to 58788. For each of the corresponding m_P movies, the pub-

lisher creates a new notification consisting of the title, year, length, rating, MPAA and genre of the movie. An example of such a notification can be seen in Figure 9. Each subscriber generates m_S subscriptions and each subscription is generated independently from the others. We randomly select a number of the available attributes to appear in a subscription. The value of each attribute can be generated using either a uniform (i.e. all values are equally preferable) or a zipf distribution (i.e. some values are more popular) according to the values appearing in the dataset. In both cases, a subscription is associated with a numeric score uniformly distributed in $[0, 1]$. Subscription examples can be seen in Figure 10.

string genre	=	Romance	
integer length	>	120	0.3
string mpaa	=	PG-13	

string genre	=	Drama	
integer length	>	100	
integer year	<	1980	0.6
integer rating	>	6	

Figure 10: Generated subscriptions.

Experiments: To run our experiments, we assume a network in which each computer node can act as a publisher, subscriber or server. A combination of these roles is also possible. The servers are organized in a hierarchical topology while clients (i.e. publishers and subscribers) can be connected to any one of the servers. Each involved client executes a series of service requests. More specifically, each publisher generates a number of notifications and injects them into the network. All notifications expire after time τ of their publication. Each subscriber generates a number of subscriptions and chooses a server to connect to and subscribe. After that, each subscriber waits for notifications to arrive.

In general, the number of delivered notifications depends on the covering relations between the various subscriptions and published notifications, the scores associated with these subscriptions and the order in which notifications are generated. The notification receipt rate for each individual user can be fine-tuned by letting the user define appropriate values for refreshing the subscriptions (so that the expiration times of the corresponding notifications are set accordingly) and by selecting k.

First, we measure the number of notifications delivered to a specific subscriber using PrefSIENA as a function of the number k of the top results the subscriber is interested in. We run this experiment with 100 matching events and for expiration time τ equal to $15t$ and $20t$, where $t = 500ms$ is the time length between the generation of two notifications. Note that t refers to real, and not simulated, time. We consider the following two scenarios. In the first one, *scenario 1*, most of the notifications with higher scores for the user are published early, while in the second one, *scenario 2*, notifications with higher scores arrive towards the end. We observe that in the first scenario the user receives fewer notifications than in the second one (Figure 11). This happens because in the first scenario, where notifications with higher scores arrive first, many of the notifications with lower scores cannot enter the top-k results, until some of the first ones expire. In the second scenario, however, the user receives both the
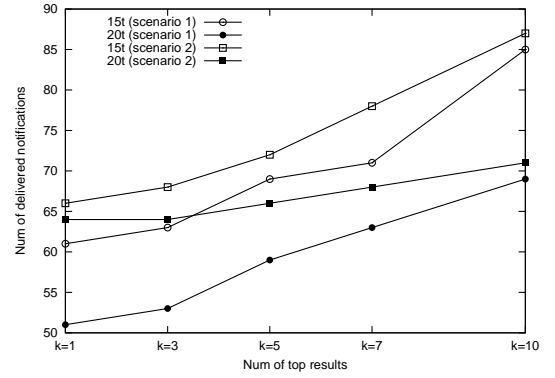

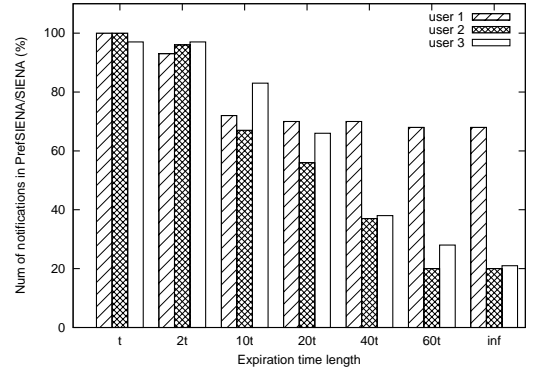Figure 11: Number of delivered notifications for various values of τ and k.

Figure 12: Percentage of delivered notifications for various expiration times (from $t = 500ms$ to infinity)

notifications with the lower scores that arrive first and the notifications with higher scores that arrive later. In both scenarios, the number of delivered notifications decreases with the increase of the expiration time. This happens because notifications with higher scores for the user remain in the top-k results for a longer time period and prevent notifications with lower scores from reaching the user.

Furthermore, we measure the number of notifications delivered to a number of different users in the following cases: (i) using SIENA, (ii) using PrefSIENA with no expiration time for notifications and (iii) using PrefSIENA with a number of different expiration times. The number of published events is 200 and their expiration time τ takes values from t to $60t$, where $t = 500ms$. All subscribers submit 5 different subscriptions and are interested in the top-1 result. We select to show results for three users according to the percentage of notifications that are covered by their subscriptions. The subscriptions of *user 1* cover 51% of the generated notifications. For *user 2* and *user 3*, the percentage values are 27% and 15% respectively. In Figure 12, we depict the results for these three users. We count the percentage of delivered notifications in PrefSIENA over the number of notifications in SIENA. By varying the expiration time, we can achieve different notification receipt rates. This rate depends on the scores of users subscriptions, and also, on the specific time that various notifications in the top-k results expire, as shown by the previous experiment.

We also experimented with using the clustering criteria described in Section 3.3 during bootstrapping. We observed a 27% reduction on average of the total messages exchanged between nodes of the system, even though in this case we have an overhead of extra messages during bootstraping.

6. 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, [18] considers the case of continuous queries in distributed systems. In this approach, only a subset of publishers provide notifications for a specific query. These publishers are selected according to the similarity of their past publications to the query. Similarity is computed via IR techniques. In [17], user preferences are employed to deliver newly added documents of a digital library to the users. Next, we discuss work related to publish/subscribe systems and preferences.

Publish/Subscribe Systems: The publish/subscribe paradigm can be applied to a number of different architectures. The naive approach is to gather all subscriptions and events to a specific node. This node will be responsible for managing subscriptions, matching the incoming events against them and notifying the appropriate subscribers. This is a centralized approach (e.g. [10]). Often, in publish/subscribe systems the number of participating nodes becomes very large. For scalability reasons, a distributed architecture seems to be more suitable. In this case, the event service is implemented via a network of interconnected servers who act as a middle level for the communication of publishers and subscribers. Various distributed architectures such as hierarchical [6] and DHT-based [7] ones, have been proposed.

There are two widely used methods for users to express their subscriptions: the topic-based method and the content-based one. In the *topic-based* method (such as [14]) there are a number of predefined event topics, usually identified by keywords. Published events are associated with a number of topics. Users can subscribe to a number of individual topics and receive all events associated with at least one of these topics. In the *content-based* method [6, 7], such as the one used in this paper, the classification of the published events is based on their actual content. Users express their subscriptions through constraints which identify valid events. An event matches a subscription, if and only if, it satisfies all of the subscription's constraints. In general, the content-based method offers greater expressiveness to subscribers but is harder to implement.

Preferences: The research literature on preferences is extensive. In general, there are two different approaches for expressing preferences: a quantitative and a qualitative one. In the *quantitative approach* (such as [5, 13, 16]), preferences are expressed indirectly by using scoring functions that associate numeric scores with data items. In the *qualitative approach* (e.g. [8, 12, 11]), preferences between data items are specified directly, typically using binary preference relations.

7. CONCLUSIONS

To control the amount of data delivered to users in publish/subscribe systems, we extend such systems to incorporate ranking capabilities. In particular, in this paper, we address the problem of ranking notifications based on preferential subscriptions, that is, user subscriptions augmented with interest scores. To maintain the freshness of data delivered to users, we associate expiration times with notifications. We organize preferential subscriptions in a graph and utilize it to forward notifications to users. We have fully implemented our approach in SIENA, a popular publish/subscribe middleware system.

There are many directions for future work. One is considering alternative approaches for achieving timeliness, such as computing top-k results over sliding windows of notifications. Among our future plans is also studying a weighting scheme based on both time and relevance for ranking notifications. Finally, the focus of this paper has been on enhancing the expressiveness of publish/subscribe systems. Besides expressiveness, performance is also central in such large-scale dynamic systems. In this respect, we plan to consider additional topologies besides the hierarchical one used in this work.

8. 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. *SIGMOD Rec.*, 29(2):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] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8):1489–1499, 2002.

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

[9] 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.

[10] F. Fabret, A. H. Jacobsen, F. Llirbat, J. a. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *SIGMOD Rec.*, 30(2):115–126, 2001.

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

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

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

[14] T. Milo, T. Zur, and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *SIGMOD*, pages 749–760, 2007.

[15] K. Stefanidis and E. Pitoura. Fast contextual preference scoring of database tuples. In *EDBT*, pages 344–355, 2008.

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

[17] 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.

[18] C. Zimmer, C. Tryfonopoulos, K. Berberich, G. Weikum, and M. Koubarakis. Node Behavior Prediction for Large-Scale Approximate Information Filtering. In *LSDS-IR*, 2007.