A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems based on Modified Binary Decision Diagrams

Guoli Li   Shuang Hou   Hans-Arno Jacobsen
Middleware Systems Research Group, University of Toronto
{gli,shou}@cs.toronto.edu   jacobsen@eecg.toronto.edu

Abstract

The challenge faced by content-based publish/subscribe systems is the ability to handle a vast amount of dynamic information with limited system resources. In current p/s systems, each subscription is processed in isolation. Neither relationships among individual subscriptions are exploited, nor historic information about subscriptions and publications is taken into account. We believe that this neglect limits overall system efficiency. In this paper, we represent subscriptions using modified binary decision diagrams (MBDs), and design an index data structure to maintain distinct predicates and manage associated Boolean variables. Our MBD-based approach can address, in a unified way, publication routing and subscription/advertisement covering and merging. We propose a novel covering algorithm based on MBDs. The algorithm can take historic information about subscription and publication populations into account and exploits relations between subscriptions. We explore merging, especially imperfect merging, and discuss an advertisement-based optimization applicable to subscription merging.

1. Introduction

A publish/subscribe (p/s) system is comprised of information producers who publish and information consumers who subscribe to information. The p/s system matches or routes relevant information to interested subscribers. There has been great interest in p/s over the past few years. In academia, the focus has been on developing efficient matching algorithms for centralized p/s [9] and on content-based routing [6, 11, 8] for distributed p/s architectures. In industry, several standards have emerged that define common application programming interfaces for p/s-style interactions, such as the CORBA Notification Service, the OMG Data Dissemination Service specification, and, partially, the Java Messaging Service. Applications of p/s technology range from selective information dissemination, location-based services, network management, to workload management and scheduling.

For Internet-scale adoption of p/s, one of the main challenges that still have to be overcome, is the ability to handle large amounts of dynamically changing information content and interest specifications (i.e., publications and subscriptions) at the p/s broker. A publication arriving at a broker has to be correlated against the entries in the routing table to identify the proper forwarding targets [6]. The larger the routing table, the more expensive this routing decision becomes.

Subscription covering and merging have been proposed to alleviate this problem [11]. However, current approaches propose fragmented solutions that are either able to do efficient routing [8, 12], compute cover relations [6], or determine perfect merging relations [11].

All existing covering approaches must compute for each subscription two relations in the worst case (i.e., is-covering and is-covered-by). Moreover, in many current p/s systems, each subscription is processed and stored independently. That is neither historic information about subscriptions and publications, nor semantic or logic relationships between individual predicates in subscriptions is taken into account in processing. Historic information, for example, may reveal popular predicates and popular combinations of predicates, which can serve to optimize routing, covering, and merging (e.g., a plane ticket always has a source, destination, and price predicate (semantic relationship)). Logic relationships can serve to infer the truth/falsehood of individual predicates, if the relationship between such predicates is tracked. Existing approaches cannot take full advantage of this kind of information.

We propose novel data structures and algorithms for the design of a p/s broker that unifies routing, covering, and merging in one model and can take advantage of the above mentioned auxiliary information (i.e., statistics on predicate popularity, etc.). The contributions of this paper are threefold. First, we present data structures for managing subscriptions based on modified binary decision diagrams (MBD). Our data structures support in a unified man-

ner publication routing, subscription covering and merging. Moreover, our underlying predicate data structure ensures that only distinct predicates are assigned different Boolean variables. Thus, overlap at the predicate level is avoided and only unique predicates are stored and evaluated by the system. This stage also ensures that *related* predicates can be placed in favorable proximity in the variable ordering underlying the MBDs, which demonstrates performance gains. Second, we propose an optimized one-phase covering algorithms to detect covering relations among subscriptions and a merging algorithm to identify mergers[1]. Third, we discuss merging-based routing and explore the characteristics of imperfect merging. Finally, an advertisement-based optimization is proposed to enhance imperfect merging.

This research is part of the Toronto Publish/Subscribe System research efforts [14, 10, 4, 13, 16] and specifically part of the PADRES project. The PADRES project is a content-based p/s middleware platform, with features inspired by the requirements of workflow management and business process execution systems. PADRES consists of a set of brokers connected by a peer-to-peer overlay network. Clients connect to brokers using binding interfaces such as Java Remote Method Invocation (RMI) and Java Messaging Service (JMS). Message routing in PADRES is based on the advertise-publish-subscribe model.

The paper is organized as follows. Section 2 combines the presentation of background material and related work, as they are closely coupled in our discussion. Section 3 proposes our MBD-based architecture for the p/s broker design. Section 4 presents our novel subscription covering algorithm. In Section 5, we discuss merging-based routing and explain how to create a new merger based on perfect and imperfect merging rules. In Section 6, we experimentally validate our approach and compare the performance of perfect merging, imperfect merging, and our covering techniques with common alternatives.

2. Background and Related Work

To keep this paper self-contained, we provide an overview of key concepts used in the reminder of this paper. This includes content-based routing, covering, merging and binary decision diagrams. We also discuss related work along these dimensions.

Content-based Routing: The goal of content-based p/s systems is to direct messages from source to destination based entirely on the content of the messages. In a content-based p/s system, a publication P is a set of attribute-value pairs. Formally, P has a set of attributes $\{a_1, a_2, ..., a_i, ...a_n\}$, and is described as $P = \{(a_1, v_1), (a_2, v_2), ..., (a_n, v_n)\}$. A subscription

---

1 With one-phase, we mean that our algorithm computes all covering relations at once

---

S is specified as relations between attributes and values. It is a conjunction of a set of predicates which are also often called attribute filters. A predicate *(attribute,operator,value)* is a constraint on the value of the attribute. In advertisement-based p/s systems, a publisher issues an advertisement before it publishes. An advertisement has the same format as a subscription. An advertisement allows the publisher to issue a set of matched publications. Fig. 1 shows a scenario of advertisement-based, content-based routing. Advertisements are used to avoid broadcasting subscriptions in the network, since subscriptions are only routed to the publishers who advertise what the subscribers are interested in. In the brokers the subscription routing table (SRT) is used to route subscriptions. Publications will trace back along the path setup by subscriptions to interested subscribers. The publication routing table (PRT) maintains the path information.
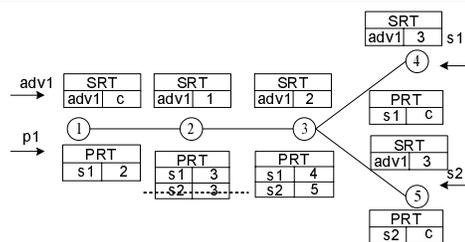

Figure 1. Advertisement-based Content-based Routing

We call a subscription or an advertisement a filter in the later part of this paper. *P(S)* refers to all publications that match *S*. Two versions of content-based routing are known, simple routing, for example Gryphon [12], and covering-based routing which is discussed in SIENA [6] and JEDI [8]. In this paper, we propose an efficient p/s broker design based on MBDs that unifies routing, covering and merging. Existing approaches, either do not perform all these operations together, or use different algorithms and data structures for each one of these operations.

Covering Definition: The goal of covering-based routing is to remove redundant subscriptions from the network in order to maintain a compacted routing table and reduce network traffic. In Fig. 1, if s_1 covers s_2, we can safely remove s_2 on Broker 2. The concept of covering includes predicate covering and filter covering. For predicate $f_1 = (a_1, op, v_1)$ and $f_2 = (a_2, op, v_2)$, we say f_1 covers f_2, denoted as $f_1 \succeq f_2$, if and only if $a_1 = a_2$ and all attribute-value pairs matching f_2 also match f_1. A filter F_1 covers another filter F_2 if the publication set matching F_2 also matches F_1, that is, $P(F_1) \supseteq P(F_2)$. The formal definition of the covering relation is as following: Sup-

pose $F_1 = f_1^1 \wedge f_2^1 \wedge ... \wedge f_n^1$, and $F_2 = f_1^2 \wedge f_2^2 \wedge ... \wedge f_m^2$, where n≤m. $F_1 \sqsupseteq F_2$ if and only if $\forall f_i^1 \exists f_j^2, f_i^1 \succeq f_j^2$ [11]. The covering relationship defines a partial order on the set of all filters. SIENA [6] and JEDI [8] exploit covering-based routing. Unfortunately, they restrict the expressiveness of content-based routing, and do not consider merging techniques. Subscriptions in SIENA [6] are represented in a partially ordered set (*poset*), and the partial order is defined by the covering relations. However, the *poset* is expensive to maintain because of the nested covering relations. We propose an index data structure to ensure that only distinct predicates are stored and evaluated. Covering rules for predicates of various data types are discussed in REBECA [11]. REBECA uses two separate covering algorithms: one to detect whether a subscription is covered by the given routing table or not, and the other to identify what subscriptions in the routing table are covered by the inserted subscription. In this paper, we develop a novel covering algorithm based on MBDs which combines these two functions.

Merging Definition: The merging technique is used for further minimizing the routing table size and the network traffic overhead in a content-based network. It is an extension of covering. If s_1 and s_2 in Fig. 1 have no covering relations but largely overlap with each other, they can be merged into a more general subscription on Broker 2. A new filter F_M, which covers the original ones, that is $P(F_M) \supseteq P(F_1) \cup P(F_2) \cup ... \cup P(F_n)$, is called a merger of $F_i (i = 1, ..., n)$. There are two kinds of mergers. If the publication set of the merger is exactly equal to the union of the publication sets of the original filters, i.e., $P(F_M) = P(F_1) \cup P(F_2) \cup ... \cup P(F_n)$, then F_M is a *perfect merger*; if $\supset$ holds, that is the merger is larger than the union, it is an *imperfect merger*. Imperfect merging can reduce the number of subscriptions. On the other hand, it may allow publications being forwarded into the network that do not match any of the original subscriptions. REBECA [11] presents an evaluation of several routing algorithms, including covering-based routing and merging-based routing, however, it only focuses on perfect merging. No details for imperfect merging are given. Crespo *et al.* [7] study the complexity of the query merging problem and show that the n-query merging problem is NP-complete. They also discuss and compare several optimal and heuristic algorithms to try to find an optimal merging pattern. In this paper, we discuss merging-based routing, especially imperfect merging, define the imperfect degree for an imperfect merger, and propose an advertisement-based optimization to reduce the number of false positives introduced by an imperfect merger.

BDD-based Approach for Matching: Ordered Binary Decision Diagrams (BDDs) are abstract representations of Boolean functions [3]. The key idea of BDDs is that by representing a Boolean function as a rooted, directed acyclic graph, Boolean manipulations become computationally simpler. BDDs are widely used in a number of areas, such as digital system design, finite-state system analysis and artificial intelligence [3]. In the BDD-based approach to p/s, each predicate is assigned a Boolean variable, which is a symbol representing the predicate. A subscription is expressed using a BDD as a Boolean function of those variables based on a predefined order. Fig. 2 shows a subscription expressed in a BDD with variable order x≺y≺z. Each non-terminal node v has two children: $low(v)$ if the variable is assigned 0, and $high(v)$ if the variable is assigned 1. The labelled values of non-terminal nodes indicate the variable ordering with respect to ≺. A terminal node denoted by a rectangular box is labelled 0 or 1. Each node represents a Boolean function $f = \bar{x} \cdot low(v) + x \cdot high(v)$, denoted as $ITE(x, low(v), high(v))$.
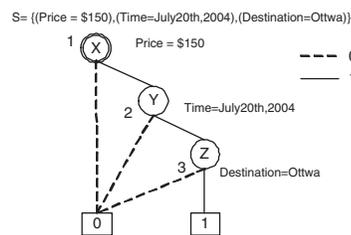

Figure 2. A Subscription BDD

Campailla *et al.* [5] proposed a matching algorithm for p/s systems using Binary Decision Diagrams. Since BDDs can be used to represent arbitrarily complicated Boolean functions, an expressive subscription language can be naturally supported. However, this approach did not demonstrate how the Boolean variables used by the BDDs are organized. Also, the impact of the choice of the variable order on the BDD representation was ignored. The approach was only applied to solving the matching problem for centralized p/s systems, and the evaluations were limited, not showing scalability results at large subscription workloads. In this paper, we use MBDs, an extension of BDDs, to represent routing state for content-based p/s (i.e., distributed p/s), and discuss unified routing, covering and merging techniques and their experimental evaluation.

3. MBD-based Architecture

3.1. Global Predicate Index

The number of distinct predicates in Internet-scale p/s systems is extremely large. We build a predicate index to manage all distinct predicates in a broker efficiently, which is shown in Fig. 3. At system start the index is empty. For each predicate of a given subscription, we traverse the in-
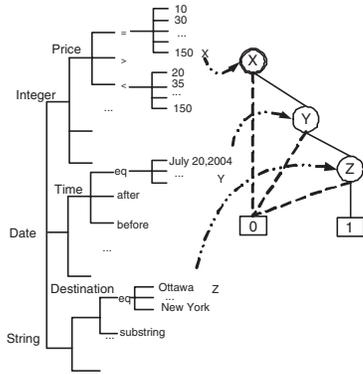
Figure 3. Boolean Variable Index

dex, if we find the predicate, it means the predicate already exists in the broker and an existing variable has already been assigned to it; if the predicate is not found in the index, we insert it as a new one and assign a new Boolean variable to it. After all the predicates have been assigned variables, a BDD for this subscription is build based on these Boolean variables. Using the indexing data structure, we assign Boolean variables to distinct predicates. All the predicates are organized according to data type, attribute, operator, and value in the index. This organization speeds up the process of finding a particular predicate and maintains the predicates more efficiently by storing distinct predicates only once in the broker.

Considering that there maybe a large number of common predicates among subscriptions, we represent several subscriptions using a Modified Binary Decision Diagram (MBD), instead of a single BDD for each subscription. An MBD is an extension of a BDD, which has multiple roots and can represent multiple-output functions [15]. An MBD can be used to express a set of subscriptions that share common predicates. For example, in Fig 3, subscription S_1=(price,=,150)∧(time,eq,'July 20,2004') ∧(destination,eq,'Ottawa') and S_2=(price,<,150)∧(time,eq, 'July20,2004')∧(destination,eq,'Ottawa') share two predicates. They can share the common predicates in one MBD with two output nodes, which is denoted as MBD(o_1,o_2). Generally, an MBD represents a set of subscriptions, denoted as MBD(o_1,...,o_i,...o_n). As a result, in a broker, the routing table of subscriptions can be organized as a set of MBDs. Fig. 4 shows the algorithm of inserting a subscription into a set of MBDs.

Using MBDs to represent subscriptions has several advantages. First, subscriptions are stored more efficiently by sharing the same predicates among each other. Second, the matching algorithm can be transformed into the evaluation of MBDs, which has a time complexity of $O(n)$, where n is the size of the MBD. Third, during the evaluation of an MBD, the evaluation result of an internal node, which is

Input: New subscription s, a set of MBDs Ω
Output: Ω with s inserted
1: Build a BDD for s; *insert* := false;
2: For each MBD $\omega \in \Omega$, which has n outputs
3:   If (*insert* == true) Break;
4:   If (*s*.output has the same child nodes as $\omega.o_i$)
5:   Insert s into ω as o_{n+1};
6:   *insert* := true ;
7:   If (*insert* == false) Add s in ω;
8: Return Ω;

Figure 4. MBD Insertion Algorithm

shared by several output nodes, can be reused. Since shared nodes are evaluated only once, redundant computation is eliminated. The matching algorithm can be improved significantly in this way. The same applies to the covering algorithm. Fourth, subscriptions organized in an MBD have more common predicates. Two output nodes in an MBD which have the same child nodes have a higher chance of being merged. In Fig. 5, for instance, two output nodes can be merged as *(price,≤,150)*. Since we chose candidates of merging among output nodes of an MBD, instead of the whole routing table, the merging process is simplified. We interpret a BDD as an MBD with only one output node in the later part of this paper.

3.2. Variable Ordering

It is well known that the variable ordering problem is NP-complete [2], and a reasonable ordering of the variables can avoid the exponential growth of the number of nodes in a BDD. We order Boolean variables in two ways. One is the order of attributes, the other is the order of variables corresponding to the same attribute. For attribute ordering, a proper order may lead to more subscriptions sharing a common predicate pattern among each other. We can design an order of attributes based on the knowledge of historical subscriptions. There are three main criterions that can be used for attribute ordering. First, the frequency of attributes based on the analysis of historic data. We can get the attribute set which includes all attributes in a particular application. The frequency of each attribute indicates the popularity of this attribute. We prefer to place attributes with
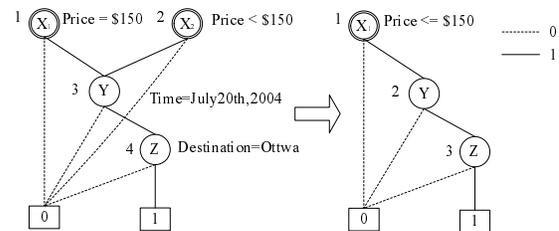

Figure 5. An MBD of Two Subscriptions

high frequency in the bottom of the MBDs in order to share it with other subscriptions. Second, distribution of an attribute. If predicates of an attribute are focused in a certain range, they are more likely to be shared. As a result, this attribute should be placed in a lower position as well. Third, application related semantics are also useful for variable ordering. For instance, a subscription about plane tickets always contains attributes *destination* and *price*. The two attributes can be placed close to each other in the order. The attribute ordering is key for sharing variables among single BDDs. For variables with the same attribute, however, the ordering among them is not so important to the MBD's size because each attribute appears once in a subscription. Our evaluation shows that proper attribute ordering can greatly reduce the number of MBDs. In other words, an MBD may have more output nodes representing more subscriptions, while the order of variables corresponding to the same attribute does not show significate changes over random ordering.

3.3. MBD-based Matching Algorithm

A publication matches a subscription if for each predicate in the subscription there is an attribute-value pair in the publication that satisfies the attribute constraint. The matching algorithm has two steps. First, for each attribute-value pair in the publication, the algorithm traverses the global predicate index and computes a truth assignment to all existing variables corresponding to the same attribute. Default value 0 is assigned to a variable whose attribute is not contained in the publication. Second, each MBD in the routing table is evaluated with respect to the truth assignment. The number of MBDs effects the matching time, instead of the subscription number. If an output node o_i in MBD(o_1,...,o_n) has value 1, it means the subscription corresponding to o_i is matched by the publication. Since BDD-based matching approach for p/s has been discussed in [5], we do not repeat the MBD computation algorithm in this paper, which is similar to the BDD evaluation method.

4. Covering-based Routing

4.1. Basic Idea of Covering

The goal of covering-based routing is to guarantee a compact routing table without information loss, so that the performance of a matching algorithm can be improved based on the concise routing table and no redundant information is forwarded into the network. When a broker receives a new subscription from a neighbor node, it will do the following steps to determine whether to forward it or not. First, it searches the routing table to determine if the subscription is covered by some existing subscription from the same neighbor. If it is, the new subscription can be safely removed without inserting it into the routing table

and, of course, without forwarding it further. If the new subscription is not covered by any existing subscriptions, the broker needs to check if it covers any existing subscriptions. If it does, those subscriptions which are covered should be removed. *ACK* messages are used to ensure that all the covered subscriptions are removed completely. Before removing them from the routing table, the broker informs its next-hop neighbors about the deletion by sending unsubscription messages to them, and waits for *ACK* messages back from these neighbors. If a neighbor broker finishes processing the unsubscription message, it will send back an *ACK* message. The unsubscription message is sent periodically with a timeout value of t_1 ms until its *ACK* is received. If the *ACK* does not arrive within an interval of time t_2 ms, we assume the neighbor broker has failed. A fault tolerant module is called to recover SRTs/PRTs. The details are beyond the scope of this paper. After the broker receives all the *ACK* messages, it can safely remove the subscriptions from its routing table[2]. Finally, the broker inserts the new subscription into its routing table and forwards it to its neighbors. These steps guarantee an atomic transaction based on the *ACK* messages for unsubscriptions. In this way, the routing tables remain consistent throughout the p/s system. If this protocol is not used unsubscriptions may lead to inconsistency in routing tables. To summarize, a subscription is not forwarded to the next-hop node in the network if it has been covered by an existing subscription.

Processing an unsubscription message is not trivial, since covering actions may have removed subscriptions from a broker. If a subscriber wants to unsubscribe a removed subscription, it may cause a "no subscription found" error. Therefore we cannot simply forward unsubscriptions into the broker network. In the context of covering, an unsubscription, $unsub(S)$, is processed as following: if S is covered by another subscription, the node simply sends an *ACK* back without forwarding it. If S is not covered, the node should forward $unsub(S)$ to its next-hop neighbors, and recover those subscriptions which previously were covered by S until the broker receives all *ACKs* from its neighbors. Last, S is removed from the routing table and an *ACK* is sent.

In advertisement-based p/s systems, since advertisements have the same structure as subscriptions, the advertisement covering can be solved in the same way.

4.2. MBD-based Covering

For subscription covering, we need to identify whether a subscription is covered by existing subscriptions, or what existing subscriptions it covers. There are four covering relations between two subscriptions S_1 and S_2: (0) denoting

---

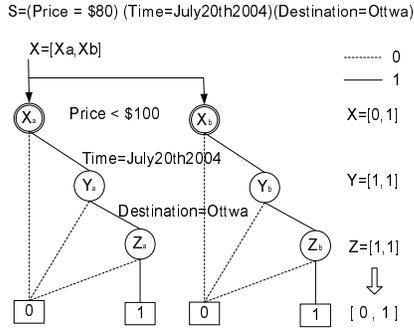2 *ACK* messages are parameterized by unsubscription message IDs to not confuse multiple concurrent messages

Figure 6. Covering Relation Detection

Input: A variable assignment Γ, A set of MBDs Ω
Output: cover, subscription set (R) covered by a new subscription
1: cover = false; RHS = true; R = Φ;
2: For each MBD $\omega \in \Omega$
3:    If (RHS)
4:        ComputeMBD($\omega_R, \Gamma(R)$);
5:        If ($\exists$ value(o_i) == 1)
              then cover = true; R = Φ; return;
6:        ComputeMBD($\omega_L, \Gamma(L)$);
7:        For each output node o_i;
8:            If value(o_i) == 1
                  then RHS = false; R = R $\cup o_i$
9: Return cover and R;

Figure 7. Covering Algorithm

no relationship between them; (1) denoting S_1 is covered by S_2; (2) denoting S_1 covers S_2; and (3) denoting the equality between S_1 and S_2.

Generally, an MBD can be used to solve N-value functions. Consider a finite set of elements V, where $|V| = N$. We can encode each element in V as a vector of n bits, where $n = \lceil log_2 N \rceil$. Suppose function $\Gamma : V \rightarrow \{0,1\}^n$ maps each element of V to a distinct n-bit vector. In our case, N=4. We define Γ as following: $\Gamma(0) = [0,0], \Gamma(1) = [0,1], \Gamma(2) = [1,0], \Gamma(3) = [1,1]$. Boolean operations $\vee^*$ and $\wedge^*$, are defined as:
$X \vee^* Y = [X_a, X_b] \vee^* [Y_a, Y_b] = [X_a \vee Y_a, X_b \vee Y_b]$
$X \wedge^* Y = [X_a, X_b] \wedge^* [Y_a, Y_b] = [X_a \wedge Y_a, X_b \wedge Y_b]$

A pair of MBDs can be used to identify the covering relations, as shown in Fig. 6. For example, with the new subscription *(price,=,80)$\wedge$(time,eq,'July 20,2004')$\wedge$(destination,eq,'Ottawa')*, the assignment of X is [0,1], Y is [1,1], and Z is [1,1]. After evaluation, the left-hand side MBD is 0 and the right-hand side MBD is 1. So the new subscription is covered by the existing subscription. An MBD has a left-hand side MBD and a right-hand side MBD. For an output node in an MBD, if its evaluation result in the LHS MBD is 1, it means a new subscription covers or equals the subscription corresponding to the output node; if the evaluation result in the RHS MBD is 1, it means a new subscription is covered by or equal to this subscription. The evaluation results of the output node in the two MBDs together determine the covering relationship.

4.3. Covering Algorithm

Existing approaches use two covering algorithms: one algorithm to decide whether a new subscription is covered by existing subscriptions, and another algorithm to determine a set of subscriptions that are covered by the new subscription. In the covering-based context, if a newly arriving subscription is covered by an existing subscription, the new subscription could not cover any subscriptions in this routing table. On the other hand, if a new subscription cov-

ers an existing subscription, it could not be covered by any subscriptions in the routing table. Based on the above two facts, we propose a completely novel algorithm which combines the functionality of these covering relation determination operations into one single algorithm. Our algorithm scans the routing table only once to determine the covering relations.

Our algorithm has three main steps. First, when a new subscription arrives, we go through the index checking if all its predicates have been assigned a variable. If there is a new predicate, we insert it into the index and assign a new variable to it. As a result, all the predicates have a variable. We need to identify the covering relations among predicates using values in {0,1,2,3}, which have the same meaning as subscription covering. Instead of assigning 0 or 1 as we do in the matching algorithm, we assign a value in format of $[v_l, v_r]$ to variables, where $v_l, v_r \in \{0, 1\}$. For example, a value [0, 1] indicates the new predicate is covered by existing predicates. The new subscription gives an assignment to all Boolean variables. Second, with the truth assignment, we can evaluate the pair of MBDs to determine the covering relation. If there is an output node in a RHS MBD that has value 1, then this means the new subscription is covered by the routing table, we do not need to evaluate the remaining MBDs, and the algorithm stops. Similarly, if there is an output node in a LHS MBD that has value 1, then this means the new subscription covers an existing subscription and, from this point forward, we do not need to evaluate the RHS MBDs of the remaining MBDs. In this way, we can speed up the covering algorithm. Fig. 7 shows the details of step 2. In the worst case, the algorithm has a time complexity of *O(mn)*, where *m* is the number of MBDs in the routing table and *n* is the average size of an MBD graph. Third, if the subscription is not covered by the routing table, we need to remove all the subscriptions it covers, which we can get from step 2, and insert it into the routing table. To insert the subscription, we build a BDD for the subscription based on its variables, and decide which MBD it should be inserted into. If in an MBD there exists an output node

which has the same child nodes as the new BDD, we insert the BDD into the MBD by adding an output node on the child nodes. We can benefit from the insertion in the merging algorithm, which will be discussed in Section 5.2.

4.4. Variable Dependency Optimization

In the covering algorithm, we need to check the covering of predicates repeatedly when a new subscription arrives. Those redundant checking can be eliminated by caching the variable dependencies. For example, if $x_1=(price,\geq,10)$ is true, then $x_2=(price,\geq,8)$ must be true, denoted as $x_1 \rightarrow x_2$. In the indexing data structure, the two predicates are represented by different variables, but there is a variable dependency between them. To make the cache more efficient, we maintain a concise variable dependency table, which is complete and minimal in size. For example, if there is another variable $x_3=(price,\geq,5)$, then we also have $x_2 \rightarrow x_3$ and $x_1 \rightarrow x_3$. However, $x_1 \rightarrow x_3$ is redundant since we can deduce it from the other two variable dependencies. As a result, the table with the first two dependencies is complete and minimal. Fig. 8 shows the algorithm for computing the minimal closure for variable dependencies. For the worst case, the time complexity of the algorithm is $O(n^2)$, where n is the number of variable dependencies. Caching these variable dependencies can improve the performance of the covering algorithm, especially when the index grows large.

5. Merging-based Routing

Subscriptions which are not in any covering relation can be merged into a new subscription thus creating a more concise routing table. The efficiency of the p/s system can be further improved by merging-based routing which takes advantage of this merged routing table. Advertisements can be merged in the same way. We prefer to do perfect merging for advertisements in order to not route subscriptions to wrong destinations, as would be the case of imperfect merging-based advertisement merging.

5.1. Basic Idea of Merging

The three core problems of merging in p/s systems are when to do the merging, what should be merged and how

---

Input: New variable v, A set of v.d.'s *VD*
Output: Minimal closures of variable dependencies
1: Add all the variable dependencies of $v \rightarrow u$ in VD, where u∈ existing variable set
2: Do until VD does not change
3:   For each v.d. $v_i \rightarrow u_i$ in VD do
4:   If VD - $\{v_i \rightarrow u_i\} \models v_i \rightarrow u_i$ then remove $v_i \rightarrow u_i$ from VD
5: Return VD

Figure 8. Minimal Variable Dependencies

---

to do the merging. It is not wise to try to merge every subscription when it is received, as it would cost too much time and system resources, thereby degrading the efficiency of the whole system. We need some merging policies to decide when to do the merging. For example, we can simply do merging periodically, or start the process when the routing table size has reached a certain threshold. The incoming rate of new subscriptions and the usage of network bandwidth can also be considered as merging policies. In our evaluation, we start merging when the average number of output nodes in an MBD reaches a given value.

5.1.1. Creation of a Merger Given a routing table, there are many ways to combine a set of subscriptions into merged ones. For example, three subscriptions could be merged pairwise, merged to one subscription, or not merged at all. As a result, we have to consider all the possible ways for subscription merging, which has been proven to be NP-complete. Merging rules are used for what and how to merge. Two subscriptions that come from the same broker can be merged, if they satisfy merging rules. There are two kinds of rules, perfect merging rules and imperfect merging rules. For example, it is a perfect merger if we merge two output nodes in Fig. 5 into *(price,≤,150)*.

Imperfect merging will introduce false positives, that is some publications that do not match any of the original subscriptions will match the imperfect merger. The false positives increase the network traffic overhead. Since we only merge subscriptions coming from the same broker, the false positives only occur in the broker network. Clients will not receive any unwanted publications. Although imperfect merger may cause some unnecessary publications to be forwarded into the broker network, thus increasing the network traffic overhead, an imperfect merger is useful in some situations in which a perfect merger is hard to compute. For instance, there are several subscriptions which have only one predicate different from each other:*(price,>,150), (price,=,145), (price,=,147) and (price,=,148)*. It is difficult to find a perfect merger for them. In this situation, an imperfect merger is a good compromise. We can merge the different predicates into *(price,≥,145)*. Although we introduce false positives, we reduce the routing table size. An imperfect merger is created by imperfect merging rules. Tab. 1 shows some rules which are suitable to subscriptions that have the same attribute set and have only one different predicate. More rules can be applied, for example, some

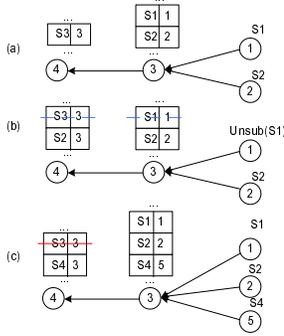| p_1	p_2	Conditions	Merger
(attr,<,v_1)	(attr,>,v_2)	$v_1 < v_2$	(attr,isPresent,0)
(attr,<,v_1)	(attr,=,v_2)	$v_1 < v_2$	(attr,≤,v_2)
(attr,>,v_1)	(attr,=,v_2)	$v_1 > v_2$	(attr,≥,v_2)

Table 1. Some Imperfect Merging Rules

Figure 9. Merge-based Routing

subscriptions which have one or two predicates are different can be merged together by simply dropping the different predicates.

5.1.2. Cancellation of a Merger A merger may be removed in the following situations: if an unsubscription for a part of the merger is received, or there is a new subscription which covers the new merger.

We explain this with the following example. Suppose subscription S_1 and S_2 have no covering relations, but they can be merged into a new merger S_3, as shown in Fig. 9 (a). The new merger will be cancelled if there is an unsubscription message of part of the merger that arrives from node 3, say unsub(S_1), then S_3 needs to be unmerged and decomposed into the original smaller subscriptions. The message unsub(S_3) is forwarded instead of unsub(S_1). Meanwhile the remainder of the merger (S_2) should be recovered, as shown in Fig. 9 (b). A merger may be removed because of a covering relation. For instance, in Fig. 9 (c), a new S_4 is received by node 4 which covers the merger S_3, and node 4 will remove S_3 from its routing table.

5.2. MBD-based Merging Algorithm

To merge subscriptions in a given routing table, we need to select proper candidates among subscriptions in the routing table. The candidates should have similar structures, for instance, their predicates are based on the same attribute set, and have only one or two different predicates. It is not trivial to select such candidates for merging, especially when the routing table is extremely large and subscriptions are not organized properly. In our routing table, subscriptions are grouped by MBDs. Only subscriptions that share most of their predicates can be built in the same MBD. This requirement is similar to the criteria of selecting candidates for merging. In other words, subscriptions built in the same MBD have a better chance to be merged. Two output nodes corresponding to the same attribute in an MBD have the same child nodes means that subscriptions represented by the two output nodes have only one predicate different from

each other. Therefore the two output nodes can be merged if some merging rule is satisfied. If two subscriptions are build in different MBDs, we do not merge them by assuming that they have less chance to match the merging rules. Our merging algorithm is described in Fig. 10. A new merger may cause new covering relations, before inserting the new merger into the routing table, we need to remove all subscriptions it covers.

5.3. Imperfect Merging

Imperfect merging is a way to further reduce the routing table size based on perfect merging. The goal of perfect merging is to reduce the routing table size by compacting highly overlapping subscriptions, and at the same time avoid introducing false positives of matched publications. It is an ideal way to do merging. If a system can allow a certain degree of false positives, we can apply imperfect merging rules to those subscriptions that can not be merged perfectly. The routing table size can be reduced significantly by imperfect merging. On the other hand, the network traffic would be increased because of the false positives introduced. To avoid sending false positives to clients, we merge subscriptions with the same last-hop broker. Only the network traffic overhead of the internal broker network would be increased. In Fig. 9 (a), for instance, merger S_3 is created on node 4. If S_3 is an imperfect merger, the false positives exist between node 4 and node 3. Node 3 would not forward these unwanted publications to its next-hop brokers. There is a tradeoff between routing table size and network traffic. We define the *imperfect degree*, which measures the imperfectness of an individual new merger, to balance this tradeoff. A formal definition is given below.

Definition 1 *Suppose F_M is a merger of F_1, F_2,...,F_n, the imperfect degree of F_M is :*

$$D_{imperfect} = \frac{|P(F_M) - \cup_{i=1}^n P(F_i)|}{|P(F_M)|}$$

Intuitively, this means that the imperfect degree of F_M is the ratio of the shaded area to the whole square in Fig. 11 (a). Whether or not two subscriptions can be merged depends on what imperfect degree the system allows. The bigger the imperfect degree allowed, the more subscriptions will be merged. The imperfect degree of a perfect merger is 0.

The imperfect degree itself is not enough to balance the tradeoff between routing table size and network traffic. For example, given two imperfect mergers M_1 and M_2, at M_1's imperfect degree be smaller than M_2's, then the number of publications matching M_1 may be larger than that matching M_2 considering the distribution of publications in the system. In other words, even though M_2 has a larger imperfect degree it may introduce fewer false positives than M_1. Thus, we prefer merging M_2 instead of M_1, although M_1's

Input: A set of MBDs Ω
Output: A set of merged MBDs Ω
1: For each MBD$(o_1,...,o_n) \in \Omega$
3:   If $(|n| > 1)$
4:   Divide output nodes into groups. Output nodes which have
            different attributes and child nodes are put into the same group;
5:   For each group
6:   check the perfect/imperfect merging rules;
7:   If there is a rule can be applied
8:   create a merger;
9:   remove subscriptions covered by the new merger;
10:   insert the merger into Ω;
11: Return Ω

Figure 10. Merging Algorithm

imperfect degree is smaller. We define a new parameter to quantify this effect, the *imperfect merging threshold (IMT)*. The IMT is derived from the imperfect degree by multiplying the probability of a merger and its imperfect degree. A merger is created if its IMT is smaller than a specified value. The IMT considers the distribution of the publications and the number of predicates of a subscription. It is a more accurate way to ensure that the number of unwanted publications introduced by imperfect merging is as little as possible.

5.4. Advertisement-based Optimization

Imperfect merging can be improved if we take advantage of knowledge of advertisements in the broker. For instance, in a broker there are three subscriptions for plane tickets with different predicates *(price,>,150), (price,=,130)*, and *(price,=,120)*, and an advertisement *(price,>,150)*. From this we know that all the publications coming to this broker have a price value larger than 150. So we generate an imperfect merger *(price,≥,120)* for the three subscriptions. Although the merger is an imperfect merger, it does not include any false positive because no publications in the range of $120 \le price \le 150$ will come to this broker. With the knowledge of advertisements, the imperfect degree of a merger can be modified as:

Definition 2 *Suppose F_M is a merger of F_1, F_2,...,F_n, and $A_1, A_2, ..., A_m$ are the advertisements available in the same broker, then*

$$D_{imperfect} = \frac{|(P(F_M) - \cup_{i=1}^n P(F_i)) \cap (\cup_{j=1}^m P(A_j))|}{|P(F_M) \cap (\cup_{j=1}^m P(A_j))|}$$

Intuitively, this means that the shaded area in Fig. 11 (b) represents the publications that would introduce false positives. The new imperfect degree is the ratio of the shaded area to the publications being advertised. Under the new definition, the imperfect degree of some imperfect merger maybe 0, since there is no advertisements for the publications that would cause the false positives.
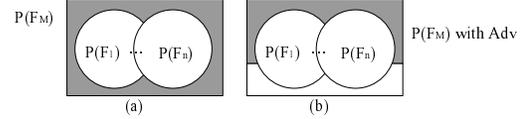

Figure 11. Imperfect Degree of a Merger

6. Evaluation

We implement the algorithms in Java with JDK1.4.2 using the JavaBDD package [1]. All our experiments are performed on a computer with an Intel Xeon 3GHz processor 2GB RAM, of which 1GB RAM is allocated to the JVM. Subscriptions and publications were generated by a workload generator which produces the data randomly by selecting attributes from a list of twenty pre-ordered attributes and selects values from given value ranges. We assume that the value of each attribute in a publication is uniformly selected from its value range. We vary the range of values, number of predicates and type of operators in a subscription to obtain data sets with different number of distinct predicates. These workloads represent different application scenarios. In order to simplify the calculation of IMT, we choose integer as the experimental data types. The performance metrics we take includes routing table size, publication routing time, and IMT in a single broker.

Routing Table Size (RTS): Our algorithms exploit the covering and merging relations among subscriptions. To verify this fact, we generate two data sets which have 200,000 subscriptions each. Set A is based on 2,000 distinct predicates, and Set B is based on 5,000 distinct predicates. The subscriptions in A have a higher degree of overlap. Intuitively, the algorithm should perform better on A, due to the higher degree of overlap. The routing table size is the number of subscriptions in the table. For each data set, we compare the results of covering, perfect merging and imperfect merging with the original system without these techniques. The routing table size of Set A is reduced dramatically by covering (see Fig. 12). About 75% subscriptions are covered by other subscriptions in Set A; while the covering rate of Set B is about 45%. Perfect merging can further reduce the routing table size, so does imperfect merging. The less the number of distinct predicates in the data set, the more subscriptions we can merge. With a compact routing table, we can improve the routing of publications and reduce the network traffic.

The attribute ordering effects the number of MBDs in the routing table. As the number of MBDs increases, the average number of output nodes in an MBD will decrease. As a result, less subscriptions can be merged. We use a random attribute order to do perfect merging, Fig. 12 shows that the routing table size of the random attribute order is higher by 6,076 subscriptions than the pre-defined order.
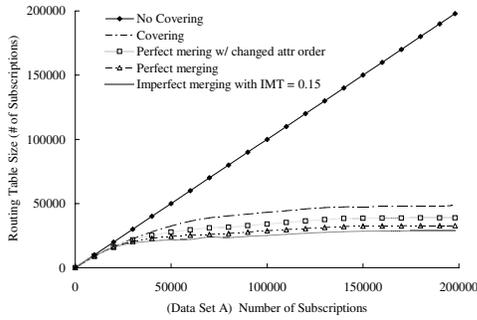
Figure 12. Routing Table Size

Figure 13. Subscription Insertion Time

Publication Routing Time: We compare MBD-based routing time with a naive approach, which linearly scans the routing table, using two data sets. Tab. 2 shows the routing time of publications against 200,000 subscriptions. The measurements were obtained by averaging the time taken to route 50,000 publications. Without covering, the MBD-based routing time of Set B is longer than Set A, because the number of MBDs in B is larger than A due to overlap, which effects the routing time. After applying covering and merging algorithms on subscriptions, the matching time of Set A is reduced by 85.7%. Obviously, the MBD-based routing is much faster than the naive approach, because distinct predicates are evaluated only once. Since the two data sets have different covering rates, routing time of the two data sets exhibit different benefits. The benefits are derived from subscription covering and merging.

Because subscriptions are used to build up the index and MBDs, adding a subscription is the most time-intensive operation. Each data point in Fig. 13 is the average time of processing 2,000 subscriptions. Processing a subscription in a covering-based context consists of two steps. For the first step, the algorithm checks the covering relations among predicates using the index. When the index grows to about 5,000 predicates, it takes 23.41ms to traverse the index, which can be improved by caching the predicate covering relations. However, if a predicate is new to the index, more time is spent to maintain the minimal cache table. It happens during the buildup of the index. When the number of subscriptions is less than 50,000, *MBD-CwC*(with cache) is higher than *MBD-C*(without cache). After the index is build up, that is no new predicate arrives later, the
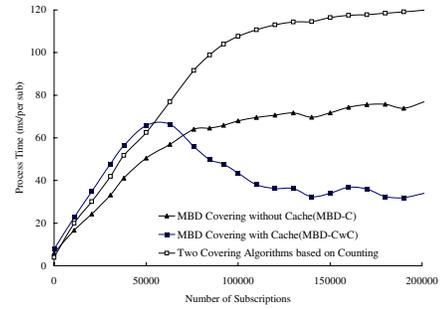
MBD-CwC is lower because the algorithm can get the covering results from the cache table, which speeds up the processing and uses 2.46 *ms*. For the second step, the algorithm evaluates the MBDs. At system start, the number of MBDs grows with the number of subscriptions, and the processing time increases accordingly. The number of MBDs increases much slower later on, since most of the subscriptions can share an MBD with already processed subscriptions or be covered. This results in a decrease of processing time. The results also clearly show that detecting covering with two separate covering algorithms is slower than our optimized covering algorithm.

Imperfect Merging Threshold (IMT): If the system allows a larger error tolerance, more subscriptions can be merged when doing imperfect merging. For a real-world data set, we could analyze its probability distribution and get a more accurate IMT for a particular merger. Unfortunately, no p/s workloads are available in the literature. In our experiments, we assume that all publications are uniformly distributed on each attribute so that the probability of a merger is easy to compute. As we show in Fig. 14, after covering and perfect merging, where IMT equals 0, all 100,000 subscriptions have been compacted to about 32,852. With increasing IMT, the results show that the routing table size is further reduced. For imperfect merging, the IMT varies theoretically from 0 to 1, while the reduction of the routing table size is not evenly distributed along this
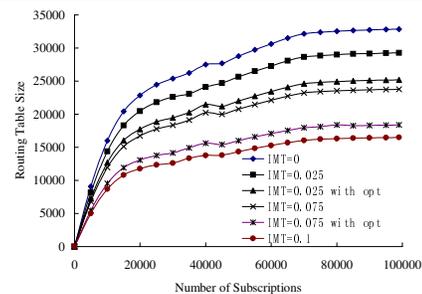
Method	Set A(ms)		Set B(ms)	
	MBD	Naive	MBD	Naive
No Covering	68.73	300.71	82.28	300.92
Covering	19.16	73.41	38.15	165.38
Perfect Merging	13.31	49.89	33.37	152.21
Imperfect Merging	9.96	43.83	29.19	141.06

Table 2. Publication Routing Time
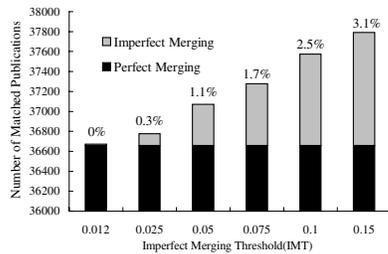

Figure 14. Imperfect Merging Threshold

Figure 15. False Positives

range. In our experiments, most mergers are generated with IMT ranging from 0 to 0.15. With IMT larger than 0.15, the routing table size reduces only slightly. Fig. 14 also shows the advertisement-based optimization for imperfect merging. The results show that more subscriptions can be merged without increasing the false positives. Some imperfect mergers, if based on the knowledge of advertisements, will not yield false positives, and can thus be considered as "perfect ". These mergers will then further reduce the routing table size.

A larger IMT means the system allows more false positives. We show the relation between IMT and false positives in Fig. 15. The black bar shows the number of publications that match the original subscriptions. In other terms, these publications are what subscribers want. The larger the IMT is, the greater the number of matched publications, among which some are false positives introduced by imperfect mergers. If the system tolerates up to 2.5% of false positives, an IMT with value less than 0.1 can satisfy the requirement.

7. Conclusion

In this paper we have proposed novel data structures and algorithms that unify publication routing, subscription covering and subscription merging. The algorithms are based on modified binary decision diagrams. We have shown that knowledge derived from statistics about the popularity of predicates in subscriptions and semantic as well as logic relationships among predicates can be of advantage. For example, publication routing with merging technique, which is based on past statistic information, improves the routing time by 85.5%. Unlike existing approaches, our covering algorithm computes all covering relations for a newly arriving subscription in one pass, leading to a speed up of 45.8%, as compared to alternate approaches. Since advertisements have the same semantics as subscriptions, covering and merging for advertisements can be solved in the same way, thus further emphasizing the unified character of our solution.

References

[1] JavaBDD package. *http://javabdd.sourceforge.net*.

[2] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[4] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, and M. Petrovic. Disconnected Operation in Publish/Subscribe Middleware. *2004 IEEE International Conference on Mobile Data Management*.

[5] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *ICSE*, pages 443–452. IEEE Computer Society, 2001.

[6] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.

[7] A. Crespo, O.Buyukkokten, and H. Garcia-Molina. Efficient query subscription processing in a multicast environment. *ICDE*, 2000.

[8] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructrue and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), 2001.

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

[10] H. Liu and H.-A. Jacobsen. Modeling uncertainties in Publish/Subscribe System. In *In Proceedings of ICDE*, 2004.

[11] G. Mühl. Large-scale content-based publish/subscribe systems. *Ph.D Dissertation*, University of Darmstadt, September 2002.

[12] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting IP multicast in content-based publish-subscribe systems. In *IFIP/ACM International Conference on Distributed systems platforms*, pages 185–207. Springer-Verlag New York, Inc., 2000.

[13] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS - a semantic publish/subscribe system. In *Very Large Databases (VLDB'03)*, Berlin, Germany, September 2003.

[14] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS - fast filtering of graph-based metadata. In *the 14th International World Wide Web Conference (WWW2005*, Chiba, Japan, May 2005.

[15] C. Trullemans and R. P. Jacobi. A study of the application of binary decision diagrams in multilevel logic synthesis. *Thesis*, September 1993.

[16] Z. Xu and H.-A. Jacobsen. Efficient constraint processing for location-aware computing. In *6th International Conference on Mobile Data Management (MDM'05), Ayia Napa, Cyprus*, 2005.