

Pruning Subscriptions in Distributed Publish/Subscribe Systems

Sven Bittner

Annika Hinze

Department of Computer Science
University of Waikato,
Private Bag 3105, Hamilton, New Zealand,
Email: {s.bittner, a.hinze}@cs.waikato.ac.nz

Abstract

Publish/subscribe systems utilize filter algorithms to determine all subscriptions matching incoming event messages. To distribute such services, subscriptions are forwarded to several filter components. This approach allows for an application of routing algorithms that selectively forward event messages to only a subset of filter components. Beneficial effects of this scheme include decreasing network and computational load in single filter components.

So far, we can find routing optimizations that exploit coverings among subscriptions or utilize subscription merging strategies. Generally, such optimizations aim at reducing the amount of subscriptions forwarded to filter components, which decreases their computational load. This might in turn result in an increasing number of event messages routed through the network.

However, current optimization strategies only work on restrictive conjunctive subscriptions and cannot be extended to efficiently support arbitrary subscriptions. Furthermore, it is not possible to apply covering and perfect merging strategies in all application scenarios due to the strong dependency of these approaches on actually registered subscriptions.

In this paper, we present a novel optimization approach, subscription generalization, to decrease the filtering overhead in publish/subscribe systems. Our approach is based on selectivities of subscriptions and can be utilized for all kinds of subscriptions including arbitrary Boolean and conjunctive subscriptions. We propose a simple subscription generalization algorithm and show an evaluation of the results of a first series of experiments proving the usefulness of our approach.

Keywords: Distributed publish/subscribe, event filtering, subscription tree pruning

1 Introduction

Publish/subscribe systems use a push-based approach to access information that is published in the form of event messages. This means these systems continuously filter and actively deliver information to interested parties, which register their interests by the help of subscriptions. We can effectively apply publish/subscribe systems for several applications, e.g., facility management (Hinze 2003), meteorology (Mathieson, Dance, Padgham, Gorman &

Winikoff 2004), healthcare (Jung & Hinze 2005) and electronic commerce (Cilia & Buchmann 2002).

To achieve large-scale publish/subscribe solutions, these systems have to be realized as distributed services (Mühl 2002). They consist of several broker components dividing the filter load and thereby collaboratively performing the overall filtering task.

We have illustrated such a distributed publish/subscribe system in Figure 1: In the simplest case broker components B_x are connected by an acyclic (overlay) network structure. Clients, i.e., publishers P_x , subscribers S_x , and clients acting as both parties, connect to an arbitrary broker. This broker is called local broker and hides the distributed nature of the system, e.g., B_2 is local broker for P_1 in Figure 1. Subscriptions s_x are registered at the respective local brokers; event messages e_x are published to them. Matching event messages are delivered to subscribers by their local broker by the help of notifications n_x .

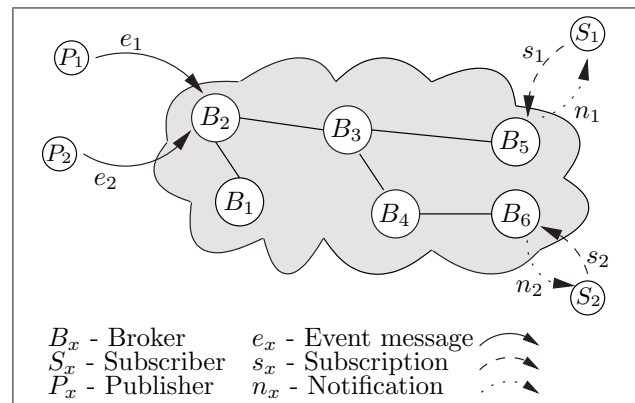


Figure 1: Distributed publish/subscribe system

In most application areas for publish/subscribe systems, the frequency of event messages is much higher than the frequency of registering and deregistering subscriptions. Thus, a profile forwarding scheme (Bittner & Hinze 2004) should be utilized to reduce the amount of event messages forwarded to brokers: Subscriptions are forwarded from local brokers to neighbor brokers. Then, brokers only deliver event messages to neighbor brokers that fulfil their subscriptions (which could again have been registered with another broker component). This forwarding of event messages and subscriptions is also referred to as event and subscription routing, respectively.

To minimize the amount of forwarded subscriptions, current approaches utilize coverings among subscriptions or merge several subscriptions. The main drawbacks of these methods are that their efficacy strongly depends on registered subscriptions and that they are applicable to conjunctive subscriptions only.

Especially in advanced application areas (e.g., e-

commerce) the constraint of conjunctive subscriptions is too restrictive and does not allow for the definition of subscriptions required to specify user interests.

In Figure 2, we present an example subscription s_1 from an e-commerce setting, in particular from on-line book auctions (letters in the figure describe the name of nodes). We will use s_1 as a running example throughout this paper: A subscriber is interested in books whose title contains the phrase "Harry Potter". According to the condition of the copy of the book (new, used), she wants to pay a different price (at most NZ\$15.0 or NZ\$10.0, respectively). To avoid unnecessary notifications, the subscriber will be notified not earlier than one day before the auction ends.

We can represent subscriptions as subscription trees as shown in Figure 2. Inner nodes represent Boolean operators; leaf nodes specify predicates, i.e., attribute-operator-value triples (Bittner & Hinze 2005b).

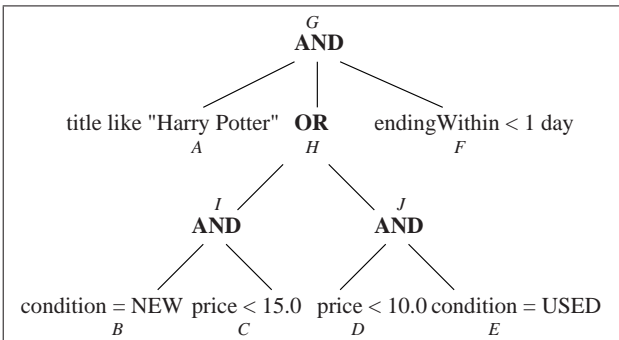


Figure 2: Example of a Boolean subscription s_1

Next to this requirement for more expressive than pure conjunctive subscriptions, it has been shown that publish/subscribe systems supporting more expressive subscription languages retain the efficiency properties of systems only offering conjunctive languages. In fact, the utilization of more expressive languages increases the scalability properties of brokers performing event filtering (Bittner & Hinze 2005a).

These beneficial characteristics of systems supporting arbitrary Boolean subscriptions (including conjunctive subscriptions) lead to the necessity of developing routing optimizations that are applicable to such services. Since we cannot use current covering and merging approaches in conjunction with expressive subscription languages, in this paper we design a novel routing optimization approach, subscription generalization, working on arbitrary subscriptions.

Subscription generalization is based on selectivities of subscriptions and aims at reducing subscription complexity to relieve resources in filtering broker components. Thus, it decreases their computational load as well as their memory requirements. This, in turn, results in increasing efficiency and scalability properties in broker components themselves.

However, this beneficial effect is counteracted by increased network load when applying subscription generalization. This behavior originates in the decrease of selectivities of subscriptions due to generalizations. Hence, more event messages are forwarded to neighbor brokers.

The rest of this paper is structured as follows: In Section 2 we present related work including routing optimization and selectivity estimation algorithms. The overall idea behind subscription generalization is elaborated in Section 3. There we also outline two specific generalization approaches, subscription pruning (Section 3.2) and predicate replacement (Section 3.3). Section 4 presents our proposal to estimate selectivities of subscriptions. Then, we continue with

an automatic generalization algorithm in Section 5. Section 6 presents a series of experiments we have undertaken to evaluate our approach as well as the evaluation of their results. Finally, we conclude and present future work in Section 7.

2 Current Optimization Approaches

We can classify most current routing optimization approaches into covering-based and merging-based solutions. These approaches are presented in the following subsections. We discuss current proposals on estimating selectivity in Section 2.4.

2.1 Subscription Covering

The definition of coverings among subscriptions, e.g., s_2 and s_3 , is based on analyzing the sets of event messages fulfilling them. These sets of fulfilling event messages are referred to as $E(s_2)$ and $E(s_3)$, respectively. If it holds $E(s_2) \supseteq E(s_3)$ then subscription s_2 covers s_3 (Mühl & Fiege 2001).

Coverings have been investigated in the publish/subscribe systems SIENA (Carzaniga, Rosenblum & Wolf 2001) and REBECA (Mühl & Fiege 2001). Both systems only support conjunctive subscriptions. Furthermore, (Mühl & Fiege 2001) restricts subscriptions to contain at most one predicate per attribute; (Carzaniga et al. 2001) does not present algorithms to compute coverings at all.

Also XML-based publish/subscribe systems, e.g., XROUTE (Chand & Felber 2003), utilize subscription covering. However, these systems restrict their subscription languages to conjunctive forms, too.

For general database systems, research also addresses a problem similar to covering: query rewriting (Halevy 2000). Since, in contrast to publish/subscribe systems, database systems can effectively utilize queries in canonical forms (Bittner & Hinze 2005b), research for database systems only targets these cases. Finding the required rewriting for general conjunctive database queries is NP-complete (Halevy 2000).

Generally, the utilization of coverings among subscriptions heavily depends on these registered subscriptions, i.e., we can only optimize by the help of coverings if subscriptions cover each other. The amount of optimization achievable by coverings is determined by the covering properties of subscriptions. So far, there is no research evaluating these properties in real-world applications.

2.2 Subscription Merging

Another optimization technique is subscription merging. There a set of subscriptions \mathcal{S}_4 is merged into a smaller set of subscriptions \mathcal{S}_5 with $|\mathcal{S}_4| > |\mathcal{S}_5|$ and $\bigcup_{s_4 \in \mathcal{S}_4} E(s_4) \subseteq \bigcup_{s_5 \in \mathcal{S}_5} E(s_5)$. In case of set equality we refer to this as perfect merging; for a proper subset relation it is denoted as imperfect merging.

Perfect merging does not increase the network traffic for event routing compared to originally registered subscriptions (Wang, Qiu, Verbowski, Achlioptas, Das & Larson 2004). However, we cannot always find such a merging; its applicability depends on the kind of registered subscriptions. REBECA (Mühl 2001) supports perfect merging for restricted conjunctive queries as described above. The work in (Crespo, Buyukkotken & Garcia-Molina 2003) analyzes several merging strategies for geographic queries, i.e., simple two-predicate conjunctive queries.

As pointed out in (Wang et al. 2004), perfect merging may not optimize the overall system throughput. Therefore, this work utilizes imperfect merg-

ing. It clusters subscriptions according to similarity to achieve a compact summary (merging) of them and evaluates several clustering techniques. Again, only conjunctive subscriptions are supported.

A general objection to subscription merging is its benefit when applying to subscriptions converted into conjunctive forms (arbitrary Boolean subscriptions might be converted into disjunctive normal forms of exponential size). Considering imperfect merging, we do not face the sole dependency on registered subscriptions as in the covering approach. However, the most possibilities to perform merging might result out of converted subscriptions, i.e., subscriptions previously converted and split into conjunctive elements are merged into a more general (conjunctive) subscription. This questions the usefulness of canonical conversions in respect to routing optimizations in addition to their drawbacks of decreasing scalability in broker components (Bittner & Hinze 2005a).

2.3 Other Optimization Approaches

Several approaches, e.g., the proposal in (Guimarães & Rodrigues 2003), target the routing in publish/subscribe systems as a multicast mapping problem: Similar subscriptions are clustered in the same multicast group. Event messages are only sent to those groups that may contain matching subscriptions. However, such approaches are also restricted to conjunctive subscription languages.

The routing scheme in (Carzaniga, Rutherford & Wolf 2004) mentions subscription simplification, which is a similar approach to subscription merging (perfect as well as imperfect). However, the work presents no computation algorithms. It supports subscriptions in disjunctive normal form and thus requires the same conversions as conjunctive approaches if it is utilized in applications involving more expressive than conjunctive subscriptions.

2.4 Selectivity Estimation

Estimating the selectivity of queries has been researched in the context of database systems, e.g., (Poosala & Ioannidis 1997, Chen, Koudas, Korn & Muthukrishnan 2000). However, such approaches require either conjunctive queries or conversions of queries into disjunctive or conjunctive normal forms. Apart from the time complexity required for these selectivity estimations and the memory consumption of involved data structures, canonical conversions lead to exponential space complexity that is not applicable in the context of publish/subscribe systems (Bittner & Hinze 2005a). These solutions are applicable to database systems that do not have to deal with a large number of continuous queries (another term used for subscriptions) and thus can convert queries into canonical forms.

3 Subscription Generalization

The overall idea of generalizing subscriptions is to decrease the computational effort for event filtering required in broker components. In the next subsection, we outline our general approach and its effect on event filtering. Afterwards, we present two simple subscription generalization methods, pruning subscription trees and replacing predicates by more general ones, i.e., covering predicates. We describe these methods in Section 3.2 and Section 3.3, respectively.

We discuss subscription generalization if utilizing one-dimensional index structure-based filter algorithms, e.g., (Bittner & Hinze 2005b, Fabret, Jacobson, Llirbat, Pereira, Ross & Shasha 2001, Hanson,

Chaabouni, Kim & Wang 1990, Pereira, Fabret, Llirbat & Shasha 2000, Yan & García-Molina 1994). Such approaches utilize one-dimensional indexes to index predicates and propose different structures to index subscriptions. Event filtering works in two steps: In predicate matching, all predicates matching incoming event messages are determined. Then, subscription matching computes all matching subscriptions (based on matching predicates). Since we focus on Boolean subscriptions, we particularly consider the algorithm in (Bittner & Hinze 2005b) that uses tree structures as subscription indexes.

3.1 Purpose of Subscription Generalization

As stated before, most application areas for publish/subscribe systems can effectively apply a subscription forwarding scheme. That is, subscriptions are forwarded to neighbor brokers of the local broker they have been registered with. This allows broker components to determine the set of neighbor brokers each incoming event message e_x should be forwarded to. In case of acyclic network structures this set includes all neighbors with registered subscriptions that are fulfilled by e_x except the neighbor forwarding e_x .

Subscription generalization aims to decreasing the complexity of forwarded subscriptions. Whenever such subscriptions require too many memory resources, broker components start to heuristically generalize registered subscriptions in order to minimize their size and thus the size of indexing structures. We do not generalize subscriptions registered by clients to avoid false notifications; only subscriptions forwarded by neighbor brokers are suitable for generalizations.

The generalization of subscriptions decreases both the memory usage and the computational effort required in brokers to determine matching subscriptions: The complexity of subscriptions is decreased, i.e., subscriptions consist of less predicates and operators, which directly decreases the memory required for subscription index structures. Furthermore, when indexes become smaller, we can determinate matching subscriptions more efficiently: In one-dimensional indexing approaches the predicate matching step works on smaller numbers of predicates; subscription matching filters on less complex subscriptions.

However, this advantageous behavior increases network load due to the forwarding of more event messages to neighbor brokers (subscriptions are more general now). This property is consistent with imperfect merging, which, similar to subscription generalization, does not depend on registered subscriptions like the proposals of perfect merging and covering. Thus, subscription generalization does always lead to decreasing memory requirements and increasing efficiency in broker components. We analyze the correlation between memory usage and network load in our experiments in Section 6.

A subscription generalization method for publish/subscribe systems should be easy to compute to allow for an efficient registering and deregistering of large numbers of subscriptions. Furthermore, it should require little additional memory to retain scalability properties of broker components.

In the next subsections, we describe two particular subscription generalization methods, subscription tree pruning and predicate replacement.

3.2 Generalization by Pruning

Pruning subscription trees reduces the number of predicates a broker has to filter on in the predicate matching step. Furthermore, it reduces the complexity of subscriptions that need to be evaluated in subscription matching. This characteristic is obtained by

pruning certain branches of subscription trees. Notice that we do not prune subscriptions in local brokers (cf. Section 3.1).

An arbitrary pruning of subscription trees does not necessarily lead to more general subscriptions. A subscription s_x is more general than subscription s_y if $E(s_x) \supseteq E(s_y)$. This definition conforms with the definition of covering for conjunctive subscriptions known from literature, e.g., (Mühl & Fiege 2001).

Excluding the case of removing the root node in a subscription tree, there is only one kind of pruning leading to a more general subscription:

Remove a child of a conjunctive node in a subscription tree.

An example is given in Figure 3. There we have illustrated the same subscription s_1 as in Figure 2 when removing node D that is a child of conjunctive node J. This pruning operation leads to s_6 , which, in contrast to original subscription s_1 , describes interest in all used books conforming to the other criteria (Title and Ending Within).

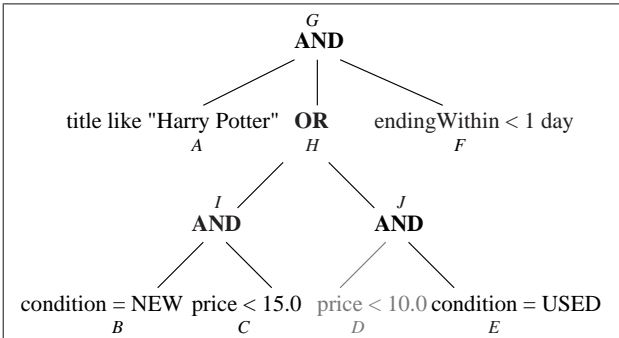


Figure 3: Valid pruning of s_1 leading to s_6

Removing a child of a disjunction is not a valid pruning as shown in Figure 4(a). There we show subscription s_1 (cf. Figure 2) when removing node J, the child of a disjunction. After restoring the general subscription tree properties (subsuming consecutive operators and eliminating unary operators (Bittner & Hinze 2005b)), the new subscription s_7 consists of only one conjunction as shown in Figure 4(b). However, it holds $E(s_7) \not\supseteq E(s_1)$, e.g., any event message describing used books does not fulfil s_7 but s_1 . Thus, this removal of node J does not lead to a more general subscription and is invalid.

However, our valid pruning option includes removing a complete disjunction in subscription trees: Disjunctions are always (except if they are a root node) the child of a conjunction due to our subsuming of consecutive operators. Hence, this option of removing a disjunction is included in our single pruning rule.

Furthermore, we can apply subscription tree pruning to pure conjunctive subscriptions. We are able to perform pruning in all cases, which makes our approach beneficial compared to current covering proposal that strongly depend on registered (conjunctive) subscriptions.

3.3 Generalization by Replacement

Our second proposal to generalize subscriptions is to replace predicates p_x by more general predicates p_y , i.e., p_x is covered by predicate p_y . The determination of coverings among predicates is relatively easy to calculate and depends on the permitted operators. Some examples are given in (Mühl 2001). However, the overall approach of subscription covering (Mühl 2001) is not applicable to arbitrary Boolean subscriptions.

The replacement of predicates results in decreasing numbers of predicates to filter on in the predicate

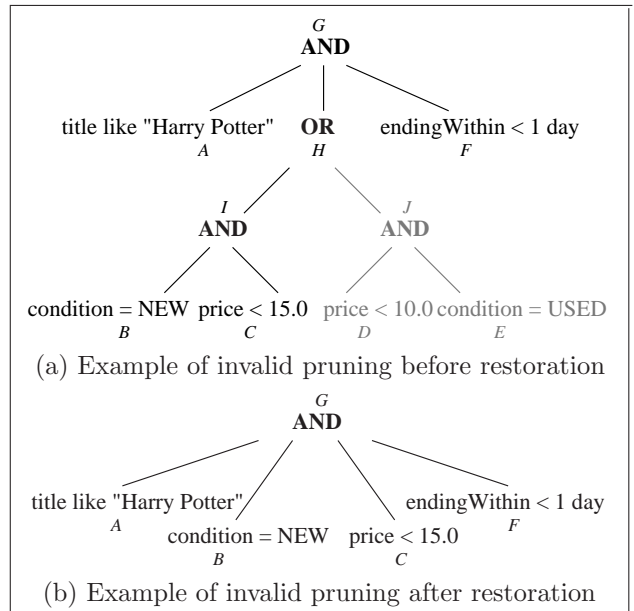


Figure 4: Invalid pruning of s_1 leading to s_7

matching step. However, it does not relieve subscription matching since subscriptions retain their complexity.

An example for the exploitation of coverings (based on s_1 that is given in Figure 2) is the replacement of predicate D by $price < 15.0$ leading to s_8 .

Replacements might also allow us to further modify resulting subscriptions. However, such modifications involve semantic knowledge (e.g., for subscription s_8 books are always new or used), which we neglect in our approach out of computational complexity and space efficiency aspects that are crucial in publish/subscribe systems (Bittner & Hinze 2005a).

We also do not focus on standard transformation rules, e.g., the integration of predicate $price < 15.0$ of s_8 into the conjunctive root node G. We could integrate such transformations into our approach but abstract from them in this paper to evaluate the effect of generalizations themselves.

3.4 Interconnection between Pruning and Replacement

A more generalized view on the presented pruning and replacement methods reveals their interconnection: Pruning could be seen as a replacement of predicates or whole subtrees by the most general predicate p_* , which is fulfilled by each event message.

In a conjunctive node we can omit a child p_* , which is fulfilled by all event messages, without changing the set of events fulfilling this conjunctive expression. This behavior is described by our pruning rule.

If a child of a disjunctive node is the most general predicate p_* , the whole disjunction is fulfilled by all event messages. Thus, we can replace this disjunctive node by p_* . In turn, we are able to remove p_* since it is child of a conjunctive node (if it is not a root itself). This describes the removal of a disjunctive node.

However, in this paper we distinguish between pruning and replacement because the computational effort required for their calculation is rather different (Section 5).

After this introduction to subscription generalization and two particular generalization methods, we introduce our method of estimating the selectivity of subscriptions in the next section. This estimation is an integral part of our automatic generalization algorithm, which is presented afterwards in Section 5.

4 Estimated Selectivity

Our later proposal for automatic subscription generalization (Sect. 5) is merely based on the selectivity of predicates. Predicate selectivity allows us to successively estimate the selectivity of a subscription.

4.1 Calculation of Estimated Selectivity

We can calculate the selectivity of predicates based on historical information: For each predicate we maintain a counter that is increased whenever this particular predicate is fulfilled by an incoming event message¹. Dividing this counter by the total number of filtered event messages leads to the selectivity of predicates. For newly registered subscriptions involving unused predicates we can estimate their selectivity, e.g., as presented in (Guimarães & Rodrigues 2003). Thus, for predicates p_x we can compute their selectivity $sel(p_x)$ both space and time efficiently.

The calculation of the selectivity $sel(s_x)$ of general Boolean subscriptions s_x would involve a huge computational effort requiring the analysis of all incoming event messages in respect to all registered subscriptions. Hence, we should focus on a simple method to estimate the selectivity $sel^{\approx}(s_x)$ of subscriptions s_x .

Our selectivity estimation $sel^{\approx}(s_x)$ for a subscription s_x consists of three easily computable values describing the minimal possible $sel^{min}(s_x)$, the average $sel^{avg}(s_x)$ (assuming a uniform distribution of all possible event messages and independent attribute values as well as predicates) and the maximal possible $sel^{max}(s_x)$ selectivity:

$$sel^{\approx}(s_x) = (sel^{min}(s_x), sel^{avg}(s_x), sel^{max}(s_x))$$

For all predicates p_x we can calculate their selectivity as described at the beginning of this subsection (count fulfilling event messages) and it holds

$$sel^{min}(p_x) = sel^{avg}(p_x) = sel^{max}(p_x) = sel(p_x)$$

We can recursively compute our selectivity estimation based on Boolean operators. For a binary conjunction the calculation works as follows

$$\begin{aligned} sel^{min}(a \wedge b) &= \min(0, sel^{min}(a) + sel^{min}(b) - 1) \\ sel^{avg}(a \wedge b) &= sel^{avg}(a) * sel^{avg}(b) \\ sel^{max}(a \wedge b) &= \min(sel^{max}(a), sel^{max}(b)) \end{aligned}$$

A binary disjunction leads to

$$\begin{aligned} sel^{min}(a \vee b) &= \max(sel^{min}(a), sel^{min}(b)) \\ sel^{avg}(a \vee b) &= sel^{avg}(a) + sel^{avg}(b) - \\ &\quad sel^{avg}(a) * sel^{avg}(b) \\ sel^{max}(a \vee b) &= \min(1, sel^{max}(a) + sel^{max}(b)) \end{aligned}$$

Our filtering approach for Boolean subscriptions (Bittner & Hinze 2005a, Bittner & Hinze 2005b) subsumes consecutive binary operators in subscription trees to n-ary ones. We can generalize the former calculations for the binary case to work with n-ary operators.

Calculating the described estimations for all subtrees of a subscription tree of s_x finally leads to the required selectivity estimate $sel^{\approx}(s_x)$. We have illustrated the estimate for our example subscription s_1 in Figure 5. Selectivities are rounded in the figure; we have given the selectivity of predicates only once (they are derived from our experiments whose setup is described in Section 6.1).

¹For shared predicates this is done only once.

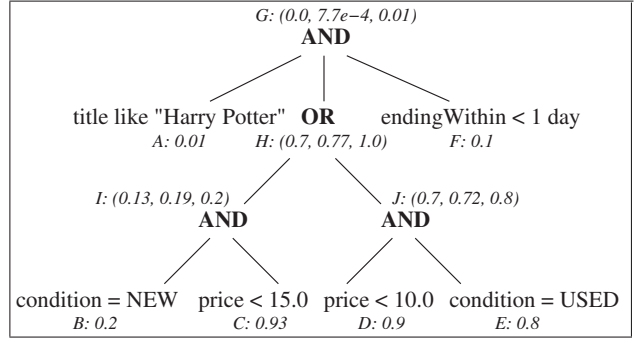


Figure 5: Example of estimating selectivity for s_1

4.2 Meaning of Estimated Selectivity

Our selectivity measure $sel^{\approx}(s_x)$ is an estimate of the real selectivity $sel(s_x)$ of s_x . It holds for all distributions of values in event messages and for all dependencies among attributes.

The value $sel^{max}(s_x)$ always describes the case that $E(s_x)$ is maximal: In case of a binary conjunction, the smaller set of event messages fulfilling a subexpression is a subset of the larger set of fulfilling event messages. For a binary disjunction, both sets are disjoint.

On the contrary, $sel^{min}(s_x)$ describes the case of a minimal $E(s_x)$: For a binary conjunction, the sets of fulfilling event messages of subexpressions exclude each other to the maximal possible extent. In case of a binary disjunction, the smaller set is included in the larger one.

Finally, $sel^{avg}(s_x)$ assumes that all possible event messages are equiprobable and subexpressions of subscriptions do not depend on each other: For a binary conjunction, the selectivity of one subexpression holds for event messages fulfilling the other subexpression. The same is true for the disjunctive case.

The real selectivity $sel(s_x)$ of s_x is always located between the two estimated extremes $sel^{min}(s_x)$ and $sel^{max}(s_x)$. The average case $sel^{avg}(s_x)$ describes which extreme is more likely if assuming independent predicates. For the subtree rooted in the disjunctive node H in Figure 5, it holds

$$sel^{\approx}(H) = (0.7, 0.77, 1.0)$$

Thus, the described average case is nearer to the estimated minimal than to the maximal selectivity value.

Our selectivity measure allows us to develop an automatic generalization algorithm targeting at the generalization of subscriptions according to the estimated effect on selectivity. This directly influences the increase in network traffic for event routing. We present this algorithm in the next section.

5 Automatic Generalization Algorithm

Our optimization algorithm targets an automatic generalization of subscriptions. Our goal is to decrease the computational costs of event filtering without increasing the network traffic to a large extent.

To achieve this goal, we generalize subscriptions s_x to s_y , which, in turn, might degrade their real selectivity as well as our estimated selectivity. We require a measure for this degradation that allows us to qualify the effect of generalizations. Such a measure is presented in the next subsection. Since the real selectivity of subscriptions is too hard to calculate (cf. Section 4) and thus its degradation $\Delta(s_x, s_y)$, we utilize its estimated counterpart to derive an estimated degradation $\Delta^{\approx}(s_x, s_y)$.

5.1 Selectivity Degradation

There are two options to describe the degradation $\Delta^{\approx}(s_x, s_y)$ of our estimated selectivity between an original subscription s_x and a generalized one s_y :

1. Absolute selectivity degradation
2. Relative selectivity degradation

Here we focus on the absolute degradation because it describes the real influence of generalizations on network traffic. Our absolute measure describes the change in selectivity as the maximal difference between the components of our estimation:

$$\begin{aligned} \Delta^{\approx}(s_x, s_y) = & \max(sel^{min}(s_y) - sel^{min}(s_x), \\ & sel^{avg}(s_y) - sel^{avg}(s_x), \\ & sel^{max}(s_y) - sel^{max}(s_x)) \end{aligned}$$

Its advantage compared to a relative measure is that degradation is expressed by its real change, i.e., $\Delta^{\approx}(s_x, s_y)$ comprises the expected additional number of event messages fulfilling the generalized subscription s_y compared to s_x .

A relative measure would, e.g., handle a decrease in selectivity in a leaf node p_x when generalizing to p_y from $sel(p_x) = 0.4$ to $sel(p_y) = 0.8$ in the same way as a decrease from $sel(p_x) = 0.04$ to $sel(p_y) = 0.08$. Thus, it does not reflect the real influence of generalization on network traffic.

5.2 Weak Points of Selectivity Degradation

We are aware about the difference between the estimated $\Delta^{\approx}(s_x, s_y)$ and the real degradation $\Delta(s_x, s_y)$.

Our estimated measure only describes the degradation in the minimal possible, average and maximal possible selectivity. The real selectivity might change more or less depending on dependencies among attributes. When generalizing a subscription s_x to s_y the worst case real degradation is

$$\Delta(s_x, s_y) = sel^{max}(s_y) - sel^{min}(s_x)$$

However, calculating the real degradation would be costly in both computational and memory resources. In publish/subscribe systems these resources are very scarce due to large subscription numbers to handle and high frequencies of incoming event messages.

Our generalization approach with its estimated selectivity degradation has still several advantages compared to existing covering and perfect merging techniques, e.g., (Mühl & Fiege 2001): Our approach might not find the optimal generalization in respect to real selectivity degradation, but it is always decreasing the size of index structures in brokers. Covering and perfect merging target a reduction of subscription numbers, which heavily depends on registered subscriptions and is not possible in all cases. Our approach works for all subscriptions, its optimization potential is merely depending on actual dependencies among attributes in event messages.

5.3 Automatic Pruning

Automatic pruning is done in broker components whenever subscriptions forwarded from neighbor brokers require too many resources. In order to execute subscription updates according to estimated degradations, we utilize a priority queue storing $(\Delta^{\approx}(s_x, s_y), s_x)$ tuples. For each incoming subscription s_x we calculate the best pruning leading to s_y . Note that we only need to calculate $\Delta^{\approx}(s_x, s_y)$ and not to determine s_y .

To perform pruning in case of exhausted resources, we generally

1. Extract the first element containing subscription s_x out of our priority queue
2. Perform the best pruning of s_x leading to s_y
3. Remove s_x from index structures
4. Insert s_y into index structures
5. Insert $(\Delta^{\approx}(s_y, s_z), s_y)$ into the priority queue, whereas s_z states the best pruning of s_y

This process is executed as long as enough memory resources have been freed².

The selectivity $sel(p_x)$ of predicates p_x might change over time. To incorporate this changing into our pruning process, we have to compare the actual estimated degradation of a pruning operation of subscription s_x to the value stored in the priority queue. If it has changed to a large extend (or is not near the minimum anymore), we can skip pruning s_x and reinsert s_x into the queue associated with the newly calculated degradation.

In case of shared predicates p_x our pruning approach is likely to remove p_x in all subscriptions containing this predicate within a short time. This is due to the general tendency of removing general before more selective predicates. Thus, all of these little selective predicates should be removed early in the pruning process and also relieve predicate indexes.

5.4 Automatic Replacement

Automatic replacement involves the determination of coverings among predicates. Generally, we can replace predicates p_x by all covering predicates p_y . The less selective p_y (chosen to replace p_x), the more our estimated selectivity degradation changes.

However, to effectively decrease memory usage, we need to remove a predicate from predicate index structures³. Otherwise, we do not save memory resources because, in contrast to pruning, subscriptions retain their complexity. Thus, the size of subscription index structures remains the same.

Since predicates p_x might be shared among subscriptions, a removal of p_x from predicate index structures is only possible if all subscriptions involving p_x replace its occurrence with p_y . This involves computations of selectivity degradation for all subscriptions involving p_x in order to find the best overall replacement option (that which should be performed first).

This property shows that the computational effort required for predicate replacement is much higher than the one required for subscription tree pruning. Additionally, pruning subscription trees leads to releasing more memory resources. Hence, subscription tree pruning should be preferred over predicate replacement as long as it does not degrade selectivity to a large extend.

6 Experiments and Evaluation

In this section, we present an evaluation and analysis of the automatic subscription tree pruning approach presented in Section 5.3. We focus on this generalization method due to its advantages compared to predicate replacement as shown in Section 5.4.

In the next subsection, we present our experimental setup. We show and analyze our experimental results in Section 6.2 and Section 6.3, respectively.

²We should execute Step 3 and Step 4 in batch to allow for a more efficient pruning.

³We might be able to save memory in subscription indexes due to implementation-specific memory overhead. Generally, we would remove one entry in the predicate subscription association table for a covered predicate, but reinsert one for the covering predicate.

6.1 Experimental Setup

In several application areas requiring publish/subscribe mechanisms, e.g., healthcare (Jung & Hinze 2005) and electronic commerce (Cilia & Buchmann 2002), more expressive than purely conjunctive subscription languages are required. In our experiments we focus on the popular application of online auctions, which particularly need active functionalities for an efficient dissemination of process-related information (Cilia & Buchmann 2002).

6.1.1 Event Messages

To obtain realistic data for our experiments we have analyzed auctions of fiction books offered on eBay⁴ on 8 July 2005. Our analysis focused on attributes shown in Table 1.

Table 1: Overview of attributes for book auctions

Attribute	Example	Values
Category	Fantasy	22
Format	Hardcover	4
Special Attribute	Signed	2
Condition	New, used	2
Ending Within	1 hour	0 sec... 10 days
Price	\$0.99	\$0.01... \$1000.00
Buy It Now	Yes, no	2
Bids	1	0... 100

We have been able to determine the exact number of books for all combinations of **Category**⁵, **Format**, **Special Attribute** and **Condition**. Furthermore, we extracted the number of books in all categories for 2 values of **Buy It Now**, 15 ranges of **Price** and 16 ranges of **Bids** (based on the search functionalities offered by eBay). For actual values in these two ranges we assume a uniform distribution, e.g., prices of all fantasy books between \$5.00 and \$6.00 (approximately 7% of all fantasy books) are uniformly distributed in this range.

For prices and bids we compared the distribution of completed and active auctions and realized only minor differences⁶. Thus, we used the distribution derived from active listings in our experiments. For the attribute **Ending Within** we assume a random distribution between 1 minute and 10 days.

We further assume 5 times less authors than books and 10% of all authors have published books in more than one category. The probability of multiple book titles is assumed as 1%. We expect authors and book titles to be given correctly in event messages (e.g., as achievable by utilizing a book database when offering items). We also experimented with other assumptions leading to similar results as presented later.

6.1.2 Subscriptions

Subscriptions in our online auction scenario potentially cover a wide range of user interests. In our evaluation we assume three different subscription classes:

Subscription class 1. Users are interested in a certain book title. According to the condition (new, used) of the copy of the book, they want to pay a different price. To avoid unnecessary notifications, users want to be notified one day before the end of the auction.

Subscription class 2. Again, users are interested in a certain book title and want to be notified one day before an auction ends. The difference to subscription class 1 is that users further distinguish between different formats, i.e., hardcover and softcover.

Subscription class 3. A collector is interested in books of a certain category, but also of a particular author. He wants to be notified one hour before the end of an auction offering a signed book copy without any bids. Furthermore, he wants notifications about signed books conforming his interests if they are Buy It Now items.

Similarly to event messages, we assume that authors are given correctly in subscriptions. To model subscriptions involving only parts of a book title, we reduce the number of possible titles and assume 100 times less titles than active auction items. We also experimented with other assumptions leading to similar results as presented later.

6.2 Experimental Results

In our experiments we analyze the interconnection between memory usage and network traffic when performing subscription tree pruning. To describe memory requirements, we use the measure of the total number of predicates registered with the system.

We present this measure in a relative manner, i.e., we show the portion of predicates compared to the original situation without applying subscription generalization. Analyzing the number of predicates allows us to directly derive the behavior of the total memory requirements for subscription index structures: For each predicate we can remove one entry from the predicate subscription table. Furthermore, subscription trees do not store removed predicates anymore⁷. Thus, memory requirements for subscription index structures at least decrease in the same manner as predicate numbers.

We neglect the memory for predicate index structures in our analysis due to their high dependency on utilized data types, supported operators and the variety of implementation variants.

For the description of network traffic we utilize the total selectivity of registered subscriptions. This measure directly implies the number of matching events, which in turn implies the network traffic created in event routing.

We separately analyze the three types of subscriptions described in Section 6.1.2 by randomly creating subscriptions conforming to the respective structure. Additionally, we present a setting with randomly chosen subscriptions out of all three classes.

Our results are presented in Figure 6 to Figure 9. Abscissae show the portion of performed pruning operations. Thereby the maximal possible pruning (1.0) describes the case that a further pruning removes a complete subscription, i.e., a subscription either contains of only one predicate or of a disjunction.

Left ordinates in the figures describe the relative decrease in predicates compared to the original situation without any pruning (0.0); right ordinates show the total selectivity of registered subscriptions.

⁷Subscription trees are actually reduced even more, since inner nodes might be deleted when removing predicates.

⁴<http://www.ebay.com/>

⁵We only looked at the first level of categories.

⁶Slightly increased bids and prices in completed auctions.

In our experiments we used 10,000 subscriptions and created 1,000,000 event messages conforming to the derived distribution in online book auctions (Section 6.1.1). The determination of initial selectivities was based on 100,000 created event messages.

We do not show individual measuring points in our figures due the large number of performed pruning operations (30,000 to almost 84,000 in our different settings) leading to the same amount of single measurements.

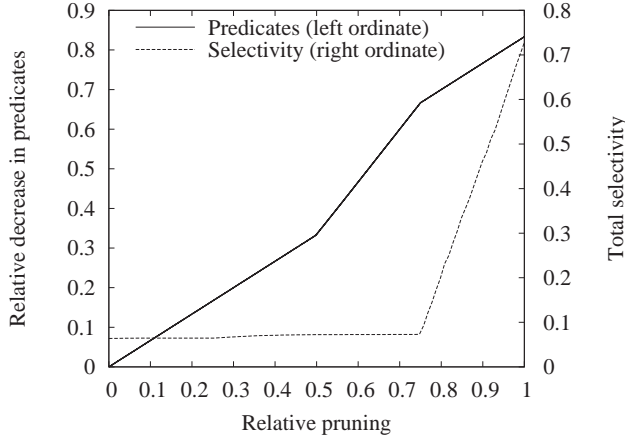


Figure 6: Subscription class 1 - influence of pruning

The behavior of subscription class 1 is given in Figure 6. Looking at the total selectivity of predicates, we realize a slight increase followed by a sharp bend and a fast increase. The bend occurs when approximately 75% of possible pruning operations have been performed (40,000 in total). At this point, the total selectivity of registered subscriptions has changed by 0.009; relative to their original selectivity (0.065) this is an increase by 14.3%.

Looking at memory requirements, we realize an always increasing behavior. Different gradients result out of pruning different subtrees. Due to the same pattern in subscriptions, similar subtrees are pruned one after the other before proceeding with another one. When reaching the sharp bend, nearly 77% of the predicates, i.e., memory for subscription indexes, have been removed. Thus, a reduction of subscription indexes to 23% of their original size has resulted in a relative increase of selectivity by only 14.3%.

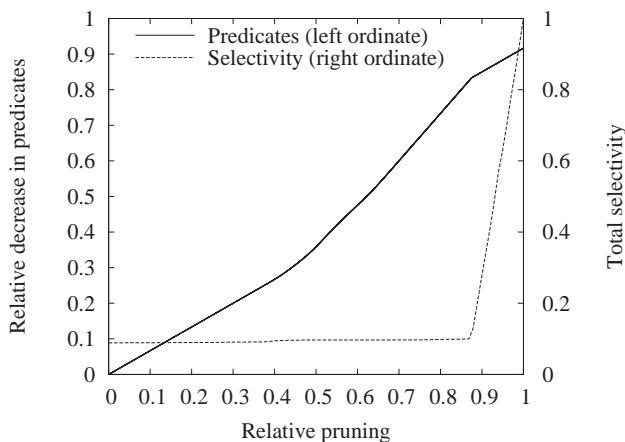


Figure 7: Subscription class 2 - influence of pruning

The behavior of subscription class 2 is depicted in Figure 7. We realize a similar behavior as for subscription class 1: The sharp bend occurs after nearly

88% of performed pruning operations (80,000 in total). At this point selectivity has increased by 0.012 (13.3%). Memory requirements of subscriptions indexes could be reduced to 17% of their original size.

Compared to subscription class 1, the sharp bend in the selectivity curve occurs after a larger amount of pruning operations has been performed. This directly results in greater savings of main memory resources compared to subscription class 1 before selectivity decreases sharply.

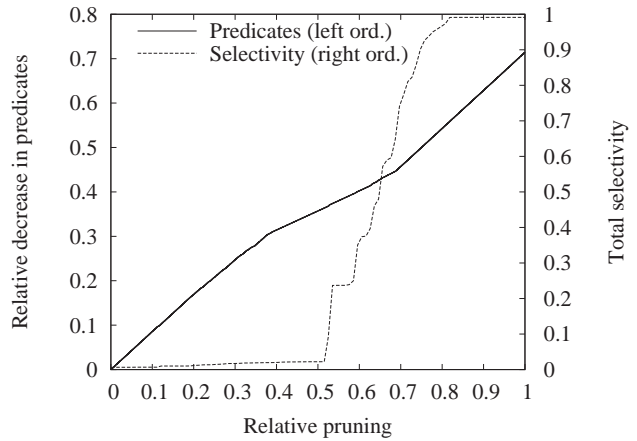


Figure 8: Subscription class 3 - influence of pruning

Subscription class 3 shows a faster decreasing total selectivity than the two previous classes of subscriptions. This is depicted in Figure 8: The sharp bend in selectivity occurs after performing almost 53% of pruning operations (30,000 in total). Up to this point, selectivity increased by 0.016; memory for subscription indexes could be reduced by 37%.

From data in Figure 8, we observe that the gradient of the selectivity curve increases in steps. These steps result from the inaccuracy of our selectivity measure in combination with the real distribution of event messages: A small amount of pruning operations abruptly increases total selectivity to a large extend (large gradients in curve). Succeeding pruning operations stick to the predicted small decrease in selectivity until the next step occurs.

In this particular class of subscriptions, the reason for this effect is the uneven distribution of signed copies among different categories of books.

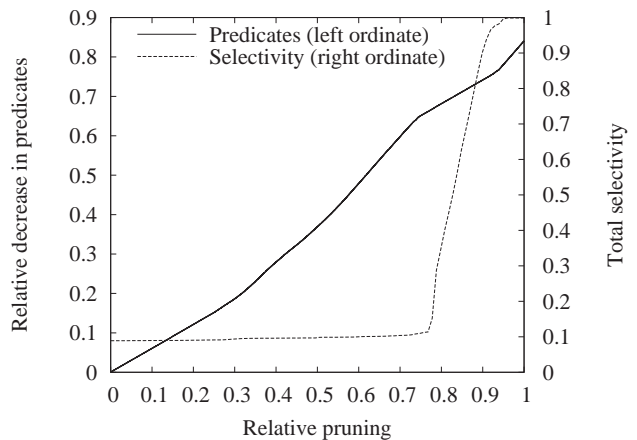


Figure 9: All classes - influence of pruning

In Figure 9 we illustrate the behavior of selectivity and memory usage in a setting with randomly created subscriptions conforming to our three subscription classes. There the sharp bend in the selectivity

curve occurs after performing 77% of possible pruning operations (approx. 50,000 in total). Up to this point, selectivity increases by 0.026 (29%). We realize a strong decrease in memory requirements: subscription indexes could be reduced to 34% of their size.

These results of random subscriptions behave better than expected when considering the results of the three single runs (Figure 6 to Figure 8). This is due to the larger number of pruning operations that are possible for subscriptions of class 1 and class 2 compared to subscriptions of class 3.

6.3 Discussion of Experimental Results

An overview of our results is given in Table 2. We present our four settings in columns 2 to 5 of the table; rows show six parameters we could derive:

- Overall possible number of pruning operations
- Original selectivity of registered subscriptions
- Relative number of pruning operations at sharp bend compared to possible pruning operations
- Relative increase in selectivity at sharp bend compared to original selectivity
- Total selectivity increase at sharp bend compared to original selectivity
- Relative decrease in predicates at sharp bend compared to original number of predicates

Table 2: Overview of results for our four test settings (the last two parameters describe changes in selectivity and memory at sharp bends)

Parameter	Class 1	Class 2	Class 3	All
Number prunings	40,000	80,000	30,000	50,117
Original selectivity	0.065	0.089	0.006	0.089
Rel. prunings	0.75	0.875	0.525	0.771
Rel. sel. increase	0.143	0.133	2.589	0.29
Total sel. increase	0.009	0.012	0.016	0.026
Rel. predicates	0.667	0.833	0.368	0.663

Our results show that subscription pruning is an effective way of decreasing the memory required for subscription indexes without increasing the selectivity of subscriptions to a large extend: For random subscriptions in our book auction application, we could reduce subscription indexes by 66% (34% of their original size) with increasing their selectivity by only 29%.

The applied pruning algorithm is straightforward and does solely depend on selectivity of predicates. This makes our optimization method efficient and does not require a large amount of memory.

The efficacy of subscription pruning depends on the structure of subscriptions and the selectivity of their predicates. Especially in cases of combining highly selective and more general predicates in subscriptions, subscription pruning leads to very good results. That is, we can remove various predicates without largely decreasing overall selectivity (which implies network load). Generally, little selective predicates are pruned early without affecting overall selectivity to a large extend. This especially holds if they are located in higher levels of subscription trees. However, the effect of pruning also depends on the structure of subscriptions, i.e., the usage of operators.

Consequently, subscriptions of class 1 and class 2 show better results than subscriptions of class 3. This is because predicates regarding **Ending Within** and **Title** are quite restrictive. Most of the other parts of subscription trees can be pruned without affecting selectivity to a large extend. Subscriptions of class 2 contain more little selective predicates than those of other classes leading to relatively more pruning operations before the sharp bend in selectivity.

For subscriptions mainly involving little selective predicates (e.g., those of class 3), subscription tree pruning can be used, too. Then, the correlation between saved memory resources and increased network usage is not that beneficial as in the former case, but pruning still results in memory requirements decreasing by 37% before strongly affecting selectivity. A selectivity of 1.0 is reached in this case after all possible prunings, because remaining subscription trees query for category that is a relatively general predicate.

We also ran experiments using different settings than presented in Section 6.1. They resulted in other magnitudes of selectivity but similarly developing curves. Especially the assumption of high selectivities for book titles results in increased selectivity.

Several application scenarios normally require subscriptions involving both highly selective and relatively general predicates: Subscriptions in our auction setting involve predicates regarding **Author** or **Title** (high selectivity) but also predicates on **Bids**, **Buy It Now** or **Format** (low selectivity).

Healthcare applications often require notifications in case of critical circumstances, e.g., abnormal blood pressure parameters or, more general, emergencies in intensive care units. These circumstances occur rarely, i.e., specifying predicates are highly selective. Other predicates, e.g., describing identifiers of monitored patients or names of medical conditions, are more general.

We can also find the same pattern in subscriptions for facility management purposes, e.g., when monitoring buildings to detect burglaries. Such events happen rarely (implying highly selective predicates describing, e.g., breakage of glass) whereas ordinary measurements from sensors arrive periodically and are leading to a low selectivity of predicates specifying, e.g., identifiers of certain buildings.

This shows that the structure of subscriptions in various application areas beneficially influences the effect of our subscription generalization approach. Consequently, subscription generalization is a valuable mechanism to increase scalability and efficiency in distributed publish/subscribe systems.

7 Conclusions and Future Work

In this paper we have proposed a novel routing optimization approach for distributed publish/subscribe systems. Our approach, subscription generalization, works on arbitrary Boolean subscriptions in combination with the well-known distribution scheme subscription forwarding. Subscription generalization aims at decreasing the complexity of subscriptions and thus at reducing the memory requirements in filtering broker components. In turn, the selectivity of subscriptions is decreased. In contrast to previous approaches, our proposal works on all kinds of registered subscriptions independent of their similarity and operators used in subscriptions.

We have presented two particular subscription generalization methods: pruning subscription trees and predicate replacement. Our pruning option relieves more memory resources than replacement whereas predicate replacement affects selectivity less than pruning. For subscription pruning, we have pro-

posed an algorithm automatically determining the order of pruning operations based on selectivities.

In order to calculate selectivities of subscriptions, we proposed a simple estimation approach focussing on the minimal, maximal and expected average selectivity of subscriptions. Our estimation is easily computable and requires little additional memory, which is an important quality criteria in publish/subscribe systems due to the large number of subscriptions.

To evaluate subscription generalization, we ran a series of experiments and evaluated our results: In an online auction scenario, subscription tree pruning is an effective way to decrease memory usage in broker components. For a typical set of subscriptions for online auctions, we could decrease memory requirements by 66% while increasing selectivity (and thus network traffic) by only 29%.

Subscription generalization leads to particularly beneficial results when combining highly selective and more general predicates in subscriptions. We can find subscriptions conforming these criteria in several applications, e.g., e-commerce, healthcare and facility management. Thus, subscription generalization is a valuable mechanism to increase scalability and efficiency in distributed publish/subscribe systems.

In the future we plan to integrate subscription generalization as routing optimization in a distributed publish/subscribe service. Then, we want to run an advanced series of experiments directly evaluating network traffic, memory requirements and efficiency.

References

- Bittner, S. & Hinze, A. (2004), Classification and Analysis of Distributed Event Filtering Algorithms, in 'Proceedings of the 12th International Conference on Cooperative Information Systems', Agia Napa, Cyprus, pp. 301–318.
- Bittner, S. & Hinze, A. (2005a), Investigating the Memory Requirements for Publish/Subscribe Filtering Algorithms, Technical Report 03/2005, Computer Science Department, University of Waikato.
- Bittner, S. & Hinze, A. (2005b), On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems, in 'Proc. of the 25th IEEE International Conference on Distributed Computing Systems Workshops', USA, pp. 451–457.
- Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2001), 'Design and Evaluation of a Wide-Area Event Notification Service', *ACM Transactions on Computer Systems (TOCS)* **19**(3), 332–383.
- Carzaniga, A., Rutherford, M. J. & Wolf, A. L. (2004), A Routing Scheme for Content-Based Networking, in 'Proc. of the 23rd IEEE Conference on Computer Communications', China.
- Chand, R. & Felber, P. A. (2003), A Scalable Protocol for Content-Based Routing in Overlay Networks, in 'Proceedings of the Second IEEE International Symposium on Network Computing and Applications', Cambridge, USA, pp. 123–130.
- Chen, Z., Koudas, N., Korn, F. & Muthukrishnan, S. (2000), Selectively Estimation For Boolean Queries, in 'Proc. of the 19th Symp. on Principles of Database Systems', USA, pp. 216–225.
- Cilia, M. & Buchmann, A. P. (2002), 'An Active Functionality Service For E-Business Applications', *ACM SIGMOD Record, Special Issue on Data Management Issues in Electronic Commerce* **31**(1), 24–30.
- Crespo, A., Buyukkokten, O. & Garcia-Molina, H. (2003), 'Query Merging: Improving Query Subscription Processing in a Multicast Environment', *IEEE Transactions on Knowledge and Data Engineering* **15**(1), 174–191.
- Fabret, F., Jacobsen, A., Lllirbat, F., Pereira, J., Ross, K. & Shasha, D. (2001), Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems, in 'Proc. of the 2001 ACM SIGMOD International Conference on Management of Data', USA, pp. 115–126.
- Guimarães, M. & Rodrigues, L. (2003), A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems, in 'Proc. of the 2nd IEEE International Symposium on Network Computing and Applications', USA, pp. 67–74.
- Halevy, A. Y. (2000), 'Theory of Answering Queries Using Views', *ACM Special Interest Group on Management of Data Record* **29**(4), 40–47.
- Hanson, E. N., Chaabouni, M., Kim, C.-H. & Wang, Y.-W. (1990), A Predicate Matching Algorithm for Database Rule Systems, in 'Proc. of the 1990 ACM SIGMOD International Conference on Management of Data', USA, pp. 271–280.
- Hinze, A. (2003), A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service, PhD thesis, Freie Universität Berlin, Institute of Computer Science.
- Jung, D. & Hinze, A. (2005), A Mobile Alerting System for the Support of Patients with Chronic Conditions, in 'Proc. of the 1st Euro Conference on Mobile Government', UK, pp. 264–274.
- Mathieson, I., Dance, S., Padgham, L., Gorman, M. & Winikoff, M. (2004), An Open Meteorological Alerting System: Issues and Solutions, in 'Proc. of the 27th Australasian Computer Science Conference', Dunedin, New Zealand, pp. 351–358.
- Mühl, G. (2001), Generic Constraints for Content-Based Publish/Subscribe Systems, in 'Proc. of the 6th International Conference on Cooperative Information Systems', Italy, pp. 211–225.
- Mühl, G. (2002), Large-Scale Content-Based Publish/Subscribe Systems, PhD thesis, Technische Universität Darmstadt.
- Mühl, G. & Fiege, L. (2001), 'Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs', *IEEE Distributed Systems Online* **2**(7).
- Pereira, J., Fabret, F., Lllirbat, F. & Shasha, D. (2000), Efficient Matching for Web-Based Publish/Subscribe Systems, in 'Proceedings of the 7th International Conference on Cooperative Information Systems', Eilat, Israel, pp. 162–173.
- Poosala, V. & Ioannidis, Y. (1997), Selectivity Estimation Without the Attribute Value Independence Assumption, in 'Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)', Athens, Greece, pp. 486–495.
- Wang, Y.-M., Qiu, L., Verbowski, C., Achlioptas, D., Das, G. & Larson, P. (2004), 'Summary-based Routing for Content-based Event Distribution Networks', *ACM SIGCOMM Computer Communication Review* **34**(5), 59–74.
- Yan, T. W. & García-Molina, H. (1994), 'Index Structures for Selective Dissemination of Information Under the Boolean Model', *ACM Transactions on Database Systems (TODS)* **19**(2), 332–364.