Classification and Analysis of Distributed Event Filtering Algorithms

Sven Bittner and Annika Hinze

University of Waikato, New Zealand,
{s.bittner, hinze}@cs.waikato.ac.nz

Abstract. Publish/subscribe middleware provides efficient support for loosely coupled communication in distributed systems. A number of different distributed message-filtering algorithms have been proposed. So far, a systematic comparison and analysis of these filter algorithms is still missing.
This paper proposes a classification scheme for distributed filter algorithms that supports the theoretical and practical analysis of these algorithms. We present a first cut theoretical evaluation and a subsequent practical evaluation of promising candidate algorithms. Factors that are considered include the characteristics of the underlying network and application-related constraints.
Based on the findings of these evaluations, we conclude with a summary of the strengths and weaknesses of the algorithms that we have studied.

1 Introduction

Large scale distributed systems increasingly rely on middleware-level publish/subscribe services to implement loosely coupled communication between components. The exchanged messages are filtered and forwarded to the appropriate components. This paper proposes a classification of distributed filter algorithms and provides an extensive theoretical and experimental analysis of selected algorithms.

An event notification system or publish/subscribe system is a middleware implementing the event-based communication paradigm. A *publisher* component sends *event messages* that announce the occurrence of events, i.e., the occurrence of something of interest within the distributed system. *Subscriber* components can subscribe to events that are of interest to them; these subscriptions are called *profiles*. Components can act as publishers and/or subscribers. The publish/subscribe system filters the incoming messages according to the subscribers' profiles and forwards matched messages to the respective subscribers. The distributed components of the publish/subscribe system are referred to as *brokers*.

We now briefly describe the current situation from which we will derive the research questions that are addressed in this paper. Several distributed algorithms have been proposed for the efficient filtering of event messages based on the context of the messages [1, 3, 7, 9–12, 14]. Rendezvous nodes [11, 12] are particular brokers that specialize in the filtering of selected event types and act as meeting points for profiles and event messages. Rendezvous nodes are a combination of a centralized and a distributed

filtering strategy, because brokers are responsible for a predefined set of profiles. Distributed filter algorithms employed in hierarchical networks exploit the hierarchical system structure [1, 3, 14]. Either every broker knows all profiles and event information is propagated down the tree starting at the root node [1], or each broker only knows the profiles registered by its children [3, 14]. Events are forwarded first up to the root and then down to the leaves. In point-to-point networks, each broker knows about its neighbor nodes and either events or profiles are forwarded within the network [3].

Several optimizations [3, 7, 9, 10] have been proposed to minimize the number of profiles that have to be forwarded to directly connected brokers. Covering uses the selectivity among profiles to decrease filtering overhead; merging unites several profiles to one profile for filtering [9, 10]. In [3], immediate computation of real covering is suggested, which results in costly computation. In [7], computation of coverings on request is proposed, which results in more network traffic, since all covered profiles are computed and forwarded if necessary.

From this brief survey of algorithms, one key problem becomes apparent: Which is the most efficient algorithm for a given network topology and application? So far, most of the algorithms have only been analyzed based on simulations of network topologies. In consequence, the results obtained in these evaluations do not consider several influential factors. In addition, most analyses have been carried out independently for single algorithms and, thus, have been performed under differing evaluation boundary conditions. As a consequence, we identify two open issues: (1) the definition of a classification scheme for distributed filter algorithms; and (2) a uniform performance analysis of filter algorithms that allows for a comparison of the algorithms' efficiency. Both issues are addressed in this paper. The contributions of this paper are as follows:

1. The introduction of a concise classification scheme for distributed filter algorithms.
2. A classification of existing filter algorithms according to the proposed scheme.
3. A theoretical performance analysis of filter algorithms.
4. An experimental performance analysis of selected filter algorithms.
5. Algorithm recommendations based on the applications and network topologies.

The remainder of the paper is organized as follows: Section 2 proposes a classification scheme for distributed filter algorithms. Section 3 briefly introduces our test system DAS. Section 4 presents the results and analysis of the experiments. The paper is rounded off by a conclusion and directions for future research.

2 Classification of Filter Algorithms

Several algorithms for distributed filtering have been proposed. A comparison of these algorithms and a general evaluation of filter approaches is difficult due to the diversity of the approaches. What is needed is a concise classification scheme for distributed filtering algorithms.

In this section, we propose such a classification scheme for distributed event filtering algorithms. This scheme provides a fundament for comparing the properties of the different types of algorithms. We classify existing filter algorithms with regard to the proposed scheme. Additionally, we introduce the results of a theoretical evaluation

of the algorithms in the proposed classification space. Finally, we identify the most promising filter algorithms to be evaluated in an experimental analysis.

Our classification scheme uses the following dimensions (see Table 1) that are subsequently explained in detail: (1) location of filtering, (2) spreading of filter complexity and memory strategy, and (3) communication with subscribers. We briefly present a description of each dimension and provide a theoretical evaluation of conceivable combinations of alternatives in all dimensions.

1. Location of filtering: Filtering can be performed close to the subscribers (flooding of events) or providers (flooding of profiles) [4], or at certain broker nodes [11, 12]. Flooding of events results in high network traffic, but less memory usage. Flooding of profiles results in the opposite: less network traffic and high memory consumption. Filtering at fixed (arbitrary) brokers gives the advantage of having control of the filtering according to available resources, but has the disadvantage of high load at filtering brokers in both network and computation.

2a. Spreading of filter complexity: The filter complexity can be spread over several brokers by exclusive filtering at certain brokers or by distributed filtering. Exclusive filtering can be implemented with little control overhead [12]. A disadvantage is the danger of multiple notifications for a single event, because the event information may be forwarded to several neighbour brokers. For distributed filtering, each broker accomplishes the filtering steps necessary to find all neighbor brokers with matching profiles [4, 11]. Beneficially, filter overhead is divided and the network traffic is minimized (only brokers with matching profiles are involved in filtering). The necessity of repeated filtering while forwarding the event message (to determine the appropriate neighbour) is a drawback. For distributed filtering, different memory strategies may be applied (see 2b).

2b. Memory strategy: Preventive storing refers to the storage of all available profiles, even duplicate and covered ones; this is beneficial in case of unsubscriptions. The resulting higher memory usage is a disadvantage. Optimistic storing minimizes the numbers of stored profiles (e.g. by discarding covered ones). In this case, unsubscriptions produce high network load, but less memory is used.

3. Communication with subscribers: We distinguish three alternatives: direct communication, forwarding via the network, and delivery via broker proxies (transparent communication). In direct communication, only the filtering broker and the subscriber are involved in communication [4] . A disadvantage is that either a connectionless protocol has to be used (resulting in unreliable communications) or new connections have to be established over time. When forwarding messages via the network of brokers, only neighbor brokers and local clients are communicating directly [12]. Local clients are publishers and subscribers that are directly connected to a broker. A drawback is the higher memory consumption: Information about the location of clients is needed, either by following the reverse path of the subscriptions, indexing all clients, or flooding notifications. When using brokers as proxies, brokers act as subscribers to their neighbor nodes [4, 11] and thus limit the number of subscribers each broker node has to deal with. Exploiting covering between profiles of several subscribers is possible and beneficial. A disadvantage is the necessity of post-filtering to notify client subscribers.

Table 1. Classification and theoretical evaluation of Distributed Filter Algorithms (EF - event forwarding, PF_x - profile forwarding, RN_x - rendezvous nodes, $\times$ = feature is supported, Evaluation: – – to ++ = poor to excellent results)

Algorithm	Filter Location			Spreading of Complexity (Memory Strategy)			Communication			Theor. Evaluation			
	Subscribers	Publishers	Arbitrary	Exclusive	Distributed: Preventive Storing	Distributed: Optimistic Storing	Direct	Forwarding	Transparent	Network Traffic	Memory Usage	Efficiency	Scalability
EF	×			×			×			– –	+ +	+ –	–
PF_1		×		×			×			+	– –	+ –	– –
PF_2		×		×				×		+ –	– –	+ –	– –
PF_3		×			×		×			+	–	+	–
PF_4		×			×			×		+ –	–	+	–
PF_5		×			×				×	+	+ –	++	+ –
PF_6		×				×	×			+	+ –	+	+ –
PF_7		×				×		×		+ –	+ –	+	+ –
PF_8		×				×			×	+	+	++	+
RN_1			×	×			×			–	– –	–	– –
RN_2			×	×				×		–	– –	–	– –
RN_3			×		×		×			+ –	– –	+ –	–
RN_4			×		×			×		–	– –	+ –	–
RN_5			×		×				×	+ –	–	+	+ –
RN_6			×			×	×			+ –	–	+ –	–
RN_7			×			×		×		–	–	–	–
RN_8			×			×			×	+ –	+	+	+ –

From the previous characteristics, we categorize filter algorithms as shown in Table 1. Columns 2–4 refer to the introduced dimensions. Our evaluation can be found in Column 5. Unfortunately, the available literature is not detailed enough to allow for a full classification of existing systems. Moreover, not all 17 variations may be implemented in existing systems. We identify three types of algorithms based on the filter location (distinguished by their names in Column 1): event forwarding (EF), profile forwarding (PF) and rendezvous nodes (RN).

Except in EF, we can find several subtypes of the algorithms. In EF each broker only filters for local subscribers; this implies exclusive filtering and direct communication. From the subtypes, we consider PF_8 and RN_8 as most promising because they have the least memory requirements due to use of coverings between profiles of several subscribers and an optimistic storage strategy. Our conclusion results from a combination of the evaluations shown above. We select one of each group for experimental analysis in our middleware: EF, PF_8, RN_8.

3 The Testbed: DAS - Distributed Alerting Service

We used the event notification service DAS as a flexible architecture with exchangeable filter components in order to evaluate different filter approaches. In this section, we first

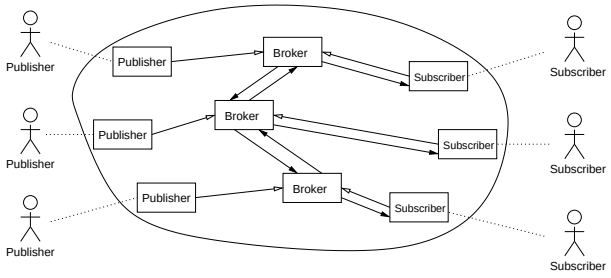


Fig. 1. Architecture of the distributed system DAS

describe DAS's architecture and then give details about the implementation of the three algorithms selected based on our theoretical analysis.

3.1 Architecture

Our distributed system DAS consists of three component types: brokers, subscribers and publishers, see Figure 1. To abstract from the physical network we use an acyclic overlay network to exchange profiles and event messages. Here, problems such as circulating messages and duplicates are displaced to lower communication layers. The acyclicity is no restriction in case of link errors, since a path between two nodes is found as long as any physical connection exists. Our reference implementation in DAS uses communication via TCP/IP. DAS is implemented in Java.

Within each broker, profiles and events are processed according to the chosen algorithm (EF, PF or RN), i.e., they are filtered or forwarded to neighbor nodes. Each broker's filter component maintains a profile repository; events are filtered against the repository. This centralized filtering uses a tree-based algorithm [6]. After the filtering, notifications are created from the processed event messages.

3.2 Implementation of the Distributed Filter Algorithms

We used three specialized implementations of the broker class for implementing the distributed filter algorithms. This section describes the algorithms' implementations in DAS. Similar algorithms have been discussed in Section 1.

Event Forwarding (EF) This is the simplest algorithm, since events are flooded through the network and brokers only filter for local subscribers. Subscriptions are added to and removed from a broker's filter structure. Profiles are registered only directly by subscribers. Events are flooded to all neighbor brokers except the sender. Events are filtered and on match, the profiles' subscribers are notified.

Profile Forwarding (PF) Profile forwarding uses covering among profiles, therefore, subscribing profile p_x and unsubscribing profile p_y are complex tasks. A profile p_x can

be registered at a broker either directly by a subscriber or by a neighboring broker. If p_x is registered by a broker, all profiles covered by p_x that are registered by this broker can be removed. If no profiles covering p_x exist, we register p_x at all neighbor brokers except the sender. If covering profiles exist and all of them were registered by the same neighbor broker, we register p_x at this broker. Then, p_x is added to the filter structure. When unsubscribing p_y, we register all profiles covered by p_y at all neighbor brokers except the sender. We also send the unsubscription to all neighbors except its sender. Finally we remove p_y from the filter structure.

Published events are filtered and subscribers of matching profiles are notified. If a subscriber is a broker, it is notified exactly once about each event even if several profiles match. When notifications arrive at a broker, the contained event is filtered and all subscribers except the sender are notified. Again, brokers are notified only once.

Rendezvous Nodes (RN) Rendezvous nodes are specified when configuring the network. When brokers connect to each other to build up the overlay network they also exchange information about known rendezvous nodes. Therefore, each broker knows which neighbor to contact to reach the rendezvous node for specific event types.

RN also uses coverings among profiles. For subscribing p_x at a rendezvous node, all covered profiles registered by the subscribing broker are removed. For subscribing p_x at a non-rendezvous node, p_x is sent towards its rendezvous node. Finally, p_x is added to the filter structure. When unsubscribing p_y at a non-rendezvous node, all covered profiles are sent towards the respective rendezvous node. Then, the unsubscription p_y is sent towards the rendezvous node. Finally, p_y is removed from the filter structure.

Events are filtered and in case of a match the neighbor brokers (except the sender) are notified exactly once. Then, the event is forwarded towards the rendezvous node. When a broker receives a notification, it filters the contained event and in turn notifies all subscribers excluding the sender. Again, brokers are notified only once per event.

Computation of Covering We used an interval-based computation of the profile covering. Our local filtering holds a separate profile tree for each attribute (a variation of [6]). The coverings are computed by analyzing the profiles in the leaves of the filter structure. For example, if a predicate contains the greater-than operator, all profiles that only occur in subsequent edges are covered. By intersecting the results from all attributes we can derive the coverings of profiles.

4 Experimental Analysis

In this section, we present the an overview of the results of our experimental analysis. A detailed discussion of the results can be found in [2].

We used our prototype in a realistic setting in a LAN with 100 mbps bandwidth, and machines with 1GHz and 256 MB main memory running under Linux. We evaluated the influence of different system parameters, namely:

1. Proportion of matching events over all events (see Section 4.1),
2. Portion of matching profiles per events (see Section 4.1),

3. Number of brokers (see Section 4.2),
4. Number of profile coverings (see Section 4.3),
5. Number of event types (see Section 4.4),
6. Locality of profiles and events (see Section 4.5), and
7. Number of profiles (see Section 4.6).

Our analysis uses the following units of measurement:

Filter efficiency: This measure refers to the system's performance, i.e., the number of events per second that can be processed by the system. We computed the filter time in the broker nodes and exclude the network forwarding time: The efficiency (i.e, the number of filtered events processed per second) is computed by dividing the number of published events by the time that the brokers took for the filtering of these events. We also evaluated parallel efficiency e, which refers to the speedup achieved by distributing the event filtering over several brokers; it is given as speedup per broker. Parallel efficiency gives an indication of the scalability of the algorithms.

Network load per event: The network load per event was computed by totaling the size of event data received by all brokers and dividing by the number of published events.

Duplication of profiles: This measure refers to the average number of brokers at which a profile is registered. For example, the value 2.0 states that each profile is registered on average by 2 brokers. The system's performance is influenced by duplication, since more memory is needed to store the same number of profiles. This memory consumption results in page swaps and less efficiency. Duplication is computed by dividing the total number of registered profiles by the number of profiles registered by clients.

We additionally use the following terms: the proportion of matching events over all events is referred to by p^e. The portion of matching profiles per event is referred to by p^p; it is computed by the number of profile notifications divided by the number of events published. The utilization of events σ is defined by $\sigma = \frac{p^p}{p^e}$. The utilization σ states how many profiles are notified by a matching event on average.

In the following experiments we only used event types with one attribute — we can easily derive the behavior of our algorithms in cases of more attributes. Figure 2 shows the filter time for the filtering of $100,000$ events against $10,000$ profiles with different numbers of attributes and values of p^e ($p^e = p^p$ since only unique profiles are used). Here, we assumed that non-matching events are recognized after the evaluation of half of the type's attributes in average (mean value of recognition after each attribute). Our filter algorithm minimizes the number of attributes evaluated to recognize non-matching events, for details see [6].

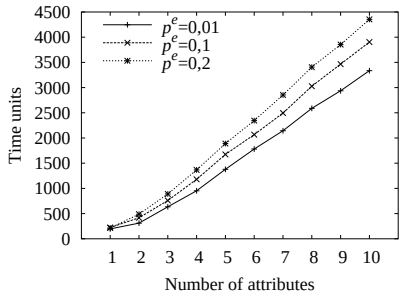


Fig. 2. Filter time depending on #attributes

Another restriction is the connection of only one publisher and one subscriber to each broker (see Figure 1). In realistic scenarios we expect more clients with individually fewer profiles and events, which leads to the same overall quantity. Our results can be generalized, because only the overall number of profiles and events influence efficiency and scalability. For example, more clients would increase the costs for synchronization, but the use of proxies that handle connections to clients would decrease the communication overhead with brokers. If not explicitly stated otherwise, events and profiles are unique, i.e., they do not overlap. In the following subsections, we describe our experimental results in detail. All experiments were performed with a standard deviation under 1% regarding efficiency.

4.1 Influence of Matching Events and Profiles

Here, we analyze the influence of the proportion of matching events p^e and the average number of matching profiles p^p on efficiency and network load. Duplication of profiles is not considered here, because it remains stable over the experiments. We used 4 brokers connected as a linear bus. The rendezvous node is located at an inner broker. Each broker managed $50,000$ local profiles. We also analyzed different values of the utilization σ.

Hypotheses: Extending our theoretical analysis (see Section 2), we expect the following behavior: With increasing p^e and p^p, the algorithms are less efficient (i.e., fewer events are filtered per second). With small p^p and p^e, PF should be more efficient than the other two algorithms. The network load is expected to be lowest in PF, followed by RN and EF. For EF we expect the maximum network load regardless of p^p and p^e.

Results: Figure 3(a) shows efficiency in number of processed events per second over the proportion of matching events p^e. As expected, PF is very efficient in case of small p^e. With increasing p^e, a strong decline in efficiency is caused by costly notifications and the post-filtering. The efficiency of EF changes less with increasing p^e, because no post-filtering is necessary. The number of created notifications increases, which results in a linear efficiency decrease. The influence of increasing p^e on RN is greater than on EF but less than for PF. The reasons are both the use of post-filtering and the creation of more notifications.

Figure 3(b) shows the influence of increasing p^p on the filtered events per second. Again, PF shows the best efficiency. With increasing σ (i.e., increasing utilization of events) efficiency increases, since less post-filtering is needed while the number of notifications remains constant. EF is less influenced by changing p^p — the event flooding causes non-matching events to be rejected earlier. The efficiency of RN lies between EF and PF for the same reasons as described above (post-filtering in rendezvous nodes and more notifications).

The network load for the three algorithms is shown in Fig. 4 as bytes per event over p^e and p^p. EF shows the highest load due to the flooding of all events. RN's forwarding of all events to the rendezvous nodes leads to less network load. The least load is caused by PF, because only matching events are forwarded. With constant p^e, the utilization of events σ does not influence the network load (leading to identical graphs in Fig. 4(a), not shown for the sake of clarity in the diagram). With constant p^p, the network load is influenced by σ (except for EF, which floods all events). Increasing σ (see Fig. 4(b)

with $\sigma = 1$ and $\sigma = 9$) results in decreasing network load, because fewer events notify the same number of profiles (i.e., decreasing p^e).

4.2 Influence of Number of Brokers

In this subsection, we analyze the influence of the number of brokers on efficiency, parallel efficiency, duplication of profiles, and network load. For the experiments, we used the network topology as shown in Fig. 5.[1] The network size was varied between 1 and 9 brokers. We used a single event type; Broker 2 acts as rendezvous node. $200,000$ unique profiles were used.

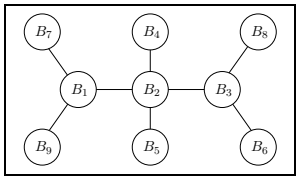
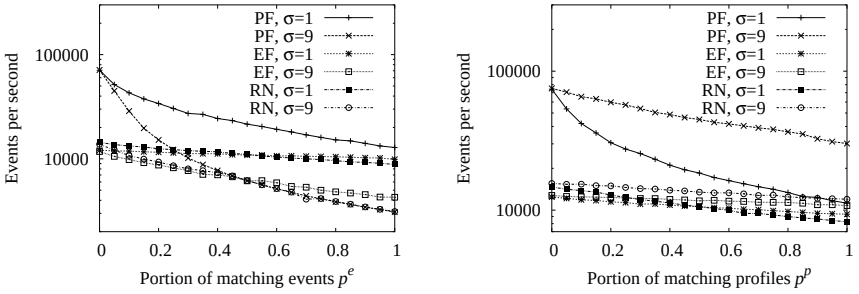


Fig. 5. Network topology for brokers

Hypotheses: Extending the theoretical evaluation from Section 2, we state the following hypotheses: When using more brokers, we expect improved efficiency for PF and less efficiency improvement for RN. The efficiency of EF should not change. PF is expected to show the best parallel efficiency and EF the worst one. The network load is expected to increase for all three algorithms, most in PF, followed by RN and EF. For profile duplication, we expect the opposite effect: EF duplicates no profiles, PF all profiles and RN is a compromise between the two.

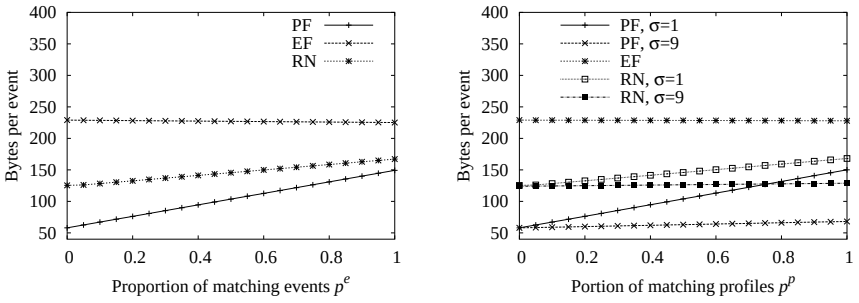
Results: Figure 6 shows the results for filter efficiency and parallel efficiency; both are given as events per second over the number of brokers. PF has a steep increase in efficiency when adding brokers (see Fig. 6(a)). The increase is highest for $p^p = 0.1$; lower p^p lowers the filter efficiency. Here, the main load is caused by the notifications. EF's efficiency is decreasing when adding more brokers (Fig. 6(b)), which is due to increased communication overhead. The influence of p^p when adding brokers is small, because the additional overhead due to notifications is small compared to the overall

[1] Many other topologies could have been tested. This is a first cut evaluation not using a simulation. Further large scale tests with more general and larger topologies are advised.



(a) Matching events p^e **(b)** Matching profiles p^p

Fig. 3. Filter efficiency depending on the on the portion of matching events p^e and profiles per event p^p.



(a) Matching events p^e **(b)** Matching profiles p^p

Fig. 4. Network load depending on the proportion of matching events p^e and matching profiles p^p

communication complexity. The efficiency is lowest when using five brokers, because Broker 2 (which is the system's bottleneck) is overloaded. RN's efficiency is nearly unchanged when adding brokers (see Fig. 6(c)). Here, the system's bottleneck is the rendezvous node, which performs the same amount of filtering steps regardless of the network size. The filter efficiency decreases when reaching a certain number of brokers. The reason is the asymmetrical network of brokers – some brokers encounter greater load than others.

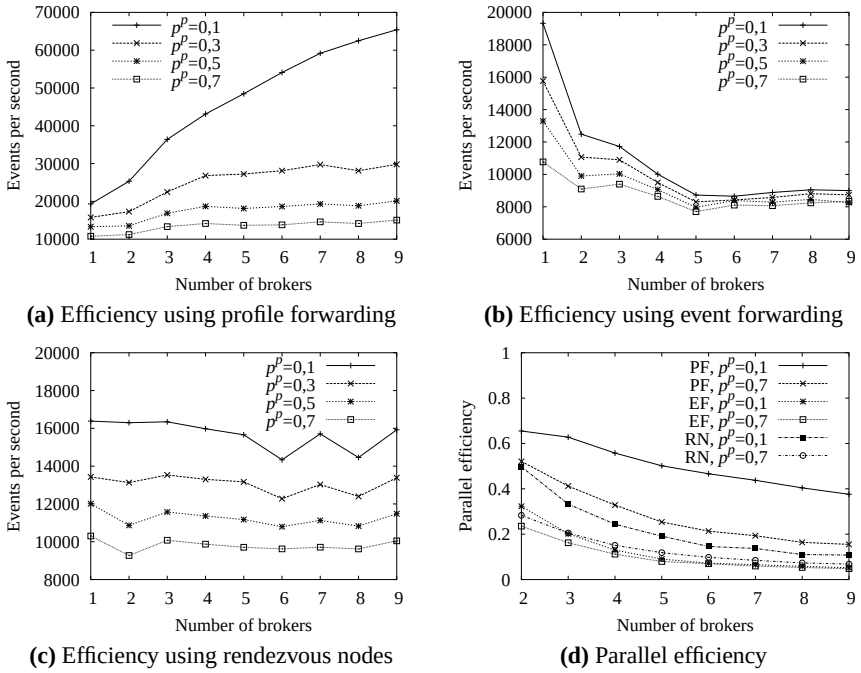
The results for parallel efficiency are shown in Fig. 6(d). Parallel efficiency e is measured in speed increase per broker over the number of brokers. The measure is computed as $e = \frac{f_{sat}^n}{n * f_{sat}^1}$ where f_{sat}^i refers to the maximal event frequency that can be processed in i brokers and n is the number of brokers in the network. The best results are recorded for PF due to its good load distribution. Overall, the parallel efficiency decreases as the number of brokers increases. The results are disappointing; the main reason for this behavior is the high communication overhead between the brokers.

The results for the duplication of profiles are shown in Fig. 7(a)). When using PF, the duplication increases linearly. Since the profiles are unique (i.e., have no overlap), each broker stores all profiles. EF shows a constant duplication value of 1.0. Results for RN lie between PF and EF with duplication values lower than 2.5.

The results for the network load are shown in Fig. 7(b); compared to the profile duplication, the order of the algorithms is reversed. EF shows a constant increase of network load. Using PF, only matching events are distributed, which results in low network load. For high values of p^p, the network loads for RN and PF are very similar. Events are always forwarded to the rendezvous node, causing little additional expense.

4.3 Influence of Covering

In this subsection, we discuss the influence of coverings. Only equality operators are used, so the utilization of events σ is equivalent to coverings (e.g., $\sigma = 5$ stands for 5 covered profiles per profile). Coverings appear only between local profiles at the brokers. We used the same network of brokers as described in Section 4.1, one event type and $200,000$ profiles. We analyzed efficiency, duplication of profiles and network load.



(a) Efficiency using profile forwarding

(b) Efficiency using event forwarding

(c) Efficiency using rendezvous nodes

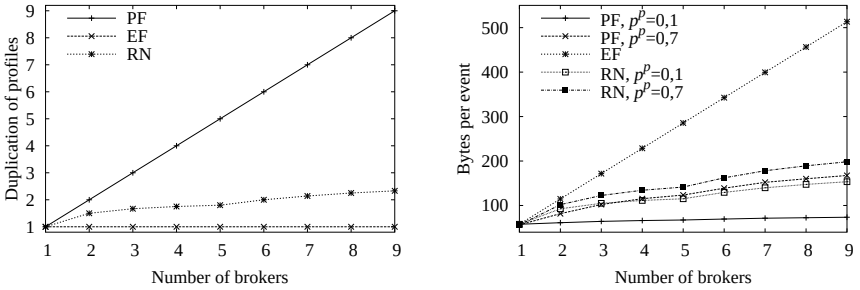
(d) Parallel efficiency

Fig. 6. Filter efficiency and parallel efficiency depending on number of brokers

Hypotheses: We expect decreasing efficiency with increasing coverings using constant p^e (more load because of more notifications). Using constant p^p, the result should be the opposite (fewer forwarding and filtering steps). The duplication of profiles is expected to decrease when using higher covering (exploiting the covering feature). The network load should remain unchanged under constant p^e and changing σ. With constant p^p and increasing σ, the network load is expected to decrease, since fewer events are forwarded (except when using EF).

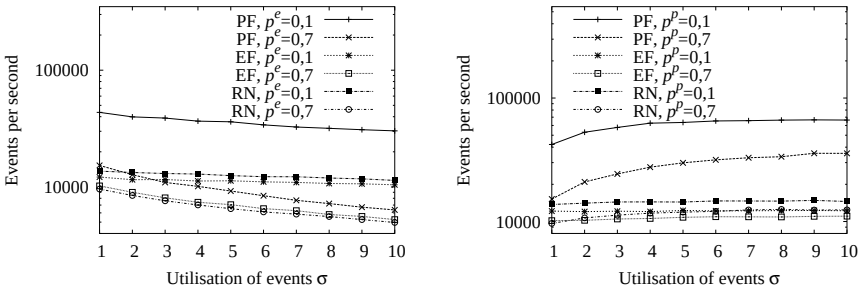
Results: Figure 8(a) shows the efficiency over the covering under changing proportion of matching events p^e. All three algorithms show decreasing efficiency when using more coverings, since more notifications are generated. With a high value of p^e, the differences among the algorithms are marginal, since nearly all events have to be flooded. With small p^e, PF is by far the most efficient algorithm. With higher p^e, efficiency decreases less (changing the proportion of complexity of notifications to all-over complexity). Using EF and RN, the decrease in efficiency is reduced.

Figure 8(b) shows the efficiency over the covering under changing proportion of matching profiles p^p. Increasing σ and constant p^p lead to higher efficiency. The reason is the constant number of notifications while filtering fewer events. PF shows the highest efficiency increase, since only events with matching profiles are forwarded. In contrast to the behavior for matching events p^e as seen in Fig. 8(a), the differences for the algorithms grow with increasing σ. RN's efficiency improves more slowly, because



(a) Duplication of profiles **(b)** Network load

Fig. 7. Duplication of profiles and network load depending on the number of brokers



(a) Matching events p^e **(b)** Matching profiles p^p

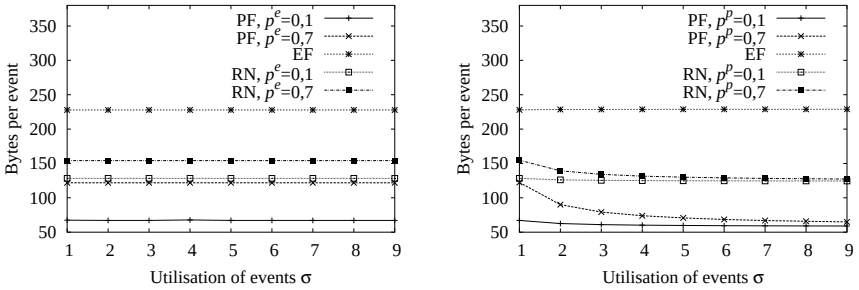
Fig. 8. Efficiency depending on the utilization of events σ and varying p^e and p^p

events are always forwarded to the rendezvous node. EF is independent of σ, because events are always forwarded to all brokers.

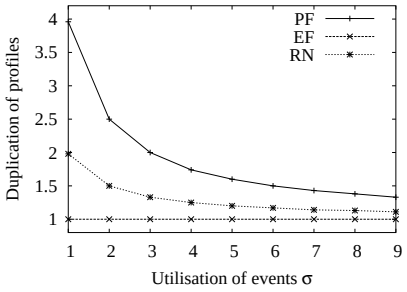
The network load remains constant with unmodified p^e (Fig. 9(a)), because the same number of events is forwarded (but there are more notifications). As expected, a high p^e increases the network load and EF causes the highest load. Constant p^p (Fig. 9(b)) results in decreasing network load because each event matches multiple profiles (which decreases communication among brokers). With growing σ, this effect becomes less influential. High p^p increases the network load (except for EF). The duplication of profiles (Fig. 9(c)) decreases with growing coverings. PF shows the largest duplication, followed by RN and EF (which never distributes profiles). For high coverings, the duplication graphs of PF and RN converge to the graph of EF.

4.4 Influence of Event Types

In this subsection, we analyze the influence of the number of event types on efficiency, duplication of profiles and network load. We used the network topology illustrated in Fig. 5 with each broker being rendezvous node for at most one event type. $180,000$ unique profiles were registered, which were evenly distributed between the event types.



(a) Network load, changing p^e **(b)** Network load , changing p^p



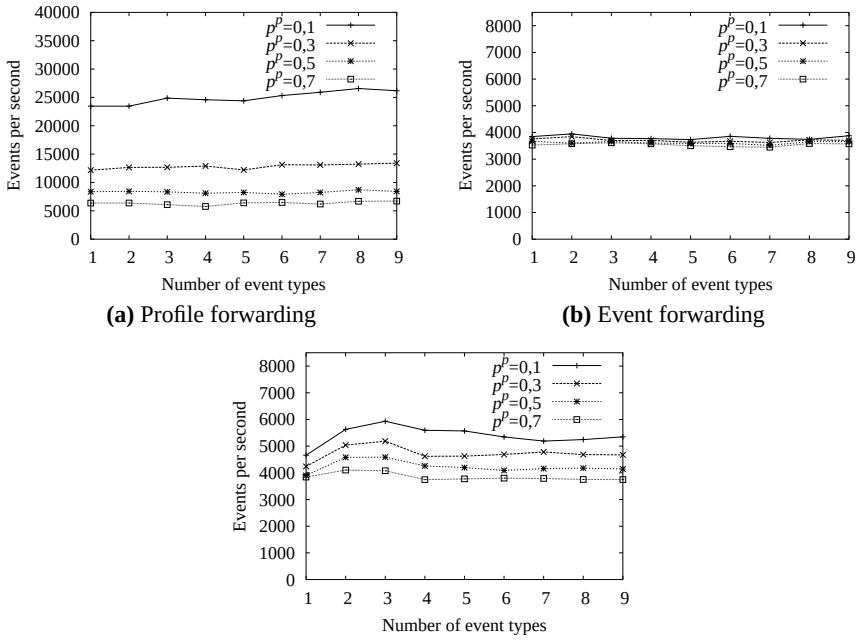
(c) Duplication of profiles

Fig. 9. Network load and duplication of profiles depending on the utilization of events σ with various portions of matching events p^e and profiles p^p

Hypotheses: We expect that the number of event types will have little effect on efficiency. PF and EF should be almost independent. RN's efficiency should increase when arranging rendezvous nodes well. Duplication of profiles and network load should be independent of the number of event types, except when using RN. There, the paths to the rendezvous nodes affect the duplication of profiles and the network load.

Results: The filter efficiency is illustrated in Fig. 10 (note the different scales). PF shows nearly constant values (see Fig. 10(a)). The small performance increase is due to our central filter algorithm, which builds a separate filter structure per event type. Increasing p^p decreases performance because more notifications are created. EF behaves similarly (see Fig. 10(b)), except that p^p has almost no influence, because the flooding overhead dominates over the processing of notifications. RN's efficiency depends on the location of the rendezvous nodes in the network (see Fig. 10(c)). For up to three nodes, if the rendezvous nodes are central nodes, the efficiency increases. The rendezvous nodes have lower burden, because fewer events have to be filtered. The efficiency decreases when using more than four event types. The reason is that some of the rendezvous nodes are outer nodes of the network – inner nodes have to forward all events and become a bottleneck.

The duplication of profiles is independent of the number of event types (Fig. 11(a)). Since only unique profiles are used, the duplication is 9.0 using PF and 1.0 using EF. Using more than two event types in RN increases the duplication based on the position

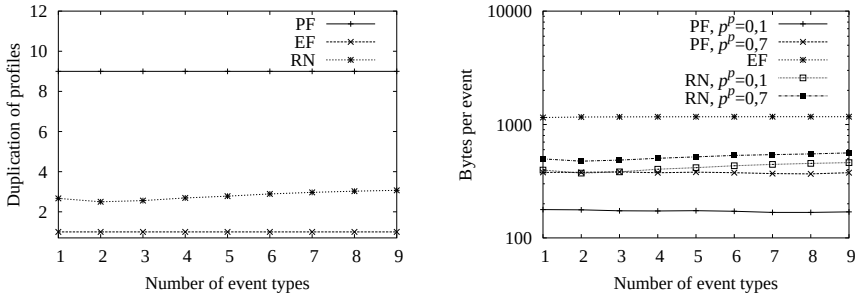


(a) Profile forwarding **(b)** Event forwarding

(c) Rendezvous nodes

Fig. 10. Efficiency depending on number of event types and various p^p

of the rendezvous nodes. The results for network load are similar (see Fig. 11(b), logarithmic ordinate). PF and EF cause stable network load. Using RN causes an increase of network load after an initial decrease (longer paths to rendezvous nodes). Increasing p^p increases the network load for PF and RN. Here, RN is less influenced due to the superfluous forwarding to the rendezvous nodes.



(a) Duplication of profiles **(b)** Network load

Fig. 11. Duplication of profiles and network load depending on number of event types

4.5 Influence of Locality

In this subsection, we analyze the influence of locality of profiles and events on efficiency and network load. Locality refers to the fact that events from a broker's local publishers only match profiles from local subscribers. We used four brokers as a linear bus, as described in Sect. 4.1. $160,000$ profiles referred to a single event type. We increased the number of matching profiles per event per broker (p^p per broker = locality). The duplication of profiles is not considered, since profiles remained unchanged.

Hypotheses: We expect an efficiency increase based on higher locality between profiles and events for PF (since fewer notifications are forwarded to neighbors). When using RN a small increase should occur: Due to the overall event forwarding to the rendezvous node only a smaller part of communication complexity is saved. For EF, we expect independence between locality and efficiency. Analogous results are expected for the network load: Less load for PF and RN, independence for EF.

Results: Figure 12 shows the efficiency depending on locality. PF's efficiency increases by a factor of 2 to 3.5 when increasing locality from 0 to 1. The reason is the early rejection of events at their local brokers. As expected, EF is independent of the locality; all events are flooded. RN is less influenced by locality than PF; the efficiency improves only by a factor of 1.25. The reason is the overall forwarding of events to rendezvous nodes. PF shows a better adaptation to locality than the other two algorithms. RN does not support the hypothesis due to the communication overhead between brokers on the path to the rendezvous node (overcomes the advantage of filtering in fewer brokers). The network load is shown in Fig. 13: EF is not influenced by the locality due to flooding. PF and RN show decreasing load due to early rejection of unmatched events.

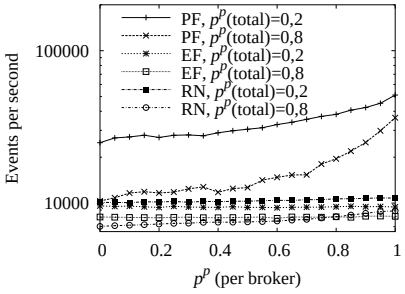


Fig. 12. Efficiency depending on locality and different p^p

4.6 Influence of Number of Profiles

In this subsection, we discuss the influence of the total number of profiles on the efficiency. We used four brokers connected as a linear bus. We subscribed different numbers of unique profiles ($\sigma = 1$). The proportion of matching events was set to $p^e = 0.8$.

Hypotheses: Efficiency is expected to decrease rapidly with increasing number of profiles. Using PF and RN, the main memory is expected to quickly be fully loaded and swapped out to secondary memory. EF should be more stable when using large numbers of profiles, since they are not duplicated.

Results: Figure 14 shows the efficiency over increasing number of profiles. As can be seen in the figure, PF and RN can process up to $100,000$ unique profiles and EP can process up to $350,000$ unique profiles in main memory (not shown in the graph: this

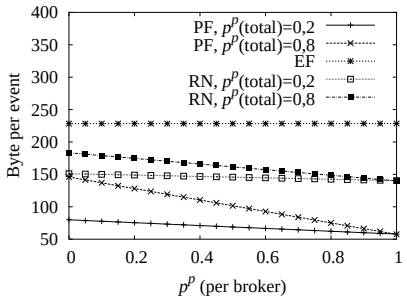


Fig. 13. Network load depending on locality

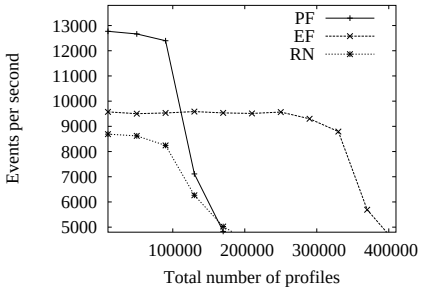


Fig. 14. Efficiency over number of profiles

value increases for PF or RN when using coverings). PF shows the best efficiency as long as the profiles are stored in main memory, followed by EF and RN. Due to the large proportion of matching events ($p^e = 0.8$), RN is less efficient than EF (as discussed in Section 4.1). Using more than $100,000$ profiles causes an efficiency plunge for PF and RN: All rendezvous nodes (RN) or all brokers (PF) create bottlenecks. Using EF, this effect appears at $350,000$ profiles, since the four brokers can process approximately four times more profiles (no duplications).

5 Conclusions

Several distributed filter algorithms have been proposed for publish/subscribe systems. So far, a systematic comparison and analysis of these filter algorithms had not been achieved. In this paper, we proposed the first classification scheme for distributed filtering algorithms for publish/subscribe systems. In a second step, we classified existing filter algorithms according to the proposed concise scheme. As a third step, we analytically evaluated 17 algorithms based on their features according to the classification dimensions. Out of the 17 algorithms, we selected the 3 most promising ones: event forwarding (EF), profile forwarding (PF), and rendezvous nodes (RN). In an extensive experimental analysis, we evaluated these three algorithms. The results of the experimental analysis support the findings of the theoretical analysis and yield a finer grained insight into the behavior of each algorithm under different conditions. A detailed discussion of the results can be found in [2]. Many existing evaluations have used simulated data, e.g., in [4, 12, 13]; others have measured different factors [8]. We used no simulation data nor a simulated network topology. The real publish/subscribe system DAS was used throughout upon real data sets. DAS has been developed for event monitoring in facility management systems.

We conclude our experimental analysis of algorithms with the following recommendations based on the underlying applications and network topologies. We refer to our key measurements filter efficiency, network load, and duplication of profiles:

Filter efficiency: For most applications, profile forwarding (PF) is the most efficient algorithm. Especially if there is a low proportion of matching profiles or events, PF

is significantly more efficient than EF or RN. For a high proportion of matching profiles, the three algorithms converge, since all events have to be flooded. In rare cases with high proportions of matching profiles, EF is the most efficient algorithm. The reason is the simplicity of the filter protocol with its low overhead. Rendezvous nodes show mediocre results for all applications and topologies: They prove to be of no benefit, as inner nodes always have to forward all events.

Network load per event: The network load in event forwarding (EF) is independent from any other system parameters (except the number of brokers), since all events are always flooded. PF causes the least network load because only matching events are forwarded. Rendezvous nodes show mediocre results, since events are always forwarded to the rendezvous nodes. When increasing p^e or p^p, the network load also increases for PF and RN, but never reaches that of EF.

Duplication of profiles: Duplication is highest when using PF. For unique profiles, the duplication is especially high, because each broker filters each profile – this implies high memory usage. The same picture holds for RN but in a smaller degree. Coverings eliminate duplications for PF and RN. In EF, profiles are not duplicated and therefore the highest number of profiles can be filtered.

Due to this dependency of the filter algorithms on the system's parameters, a publish/subscribe system should support different filter algorithms. According to the system load and application, the system can choose an optimal algorithm:

If many profiles match an event we should choose event forwarding (EF) with its simple protocol. Event forwarding does not cause significant network load since the events have to be forwarded through the network anyway. We should also use EF in case of high numbers of subscribed profiles (profile duplication and memory use are lowest). In most of the other cases (fewer profiles, small portions of matching events, coverings), profile forwarding (PF) should be used. This algorithm causes less network load and the filtering is significantly more efficient than for EF and RN. Unfortunately, rendezvous nodes (RN) have not been advantageous in any tested system configuration. One of the reasons is the limited size and variation of the used broker topology. This first cut analysis is scheduled to be extended using larger and more general topologies in computer grids.

The idea of an adaptive system that uses the appropriate filter algorithm depending on the applications parameters has been implemented in A-mediAS, an adaptable mediating event notification system [5]. A-mediAS uses adaptation only for local filtering of primitive and composite events. We plan to integrate DAS's adaptable distributed filter algorithms within A-mediAS. Another step for future research is the use of distributed filter algorithms for composite events in grid topologies and mobile environments, imposing advanced requirements on the algorithms and the system's adaptability.

References

1. G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, Austin, USA, June 1999.

2. S. Bittner and A. Hinze. Design and analysis of an efficient distributed event notification service. Technical Report 11/2004, Computer Science Department, University of Waikato, New Zealand, August 2004.
3. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Interfaces and Algorithms for a Wide-Area Event Notification Service. Technical Report CU-CS-888-99, Computer Science Department, University of Colorado, October 1999.
4. A. Carzaniga, D. S. Rosenblum, and A. L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.
5. A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universität Berlin, July 2003.
6. A. Hinze and S. Bittner. Efficient Distribution-based Event Filtering. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 525–532, Vienna, Austria, July 2002.
7. G. Mühl. Generic Constraints for Content-Based Publish/Subscribe Systems. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, pages 211–225, Trento, Italy, September 2001.
8. G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische Universität Darmstadt, September 2002.
9. G. Mühl and L. Fiege. Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs. *IEEE Distributed Systems Online (DSOnline)*, 2(7), July 2001.
10. G. Mühl, L. Fiege, and A. Buchmann. Filter Similarities in Content-Based Publish/Subscribe Systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS '02)*, pages 224–238, Karlsruhe, Germany, April 2002.
11. P. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 611–618, Vienna, Austria, July 2002.
12. A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Proceedings of the 3rd International Workshop on Networked Group Communications (NGC 2001)*, pages 30–43, London, UK, November 2001.
13. D. Tam, R. Azimi, and H. Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *Proceedings of the First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P), LNCS 2944*, pages 138–152, Berlin, Germany, September 2003.
14. H. Yu, D. Estrin, and R. Govindan. A Hierarchical Proxy Architecture for Internet-scale Event Services. In *Proceedings of IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '99)*, pages 78–83, Stanford, USA, June 1999.