Faster Event Forwarding in a Content-Based Publish-Subscribe System through Lookup Reuse

Zhenhui Shen Srikanta Tirthapura
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50010
Email {zhshen,snt}@iastate.edu

Abstract

Event forwarding in a content-based publish-subscribe system is an expensive task due to the need to match an event's content against registered subscriptions at every router. We introduce lookup reuse, a novel approach to improve the efficiency of event forwarding. Lookup reuse enables faster event forwarding through reusing matching results computed by upstream routers in making forwarding decisions at downstream routers. In many cases, this lets downstream routers replace an expensive content-match with a much cheaper hash-table lookup. We investigate the integration of lookup reuse into existing content-based event forwarding algorithms. Our simulations show that lookup reuse reduces the event processing overhead on average by 40 to 55 percent, when used with existing content-based event forwarding algorithms.

1. Introduction

Content-based publish-subscribe (pub-sub) is an idiom for building distributed event based systems, and has been useful in the design of numerous distributed applications ranging from electronic stock markets to network diagnosis. In these systems, *events* from senders (publishers) are routed to receivers (subscribers) based on their content, rather than on a fixed destination address. Receivers in turn express their interests in receiving a certain class of events by submitting *subscriptions* to the system, which are predicates on the event content. For example, the event $[\text{stock} = \text{IBM}, \text{current} = 88]$ must be delivered to a client who has issued the subscription $[\text{stock} = \text{IBM}, \text{current} < 95]$.

In contrast with a content-based system, a *subject-based* pub-sub system routes events to subscribers based on the "subject" that an event belongs to. For example, an event $[\text{subject} = \text{weather}, \text{city} = \text{Ames}, \text{temperature} = 0^\circ]$ will be delivered to all clients who have subscribed to the subject "weather". Clearly, when compared with a subject-based system, a content-based system enables a client to express her interest in a much more precise manner. This flexibility simplifies the design of distributed applications. For instance, IBM has developed a real time scoreboard, powered by a content-based pub-sub middleware called Gryphon [12], for the Australian Open.

However, such a flexibility currently comes at a significant cost. Event forwarding is an expensive task, since it is now based on the content of an event, rather than on a fixed destination address (as in IP routing), or a fixed subject name, as in subject-based pub-sub. When an event arrives at a router, its content must be matched against the set of all registered subscriptions to find interested subscriptions, an operation that we call a *content-match*. The set of subscriptions maybe large (tens of thousands), and there is no simple way known to index these subscriptions to guarantee fast matching. A large body of research (see Section 1.2) has been devoted to indexing subscriptions to perform content-match efficiently. However, it can be shown [19] that high-dimensional range-searching (for which no worst-case computationally efficient solution exists) is a special case of the content-match problem, implying that content-match also does not have a worst-case efficient solution. Thus, it is accepted that content-match is an expensive task, best done as few times as possible.

In contrast, matching events to subscriptions in a subject-based system is much cheaper. When an event arrives at a router, the router only has to identify subscriptions whose subject equals the event's subject. This can be achieved through a hash-table lookup, which can be completed in constant expected time, irrespective of the number of subscriptions registered at the router.

Many current publish-subscribe systems perform content-based routing in a distributed manner, for example, Siena [6], Gryphon [12], Rebeca [21], Hermes [16], Jedi [9] and XNet [17]. The typical setup in these systems is as follows. The system consists of a distributed network of content-based routers. Each router manages a set of local

clients and is connected to a set of peer routers. A client expresses an interest in receiving events by registering its subscriptions at a nearby router. Every router forwards its registered subscriptions to its neighboring routers. These subscriptions are further forwarded in the network to create a reverse path for matching events to flow back to the subscriber. Whenever an event is published at a router, the router performs a content-match and delivers it to all local clients who have issued matching subscriptions, and also forwards it to neighboring routers from whom matching subscriptions were received. This process is repeated at each subsequent downstream router until the event reaches all interested subscribers.

1.1. Lookup Reuse

In current pub-sub systems, event forwarding is performed independently from router to router. When an event is forwarded from one router to the next hop after a content-match, the next hop router starts from scratch and performs another content-match on the event using its own routing table. Our faster event forwarding strategy is based on the following observation. In many cases, the efforts spent by the routers in performing a content-match are repetitive. Information needed by downstream routers to forward an event has been partially or wholly computed by the upstream routers during their content-match procedure.

The above redundancy results from the way in which subscriptions are propagated in the system. A subscription that originates at a router will be forwarded to neighboring routers so that matching events can flow back. Therefore, routing tables at neighboring routers share many common subscriptions, and an event-subscription match which occurs in one router will also occur in a neighboring router with high likelihood.

We propose a novel technique called *lookup reuse* to reduce this redundancy and improve forwarding efficiency. The general approach is as follows. Once a router performs a content-match and forwards an event to a next hop router, it passes along some metadata about the content-match. Informally, this metadata tells the reason why the event is being forwarded in this direction. This metadata can be used effectively by the next hop router to speed up its own event forwarding procedure.

A simple way to implement lookup reuse, called *Always-Reuse*, is as follows. The metadata is a list of all the subscriptions that match the event at the upstream router. The upstream router passes on this list while forwarding an event. Instead of sending the actual subscriptions themselves, which can be quite bulky, the upstream router can hash each subscription into a unique key and then send the key instead. At the downstream router, a content-match can be replaced by (many) inexpensive hash-table lookups on the set of keys received from the upstream router. Since a content-match is much more expensive than a hash-table lookup, this may speed up event processing at the downstream router.

Table 1. The average time taken for a hash-table lookup, and for a content-match using two different matching algorithms. All times are in microseconds. The data is for 10,000 subscriptions. For inputs 1,2 and 3 the average fraction of subscriptions matching an event was 5%, 10% and 20% respectively.

Input	Lookup	k-d tree	Counting algorithm
1	0.85	81.29	489.42
2	0.82	74.59	385.93
3	0.73	90.84	252.69

To get an idea of the potential gain from replacing a content-match with a hash-table lookup, we conducted experiments to compare the cost of the following operations: (1)*key lookup*: each subscription is assigned a unique key and the set of all keys K is indexed using a hash table. Given a query key k, does $k \in K$? With a good hash table implementation, this cost is independent of the table size. (2)*content-match*: given an event e and a set of subscriptions S, does there exist a subscription $s \in S$ such that e matches s? This cost depends on the size of S, and also on the content-match algorithm being used. We study the cost of two popular content-match algorithms: the k-d tree and the counting algorithm (these algorithms are described in our technical report [18]). Table 1 shows the experimental results. Our experiments show that the cost of a content-match is about 2-3 orders of magnitude greater than the cost of a hash-table lookup for subscription sets of the same size.

A simple approach like Always-Reuse may not work well in many circumstances. For example, consider a case when a given event matches all subscriptions at a router. Finding all matching subscriptions and forwarding their keys along with the event may not be very efficient for the following reasons. First, finding *all* matching subscriptions imposes a heavy burden on a single router. It is also an "overkill", since the presence of a single matching subscription is enough to forward an event along. Second, the number of keys forwarded is huge. This may induce a significant message overhead and slow down the packet transmission over the network link. Third, the overhead of looking up all these keys at the downstream router could nullify any advantage gained by the elimination of a content-match. A better strategy would be a hybrid approach that switches dynamically between reusing lookups and per-

forming a content-match, depending on the matching results passed by preceding routers. The hybrid approach is better able to balance the overhead of lookup reuse with the cost of content-match. The hybrid strategy we propose in this work is called *Partial-Reuse*.

We conducted extensive simulations to evaluate the performance of various lookup reuse strategies. Our experiments show that the *Partial-Reuse* strategy performs uniformly well in all cases, and is the preferred strategy for lookup reuse.

In addition, subscription covering can be used to reduce the amount of forwarded subscriptions in a pub-sub system. Always-Reuse can not work if covering relation is exploited. But Partial-Reuse is not affected. We will discuss this further in Section 4.

In summary, the contributions of this paper are as follows:
(1) We introduce *lookup reuse* as a lightweight optimization to the most critical task of event forwarding in distributed content-based pub-sub systems. It replaces a large fraction of potentially expensive content-match operations with much cheaper hash-table lookups, by strengthening collaboration among pub-sub routers. Lookup reuse is simple, and can be profitably used in conjunction with *any* content-matching scheme, as long as the cost of a content-match is significantly greater than a hash-table lookup.
(2) We explore the design space of various lookup reuse strategies. The simple Always-Reuse just relies on key lookups (after an initial content-match), while a hybrid Partial-Reuse dynamically switches between lookup reuse and content-match. Our simulations show that Partial-Reuse performs best in most cases.
(3) We evaluated the integration of lookup reuse with two different content-match algorithms, the counting algorithm and the k-d tree. For each content-match algorithm, we measured the event processing overhead with and without lookup reuse. Our simulations showed that lookup reuse reduces the total event processing overhead by an average of 40% for the k-d tree, and by an average of 55% for the counting algorithm.

1.2. Related Work

Content-match lies at the heart of event forwarding in a distributed pub-sub system. Recently Kale *et al.* [13] formally proved the hardness of content-match by showing its equivalence to the notoriously difficult Partial Match problem. For the problem of content-match, various forms of decision trees and subscription indexes are proposed. These efforts can be classified into two broad categories: counting-based algorithms [7, 10, 15] and tree-based algorithms [13, 3, 1, 19, 8].

Kulik [14] proposed a fast event forwarding algorithm, called match-structure event forwarding. In his approach, every event or subscription is prefixed with a match-structure header, which helps to speed up event-subscription matching. In comparison with lookup reuse, Kulik's scheme differs in the way how the header is generated and interpreted, and is somewhat similar to our *Always-Reuse* strategy. However we demonstrate that Partial-Reuse performs better than Always-Reuse in most cases.

The efficiency of event forwarding also depends on the subscription management. Many systems [19, 20, 11, 22] exploit "covering" relationships among subscriptions to reduce the number of forwarded subscriptions and keep the size of routing tables small. The optimization of subscription covering influences the implementation of lookup reuse, and this aspect is discussed further in Section 4.

It has been observed that partitions can be used to confine the information propagation to smaller scopes [25, 4]. There are two dual approaches to achieve this: event partitioning and subscription partitioning. Partitions enhance the relevance of subscriptions to events inside the same subnet. As a result, events traverse fewer hops and the size of routing tables is also smaller, since each router only needs to maintain a subset of subscriptions, pertaining to events that may be routed on the networks in which it participates.

The relevance of subscriptions can also be enhanced by placing routers that manage subscribers with similar interests [2, 24] close to each other. This requires the ability to dynamically change a network topology. Note that currently many pub-sub systems only adopt a static overlay network topology. Another novel approach [5] is to compute (online) the best multicast tree for each event. This approach may eliminate most of the pure event forwarders.

2. Lookup Reuse Strategies

In this section, we describe various event forwarding strategies , starting from current forwarding algorithms, which do not reuse any lookups, and moving on to the Always-Reuse strategy, and then the Partial-Reuse strategy. Due to space constraints, the pseudocode of these algorithms are listed in our technical report [18].

To simplify our algorithm description, we assume a simple network topology, where all pub-sub routers are organized into a single spanning tree. Each router is connected to adjacent routers on the tree through interfaces. A router usually indexes all subscriptions received from an interface to improve the speed of a content-match. Note that lookup reuse does not need any assumption on the underlying topology. Section 4 shows how to adapt lookup reuse to an arbitrary topology. Furthermore we are not concerned with the exact form of a subscription index. Lookup reuse should generate a profit as long as a content-match is significantly more expensive than a hash-table lookup.

We first review the current event forwarding algorithm. When an event arrives at a router from an interface, the router checks on every other incident interface to see if there is a subscription received on that interface which matches the event. If a matching subscription exists, then the event is forwarded, otherwise it is not forwarded. The same process is repeated at all routers on the path(s) from the event source to the recipients of the event.

Since the current approach does not involve any lookup reuse, we call it the *Never-Reuse* strategy. Note that "event source" is the router at which the event is first published. The routing step is very similar at any other router. The only difference is that a downstream router should exclude the interface from which the event arrived when matching the event. This is necessary to break the loops in event delivery.

2.1. Always-Reuse

To implement lookup reuse, for each subscription entering the network, a hash function is applied on it to obtain an integer value. This integer is called a subscription's *key*. If two subscribers submit the same subscription, we can break the symmetry by prefixing each subscription with the subscriber's unique id before hashing it. We assume that the hash function will generate different keys for different subscriptions, and this can be easily achieved (with overwhelming probability) using any standard hash function, such as the simple modulo function, as long as the key space is large enough (32 bits will do). Thus, we assume that there is a one-one mapping between subscriptions and keys.

We use a structure called the *KeyTable* at every router. KeyTable is a hash table mapping a subscription's key to the interface from which that subscription arrived. Given the key of a subscription, KeyTable returns the associated interface by using a single hash table lookup.

In Always-Reuse, the event source is responsible for finding *a*ll the matching subscriptions for an event. The strategy eliminates content-match at any downstream outer. Alternatively event forwarding is achieved by performing hash table lookups on the set of matching subscription keys that are piggybacked over the event.

Always-Reuse can be useful when an event matches only a few subscriptions. Under this circumstance, the load of finding all matching subscriptions at the event source is moderate. Furthermore the event forwarding inside the network is realized using pure key lookups. The benefit is more pronounced if an event traverses many pub-sub routers. However it performs poorly under circumstances described in Section 1.1.

2.2. Partial-Reuse

In comparison with Always-Reuse, Partial-Reuse makes a few changes at both the event source and at the other routers.

At the event source, it is no longer necessary to find *all* matching subscriptions before forwarding an event. Alternatively we apply a threshold of n, which specifies an upper bound on the number of matching subscriptions to search. Suppose an event matches m subscriptions on interface I. If $m \leq n$, the content-match returns m keys. If $m > n$, the content-match just returns n keys. Therefore the number of keys is bounded by n under Partial-Reuse.

We say the key list is *complete* if it contains the keys of all matching subscriptions. At a downstream router, if the received key list is complete, Partial-Reuse forwards an event only based on key lookups. Otherwise it first uses key lookups to eliminate content-match on as many interfaces as possible. Content-match is then performed only on interfaces which are not covered by any key.

Partial-Reuse achieves good load balance, as we no longer rely on the event source to find all the matching subscriptions. By imposing a threshold on the number of matching subscriptions to search, it is able to control the overhead of content-match as well as key lookups at every router.

3. Simulation Results

We simulated the following forwarding strategies: *Never-Reuse*, *Always-Reuse* and *Partial-Reuse* using the `OMNeT++`[23] discrete event simulator. The *Partial-Reuse* strategy was simulated using different thresholds: $40, 60, 80, 120, 160$ and 240. The strategies were compared under different input conditions, which were characterized by the *matching density*. Informally, the matching density is the average fraction of the subscriptions that match an incoming event.

3.1. Experimental Setup

We considered a system with 100 routers arranged in a balanced binary tree topology, whose depth was $\lfloor \log 100 \rfloor = 6$, and diameter was 12. Numeric subscriptions and events were considered. For instance, a subscription to textbooks might be $S = (\text{year} \in [1999, 2004])$ and an event advertising a book might be $E = (\text{year} = 2002)$. In the context of numeric data, an event can be thought of as a point in high-dimensional space and a subscription is a hyper rectangle in the same space. We generate each event by choosing values for each attribute randomly from the interval $[0, 1]$. Each subscription is generated by defining

predicates to be random sub-intervals of $[0, 1]$. By controlling the length of each subinterval, we are able to control the probability that a subscription matches an event. In the experiments described here, each event has 5 attributes, and a subscription defines predicates on these attributes. Note that lookup reuse is not restricted to numeric data, and can be used with any form of events and subscriptions.

At the start of the simulation every router issues some local subscriptions. These subscriptions are propagated through the system, and routing tables are populated. After all subscriptions have been registered and indexed at the routers, routers start publishing events, which are delivered to interested subscribers by using the selected event forwarding strategy. Each experiment uses a total of $10,000$ subscriptions and $1,000$ events. We evaluated the performance of forwarding strategies under different inputs, which differed in their matching densities. We consider inputs with the following matching densities: $0.1\%, 0.2\%, 0.5\%, 1\%, 5\%, 10\%, 15\%$ and 20%. For example, with a matching density of 0.5%, each event will match approximately $10,000 \times 0.5/100 = 50$ subscriptions.

Performance Metric: We measured the *event processing overhead*, which is the total time spent by all routers in processing events. Note that this does not include the time for setting up the system, or for propagating subscriptions. An event forwarding strategy is better if the event processing overhead is smaller.

3.2. Results

We conducted experiments employing lookup reuse in conjunction with two separate content-match algorithms, the k-d tree and the counting algorithm. At a high level we make the following conclusions:

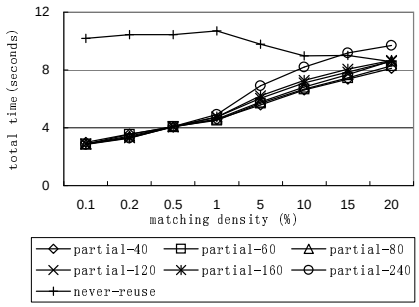
(1) Lookup reuse provides significantly faster event forwarding regardless of the underlying content-match algorithm. It reduces the event processing overhead by about 55% on average for the counting algorithm, and by about 40% on average for k-d tree, where the average is taken over various inputs with small and large matching densities. The advantage of lookup reuse is more pronounced for the counting algorithm, since the counting algorithm is less efficient than the k-d tree for numeric subscriptions.

(2) In most cases, *Always-Reuse* is not the best strategy for reusing lookups. In contrast, *Partial-Reuse* outperforms *Never-Reuse* and *Always-Reuse* on most inputs, suggesting that *Partial-Reuse* is the best strategy for reusing lookups.

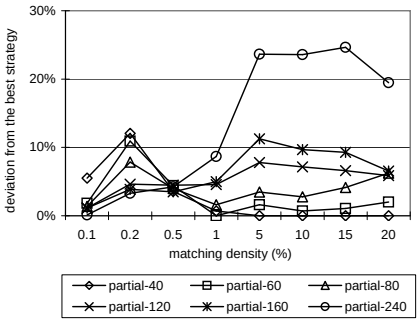
(3) Most benefits of *Partial-Reuse* can be realized by reusing a fairly small number of keys, about 60-80. Even when the matching density is very high (say 20%, where each event matches about $2,000$ subscriptions), a threshold of 60 lookups works well, and yields the best performance among the tested strategies.

The low threshold for *Partial-Reuse* is good news from the standpoint of message overhead. A threshold of 60 means that each event carries at most 60 keys. Since each key is a hash value taking 4 bytes, the message overhead is no more than 240 bytes, maybe much smaller. When compared with the size of an event, which is typically an XML document or database tuples, the message overhead may not be significant. Furthermore, a low threshold means that no matter how popular an event is, the load of matching an event or performing key lookups at a router does not increase too much due to lookup reuse. This leads to better load balanced event forwarding.

We now present the results for the counting algorithm and the k-d tree. In all the graphs, partial-n denotes the Partial-Reuse strategy with a threshold of n. For brevity, we describe the results for the counting algorithm in detail, and then discuss the aspects where the k-d tree differs from the counting algorithm.



(a)



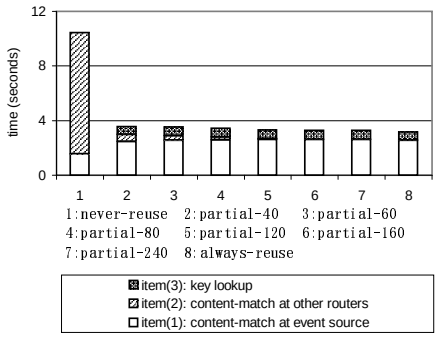
(b)

Figure 1. Counting Algorithm : (a) the event processing overhead of various lookup reuse strategies and (b) the relative performance of these strategies as compared to the best strategy observed for each matching density.

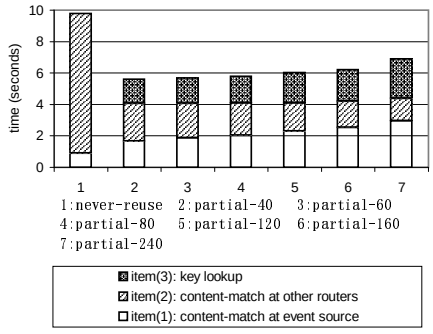
3.2.1 Counting Algorithm

Figure 1(a) shows the event processing overhead, plotted against the matching density, for different lookup reuse strategies when used with the counting algorithm. The time for Always-Reuse is not shown because it has an overhead ranging from 2.8 to 170 seconds, and does not fit easily into the graph.

Figure 1(b) shows the relative performance of the various strategies, when compared to the optimal strategy observed for each data set. For example, if the overhead of *partial*(40) on an input is 10 seconds, and the shortest time measured among all forwarding strategies for the same input is 8 seconds, then the graph shows a deviation of 25% for *partial*(40). In our experiment, Never-Reuse has a deviation declining from 260% down to 5.6%. Always-Reuse has a deviation climbing from 0% up to 1990%. Thus, both curves are not drawn in Figure 1(b). From Figure 1(b), we infer the following.



(a)



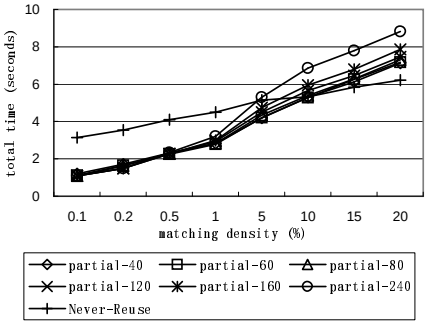
(b)

Figure 2. Counting Algorithm : Breakdown of the event processing overhead. (a) sparse case : 0.2% matching density and (b) dense case : 5% matching density

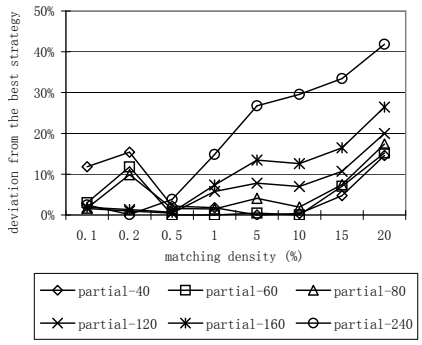
Firstly, there is no clear winner, in that there is no single forwarding strategy which performs the best over all inputs. However, most Partial-Reuse strategies illustrated come close to the optimal in every case. It can be seen that on every input their overhead is within 10% of the optimal for that specific input.

To better interpret the displayed results, we break down the event processing overhead into three parts: (1)time of content-match at the event source (2)time of content-match at the other routers and (3)time of lookup reuse.

Figure 2 shows the breakdown of the overhead for the sparse and dense cases respectively. Due to the same reasons as earlier, graph representing Always-Reuse is not shown. For *Never-Reuse*, item (3) is zero since it never reuses lookups. For *Always-Reuse*, item (2) is zero, because it completely eliminates content-match at every router but the source. For *Partial-Reuse*, all three items may be non-zero.



(a)



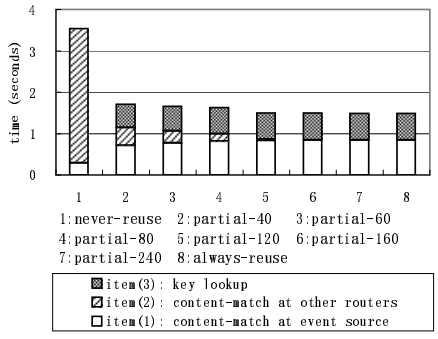
(b)

Figure 3. k-d Tree : (a) the event processing overhead of various lookup reuse strategies and (b) the relative performance of these strategies as compared to the best strategy observed for each matching density.

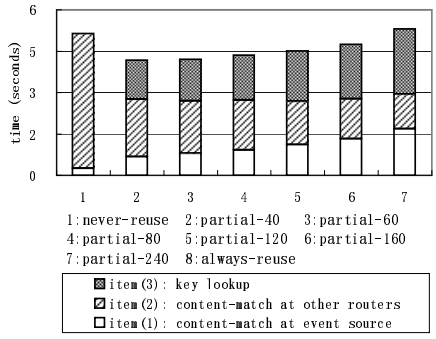
From Figure 2, we infer that by investing a little time on lookup reuse, the cost of content-match at the other routers has reduced drastically, leading to a vastly shortened total running time.

However we also notice that increasing the reuse thresh-

old further does not yield a reduction on item (2) as dramatic as before. For instance, in Figure 2(b) item (2) does not drop much even if we increase the threshold by 6 times. The reason is that item (2) also includes the time spent on content-match at the local interface. Different from *event forwarding*, where finding one matching subscription on an outgoing interface is enough, we are required to find *all* matching subscriptions on the *local* interface. Clearly the cost is much more expensive. If we want to eliminate content-match on the local interface, then a router has to receive a *complete* list of keys of matching subscriptions. Such a condition can barely be satisfied under a high matching density. For example, with a 20% matching density, an event has 2000 matching subscriptions on average, which is much larger than 240, the maximum threshold selected for Partial-Reuse.



(a)



(b)

Figure 4. k-d Tree : Breakdown of the event processing overhead (a)sparse case : 0.2% matching density and (b)dense case : 5% matching density.

3.2.2 k-d tree

Figures 3 and 4 show the corresponding data for the k-d tree algorithm. Qualitatively the trend is similar to the counting algorithm, and lookup reuse still yields a significant speedup. However, the advantage of lookup reuse is less pronounced, since k-d tree is more efficient than the counting algorithm for numeric subscriptions. The results for the k-d tree reinforce our earlier observation that a threshold of reusing 60-80 lookups is enough to realize most of the performance benefits of lookup reuse.

4. Extensions

4.1. General Graph Topology

So far we have assumed a tree topology for the pub-sub network. However, a tree is not a robust infrastructure from the perspective of fault-tolerance and network congestion. So, in many cases a more redundant topology with cycles is used to connect the pub-sub routers.

In case the interconnection topology is not a tree, we still have to preserve the following invariant: a router receives no more than one copy of a given event or subscription. Thus, each event or subscription must be propagated over a tree, though the tree used by different sources may be different. The difference from the case of a shared spanning tree is that there are now multiple trees embedded over the same topology. A downstream router, when forwarding an event, only needs to examine incident interfaces which are part of the tree on which the event is being forwarded. For an acyclic topology, this set of interfaces happens to be all incident interfaces except the one from which the event arrived. Whereas in a topology with cycles, this set of interfaces may be smaller. Such a change in the algorithm does not affect lookup reuse, which still works the same way as before. A pub-sub router can maintain information about trees for different sources by getting the direct support from network protocol [6] or by the utilization of a structured P2P overlay [21].

4.2. Subscription Covering

Subscription covering (defined below) is an effective way to reduce routing table sizes and speed up event matching.

Definition 4.1 *Let $N(s)$ denote the set of all events that match subscription s. Let s_1 and s_2 be two arbitrary subscriptions. We say that s_1 covers s_2, denoted by $s_1 \supseteq s_2$, iff $N(s_1) \supseteq N(s_2)$.*

A pub-sub router can exploit covering relationships among subscriptions as follows. If a newly arrived subscription s_2 is covered by an existing subscription s_1, then s_2 is not forwarded since all events matching s_2 will be received due to the prior propagation of s_1. If covering between subscriptions is used, then a router just forwards a minimal set

of received subscriptions to its neighbor. Under this condition, Always-Reuse can be erroneous, because a list of keys of matching subscriptions passed by an adjacent router is no longer complete (some matching subscriptions are only stored at the local router due to the covering). Thus there is the risk of the event not reaching interested subscribers.

However the covering optimization will not impact Partial-Reuse, since the design of Partial-Reuse already takes into account the situation that the received key list is incomplete. Additional checking is used to find missing matching subscriptions.

References

[1] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC 1999)*, pages 53–61, 1999.

[2] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. In *Proceedings of the International Conference on Autonomic Computing*, pages 332–333, May 2004.

[3] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *Proc. of the 19th Conference on Software Engineering*, Toronto, Canada, May 2001.

[4] F. Cao and J. P. Singh. Efficient event routing in content-based publish/subscribe service network. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM'04)*, 2004.

[5] F. Cao and J. P. Singh. MEDYM: Match-early and dynamic multicast for content-based publish-subscribe service networks. In *4th Intl. Workshop on Distributed Event-Based Systems (DEBS'05)*, pages 370–376, Columbus, Ohio, USA, June 2005.

[6] A. Carzaniga, M. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM'04)*.

[7] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'03)*, pages 163–174, 2003.

[8] C. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *VLDB Journal*, 11(4):354–379, 2002.

[9] G. Cugola, E. Di 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):827–850, 2001.

[10] F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proceedings of the 20th Intl. Conference on Management of Data (SIGMOD 2001)*, pages 115–126, 2001.

[11] 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. In *The 25th International Conference on Distributed Computing Systems(ICDCS'05)*, Columbus, Ohio, June 2005.

[12] Gryphon: Publish/subscribe over public networks.

[13] S. Kale, E. Hazan, F. Cao, and J. P. Singh. Analysis and algorithms for content-based event matching. In *4th Intl. Workshop on Distributed Event-Based Systems (DEBS'05)*, pages 363–369, June 2005.

[14] J. Kulik. Fast and flexible forwarding for internet subscription systems. In *In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.

[15] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proc. of the Int. Conf. on Cooperative Information Systems (CoopIS)*, volume 1901 of *LNCS*, Eilat, Israel, 2000.

[16] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July.

[17] R.Chand and P. Felber. Xnet: A reliable content-based publish/subscribe system. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, Oct. 2004.

[18] Z. Shen and S. Tirthapura. Faster event forwarding in a content-based publish-subscribe system through lookup reuse. Technical Report TR-2006-05-0, Iowa State University, Electrical and Computer Engineering, 2006. http://archives.ece.iastate.edu/archive/00000248/.

[19] Z. Shen, S. Tirthapura, and S. Aluru. Indexing for subscription covering in publish-subscribe systems. In *ISCA 18th International Conference on Parallel and Distributed Computing Systems(PDCS'05)*, Las Vegas, USA, Sept. 2005.

[20] S. Tarkoma and J. Kangasharju. A data structure for content-based routing. In *Internet and Multimedia Systems and Applications (EuroIMSA)*, Feb. 2005.

[21] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.

[22] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 562–571, 2004.

[23] A. Varga. Omnet++ web page. http://www.omnetpp.org.

[24] A. Virgillito, R. Beraldi, and R. Baldoni. On event routing in content-based publish/subscribe through dynamic networks. In *Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, pages 322–328, 2003.

[25] Y.-M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription partitioning and routing in content-based publish/subscribe networks. In *16th International Symposium on DIStributed Computing (DISC'02)*, Oct. 2002.