Content-based Publish/Subscribe using Distributed R-trees

Silvia Bianchi[1], Pascal Felber[1] and Maria Gradinariu[2]

[1] University of Neuchâtel, Switzerland
[2] LIP6, INRIA-Université Paris 6, France

Abstract. Publish/subscribe systems provide a useful paradigm for selective data dissemination and most of the complexity related to addressing and routing is encapsulated within the network infrastructure. The challenge of such systems is to organize the peers so as to best match the interests of the consumers, minimizing false positives and avoiding false negatives.
In this paper, we propose and evaluate the use of R-trees for organizing the peers of a content-based routing network. We adapt three well-known variants of R-trees to the content dissemination problem striving to minimize the occurrence of false positives while avoiding false negatives. The effectiveness and accuracy of each structure is analyzed by extensive simulations.

1 Introduction

Publish/subscribe is an appealing communication primitive for large scale dynamic networks due to the loosely coupled interaction between the publishers and subscribers. In this paradigm, publishers produce *events* and subscribers express their interests through *subscriptions*; any *event* matching the subscription is delivered to the corresponding subscriber. The matching procedure is performed by brokers, which are also responsible for the event delivery. In this way, publishers and subscribers are completely desynchronized in time and space. Many applications such as stock quotes, network management systems, RSS feed monitoring, already benefit from this paradigm.

Publish/subscribe systems designed so far follow two main directions: *topic-based* and *content-based* systems [10]. The *topic-based* systems are similar to group communication where events published on a specific topic are forwarded to all clients subscribed in this topic. The *content-based* systems provide a finer granularity, where subscribers specify their interests based on event contents.

Traditional solutions for content routing are usually based on a fixed infrastructure of reliable brokers. While subscriptions are dynamic, the event routing structure remains mostly static. This approach limits the scalability and routing accuracy with the increase and dynamism of subscription populations. Moreover, this solution introduces single point of failures and bottlenecks.

Another approach to content routing is to design it free of brokers infrastructure, and organize subscribers and publishers in a peer-to-peer overlay through which messages flow to interested parties. By using an adequate structure and gathering subscribers with similar interests to form *semantic communities*, events can be quickly disseminated within a community without incurring significant filtering cost [7]. Obviously, for such techniques to be efficient, one needs to properly structure the overlay to: avoid *false negatives* (a subscriber failing to receive an event it is interested in); minimize the occurrence of *false positives* (a subscriber receiving an event that it is not interested in); *self-adapt* to the dynamic nature of the systems, with peers joining, leaving, and failing; and maintaining the *overlay balanced* in order to provide a publication service time logarithmic in the size of the network similar to the DHT-based implementations. Our challenge was to propose an efficient overlay that positively responds to the above mentioned requirements.

In this paper, we present a novel approach, called distributed R-trees, to address the limitations of content routing in publish/subscribe systems. Distributed R-trees are a class of content-based publish/subscribe overlays where subscribers and publishers are organized in peer-to-peer balanced structures based only on their interests. Our overlays are derived from R-trees [12] and R*-trees [4] that are well-known indexing structures that are specially designed to support spatial database queries. We have implemented the distributed, scalable, and fault tolerant version of these particular data structures and analyzed their impact on the false positives/false negatives via extensive simulations.

Our overlays achieve the efficiency through: 1) organizing subscribers in a distributed and completely decentralized virtual balanced tree, based only on their interests; 2) providing a zero risk of false negatives and maintaining a low level of false positives; 3) masking faults via self-stabilization techniques. The self-stabilization of our structure and its correctness analysis are proposed in a companion paper [5].

The rest of the paper is organized as follows: Section 2 reviews some related work in this domain. Section 3 introduces the considered publish/subscribe model and revisits the R-tree characteristics and its variants. Section 4 presents our distributed R-tree overlays. Section 5 evaluates the effectiveness and accuracy of distributed R-trees for publish/subscribe and Section 6 concludes the paper.

2 Related Work

Content-based publish/subscribe over peer-to-peer systems has been widely addressed in recent years (e.g., [15, 18, 7, 2, 19]). Surprisingly, most of these systems aim at providing scalability and fault-tolerance but very few of them address the central problem in publish/subscribe systems: the presence of false positives and false negatives.

One of the techniques used in publish/subscribe is the *rendezvous*. This technique steams in identifying particular nodes where subscriptions meet publications [1, 3, 16]. The main drawback in such solutions is the high load reported on the *rendezvous* nodes since they centralize all the filtering performed in the system. In contrast, our overlay is totally distributed (i.e., decentralized) and every peer in the system participates in the matching and event dissemination.

Another popular technique in the design of publish/subscribe systems is the use of DHT-based overlays (e.g., Pastry or CAN). The advantage of using overlays is the logarithmic guaranties on the hit time for the publication. HOMED [8] presents a peer-to-peer hypercube overlay for distributed publish/subscribe systems. Also, the peers are organized based on their interests. Similarly, Meghdoot [11] uses the CAN infrastructure. In this system, the subscriptions composed by multiple predicates are partitioned and distributed onto CAN nodes. These approaches have two main drawbacks: the lack of scalability for publish/subscribe systems that require complex subscriptions and the large number of false positives/negatives. The first problem was addressed in [17] by using multi-dimensional spaces. Terpstra *et al.* [17] partition the event space among the peers in the system, but they broadcast the events and the subscriptions to all the peers in the system. Consequently, the number of false positives is in the order of the number of subscriptions in the system.

One of the techniques that focused on minimizing the false positives/false negatives is the organisation of subscribers based on their similarity [7, 2, 19]. In the first two systems, subscriptions form unbalanced trees and the publication complexity is strongly dependent on the subscription distribution. Contrary to this approach, our structure is balanced and offers guarantees comparable to the DHT-based implementations. Sub2Sub [19] is constructed on top of an epidemic semantic-based group membership. Nodes that share the same subscription are linked together in a ring. The impact of this architecture on the level of false positives is studied only inside the similarity groups. Our work does not only provide logarithmic guarantees with respect to the publication hit time but also extends the study of false positives to the whole trajectory of events.

Another approach is the subscription merging [9, 13], which also groups subscriptions based on their similarity and creates a new subscription containing the set. This new subscription is similar to the MBR[3] and it is used in the matching procedure.The merging algorithm used to identify the groups implies in the number of false positives generated in the system. However, the merging problem was proved to be NP-hard [9]. An optimization of subscription merging is presented by Ouksel *et al.* [14] where they propose a Monte Carlo type algorithm. Contrary to our approach, the algorithm introduces false negatives due to the probabilistic nature of the algorithm.

3 Background and System Model

3.1 Content-based Publish/Subscribe Systems

Data Model. As most other publish/subscribe systems, we assume that an event is a set of attribute-value pairs. Each attribute has a name and a numeric or string value. A subscription is a conjunction of predicates over the attribute values, i.e., $S = f_1 \wedge \ldots \wedge f_j$, where f_i is defined as a tuple $f_i = (n_i \;\; op_i \;\; v_i)$ with n_i the name of the attribute, op_i an operator ($<$, $>$, $\leq$, $\geq$, $=$, $\neq$, etc.), and v_i a constant value. For example, a subscription expressed on the attributes a and b may be of the form $(v_i < a < v_j) \wedge (v_k < b < v_l)$.

A schema composed by n attributes may be represented in a cartesian space with n dimensions. The subscriptions correspond to poly-rectangles and events as points. Figure 1 shows a set of subscriptions defined on 2-dimensional space with two attributes. Note that, if one attribute is undefined, then the corresponding rectangle is unbounded in the associated dimension.

[3] MBR is usually represented by the coordinates of the upper left corner and the bottom right corner of the corresponding rectangle.

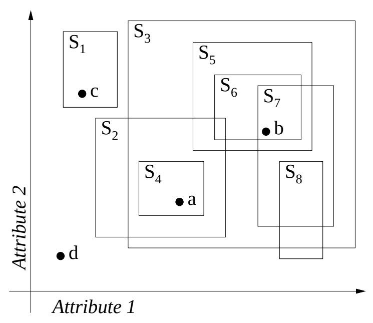

Fig. 1. Sample subscriptions with two attributes.

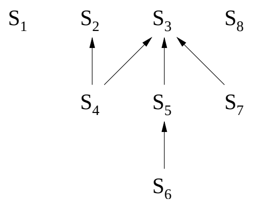

Fig. 2. Containment graph for the subscriptions of Figure 1.

Publish/subscribe systems can take advantage of the property of *subscription containment* in order to improve the filtering procedure. A subscription S_1 contains another subscription S_2, or $S_1 \supseteq S_2$ if S_1 has a larger scope than S_2. This means that any event E_1 that matches S_2 also matches S_1. Conversely, we say that S_2 is contained by S_1, or $S_2 \subseteq S_1$. Note that the containment relationship is transitive and defines a partial order. Geometrically, the containment corresponds to the enclosure relationships between the poly-space rectangles (see Figure 1).

Content-based Routing Protocols. An efficient publish/subscribe overlay should minimize the occurrence of false positives (a peer receiving a message that it is not interested in) and avoid false negatives (a peer failing to receive a message that it is interested in).

A straightforward approach for avoiding false positives and false negatives is to organize the subscribers in a tree structure according to containment relationships [6], such that the subscription of a peer contains the subscriptions of its descendants. Indeed, if an event matches the containee, it *has* to match the container (this guarantees no false negatives); conversely, if it does not match the container, it cannot match the containee (this guarantees no false positives). Figure 2 illustrates the containment graph from the mapping of the example in Figure 1.

A direct mapping of the containment graph to a tree structure [7] is often inadequate. First, it requires a virtual root with as many children as subscriptions that are not contained in any other subscription. Second, depending on the subscription workload, the resulting tree might be heavily unbalanced with a high variance in the degree of internal nodes. Another approach consists in building one containment tree per dimension and adding a subscription to each tree for which it specifies an attribute filter [2]. This solution tends to produce flat trees with high fan-out and generates a significant number of false positives.

Our objective in this paper is to improve these approaches by using bounded-degree height-balanced trees, while preserving the containment relationships that ensure accurate content dissemination. To that end, we propose distributed extensions of the R-tree index structures.

3.2 R-tree Index Structures

R-trees were first introduced by Guttman [12]. An R-tree is a height-balanced tree where each node in the tree is represented by the smallest poly-space rectangle enclosing all the rectangles in its subtree, called *minimum bounding rectangle* (MBR).

An R-tree is characterized by the following properties: *(a)* Every non-leaf node has between m and M children, except for the root that has at least two children; *(b)* The height of an R-tree containing N objects is $\lceil log_m(N) \rceil - 1$; *(c)* The worst space utilization for each node except the root is m/M.

In a classical R-tree structure, the actual objects are only stored in the leaves of the tree and internal nodes only maintain MBRs. An R-tree constructed from the sample subscriptions of Figure 1 is shown in Figure 3 (for $m = 1$ and $M = 3$) and its spatial representation in Figure 4. Note that all subscriptions are stored in the leaves and the role of internal nodes $B_1, \ldots, B_5$ is to keep track of the bounding rectangles that contain their descendants. In distributed settings, obviously, internal nodes must be managed by specific peers in the system.

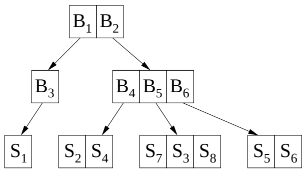

Fig. 3. R-tree for the subscriptions of Figure 1.

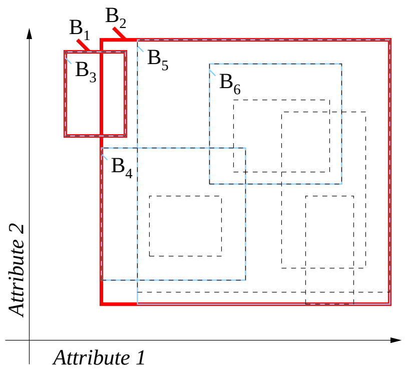

Fig. 4. Spatial representation of the R-tree of Figure 3.

Upon a join of a new node, if the children set becomes bigger than M entries, the children set must be split. There are several well-known methods for splitting an overflowing node during the join. In the following, we present three classical methods, which are supported by our distributed R-tree structures: *(i)* The *linear* method [12] chooses two children from the overflowing node such that the union of their MBRs waste the most area and places each one in a separate node. The remaining children are assigned to the nodes whose MBR is increased the least by the addition. This method takes linear time. *(ii)* The *quadratic* method [12] chooses two children from the overflowing node that would waste the most area if they were in the same node, and place each one in a separate node. The remaining MBRs are examined and the one whose addition maximizes the difference in coverage between the MBRs associated with each node is added to the node whose coverage is minimized by the addition. This method takes quadratic time. *(iii)* The *R*-tree* splitting method [4] attempts to reduce not only the coverage, but also the overlap. Instead of just splitting the node when it overflows, it also tries to allocate some entries to a better suited node through reinsertion.

In the next section, we extend R-trees and show how the resulting *distributed R-tree* variants can be used to produce efficient content dissemination networks.

4 Distributed R-trees for Content-Routing

Our distributed R-tree is a virtual height-balanced tree for content-routing in publish/subscribe systems, where the peers are organized according to their interests (subscriptions). Unlike traditional R-trees, each node in the tree maps to one peer in the network.

Every peer has two roles: publisher/subscriber and router. A peer that registers a subscription will participate in the overlay (the nature of the subscription will influence the position of the peer in the tree) and may or not publish events. Also, every non-leaf node acts as a forwarder during event dissemination. Node positions may change due to the dynamism of the system: a leaf node may become an internal node and inversely.

Each node in the tree corresponds to a subscription. The peer associated with the node keeps track of a set of neighbors: its parent and, for non-leaf nodes, a set of children (between m and M, except for the root).

Each peer p maintains its own subscription S_p, as well as an MBR that encloses the MBRs of all its children. Therefore, each node keeps track of the MBRs of its children and updates its own MBR dynamically when it changes (e.g., due to the arrival or departure of a peer). Note that the MBR of a leaf node is identical to its subscription.

In order to maintain the balanced nature of the R-trees, a subscriber may appear at different locations in the tree. More precisely, a node must appear in exactly one leaf, and may appear on all the nodes of a suffix of the path from the root to that leaf. Thus, an interior node at level l of the tree is recursively its own child at level $l + 1$. We consider separately every occurrence of a peer in the tree: each one has its own set of neighbors and related data.

The choice of which peers are promoted as internal nodes is performed according to existing containment relationships so as to minimize the likeliness for false positives. We typically choose the node whose current MBR is largest. Hence, if a peer whose MBR covers all the other MBRs in the children set, then it trivially becomes the new parent: the containment relationship is preserved and there is no occurrence of false positives. If MBRs in the children set intersect or are disjoint, the peer with the largest MBR is chosen in order to minimize the size of the area corresponding to false positives.

Figure 5 illustrates a possible configuration of the distributed R-tree structure for the subscriptions of Figure 1; and Figure 6 shows the associated communication graph.

Selective Event Dissemination. Event filtering and dissemination in distributed R-trees are fully distributed among the peers in the system. This process is very simple and only relies on local information.

Upon receiving an event, a peer forwards it to each child whose MBR contains the event, unless the message was received by that child. If the message originated from a

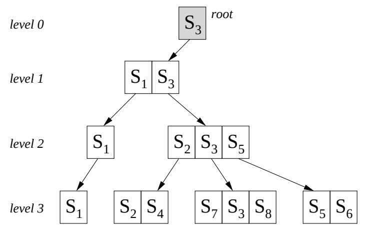

Fig. 5. Distributed R-tree for the subscriptions of Figure 1.

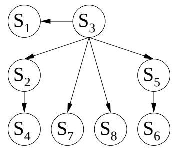

Fig. 6. Communication graph for the distributed R-tree of Figure 5.

descendant (i.e., it is propagated upward the tree), then the peer forwards it to its parent as well. If the event matches the local subscription, it is delivered to the user.

An event E_i matches a subscription iff the subscription's attributes are satisfied by the event, i.e., the event falls in the attribute range for each of the dimensions. If a node receives an event that does not match its subscription, the event is considered a false positive. As leaf nodes have an MBR equal to their subscriptions, only internal nodes can experience false positives. Moreover, if the subscriptions of all the descendants of a subscriber p are contained within p's subscription, then p's MBR is identical to its subscription and it does not experience false positives.

By construction, our R-tree structures cannot produce false negatives during dissemination, i.e., all the subscribers that have subscribed for an event eventually receive it (unless there is a failure).

Registering a Subscription. A peer joining the network by registering a subscription can contact any existing peer. The subscription request is redirected upward the tree until it reaches the root. Then, it is pushed downwards to the last non-leaf node whose interests are closest to those of the new subscriber (as determined by comparing MBRs). Having neighbors with similar interests helps minimize the occurrence of false positives. Our distributed R-tree structures support two variants for selecting the best branches when traversing down the tree to register a new subscription: 1) *R*: we choose the subtree that needs the least enlargement of its MBR to insert the new subscription; upon tie, we select the subtree with the smallest MBR [12]; 2) R^*: we proceed as above until we reach the last non-leaf nodes; then, we insert the new subscription in the node that needs the least overlap enlargement; upon tie, we select the node whose MBR needs the least area enlargement [4].

The concepts of coverage and overlap are important to minimize the occurrence of false positives. Each of these variants attempts to minimize the coverage or/and the overlap of the MBRs at the same level. Minimizing the coverage of a node's MBR permits to minimize the dead space between its MBR and the MBR of its children. Minimizing the overlap of MBRs at the same level avoids an event being disseminated to all the subtrees.

As previously discussed, each node has between m and M entries by level (except for the root node). If the new parent has less than M children, it inserts the new subscription in its children set. Otherwise, the parent creates two children sets with at least

m subscriptions, effectively creating a new subtree. Splits propagate upward the tree: a split at a node may trigger a split at its parent when inserting the newly created subtree. We have implemented and compared the efficiency of the different splitting variants (quadratic, linear, R^*) applied to our distributed R-trees.

Dynamic Reorganization. In order to improve the accuracy of event dissemination, the nodes are dynamically reorganized during the join and splitting procedures. Each internal node in the tree checks if it is the best cover for its subtree. If one of its children provides better coverage (e.g., because its MBR has grown after the insertion of a new node), then the child is promoted and replaces its parent.

Canceling a Subscription. We only discuss controlled departures here, i.e., peers explicitly unregistering their subscriptions from the system. Fault tolerance is supported by the means of a self-stabilizing tree maintenance protocol discussed in [5].

A peer leaving the system notifies its parent(s). When receiving such a notification, the parent removes its departing child from its children set. After the peer leaves the system, the whole branch that used to contain the departing peer, must be repaired.

If the children set drops below m after the peer leaves the system, the children set is reinserted in the tree. Thus, for each child, the whole subtree rooted by the child is reinserted in the tree at the same level. Conversely, if the children set remains between m and M, there are two scenarios: if the parent is the departing peer, then a new parent is promoted to occupy the vacant position; otherwise, the MBR of the parent is updated since it may become smaller.

5 Evaluation

This section describes the results of our evaluation and comparision of the different distributed R-tree structures presented earlier.

Parameter	**Values**
Splitting Method	[quadratic], linear, R*
Number of subscriptions	$1,000$, $2,500$, $5,000$, [$10,000$], $25,000$, $50,000$
Number of events	[$2,500$]
Subscription distribution	uniform, uniform-25:75, uniform-10:90, Zipf, [Zipf-25:75], Zipf-10:90
Event distribution	[uniform]
Number of dimensions	2, [4], 6, 8, 10, 12
Degree of the tree (m, M)	$(2,5)$, [$(5,10)$], $(10,20)$, $(15,30)$, $(20,40)$

Table 1. Parameters used for the experiments

Experimental Setup. Subscriptions are defined as a set of d attribute-range pairs, each of which corresponds to a dimension. The range specifies the set of values that the consumer is interested in. Without loss of generality, we used range values between 0 and $1,000$. Note that a range may represent a single value. Events are points in the d-dimensional space.

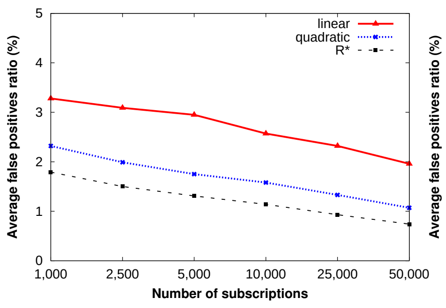

Fig. 7. False positives ratio for different subscription set sizes.

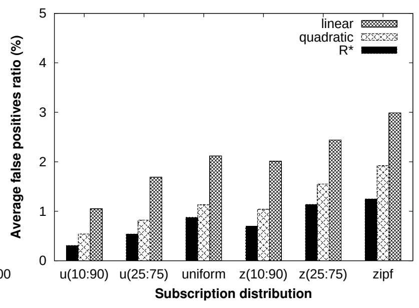

Fig. 8. False positives ratio for different subscription distributions.

We analyzed the performance of the system under uniform and skewed subscription workloads; and with a uniform event distribution. Skew is simulated using a power-law distribution (Zipf with $\alpha = 1$) and is applied to the origin of subscriptions only: their size is always chosen according to a uniform distribution.

To model and observe the influence of containment relationships, we have generated some subscription sets with a given ratio of container/containee subscriptions. Given a ratio of $X{:}Y$, we have first generated $X\%$ of the subscription population according to the current distribution. For each subscription in the remaining $Y\%$, we have taken the following steps: select a random subscription S from the current set; generate a uniform random subscription S' such that $S \supseteq S'$; and insert S' in the set. This method guarantees that at least $Y\%$ of the subscriptions are containees. We considered uniform and Zipf distributions, as well as two $X{:}Y$ ratios: $25\%{:}75\%$ and $10\%{:}90\%$.

In the experiments, we evaluated the efficiency of our approach in terms of *false positives ratio*, i.e., the percentage of the nodes in the system that receive events that does not match their interests. For simplicity, we assume that events are injected at the root. Note that this assumption is equivalent to having each event with at least one interested consumer being produced by a publisher with a matching subscription, i.e., producers never experience false positives locally.

Obviously, an event that does not match a single subscription is expected to show a lower false positive ratio than an event with many interested subscribers, because the latter is likely to be propagated deeper in the tree. Therefore, we shall also observe the effect of event popularity in our study. As leaves have an MBR equal to their subscriptions and a node forwards an event to each of its children whose MBR contains the event, only interior nodes can experience false positives. We do not consider false negatives since our distributed R-tree structures do not produce any.

The number of events was fixed in all simulations to $2,500$ for computing false positive ratios. We have used the parameters shown in Table 1 (default values are in brackets). For each simulation, we have varied the values of the parameter to be observed and fixed the remaining ones to their default value.

Evaluation Results. We measured the false positives ratio for different sizes of subscription sets. Figure 7 shows that the average false positives ratio is less than 5% and slightly decreases with the size of the subscription set. Comparing the four splitting methods, we observe that R* presents the best results because it reinserts nodes in case of overflow instead splitting immediately. This may improve the containment relationship along the tree and, consequently, the routing accuracy because R-trees are known to be highly susceptible to the order in which entries are inserted.

Figure 8 shows the routing accuracy when varying the distribution of the subscriptions. We observe that better results are obtained for subscriptions with high containment relationship, which confirms that our trees do indeed preserve and take advantage of containment relationships. Accuracy is also slightly better with a uniform subscription distribution.

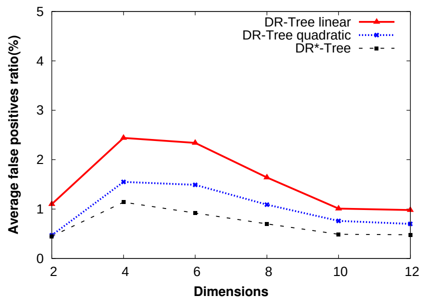

Fig. 9. False positives ratio for different dimensions.

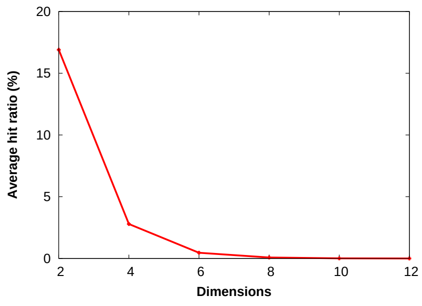

Fig. 10. Hit ratio for different dimensions.

Figure 9 illustrates the average false positives ratio for different dimensions. Surprisingly, accuracy improves with the number of dimensions. This is due to the fact that less nodes are interested in the event, as can be seen in Figure 10. We have, therefore, plotted the false positives ratio as a function of the number of hits for each experiment with three dimensions. The results are shown in Figure 11. We observe now that the average false positives ratio actually increases with the dimension but remains reasonably small, never reaching 10% even for linear. The same general trends are exhibited by all three splitting method.

As discussed before, our approach differs from traditional R-trees in that some subscriptions may appear at different levels in the logical tree. Thus, the degree of a node varies depending on its position and number of occurrences in the tree, in addition to the values of m and M. Figure 12 presents simulation results for different degrees between $M = 5$ and $M = 40$, where $m = M/2$; and Table 2 shows the maximum, average, and variance of the degree of internal nodes. We observe a clear trade-off between accuracy and nodes degree: increasing the degree improves accuracy. In the studied scenario, a value of $M = 20$ appears to be a good compromise.

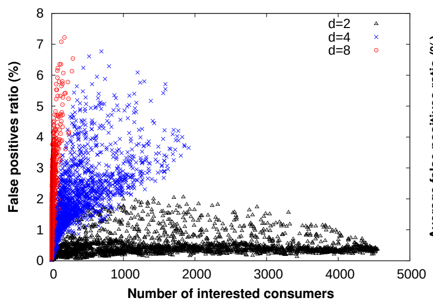

Fig. 11. False positives ratio vs. hit ratio for different dimensions (one point corresponds to one experiment).

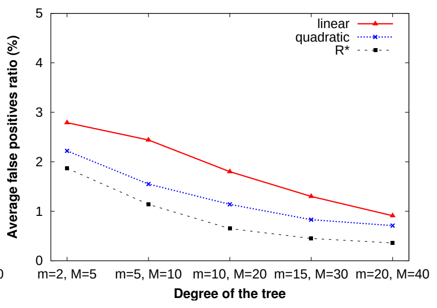

Fig. 12. False positives ratio for different degrees of the tree.

For comparison purposes, a tree built as a direct mapping of the containment graph using the same $10,000$ subscriptions (Zipf-25:75) would have a virtual root node with approximately $2,000$ children and would obviously not be height-balanced.

	quadratic			**linear**			**R***		
Degree	*Max*	*Avg*	*Var*	*Max*	*Avg*	*Var*	*Max*	*Avg*	*Var*
m=2, M=5	20	4.59	5.09	19	4.61	5.08	25	3.91	8.47
m=5, M=10	29	7.99	10.98	28	7.93	10.66	30	7.49	16.27
m=10, M=20	51	14.92	25.40	37	14.82	23.15	42	15.05	32.87
m=15, M=30	51	21.91	35.70	50	21.69	33.52	57	22.28	56.29
m=20, M=40	68	28.02	55.38	60	28.39	54.02	77	30.00	81.10

Table 2. Degree statistics

6 Conclusion

In this paper we studied a class of distributed R-trees and their applicability to build content-based publish/subscribe overlays. Our study focused on the properties of the resulting topology and the accuracy of event dissemination (occurrence of false positives and false negatives). Distributed R-trees, proposed in this paper, are a decentralized implementation of the R-tree structure and its variants, which are widely used in database systems. These overlays are fully adapted to embed a publish/subscribe system with complex subscriptions (multi-dimensional) and cope with the dynamism of the system. The overlays are designed such that they eradicate false negatives and drastically drop the number of false positives. Moreover, organizing the peers based on their interests minimizes both the amount of matchings in the system and the latency during event dissemination (the worst case being logarithmic in the size of the network). We have implemented and analyzed via simulations the evolution of false positives applying the different variants of insertion and splitting methods. Independently of the absolute effectiveness of R-trees for content-based publish/subscribe, this study provides valuable insights in the relative performance of their different variants. Note that distributed R*-trees provide the best overall performance.

References

1. I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe DHT network. In *Proceedings of 26th ICDCS*, page 23, 2006.
2. E. Anceaume, M. Gradinariu, A. K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self-* peer to peer publish/subscribe. In *Proceedings of 26th ICDCS*, page 22, 2006.
3. R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of 25th ICDCS*, pages 437–446, 2005.
4. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD*, 1990.
5. S. Bianchi, A.K. Datta, P. Felber, and M. Gradinariu. Stabilizing dynamic R-tree based spatial filters. In *Proceedings of 27th ICDCS*, 2007.
6. 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.
7. R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Proceedings of Euro-Par*, 2005.
8. Y. Choi, K. Park, and D. Park. HOMED: a peer-to-peer overlay architecture for large-scale content-based publish/subscribe system. In *Proceedings of 3rd DEBS*, pages 20–25, 2004.
9. A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Query merging: Improving query subscription processing in a multicast environment. *IEEE TKDE*, 15(1):174–191, 2003.
10. P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.
11. A. Gupta, O. D. Sahin, D. Agrawal, and A. El Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In *Proceedings of 5th Middleware*, 2004.
12. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD*, pages 47–57, 1984.
13. G. Li, S. Hou, and H. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *Proceedings of 25th ICDCS*, pages 447–457, 2005.
14. A. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient probabilistic subsumption checking for content-based publish/subscribe systems. In *Proceedings of 7th Middleware*, 2006.
15. G. Perng, C. Wang, and M. Reiter. Providing content-based services in a peer-to-peer environment. In *Proceedings of 3rd DEBS*, pages 74–79, 2004.
16. R. Renesse and A. Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *Proceedings of 3rd IPTPS*, 2004.
17. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of 2nd DEBS*, pages 1–8, 2003.
18. P. Triantafillou and I. Aekaterinidis. Content-based publish/subscribe over structured P2P networks. In *Proceedings of 3rd DEBS*, pages 104–109, 2004.
19. S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. van Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *Proceedings of 5th IPTPS*, 2006.