On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems

Sven Bittner & Annika Hinze
University of Waikato
Hamilton, New Zealand
{s.bittner, a.hinze}@cs.waikato.ac.nz

Abstract

Current matching approaches in pub/sub systems only allow conjunctive subscriptions. Arbitrary subscriptions have to be transformed into canonical expressions, e.g., DNFs, and need to be treated as several conjunctive subscriptions. This technique is known from database systems and allows us to apply more efficient filtering algorithms. Since pub/sub systems are the contrary to traditional database systems, it is questionable if filtering several canonical subscriptions is the most efficient and scalable way of dealing with arbitrary subscriptions. In this paper we show that our filtering approach supporting arbitrary Boolean subscriptions is more scalable and efficient than current matching algorithms requiring transformations of subscriptions into DNFs.

1. Introduction

Most matching algorithms in pub/sub systems exclusively measure quality in terms of efficiency. Hence, they only support very limited subscription languages, i.e., subscriptions are defined as conjunctions of attribute-operator-value triples (predicates). Restricting the expressiveness of subscriptions in that way allows for the application of efficient matching algorithms. These algorithms do not have to take into account the various combinations of predicates, they only have to determine fulfilled predicates and test if the number of matching predicates equals the total number of predicates of a subscription (this is known as the counting algorithm [15, 17]).

For more sophisticated subscriptions (i.e., subscriptions involving arbitrary Boolean combinations of predicates) current approaches require transforming subscription definitions into disjunctive normal forms (DNFs) and treating each disjunction as a separate subscription [8, 14]. Hence, we can apply fast matching algorithms that only support conjunctive predicates. However, this approach has the drawback of limiting existing scalability restrictions even further: Filtering algorithms are designed as pure main memory solutions, hence their scalability depends on available resources. Boolean expressions transformed into DNFs are exponential in size (worst case) compared to their original expressions. Thus, after transformations already limited memory resources are even more utilised. Consequently, it is very questionable if such transformations into canonical expressions have even advantages for routine employment, i.e., conventional machines do not have processors fast enough nor main memories large enough for current (i.e. canonical) matching approaches. In typical real world situations we will find peer-to-peer networks of less equipped machines, such as laptops and mobile devices to perform event filtering. Thus, filtering of general subscriptions in practice is still an open problem.

Our approach is to allow arbitrary Boolean subscriptions in pub/sub systems and to obtain filtering using indexes and these explicitly defined subscriptions. Thus, we improve the scalability of pub/sub systems and allow efficient event filtering on other machines than designated servers with "inexhaustible" resources. In detail the contributions of this paper are:

1. We present a novel matching algorithm applicable to pub/sub systems (Sect. 3)

2. We show that our approach is more space efficient and thus scalable than current matching algorithms (theoretically in Sect. 2 and practically Sect. 4)

3. We show that filtering original subscriptions is more time efficient than filtering subscriptions transformed into DNFs (theoretically in Sect. 2 and practically in Sect. 4)

The remainder of this paper is structured as follows. In Sect. 2 we give a short overview of state-of-the-art matching algorithms and reasons why they use conjunctive subscriptions only. The essentials of our new filtering approach are presented in Sect. 3. Section 4 describes experiments showing the advantages of our non-transforming attempt over a

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

transforming algorithm. Finally, some of the future work is presented in Sect. 5 as well as the conclusion of this paper.

2. Event filtering algorithms

Nearly all filtering approaches in general purpose pub/sub systems only support conjunctive subscriptions (e.g. Gryphon [1, 3], Le Subscribe [8], SIENA [5] and the approaches in [2, 9, 10, 14]). Thus, arbitrary Boolean subscriptions need to be converted to canonical expressions [7, 10, 14] to allow for their matching. Then, these algorithms treat disjunctions as several subscriptions [8, 14]. This results in heavy memory consumption, since Boolean subscriptions transformed into DNFs are exponential in size compared to their original expressions. In [6] disjunctions of conjunctive predicates are allowed in subscriptions. Indeed this is an improvement of solely conjunctive subscriptions, but arbitrary Boolean subscriptions still have to be transformed into the supported canonical form and are not polynomial in size compared to their original expressions.

The motivation for these current approaches with restricted subscriptions is to achieve very fast event filtering to obtain notifications in a short delay [8] assuming designated filtering servers equipped with fast processors and tremendous main memories. Unfortunately, the improvement of efficiency comes with a restriction of the expressiveness of subscriptions. To balance this behaviour, techniques known from database systems have to be applied, i.e., arbitrary Boolean subscriptions are required to be transformed into canonical expressions. However, it remains questionable if this technique should be used in the context of pub/sub systems. We claim it is a more promising option to filter on general subscriptions and show it practically in Sect. 4. In the following section we firstly present current filtering algorithms. Secondly, we show the impact of canonical expressions in these approaches on filtering efficiency and scalability.

2.1. Brief overview of filtering approaches

We examine filtering algorithms for pub/sub systems according to three quality measures: We firstly have a look on time efficiency, i.e., the performance of an algorithm for its processable problem size. Secondly, we focus on space efficiency, i.e., how cautious main memory resources are used by algorithms. Our last important quality measure is scalability, i.e., with which performance growing problem sizes are handled by an algorithm. In current pub/sub systems space efficiency directly influences scalability, since algorithms are designed as main memory solutions. Hence, their processable problem sizes depend on resource usage, i.e., space efficiency.

Current filtering algorithms in pub/sub systems can generally be classified into three categories applying

- no index structures, e.g. [4, 16]
- one-dimensional index structures, e.g. [8, 10, 15, 17]
- multi-dimensional indexes, e.g. [1, 9]

A popular notification service applying no indexes for filtering is Elvin [16]. Prominent examples of one-dimensional algorithms are the counting algorithm [15, 17] and Hanson's approach [8, 10], both exploiting the restriction to conjunctive subscriptions for efficiency improvements. Popular multi-dimensional algorithms are tree-based, such as the approaches from Gough [9] and Aguilera [1]. There traversing a matching tree results in obtaining all matching subscriptions, since only conjunctive subscriptions can be used.

Applying indexes means to evaluate each attribute only once, i.e., all predicates of subscriptions involving a certain attribute are tested using an index. Without indexes several evaluations per attribute are performed, i.e., the predicates are tested independently. Matching times in case of non-index filtering grow linearly with the number of subscriptions and have a strong gradient which makes this approach inapplicable for practical solutions with time constraints targeting large subscription quantities. However, non-index approaches mostly allow non-canonical expressions in subscriptions, such as Elvin [16] and [4], resulting in more expressive subscription languages.

Exploiting indexes results in faster filtering [11, 16]. One-dimensional index structures need two steps to determine matching subscriptions, multi-dimensional ones allow filtering in one step. Furthermore, matching using multi-dimensional indexes allows for the evaluation of required predicates only, i.e., evaluated predicates depend on already fulfilled ones. Applying one-dimensional indexes instead means to evaluate predicates of attributes independently from previous matching steps. Hence, regarding time efficiency multi-dimensional indexes are a better choice than one-dimensional ones [11].

In addition to time efficiency we also have to have a look at space efficiency of algorithms. Assuming that algorithms in all three categories have to store subscriptions, applying no indexes shows the best space efficiency. Storage of subscriptions is always required if a more sophisticated subscription language than conjunctions only is supported by a system. Furthermore, to efficiently support unsubscriptions we require an association between subscriptions and predicates, e.g., by storing lists of predicates of subscriptions. Since current indexing algorithms assume conjunctive subscriptions they do not need to store subscriptions explicitly, but only require an association list as mentioned before.

Generally, one-dimensional indexes have advantages in respect to space efficiency over multi-dimensional ones, be-

cause one-dimensional indexes store predicates only once. Multi-dimensional ones might index predicates several times depending on other predicates of their subscriptions.

According to our reasoning in the beginning of this section out of space efficiencies of main memory algorithms we can derive scalability characteristics. Thus, concerning scalability we should prefer non-indexing approaches over algorithms exploiting one-dimensional indexes. Worst scalability is achieved if applying multi-dimensional indexes.

Concluding, when dealing with restricted resources applying one-dimensional indexes is the best option, since they require far less memory than multi-dimensional ones and show much better time efficiency than non-index approaches [11, 16]. Indeed, one-dimensional indexes are slower than multi-dimensional techniques, but this can be neglected out of scalability reasons which we have to look at in practical use. Non-indexing approaches should be avoided due to their characteristic to evaluate all subscriptions for each event.

2.2. On the usage of canonical expressions

The practice of query rewriting into canonical forms is adopted from database systems. For database queries, it is fundamental to transform restrictions on result sets into canonical expressions: Queries are transformed to have a common starting point for all queries to perform query optimisation [12]. Then, after a transformation into canonical expressions, queries are simplified by applying various transformation rules. When using DNFs each element within the disjunction is handled and optimised separately [13]. Finally, access plans are created and the cheapest one is executed [12]. These query rewriting and execution techniques are chosen because the main costs in databases are caused by accessing and combining a huge amount of data over several tables and columns. Hence, time efficient query execution depends on how efficiently data is accessed, which in turn, depends on physical data storage. Thus, regarding main memory usage it is not crucial if queries are transformed to improve performance: A database system can deal with the problem of expanded memory usage because of the small quantity of queries evaluated at one time.

Pub/sub systems are the contrary to traditional databases: They deal with huge amounts of subscriptions and data occurs in form of event messages. Because of this converse problem definition we argue that it is not sufficient to apply solutions from database systems to pub/sub. Current research has addressed these differences in regard to filtering by developing algorithms better applicable to pub/sub systems [2, 15].

The problem still persists regarding internal representation and processing of subscriptions in pub/sub systems. It is very questionable if the best strategies are either transforming all subscriptions (which are really numerous in contrast to queries in database systems) into canonical expressions (e.g. [7, 10, 14]) requiring more than polynomial space in worst case or solely focusing on very restricted subscriptions (e.g. [5, 6, 8, 15]) and leaving transformations to users. Unlike in database systems, in pub/sub systems this means to multiply memory problems. Furthermore, current matching approaches do not optimise subscriptions, which is a main reason for query transformations in database systems.

Besides this, filtering over canonical expressions means to execute redundant computations. Consider, e.g., the counting algorithm after the registration of non-canonical subscriptions with unique predicates (the pub/sub system either transforms itself or requires a transformation by users): If one unique predicate is fulfilled we have to increase a counter for several subscriptions in the predicate matching step (this is because after a transformation predicates are part of several conjunctive subscriptions). Furthermore, the subscription matching step works on a multiple of the number of original registered subscriptions. Hence, a large extend of performed computations are redundant and a direct result of canonical transformations. This also holds if users have to transform their subscriptions themselves.

However, the advantage of transforming into canonical expressions in pub/sub systems or the neglect of arbitrary Boolean subscriptions is to circumvent evaluations of Boolean expressions. Subscription matching can be obtained by simply counting the number of fulfilled predicates per subscription and comparing this value to the total number of predicates. Consequently, algorithms become faster and easier to implement. Up to now it was questionable if these latter advantages outbalance the former mentioned disadvantages in practice. Section 4 presents an answer to this question, but beforehand we present an essential detail in the next section: our matching approach.

3. Non-canonical filtering

This section introduces our filtering approach for non-canonical subscriptions. This algorithm overcomes existing expressiveness restrictions in conjunctive subscriptions and allows for the direct filtering of arbitrary Boolean subscriptions. Furthermore, it prevents scalability constraints due to the avoidance of transforming Boolean subscriptions into canonical forms.

3.1. Subscription representation

We define a subscription s as an arbitrary Boolean expression with predicates p as variables using Boolean operators AND, OR and NOT. Predicates p are filters in form of

attribute-operator-value triples and might be shared among different subscriptions. Each subscription may have an arbitrary number of predicates regarding any attribute of an event. Both predicates p and subscriptions s can be uniquely identified by their identifiers id(p) and id(s), respectively. An event e matches a subscription s if the Boolean expression of s evaluates to *true*. Thereby the variables of this expression are represented by the results of the respective predicates of s applied to e.

Internally, subscriptions are compiled into subscription trees representing their Boolean expression and their predicates, i.e., inner nodes are marked with Boolean operators and leaf nodes represent predicates. Binary operators are treated as n-ary ones due to compacting subscription trees. Predicates p are represented by their identifiers id(p) instead of their filter operations. A simplified example of a subscription tree is illustrated in Fig. 1 for the subscription $s = (a > 10 \vee a \leq 5 \vee b = 1) \wedge (c \leq 20 \vee c = 30 \vee d = 5)$. To register this subscription s in canonical approaches, s has to be transformed into DNF. Thus, s results in 9 disjunctions that are required to be treated separately.

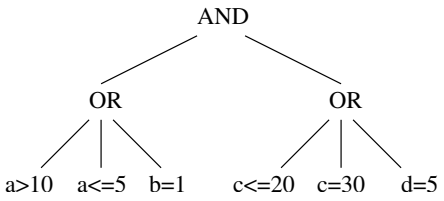


Figure 1. Example of a subscription tree.

Internally we store predicate identifiers id(p) in leaf nodes in contrast to predicates p themselves as shown in Fig. 1. For predicates we utilise indexes to allow for a fast determination of all predicates matching an event. This filtering process is described in detail in the next section. Subscription trees are encoded to decrease memory usage. The memory address of a subscription is denoted by loc(s).

3.2. Event filtering

The process of event filtering involves four major data structures that are shown in Fig. 2: We utilise one-dimensional indexes, a predicate subscription association table storing (id(p), $\{$id(s)$\}$) tuples, a subscription location table containing (id(s), loc(s)) tuples and subscription trees to store subscriptions themselves.

In the first step of event filtering (predicate matching) all predicates matching an event e are determined, i.e., all predicates evaluating to *true* if applied to e. This is accomplished by the application of one-dimensional index structures such as hash tables or B+ trees. These indexes are applied based on operators used in predicates, e.g., point predicates utilise hash tables, for range predicates we deploy B+ trees. This first part of filtering is illustrated in the upper part of Fig. 2. The output of predicate matching is a list of identifiers $\{$id(p)$\}$ of matching predicates.

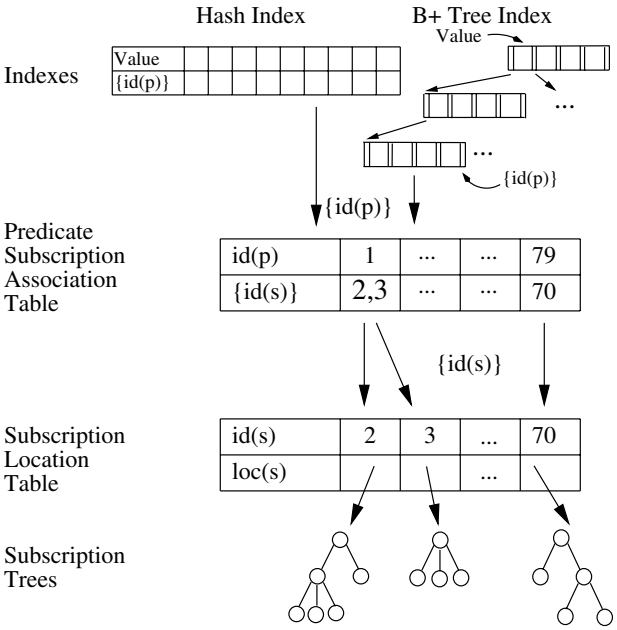


Figure 2. Overview of data structures involved in the filtering process.

The second step of event filtering (subscription matching) is depicted in the lower parts of Fig. 2 and works as follows: We firstly determine all candidate subscriptions, i.e., subscriptions including at least one of the matching predicates obtained in step one. This is achieved by the help of the predicate subscription association table (cf. Fig. 2) resulting in a list of subscription identifiers $\{$id(s)$\}$. Secondly, we utilise the subscription location table to obtain memory addresses loc(s) of subscription trees of candidate subscriptions s. Thirdly, we evaluate the Boolean expressions of all candidate subscriptions (the values of variables, i.e., matching predicates, have already been obtained in step one and are known in step two). Finally, all subscribers with subscription trees evaluating to *true* are notified.

Hence, predicates are only evaluated in predicate matching. Subscription matching evaluates Boolean expressions of candidate subscriptions with information obtained in step one. Several optimisations could be applied to the process of subscription matching presented here (e.g. reordering subscription trees or a general space optimisation); their impact remains to be investigated.

Our approach does not multiply memory usage by transforming subscriptions into canonical expressions. But unlike current algorithms, we explicitly store subscriptions

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'05)
1545-0678/05 $20.00 © 2005 **IEEE**

and thus require memory for their storage. Several matching approaches neglect the storage of subscriptions, e.g., the counting algorithm only stores the number of predicates each subscription consists of. This entails complications when supporting unsubscriptions as already outlined in Sect. 2.1[1].

3.3. Prototype

We implemented a prototype of our filtering approach. Subscriptions are encoded in a basic and thus not the most space efficient way (in practical employment we should further improve this aspect). We only encode them on a byte level, e.g., to encode a Boolean operator we require one byte, also the number of children for inner nodes is encoded by one byte. Furthermore, the width of children is stored using two bytes each and predicate identifiers require four bytes. Improvements to this approach are left to future work here.

Besides this "waste" of memory in our prototype we want to compare the behaviour of our algorithm to a memory friendly implementation of the counting algorithm [2] without the support of unsubscriptions. Thus, we require a predicate bit vector, a hit vector, a subscription-predicate count vector and a predicate subscription association table[2]. According to [2] we assume a maximum of 256 predicates per subscription and use 1 byte per entry in hit and subscription-predicate count vector.

We also implemented a variation of the counting algorithm: In subscription matching we do not compare the whole hit vector and subscription-predicate count vector. Instead, in the beginning of step two for matching predicates we record all subscriptions they belong to. Afterwards, we only compare the entries of these subscriptions in hit vector and subscription-predicate count vector. Thus, our variation of the counting algorithm rather depends on the number of matching predicates than on the total number of subscriptions.

We choose the counting algorithm for our comparison because it easy to implement, requires no additional statistical information and makes no further assumptions about subscriptions and events, such as [8]. Space efficiency of [8] shows only slight differences to the counting algorithm. Thus, in respect to scalability the results of out experiments also hold for the algorithm presented in [8]. We are aware that the counting algorithm is not the most efficient filtering solution, but our implementation variation improves the time efficiency of the counting algorithm in a similar manner as [8], since only candidate subscriptions are considered in subscription matching.

[1] If we do not use data structures to determine all predicates that belong to each subscription we have to access the whole predicate subscription association table to find the predicates of a subscription.

[2] We choose an implementation similar to the list-based one in [2] to require as little memory as possible and thus improve the memory usage of the algorithm. Since we know the number of subscriptions per predicate we use arrays instead of a subscription list.

4. Experiments

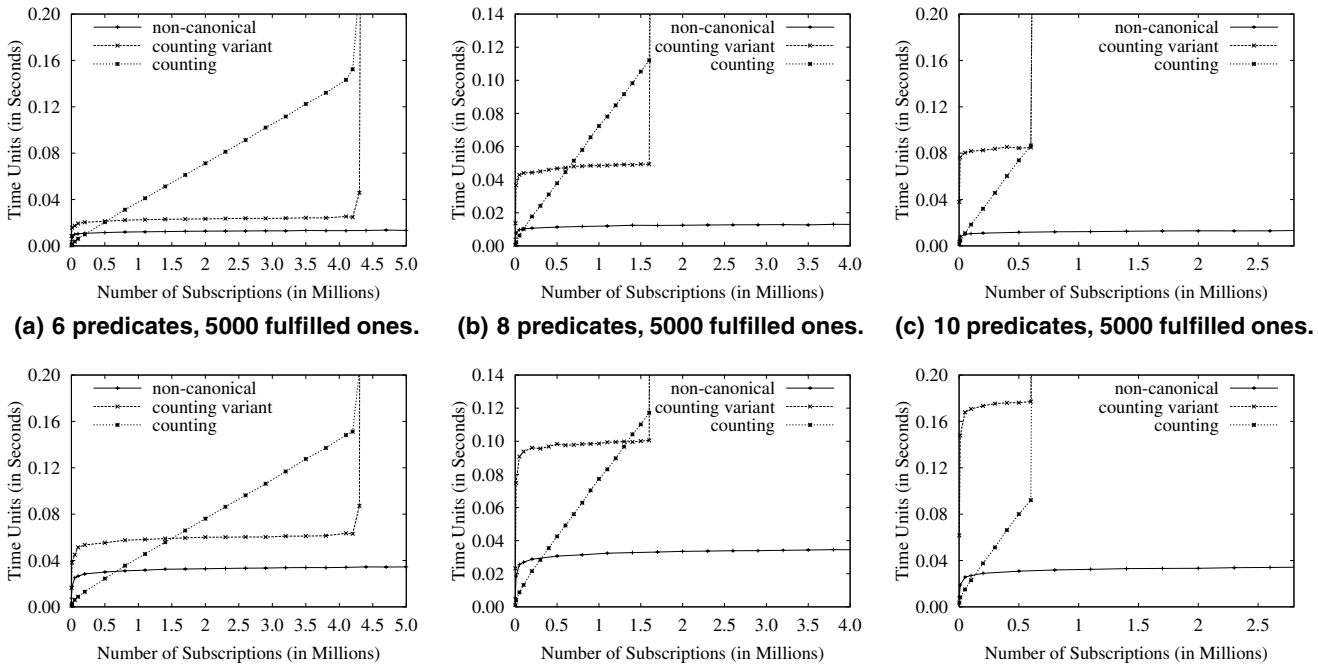
In this section we describe a first practical evaluation of our non-canonical matching algorithm. We have run several experiments to show the impact of transformations into canonical expressions. In these experiments we compare our filtering algorithm to the two variants of the counting algorithm presented in the last section. We only need to compare the second phases (subscription matching) of the algorithms, since the first phases use the same indexes in the same way in both approaches resulting in the same computation times for predicate matching.

We avoid the usage of shared predicates in order to directly observe the influence of increasing numbers of subscriptions on both time and space efficiency. We do not assume high predicate redundancy, i.e., domains are supposed to have relatively large sizes and subscribers are interested in different events. Our subscriptions are non-DNF expressions and are characterised by their numbers of predicates $|p|$. If they are transformed into DNFs to be usable by the counting algorithm one original subscription results in $2^{\frac{|p|}{2}}$ subscriptions with $\frac{|p|}{2}$ predicates each. The other test and test system parameters are shown in Table 1. We have run our experiments several times in order to obtain variances under 1 %. Hence, it is not required to present variances in our results.

The experiments and their results presented in this paper are an initial comparison of our non-canonical filtering approach to recent canonical filtering attempts. Thus, our examination does not allow for an all-embracing analysis of our approach but gives a first demonstration of its usefulness and advantages and shows the impact of transformations into canonical expressions.

Parameter	Value
CPU speed	1.8 GHz
Total machine memory	512 MB
Number of subscriptions	2,000 – 5,000,000
Number of original (unique) predicates per subscription	6 to 10
Number of subscriptions per subscription after transformation	8 to 32
Used Boolean operators	AND, OR
Matching predicates per event	5,000 – 10,000

Table 1. Parameters in experiments.



(a) 6 predicates, 5000 fulfilled ones. **(b) 8 predicates, 5000 fulfilled ones.** **(c) 10 predicates, 5000 fulfilled ones.**

(d) 6 predicates, 10000 fulfilled ones. **(e) 8 predicates, 10000 fulfilled ones.** **(f) 10 predicates, 10000 fulfilled ones.**

Figure 3. Results using 6 to 10 predicates with varying numbers of fulfilled predicates.

4.1. Experimental results

Our experimental results are shown in Fig. 3. Abscissae show an increasing number of subscriptions (in millions), ordinates present matching times (in seconds) for subscription matching per event with the given number of fulfilled predicates per event.

The matching time of the counting algorithm increases linearly with the number of registered subscriptions. This observation is consistent with other works [2, 8] and results out of the fact that the number of matching predicates has to be compared to the total number of predicates for all registered subscriptions. Furthermore, if less subscriptions are created by transformations the counting algorithm shows better scalability (Fig. 3(a) and Fig. 3(d) compared to Fig. 3(c) and Fig. 3(f), respectively). Sharp bends in the curves denote the point when available main memory resources are exhausted and the operating system starts page swapping, e.g., from approx. 1,600,000 subscriptions in Fig. 3(b). For small subscription numbers (e.g. up to 700,000 subscriptions in Fig. 3(d)) the counting algorithm behaves most efficient compared to other approaches due to the small number of required comparisons.

The variant of the counting algorithm does not behave linearly with the number of subscriptions. This is because not all subscriptions have to be evaluated in subscription matching. Hence, the variant behaves better than the original counting algorithm in cases of large quantities of subscriptions. Small numbers of subscriptions require more overhead for creating a list of candidate subscriptions than saved computation costs for comparisons of the numbers of fulfilled predicates for non-candidate subscriptions. However, the scalability of the variant is restricted in the same way as the one from the original counting algorithm: Transformations (required due to the support of conjunctive subscriptions only) result in the same large number of registered subscriptions, i.e., approx. 700,000 original subscriptions fill available main memory resources when using 10 predicates (Fig. 3(c) and Fig. 3(f)).

Our non-canonical approach shows several advantages. On the one hand it shows much better scalability than transforming algorithms (in case of 10 predicates in Fig. 3 it easily handles more than 4 times as many subscriptions). On the other hand it shows better performance than the counting approach in all cases except for small subscription quantities. And it always achieves better time efficiency than the implemented variant of the counting algorithm. Here it becomes obvious that the overhead of redundant computations after transformations in the counting approach is enormous compared to the overhead of evaluating Boolean subscriptions in our approach (as argued in Sect. 2.2). Due to the increasing overhead after transformations the difference in time efficiency between our approach and the variant of the counting algorithm becomes larger in cases of growing numbers of transformed subscriptions (cf. Fig. 3(d) over Fig. 3(e) to Fig. 3(f)). In cases of increasing subscription

numbers our algorithm shows only slightly decreasing time efficiency. Its performance (as well as the one from the variant of the counting algorithm) is more dependent on the number of fulfilled predicates per subscription than the performance from the original counting approach. This results out of the different handling of non-candidate subscriptions.

Concluding, our experiments show that

1. Transformations into DNFs radically drop the scalability properties of an event filtering algorithm
2. The Filtering of several conjunctive subscriptions instead of arbitrary Boolean ones decreases efficiency

The scalability decrease of transformations arises from the increasing problem size, i.e., subscription numbers. This overhead cannot be outbalanced by cutting down the complexity of each subscription, i.e., to allow only conjunctive subscriptions (this effect becomes even more apparent if supporting unsubscriptions, cf. Sect. 3.2). Decreased efficiency characteristics are caused by the overhead of redundant computations that are required due to transformations (cf. Sect 2.2).

5. Conclusion and future work

In this paper we have investigated the impact of the transformation of subscriptions on efficiency and scalability of filtering algorithms. We firstly analysed motivations of current approaches to either perform transformations or solely support conjunctive subscriptions and leave the tasks of transformations to users. Secondly, we argued that non-transforming algorithms behave better. Then, we presented our matching approach treating subscriptions as they are without a generation of canonical expressions. Finally, we compared our algorithm to two variants of the counting approach.

In experiments, we have shown the advanced scalability properties of our approach compared to algorithms requiring transformations. We also found that our approach outbalances the efficiency properties of other algorithms in case of non-canonical subscriptions. These two advantages of our approach are based on less overhead for filtering of arbitrary Boolean subscriptions compared to additional computation costs after transforming such subscriptions into canonical expressions.

In the future we want to theoretically investigate memory consumptions of different filtering algorithms. Additionally, we will perform experiments with more general subscriptions using an improved encoding and analyse the influence of various system and subscription parameters on efficiency and scalability. A further step is the development of filtering strategies exploiting other resources than main memory.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching Events in a Content-Based Subscription System. In *Proceedings of PODC '99*, pages 53–61, Atlanta, USA, May 4–6 1999.

[2] G. Ashayer, H. A. Jacobsen, and H. Leung. Predicate Matching and Subscription Matching in Publish/Subscribe Systems. In *Proceedings of ICDCSW '02*, Austria, July 2002.

[3] 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 ICDCS '99*, Austin, USA, May 31–June 4 1999.

[4] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams. In *Proceedings of ICSE 2001*, pages 443–452, Toronto, Canada, May 12–19 2001.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[6] A. Carzaniga and A. L. Wolf. Forwarding in a Content-Based Network. In *Proceedings of SIGCOMM '03*, pages 163–174, Karlsruhe, Germany, March 24–26 2003.

[7] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of SIGMOD 2000*, USA, May 2000.

[8] F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *Proceedings of SIGMOD 2001*, Santa Barbara, USA, May 21–24 2001.

[9] J. Gough and G. Smith. Efficient Recognition of Events in a Distributed System. In *Proceedings of ACSC-18*, 1995.

[10] E. N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A Predicate Matching Algorithm for Database Rule Systems. In *Proceedings of SIGMOD 1990*, USA, May 1990.

[11] A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universität Berlin, Institute of Computer Science, July 2003.

[12] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16(2):111–152, 1984.

[13] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing Disjunctive Queries with Expensive Predicates. In *Proceedings of SIGMOD 1994*, USA, May 1994.

[14] G. Mühl and L. Fiege. Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs. *IEEE DSOnline*, 2(7), 2001.

[15] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient Matching for Web-Based Publish/Subscribe Systems. In *Proceedings of CoopIS 2000*, Israel, September 2000.

[16] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Australia, September 1997.

[17] T. W. Yan and H. García-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM TODS*, 19(2):332–364, 1994.