

Adaptive Content-based Routing based-on General Overlay in Publish/Subscribe Systems

Guoli Li and Hans-Arno Jacobsen

Middleware Systems Research Group, University of Toronto, Toronto, ON, Canada

Abstract.

1 Introduction

Publish/subscribe systems provide a simple and effective method for disseminating data while maintaining a clean decoupling of data sources and data sinks. This decoupling can enable the design of large, distributed, loosely coupled, and service oriented systems that interoperate through simple publish and subscribe-style operations. A wide variety of applications can use publish/subscribe as a messaging substrate, such as business process execution, business activity monitoring and workflow management. Over the past decade, most interesting issues have been explored and made significant progress, such as efficient matching algorithms [3] and content-based routing [1, 8, 6].

A limitation of existing publish/subscribe systems [9, 2, 10] is the acyclic broker network based on which the content-based message routing is performed. Acyclic means there is only one path between publisher and subscriber. The drawback of acyclic overlay network is that this topology is not flexible to route messages according to the dynamic network environment and it is not robust with respect to failures of brokers. For example, failure of an internal broker

may introduce network partitions which broken the routing pathes of related publishers and subscribers. Furthermore, solutions of failure recovery and load balancing in an acyclic broker overlay are complicated. For instance, the solutions for failure recovery must be loop free. Detecting cycles in a network is expensive. Removing the acyclic constraint of the broker network, the content-based routing protocol in publish/subscribe needs to be changed to avoid loops and select the best routing paths. The advantages of an arbitrary broker network are: first, it is more flexible to config the connections among brokers and easy to support dynamic changing network topology. Second, having more paths from a data source to a data sink, this overlay topology is more robust to broker failures. Third, message routing protocols are suitable for dynamic network environment. It is possible to chose different routing paths according to the QoS, e.g. delay or bandwidth, of the overlay network.

Publish/subscribe systems are nature to handle event correlations in a distributed manner [7]. With *composite subscriptions*, subscribers could correlate independent events from multiple publishers using logical or temporal operators. These independent events potentially occur at different locations and times. Routing composite subscriptions based on the topology of the broker network does not guarantee an efficient composite event detection solution. An efficient detection solution should report a composite event happening in a distributed network as fast as possible with minimized network traffic overhead. To satisfy the QoS constraints of applications, the expected routing solution of composite subscriptions should dynamic and adaptive to the QoS of broker network.

To address the above outlined problems, we propose a publish/subscribe protocol for an arbitrary broker network and an adaptive routing algorithm for

composite subscriptions in the PADRES project [5, 7]. The rest of this paper is organized as follows. Section 2 contains the related work and background. The extended publish/subscribe protocol in PADRES is described in Section 3. The protocol remains the original interface to publish/subscribe clients. It is easy to be extended in existing publish/subscribe systems without changing their matching algorithms. This solution takes the advantage of redundant routing paths so that publications could propagate to subscribers via a best path. Section 4 discusses the adaptive composite subscription routing algorithm. The novel routing algorithm propagates composite subscriptions according to the dynamic broker overlay network. The approach minimizes message delay and network traffic while detecting composite events in a distributed environment. We show our experiment results in Section 5. Conclusions are given in Section 6.

2 Background and Related Work

2.1 Content-based Publish/subscribe

A publish/subscribe system is comprised of information producers, information consumers and a broker overlay network. Information producers publish data, referred as publications, into the broker network; information consumers subscribe to the information by issuing subscriptions. Both of them are clients to the publish/subscribe system. The clients may join and leave the system at any time because of the decoupling feature. The broker overlay network performs message matching and routing based on the content of messages. It sends notification to proper subscribers if there is a match between publication and subscription messages.

2.2 The PADRES System

The PADRES project (Publish/subscribe Applied to Distributed Resource Scheduling) [5, 7] develops a distributed, content-based publish/subscribe middleware platform. The system consists of a set of brokers connected by a peer-to-peer overlay network. Clients connect to brokers using various binding interfaces such as Java Remote Method Invocation (RMI) and Java Messaging Service (JMS). Each PADRES broker has an intelligent rule-based engine to perform scalable message routing and matching. It supports an expressive subscription language which is powerful to describe message content and correlate publications in a distributed environment. For example, variables are supported in the subscription language so that we can extract values of attributes and correlate several publications by attribute values. Failure detection, recovery and dynamic load balancing are explored in PADRES as well.

The overlay network connecting the brokers is a set of connections that form the basis for message routing. The overlay routing data is stored in Overlay Routing Tables (ORT) at each broker. Specifically, each broker knows its neighbors from the ORT. Message routing in PADRES is based on the publication-subscription-advertisement model which is adopted from the SIENA project [1]. With ORT, advertisements are broadcasted throughout the network forming an advertisement tree that reaches every broker. The subscriptions are propagated in reverse along the advertisement trees, which we call Subscription Routing Table (SRT). The SRT is essentially a list of `[advertisement, last hop]` tuples. If a subscription overlaps an advertisement in the SRT, it will be forwarded to the last hop of that advertisement. Subscriptions are routed hop by hop to the advertiser, who may publish information of interest to the subscriber. Meanwhile,

the subscription will be used to construct the Publication Routing Table (PRT). Like the SRT, the PRT is logically a list of [subscription, last hop] tuples, which is used to route publications. If a publication matches a subscription in the PRT, it will be forwarded to the last hop broker of that subscription until it reaches the subscriber.

Topology-based CS Routing ...

2.3 Related Work

REBECA: (in Ph.D Thesis) based on acyclic broker overlay network;

JEDI: based on hierarchical overlay

SIENA:

3 Publish/subscribe On Arbitrary Broker Network

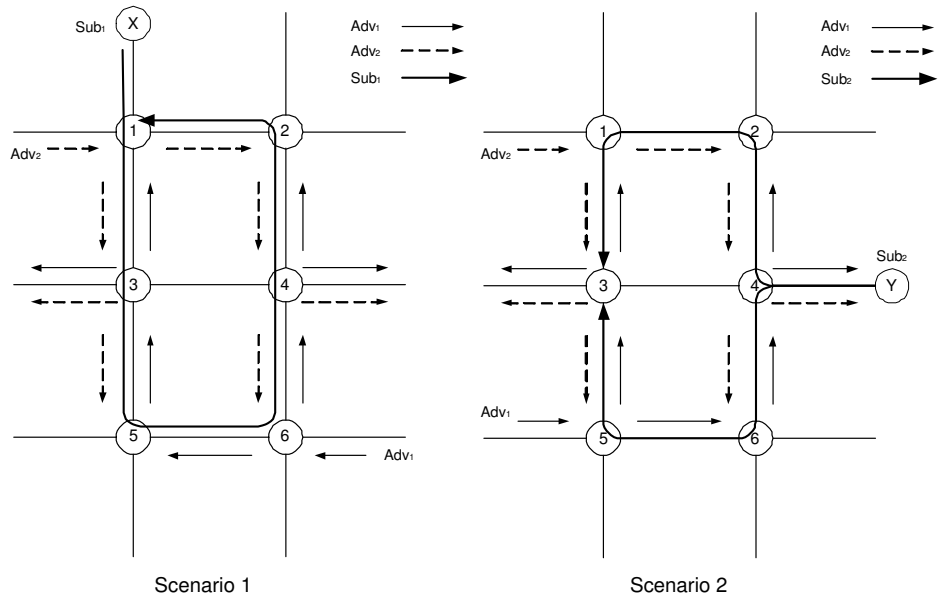


Fig. 1. Subscription routing on a broker network with cycles

Cycles in the broker network introduce many problems in terms of misleading message routing paths and redundant network traffic. In an arbitrary broker network, suppose there are two publishers issuing advertisement Adv_1 and Adv_2 separately. Advertisements are broadcasted in the network to form an advertisement tree for each publisher, as shown in *Scenario 1* of Fig 1. A subscription Sub_1 , which matches both Adv_1 and Adv_2 , is coming from Broker X. When it arrives at Broker 1, Sub_1 is forwarded to Broker 6 along the advertisement tree built up by Adv_1 . Unfortunately, Sub_1 does not stop at Broker 6 as we expected. It matches Adv_2 and continue to route back to Broker 1. Obviously, the subscription forms a loop.

The cycle path of subscription is misleading for later publication messages. Beside the publications delivered to the subscribers, extra copies of the publications are routing in the loop and being discarded at the end. Moreover, since subscriptions may overlap with each other, if a publication matching all of them may propagates in cycles as well. The duplicated messages may be advertisements, subscriptions and publications.¹ That means a large number of messages are discarded, which are redundant and waste the network bandwidth. Moreover, special algorithms and data structures are needed to handle the duplicated messages.

Since a subscription may have multiple matching publishers, one subscription is multicasted to the data sources. Each copy of the subscription has a chance to form a loop, which exacerbates the problems of cycles and redundant traffic, especially in a well-connected network. Take *Scenario 2* of Fig 1 as an example:

¹ A general assumption in publish/subscribe is that the number of advertisements is great less than the number of subscriptions, which is great less than the number of publications.

when Sub_2 arrives at Broker 4, it is forwarded to Broker 6 and Broker 2 following the paths built by Adv_1 and Adv_2 . However instead of stopping at Broker 5 and Broker 1, the two copies continue to route as loops. For instance, Sub_2 propagates along the advertisement tree of Adv_1 to Broker 5, at which it matches Adv_2 as well. As a result, Sub_2 will continue routing to Broker 3, which is not necessary. The duplicated subscriptions are detected until they both arrive to a broker, say Broker 3.

3.1 Publish/subscribe Routing Protocol For Arbitrary Broker Network

The reason of creating subscription and publication loops is that advertisement trees intersect with each other, as a result, matching subscriptions may route among the trees. That is, a subscription may start from a node of an advertisement tree and go back to the original node by branches of other advertisement trees. The key idea of our approach is distinguishing advertisement trees using unique identifiers so that subscriptions do not switch between trees while routing to their publishers so that subscriptions may avoid loops.

The new routing protocol remains the interface to clients unchanged. We explain the new strategy with *Scenario 1* in the previous example, as shown in Fig. 2.

Advertisement When a publisher advertises, each advertisement is assigned to a unique identifier, called *tree identifiers* (TID). The identifier is assigned within the system so that clients are not aware of TID. The easiest way is using the message identifier as TID, since each message in PADRES has a unique message identifier.

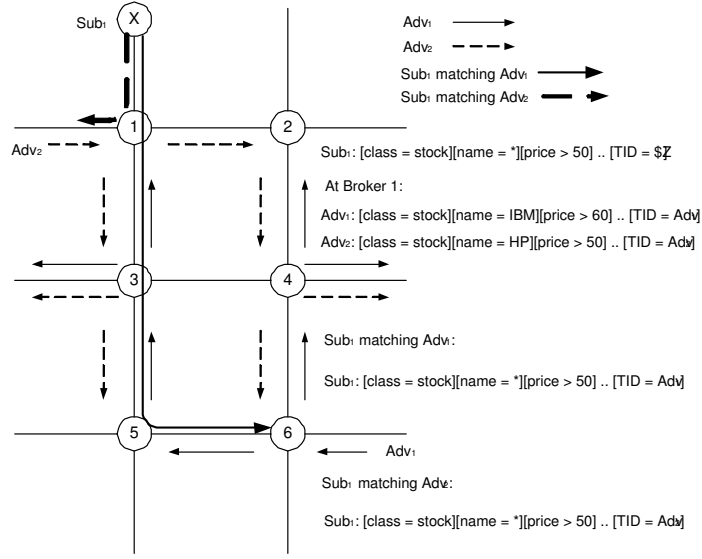


Fig. 2. Subscription routing with TIDs: Scenario 1

In details, when a broker receives an advertisement from a client, it will broadcast the advertisement to all its neighbors according to ORT and insert the advertisement to its SRT. When a broker receives an advertisement from another broker, it looks up its SRT. If it is not in the SRT, it forwards the advertisement to all its neighbors except the incoming broker and update the SRT. Otherwise, the advertisement will be dropped. Each advertisement forms an advertisement tree which reaches every broker in the overlay network. Advertisement trees are distinguish from each other by the TIDs. In our approach, duplicated messages received by a broker could only be advertisements.

Subscription When a broker receives a subscription from a subscriber, it adds one more predicate of TID to the subscription, e.g. $[TID = \$, \$Z]$, where $\$Z$ is a variable supported in PADRES which extracts the value of matching attribute. In other words, if the subscription matches an advertisement in the

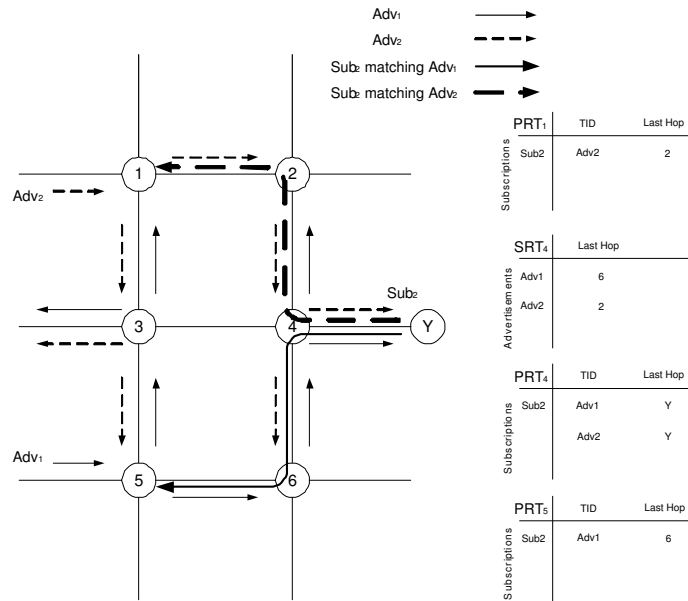


Fig. 3. Subscription routing with TIDs: Scenario 2

SRT, the advertisement's TID is assigned to variable \$Z. In Fig. 2, for example, *Sub*₁ is matched with both *Adv*₁ and *Adv*₂ at Broker 1. A copy of *Sub*₁ with TID equals to *Adv*₁ is forwarded to Broker 6 and another copy matching *Adv*₂ is forwarded to its publisher. Two routing paths replace the original cycle. In PADRES, attribute *TID* may have a set of values so that only one copy of subscription is send out if subscriptions with different TIDs have the same next hop destination. If a broker receives a subscription which already has a TID, it forwards the subscription to neighbor brokers according to SRT and TID, except the broker where the subscription comes from. The subscription only reverses the advertisement tree indicated by the TID.

Subscriptions set up paths for publications. The new PRT consists of a list of [subscription, set of {TID, last hop of subscription} pairs],

such as PRT_4 in Fig. 3. Since in an arbitrary broker overlay network, a subscription may arrive a broker by different paths labelled by TIDs, the PRT records TID and the last hop broker of the incoming subscription. If a new subscription is not in the PRT, it is inserted as a new record; if it is already in the PRT, update the existing record with the new `{TID, last hop of subscription}` pair. A subscription may have many TIDs because of multiple matching advertisements. Each copy of the subscription may take different paths from the subscriber to the current broker. Thus, a broker may receive many copies of the same subscription with different TIDs. These are not duplicated messages and they carry different path information, which is stored in the PRT as potential routing paths for publications from this broker to a subscriber.

Publication When a publisher sends out a publication, the publication is assigned to an identifier, which is the TID of its matching advertisement. The publication will be propagated along the paths set by matching subscriptions which have the same TID. By the TID, subscriptions form a directed, loop-free graph between a publisher and its subscribers so that publications would not be forwarded in loops. Brokers do not need to detect duplicated publications because no bandwidth is wasted for such redundant traffic. Lemma 1 guarantees the efficiency of our approach. We give a proof from publish/subscribe’s the point of view.

Lemma 1. *No duplicated publications are received by a broker or a subscriber.*

Proof. If a broker receives a publication p twice, it must receive them from two different neighbors or there must be another broker which receives p from different last hop brokers, since a publisher would not publish a publication with the same message ID twice.

Proof by contradiction: Assume that a publication p is received from a publisher at Broker X and there is a Broker Y receives p twice from its neighbor Broker B_1 and B_2 separately.

At Broker X, p is assigned to a TID of its advertisement and p is propagated to its subscribers along the paths set by its matching subscriptions which have the same TID as p . All the matching subscriptions are forwarded to the publisher, which is connected to Broker X, in reverse of the advertisement tree of TID. Broker X is the root of the advertisement tree. Broker Y is a node in the tree. The publication p arrives at Y from B_1 and B_2 , which means there are two different paths from root X to a node Y in the advertisement tree. There is a contradiction with the tree data structure, which says there only one path from the root to a node in a tree.

That is, no broker would receive duplicate publication messages. Furthermore, subscriber clients are connected to the publish/subscribe system through one broker. As a result, subscribers would not receive duplicated publications neither. \square

Unsubscription A subscriber could cancel its subscription by issuing an unsubscription. Similar to subscription forwarding, a broker looks up its SRT to decide the TIDs and destinations for the unsubscription message and removes corresponding subscription from the PRT.

3.2 Publication Routing Path Selection

Subscriptions are routed to its publishers along the advertisement trees. If the advertisement trees are intersect, at a jointed broker, there may be multiple paths to the subscriber. For example, in Fig. 4, *Sub* is forwarded to Broker 1

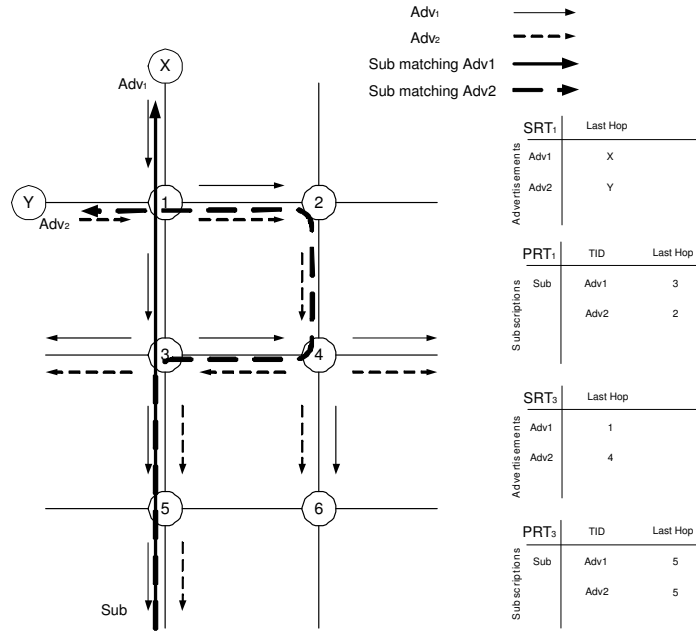


Fig. 4. Composite Subscription Matching

from two different paths. Potentially, publications matching *Sub* have multiple choices for the routing path. Publications of *Adv₁* will take the path through Broker 3; while publications of *Adv₂* will go through Broker 2. If the TID of a publication allows to change, a publication may chose the “best” routing path. In other words, if a publication of *Adv₂* arrives Broker 1, instead of route to Broker 2, it could route to Broker 3 by changing its TID to *Adv₁*.

A routing algorithm is needed to decide which path is the “best”. Typically, a “best” path is the one that has “least cost”, e.g. the number of hops or the delay from source to destination. This problem coincides with the well know routing problem in computer network. The solutions are classified according to whether they are global or decentralized. Global routing algorithms are based on complete, global knowledge of the broker network, which is not scalable with

large broker overlays. We prefer a decentralized solution which based on local connection and link status information. Algorithms that are widely used in practice, such as distance vector algorithm, hot potato forwarding and reverse path forwarding, could be used here to select a “shortest” path for publications. Different algorithms may have different performance, extra traffic of exchanging information between neighbors, and different implementation complexity.

We use idea similar to the simplest routing algorithm *hot potato routing*, which may not grantee the best solution but has been observed to work well in practice. The algorithm derives its name from its behavior: a broker tries to forward a message through the link with minimal delay time. Our algorithm is explained in Algorithm 1. To dynamically select paths as the network traffic loads or topology changes, we update the link latency whenever a broker receives a message from the link. Each broker maintains a *Link Delay Table* for all its outgoing links. When a broker receives a publication, it finds out all matching subscriptions. For each subscription with multiple paths, select a path with minimal link latency, assign the path’s TID to the publication, and put the copy of publication to result set. Only one publication is forwarded to each neighbor broker. That means the size of the result set is less than $D_{outgoing} - 1$, where $D_{outgoing}$ is the outgoing degree of the broker. Lemma 2 guarantees the correctness of Algorithm 1

Lemma 2. *Changing a publication’s TID while it is propagating will not change the set of subscribers.*

Proof. Say a publication p is changed to p' whose TID is different. $p.TID = Adv_1$ and $p'.TID = Adv_2$. This lemma indicate the following two cases:

case 1: No subscribers who does not subscribe to p but receives p' .

Suppose there is a subscriber S who does not subscribe to p but receives p' . S receives p' that means one of its subscription s_i matches Adv_2 , then we have $P(s_i) \cap P(Adv_2) \supseteq p'$, where $P(s_i)$ is the publication set of s_i . Since p and p' have the same publication content, we have $P(s_i) \cap P(Adv_1) \supseteq p$ as well. That is, subscriber S subscribes to p . There is a contradiction.

case 2: No subscribers who subscribes to p but does not receive p' .

Suppose there is a subscriber S who subscribes to p but does not receive p' . S does not receives p' indicating that $\bigcup P(s_i) \cap P(Adv_2) = \Phi$. Since S subscribes to p and p is changed to p' at Broker X, according to Algorithm 1, there must be a subscription of S , say s_p which matches with both Adv_1 and Adv_2 and forms two different paths, say $[Adv_1, lasthop_1]$ and $[Adv_2, lasthop_2]$. Then we have $s_p \cap P(Adv_2) \neq \Phi$, that is, $\bigcup P(s_i) \cap P(Adv_2) \neq \Phi$. There is a contradiction.

In both cases, we get contradictions. That means changing a publication's TID will not change the set of subscribers.

□

The advantages of our solution are: first, our approach makes little changes to the routing protocol. The interface to clients is remain the same. We only attach one predicate/attribute-value pair to each message and do not change the matching algorithm. In terms of matching, TID is the same as other attributes. In the rest of this paper, we do not distinguish TID from other attributes. Adding an attribute to solve cycles in the overlay is easy to implement in existing publish/subscribe systems, besides PADRES. Second, our approach generate duplicated messages only when broadcasting advertisements. Subscription and publication forwarding do not create redundant traffic. Third, subscriptions may set multiple paths for publications so that the system is more robust to broker fail-

Algorithm 1 Publication Routing Path Selection

Require: An incoming publication p with TID equals to TID_p **Ensure:** $routeMsg$: A set of publication messages with next hop destination and updated TID

```

1:  $routeMsg = \Phi$ 
2: Matching  $p$  against PRT and get  $S = \{s_i \mid p \text{ matches } s_i, i = 1 \dots n\}$ 
3: for Each  $s_i \in S$  looks up PRT do
4:    $paths =$  a set of  $[TID_j, LastHop_j]$ 
5:   Look up the Link Delay Table
6:    $nextHop = \min\{\text{Link delay of } LastHop_j\}$ , say  $j = m$ 
7:   if  $nextHop$  is not  $\in routeMsg$  then
8:      $pub = p$  with  $TID_m$ 
9:      $pub$ 's next hop destination =  $nextHop$ 
10:     $routeMsg = routeMsg.add(pub)$ 
11:   end if
12: end for
13: return  $routeMsg$ 

```

ure. Last, the routing algorithm is dynamic. A routing path changes according to the current network traffic. Our approach takes the QoS of the overlay into account, which is suitable for applications having QoS constraints.

4 Adaptive Composite Subscription Routing

As a composite subscription is propagated into the network, it travels as a unit until it reaches the *joint point* broker [4]. This broker is the first broker at which data sources are in different directions on the broker overlay network. This topology-based composite subscription routing does not necessary result

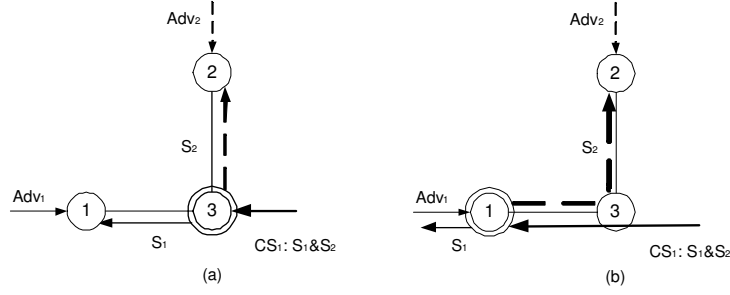


Fig. 5. Adaptive Composite Subscription Routing

in efficient use of network resources, especially when the amount of publication data from one data source is significantly greater than (or less than) the amount of data from other data sources. As a simple example shown in Fig. 5, suppose a composite subscription CS is coming to Broker 3. The data sources of publications can be determined from the advertisements. In this case, Broker 3 is the *joint point* broker of CS . At the *joint point* broker, the composite subscriptions is split into parts and each part is forwarded to the appropriate neighbors. In this way, all publications are collected at the *joint point* broker and composite event detection happens here as well, using the standard composite subscription mechanism in PADRES [7]. If the amount of publication data matching S_1 outweighs the amount of data matching S_2 , sending the data from Broker 1 to Broker 3 has a significant cost in network usage. It would be a better choice to move the composite event detection from Broker 3 to Broker 1. Performing the composite event detection at Broker 1 will increase the cost (e.g. the number of hops, bandwidth, delay etc.) of moving publications of S_2 from Broker 3 to Broker 1, but will decrease the cost of moving publications of S_1 . The savings in network traffic will be significant if the imbalance in detecting composite events at different locations is great.

To determine the most efficient location for composite event detection, it is necessary to estimate data set sizes of publishers at a particular broker and overheads of evaluation composite subscriptions at the broker and its neighbor brokers.

4.1 Estimating Data Set Sizes

Estimating the data set sizes of atomic subscriptions and composite subscriptions, we need to maintain statistical data associated with each publisher, for example, a publication rate and a distribution function of an advertisement. The overlap between a subscription and an matching advertisement indicates what publications can be forwarded to this broker. The ratio of actual coming publications to all potential publication of an advertisement is called *reduction factor* of the advertisement by the subscription. The data set size of an atomic subscription is :

Definition 1. *Data Set Size of Atomic Subscription*

Suppose an atomic subscription S matches a set of advertisement at a broker,

$$A = \{a_i \mid P(S) \cap P(a_i) \neq \Phi, i = 1..n\}$$

$$\text{Data Size of } (S) = \sum_{i=1}^n Pr_i * \frac{\int d_{a_i} f(S \cap a_i)}{\int d_{a_i} f(a_i)}$$

In Definition 1, Pr_i is the publication rate of a_i and d_{a_i} is the distribution function of advertisement a_i . Multiplying the publication rate by the reduction factor gives the expected publication rate for a_i . The publication set size of S is the sum of publications from its matching advertisements.

For composite subscriptions, the estimation of data set size depends on the logical operators in composite subscriptions.

Definition 2. *Data Set Size of Composite Subscription*

Suppose a composite subscription $CS = S_l \text{ op } S_r$, where S_l and S_r may be composite subscriptions as well.

$$\text{Data Size of } (CS) = \begin{cases} \text{Data Size of } (S_l) + \text{Data Size of } (S_r) & \text{if } op = ||; \\ \min(\text{Data Size of } (S_l), \text{Data Size of } (S_r)) & \text{if } op = \&. \end{cases}$$

Based on Definition 2, we can estimate the data set size of an arbitrary composite subscription using bottom-up algorithms.

4.2 Estimating Composite Event Detection Costs

The overhead costs is essentially the additional cost of moving the detection location from the *joint point* broker to its neighbors. We discuss the overhead costs in two different network models with the example shown in Fig. 5.

Traffic-based model In this model, we simplify the broker overlay network by assuming that each link has the same bandwidth capacity (e.g. bd bit/sec) and message delay (e.g. d ms). The parameter effecting the detection location is the publication traffic for a given composite subscription. The traffic overhead cost of evaluating composite subscription at a broker where one of the child subscriptions comes from, say Broker 1 in Fig. 5, is:

$$\text{Traffic Cost}(S_1) = \text{DS Size of } (S_2) - \text{DS Size of } (S_1) + \text{DS Size of } (CS)$$

The traffic costs includes the cost of sending publications of S_2 to Broker 1 and the cost of sending CS 's notification set to Broker 3, but we save in sending publications of S_1 to Broker 3. Similarly, we can estimate the cost of evaluating CS at Broker 2. The efficient location is the broker who has the minimal costs comparing to the *joint point* broker. Since all link have the same bandwidth and delay, the delay cost is:

$$\text{Delay Cost}(S_1) = \frac{\text{Traffic Cost}(S_1)}{bd} + d$$

QoS-based model In this model, we estimate the overhead cost based on the publication traffic and the QoS of the overlay network. If network links have different bandwidth and delay properties, the delay cost function is changed to:

$$Delay\ Cost(S_1) = \frac{DSSizeof(S_2) - DSSizeof(S_1) + DSSizeof(CS)}{bd_{13}} + d_{13}$$

$$Delay\ Cost(S_2) = \frac{DSSizeof(S_1) - DSSizeof(S_2) + DSSizeof(CS)}{bd_{23}} + d_{23}$$

The link properties are updated dynamically according to the network traffic.

4.3 Adaptive Composite Subscription Routing Algorithm

Based on the dynamic overhead estimation, we propose an adaptive composite subscription routing algorithm, as shown in Algorithm 2. This is a recursive algorithm, which computes the destination tree for a composite subscription.

Each composite subscription is represented in a tree data structure. The root node represents the composite subscription. Each internal node is an operator which has two child nodes and represents part of the subscription. Leaf nodes are atomic subscriptions. The algorithm traverses the tree of composite subscription as following: if the root of the tree is leaf node, that is, an atomic subscription, the atomic subscription's next hop destination in SRT is assigned to the leaf node. Otherwise, compute the left and right child nodes' destination trees separately. If the two child nodes have the same destinations, the root node has the same destination as its children. In other words, the composite subscription is routed to the next hop as a whole unit. If the child nodes have different destinations and the operator connecting the two children is OR, the root's destination is current broker, which means the composite subscription is split at the current broker. If the child nodes have different destinations and the operator connecting the two children is AND, the algorithm estimates the data set sizes of the child nodes, estimates the overhead cost at children's destinations, and selects a destination

which has minimal overhead costs. This destination is assigned to the root node. The recursive algorithm computes the destination tree from bottom to top. A broker splits and routes the composite subscription based on the destination tree. If a node has a destination which is not the current broker, forward the node and its sub-tree as a whole to its destination. If the destination is the current broker, split the composite subscription represented by this node into two parts. As a result, the composite subscription is split into parts according to its destination tree. Each part is forwarded to their own destination. The TID-based approach could be applied when an atomic subscription is split from a composite subscription.

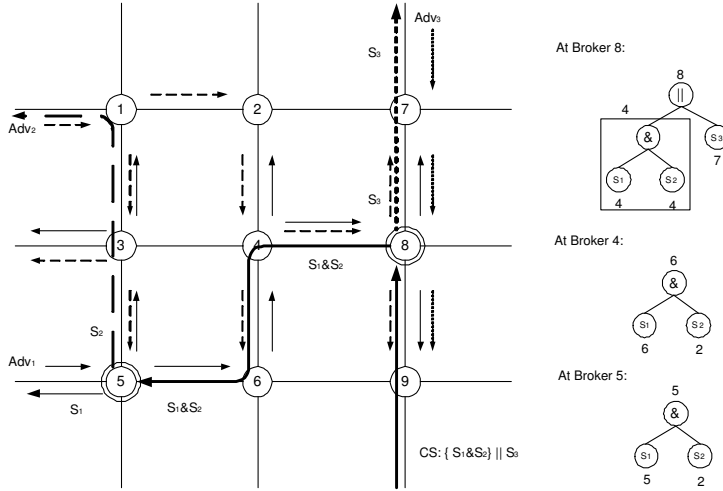


Fig. 6. Adaptive Composite Subscription Routing

This is a dynamic algorithm which route a composite subscription based on the potential publication traffic and the status of current overlay network. Fig. 6 gives a possible routing solution for composite subscription $CS = \{S_1 \& S_2\} \parallel S_3$. At Broker 9, CS is routed to Broker 8 as a whole. At Broker 8, both S_1 and S_2 's destination is Broker 4, as a result the destination of their parent node

is Broker 4 as well. S_3 should be forwarded to Broker 7. Since the operator of root node is OR, CS is split at Broker 8. When $S_1 \& S_2$ arrives Broker 4, according to SRT, Broker 4 is the *joint point* broker of $S_1 \& S_2$ in topology based routing algorithm. If the data set size of S_1 is significantly larger than S_2 , it may be more efficient to evaluate $S_1 \& S_2$ at Broker 6 than Broker 4. Similarly, the dynamic evaluation happens at each broker until a broker decides to split the composite subscription, say at Broker 5. According to the adaptive routing algorithm, Broker 8 and Broker 5 are the *joint point* brokers for CS , instead of Broker 8 and Broker 4.

The advantage of performing an adaptive composite subscriptions routing algorithm is that it minimizes the cost of evaluating composite subscriptions in terms of network traffic and message delay. Moreover, together with the TID-based approach, the detection of composite event in a distributed environment is faster and more efficient.

5 Evaluation

6 Conclusions

References

1. 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.
2. G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), 2001.

3. 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 Rec.*, 30(2):115–126, 2001.
4. E. Fidler. PADRES: a distributed content-based publish/subscribe system. Master’s thesis, Department of Electrical & Computer Engineering, University of Toronto, 2006.
5. E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. *International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI’05)*, Leisester, UK, 2005.
6. G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. *International Conference on Distributed Computing Systems (ICDCS’05)*, Columbus, Ohio, USA, 2005.
7. G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *ACM/IFIP/USENIX 6th International Middleware Conference, Grenoble, France*, 2005.
8. G. Mühl. Generic constraints for content-based publish/subscribe systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS ’01)*, volume 2172 of *LNCS*, pages 211–225, Trento, Italy, 2001. Springer-Verlag.
9. G. Mühl. *Large-scale content-based publish/subscribe systems*. PhD thesis, Department of Computer Science, Darmstadt University of Technology, 2002.
10. 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.

Algorithm 2 Adaptive Composite Subscription Routing Algorithm

Require: An incoming composite subscription cs

Ensure: Destination tree of a composite subscription

```

1: desTree = Represent  $cs$  in a composite subscription tree
2: if  $cs.root$  is a leaf node then
3:   Look up the SRT and find the destination set for  $cs.root$ 
4:    $desTree.root.destination = cs.root.destination$ 
5: else
6:    $desTree.left = adaptive-cs-routing(cs.left)$ 
7:    $desTree.right = adaptive-cs-routing(cs.right)$ 
8:   if  $desTree.left.destination == desTree.right.destination$  then
9:      $desTree.root.destination = desTree.left.destination$ 
10:  else
11:    if  $desTree.root.op == OR (||)$  then
12:       $desTree.root.destination = current\ broker$ 
13:    end if
14:    if  $desTree.root.op == AND (\&)$  then
15:      Estimate data set size of  $desTree.left$  and  $desTree.right$ 
16:      Estimate overhead cost of composite event detection at current broker,  $desTree.left.destination$  and  $desTree.right.destination$ 
17:       $desTree.root.destination = the\ destination\ with\ minimal\ overhead\ cost$ 
18:    end if
19:  end if
20: end if
21: return  $desTree$ 

```
