Efficient Filtering and Routing in a Scalable XML-Based Publish-Subscribe System

Yuanyuan Tian Jignesh M. Patel Farnam Jahanian
Department of Electrical Engineering and Computer Science
University of Michigan
{ytian, jignesh, farnam}@eecs.umich.edu

Abstract

This paper introduces YAK – a scalable content-based publish-subscribe system. YAK employs XML documents and expressive XPath queries as the publication and subscription model. To achieve high scalability, it combines the advantages of content routing in existing publish-subscribe systems and the efficient query indexing technique in the context of XML filtering. The filtering and routing strategy used in YAK exploits the locality of subscribers and therefore dramatically reduces the network communication overhead. Through performance tests, we conclude that our YAK publish-subscribe model is highly scalable, especially when subscriptions exhibit high selectivity and regionalism.

1. Introduction

The publish-subscribe paradigm is a loosely coupled communication model, in which a subscriber registers her interest in a subject or a pattern of events, and is notified whenever a published message matches the registered interest. While publish-subscribe systems have traditionally been subject-based, where publications are grouped into different topics and a subscriber registers an interest in a particular topic, there is emerging popularity of the research on the content-based approach to information dissemination. In content-based publish-subscribe systems, subscribers are not constrained by topics, they may register their interests as predicates or queries based on the contents of publications.

In this paper, we introduce YAK – a scalable content-based publish-subscribe system. The main goals of YAK are scalability of the system and expressiveness of subscriptions. To achieve high scalability, YAK combines the advantages of the content routing technique in traditional content-based publish-subscribe systems and the query indexing method in XML filtering systems. YAK contains a network of brokers that distribute the workload of delivering publications from publishers to subscribers. Each broker receives subscriptions from its local clients or neighbor brokers, and builds an index on the subscriptions, to facilitate the efficient matching of publications against subscriptions. Publications are delivered to interested local subscribers, and forwarded to a subset of neighbor brokers who have interested clients. The filtering and routing strategies used in YAK, exploit the locality of subscribers and therefore dramatically reduce the network communication overhead. Subscription expressiveness is another crucial criterion for evaluating a publish-subscribe system. In YAK, messages are published as XML [1] documents, and interests of subscribers are expressed as XPath [2] queries. XML includes both content and structure information within documents and XPath queries enable subscribers to express more complex interests that exploit both content and structure of published documents. Besides, XML has become a standard for information exchange on the internet. There are emerging XML standards and tools in a number of fields, including finance, biology, chemistry, astronomy, library science, and meteorology. This makes XML more appealing to YAK system. Our simulation study demonstrates the scalability of YAK especially when subscriptions exhibit high selectivity and regionalism.

Recent studies [16, 17] argue that the publish-subscribe paradigm is more applicable to the challenging requirements of the mobile communities than the traditional client-server model is. A number of researches on extending the static content-based systems to support mobile computing, including [14, 22], are conducted. In [15], a generalized peer-to-peer approach to content-based publish-subscribe is

proposed. Although we implement the routing and filtering algorithms in a static publish-subscribe system, actually our algorithms can be easily adjusted to mobile and peer-to-peer publish-subscribe paradigms.

The remainder of the paper is organized as follows. Section 2 introduces the related work that motivates our study. Section 3 provides a system overview. Section 4 focuses on the filtering and routing algorithms in YAK system. The YAK communication infrastructure is discussed briefly in section 5, and performance is evaluated in section 6. Finally, we conclude the paper in section 7.

2. Related work

2.1. Scalable content-based publish-subscribe systems

There are a number of well-known scalable content-based publish-subscribe systems that are built on a distributed network of brokers as YAK does, such as Gryphon, Siena and Rebeca. In these systems, a published event is expressed as a set of attribute-value pairs and a subscription is a conjunction of predicates. These systems typically focus on the efficient routing of published messages. The main idea is to only forward publications to brokers who have interested subscribers. In Gryphon [3], publications are routed by means of the link-matching algorithm, in which every broker uses a decision tree to determine which subset of its neighbors to send a publication to. Gryphon's limitation, however, lies in the fact that each subscription is propagated to all brokers in the network and each broker keeps a whole list of subscriptions in the system, which results in the increase of subscription network traffic and redundant subscription information. Siena [4] exploits the superset and subset relationships between the selection ranges of subscriptions (e.g. the selection range of a>5 covers that of a>10) and avoids an unnecessary subscription forwarding if a covering subscription has already been forwarded. Rebeca [13] provides a more general content-based routing framework, which includes 3 types of routing mechanisms: identity-based routing, covering-based routing and merging-based routing. In identity-based routing, a subscription is not forwarded if a former identical subscription is already forwarded; the covering-based routing resembles the routing technique in Siena; and in the merging-based routing, several subscriptions can be merged into a more general covering subscription and only the merger is forwarded. Through content routing, the above systems can achieve high scalability in a wide-area network. However, the simple subscription and publication model is a major shortcoming in these systems. Although the newest version Siena also supports publications written as XML documents, essentially the Siena-XML [5] is just a wrapper on top of Siena. It converts XML documents to the Siena's event formats, and calls the Siena processing engine to do the matching. As the event format in Siena does not support structure information, the translation from XML documents simply ignores the hierarchical structure information within documents, which implies the loss of information and inaccuracy of the matching results. On the other hand, our YAK model provides a real XML-based solution and implements a covering-based content routing similar to the one used in Siena and Rebeca.

How to extend the existing publish-subscribe system to support mobile event notification service has recently become a hot research topic. [14] and [22] propose solutions to this problem, without changing the underlying content routing algorithms in existing static systems. Both approaches resemble the mobility support in mobile IP protocol and mainly differ in whether mobility is controlled by the application or the middleware. Building publish-subscribe systems on peer-to-peer networks is another recently studied topic. In [15], Terpstra and his colleagues combine the content routing algorithms in Rebeca and the Distributed Hash Tables (DHT) technique in Chord, and provide a peer-to-peer approach to content-based publish-subscribe. It is not hard to see that, this solution can be generalized to other publish-subscribe systems which have similar content routing mechanisms as in Rebeca. As a result, although our routing and filtering algorithms are implemented in a static system, it is not hard to adapt them to the mobile and peer-to-peer environment.

2.2. Efficient XML matching

As XML technology becomes more and more popular, a number of XML-based publish-subscribe systems emerged. In these systems, published messages are written as XML documents and interests of subscribers are expressed in XML query languages. Efficient matching of XML documents is widely studied in the context of query indexing. The main idea of query indexing is that since user interests expressed as XML queries are in a great amount and relatively stable in the system, indexes can be built upon the queries to efficiently identify

the queries that match an incoming XML document. In XFilter [6], a subscription written in XPath is translated into a modified Finite State Machine (FSM). Based on the FSM representation, an FSM-based indexing mechanism and a matching algorithm are developed to quickly locate the relevant user queries for a given XML document. Whereas, YFilter [7], the subsequent project of XFilter, combines the FSM for each subscription into a single Nondeterministic Finite Automaton (NFA), which attempts to merge the common query paths to further improve performance. Xtrie [8] is another query indexing technique, which builds indexes based on the substring and superstring relationships among XPath expressions and shares the filter processing of common substrings. However, in all of the above XML filtering systems, filter processing is done in a centralized server. None of them addresses the distribution of filtering and the routing of messages.

Besides query indexing, [21] proposes an alternative approach to XML matching. Subscriptions expressed as XPath queries are divided into value predicates and tree structures, and stored into relational databases. The matching procedure is implemented as a recursive SQL query, which basically joins the results of value predicate matching and tree structure matching. This solution breaks the main memory limitation – a disadvantage of the query indexing technique – at the cost of efficiency. Again, message routing among multiple brokers is not explored in this study.

2.3. Continuous query

If subscriptions are seen as queries, then publish-subscribe systems are closely related to the research direction of continuous query. Continuous queries are relatively persistent queries in the system and continuously evaluated against incoming data stream. NiagaraCQ [9] is such a model that uses the incremental group optimization strategy to achieve scalability. Based on the observation that there is similarity among a large number of queries, NiagaraCQ splits queries into sub queries and group similar sub queries to share the common computation. However, NiagaraCQ is implemented in a non-distributed way, which potentially becomes a bottleneck for further enhancing scalability. Aurora [10] with its subsequent projects Aurora* and Medusa [11], provides a preliminary work on the distributed streaming paradigm. Aurora is a centralized DBMS-Active Human-Passive prototype for monitoring applications. A special storage component called a connection point is introduced in the Aurora operation network, to provide persistent storage for support of dynamic modification to the network. Aided by connection points, Aurora query optimizer dynamically optimizes query plans by modifying portions of the network. Aurora* and Medusa were developed to extend Aurora to distributed stream processing models. Aurora* works in an intra-participant distribution environment, where all nodes belong to the same administrative domain; whereas, Medusa uses a market-based economic paradigm to support services across administrative boundaries. Aurora* and Medusa share the basic mechanisms for scalable communication, load sharing, and high-availability. The Aurora series provide scalable distributed stream processing models, but they fail to adequately explore the sharing of common computation among numerous queries.

3. Distributed system architecture

One of the major goals in the YAK project is to achieve high scalability. So, in YAK, there are multiple interconnected brokers who are responsible for delivering messages between publishers and subscribers. Each broker has a number of local clients, either publishers or subscribers, connected with it. There is a special broker called rendezvous (RV) server who acts as a coordinator for all brokers. Besides the normal responsibility of a broker, the RV server maintains the topology of the broker network and manages the group membership of brokers. Although the RV server has no special role in the filtering and routing algorithms, it is crucial for the communication infrastructure. Figure 1 shows the overview of the YAK system.

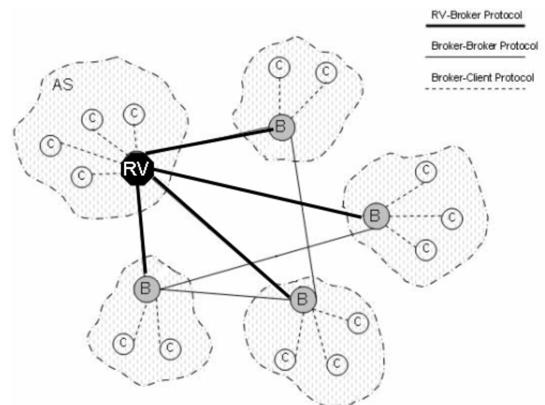


Figure 1. An overview of the YAK system

In YAK, publications are expressed as XML documents and subscriptions are XPath queries. So in this paper, document and publication are substitutable for each other and likewise query and subscription are used interchangeably. Like [7], in this work, we use XPath to select entire documents rather than parts of documents. That is if an XPath query matches at least one element of a document, then the document satisfies the XPath query.

As shown in Figure 2, for each broker, there are three crucial components based on the underlying communication layer: publication filtering, publication routing, and subscription routing. The publication filtering component decides which local clients are interested in the published documents; the publication routing component resolves to which neighbor brokers the published messages should be forwarded; whereas, the subscription routing component controls whether or not to propagate an incoming subscription to some of its neighbor brokers so that the neighbors know what this broker is interested in, and therefore can forward related publications to it. As brokers are organized in a general graph topology, acyclic paths must be followed to avoid repeated message deliveries. Therefore, the message routing in YAK is based on a global minimum spanning tree extracted from the broker network. If a broker crashes, the RV server is responsible for installing a new global spanning tree (refer to Section 5 for fault tolerance issues). The organization of brokers and clients can be consistent with the real deployment of the network. The connections between a broker and its clients can be expected to be high-speed intra-Autonomous-System (AS) connections, while brokers are connected by inter-AS links. The strategy, that every broker is only responsible for filtering incoming messages against the interests of its local clients and forwarding publications only to neighbors who have potential interested subscribers, exploits the locality of subscribers, and reduces the network communication overhead.

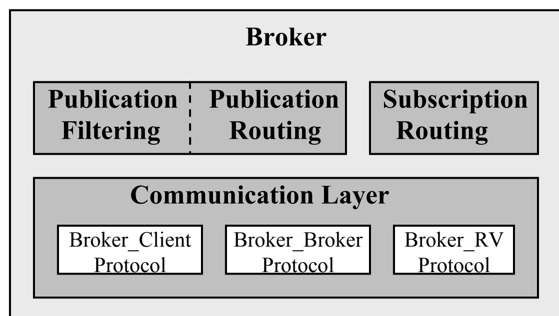


Figure 2. Internal architecture of a broker

4. Efficient filtering and routing strategy

4.1. Organizing XPath queries

In YAK, subscriptions are expressed as XPath queries. XPath takes an XML document as a tree of nodes. An XPath query can both match parts of the tree and support predicates on text data, attributes or positions of addressed elements. For example, the query **/product[book]/electronics//camera[@brand= "canon"]**, matches every XML document starting from the **production** element that has a child element **book** and another child element **electronics** that has a descendent element **camera** with the attribute **brand** equal to **canon**.

We adopt the query indexing approach in YFilter [7], so as to efficiently organize queries. In YFilter, every XPath query is represented as an NFA, in which location steps of the query are mapped into machine states. To share the filtering of common prefixes of queries, YFilter combines the state machines of multiple queries into a single NFA. Figure 3 illustrates the NFA representations for 4 queries and their combined NFA. The shared states in the combined NFA are indicated in the shadow. In YAK, each broker organizes all of the subscriptions (XPath queries) that it received into one NFA. As in YFilter, each accepting state in the NFA maintains an ID list of subscriptions that share the structure along the path from the starting state to this accepting state. In order to support routing algorithms in YAK, we extend the NFA structure in YFilter to append to each accepting state a *forward list*, which is an ID list of brokers that the corresponding query structure has been forwarded to. Note that the NFA is purely based on the hierarchical structure of elements in XPath queries. So, the NFA index is used to match the structure of queries against incoming documents. The value-based predicates in queries are evaluated by the Selection Postponed approach in a post-processing phase after the structure matching. Queries with nested paths are decomposed into several absolute paths. For example, the query **/a/b[c]/d** is decomposed into **/a/b/d** and **/a/c**. The decomposed paths, uniquely identified by the query id and path id, are added into the NFA and processed normally in the structure matching phase. In the post-processing phase, an operator called Nested Path Filter is assigned to each query that contains nested paths to check whether the incoming document matches all the paths of the query. A more detailed description of the above techniques can be found in [7].

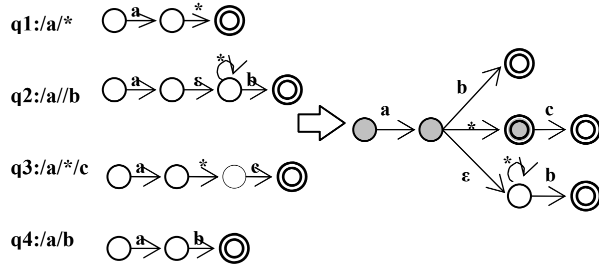


Figure 3. XPath queries and their NFA representations

4.2. Relation on queries

A relation can be derived based on the structures of XPath queries. Ignoring all the value-based predicates, if an XPath query **q** is a prefix of another XPath query **q'**, then **q** is a structure_prefix of **q'**, with the notation $\mathbf{q} \prec \mathbf{q'}$. For example, in Figure 3, q1 is a structure_prefix of q3, i.e. q1 $\prec$ q3. When $\mathbf{q} \prec \mathbf{q'}$ and $\mathbf{q'} \prec \mathbf{q}$, we say **q**=**q'**. For the convenience of expression, let the notation $\mathbf{S_q}$ be the set of documents that matches the structure of **q**. That is, $\mathbf{S_q}$ = {**d** | **d** is an XML document $\wedge$ **d** matches the structure of **q**}.

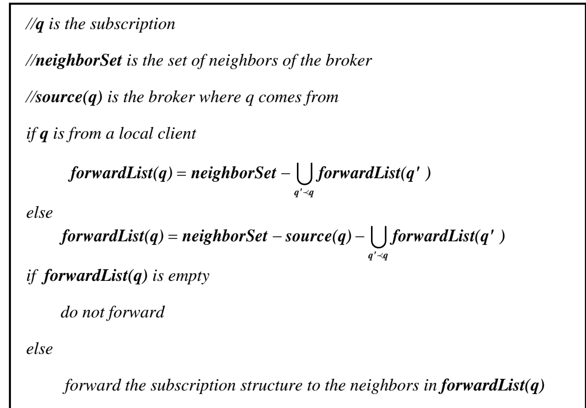
The structure_prefix relation has very good properties which are significantly useful for the filtering and routing algorithms. First, if an XML document matches the structure of a query **q'**, it matches any structure_prefix of **q'**. In other words, given $\mathbf{q} \prec \mathbf{q'}$, $\mathbf{S_q} \supseteq \mathbf{S_{q'}}$ ($\mathbf{S_q} = \mathbf{S_{q'}}$ when **q**=**q'**). Second, if $\mathbf{q} \prec \mathbf{q'}$, then **q** and **q'** share the same path from the starting state to the accepting state of **q** in the combined NFA.

4.3. Subscription routing

In YAK, the subscription routing algorithm is based on a global minimum spanning tree of the broker network. Each broker builds an NFA for the subscriptions (expressed in XPath queries) submitted from its local clients and forwarded from its neighbor brokers. Note the fact that a document matches the structure of an XPath query doesn't mean that it matches the query. As we only exploit the covering relation on the structures of subscriptions, we don't forward the subscription itself but a modified one, which contains only the structure of the XPath query. For example, if a subscription **/a/b/c[@d="10"]** needs to be forwarded, actually only the structure **/a/b/c** is forwarded. We can guarantee that a subscription that covers the original one is forwarded.

4.3.1. A basic algorithm

Adding a subscription. When a new subscription is submitted or forwarded to a broker, the broker combines the subscription into the NFA, and decides whether to forward its structure to its neighbors, and if so, decides which neighbors to forward. The algorithm is shown in Figure 4.


*//**q** is the subscription*

*//**neighborSet** is the set of neighbors of the broker*

*//**source(q)** is the broker where q comes from*

*if **q** is from a local client*

$$forwardList(q) = neighborSet - \bigcup_{q' \prec q} forwardList(q')$$

else

$$forwardList(q) = neighborSet - source(q) - \bigcup_{q' \prec q} forwardList(q')$$

*if **forwardList(q)** is empty*

do not forward

else

*forward the subscription structure to the neighbors in **forwardList(q)***

Figure 4. Algorithm for adding a subscription

The core of the algorithm is to decide the *forward list* of the newly added subscription. The *forward list* of the newly added subscription **q** is the set difference of the broker's neighbors and all the brokers in the *forward lists* of structure_prefixes of **q**. And if **q** is from a neighbor broker, this neighbor where **q** comes from is also taken away from the *forward list* of **q**. The rationale is explained below. Let **q'** be a structure_prefix of **q**. If broker **b** is in the forward list of **q'**, and **b** receives a publication that matches the structure of **q**, then according to the first property of structure_prefix, this publication also matches the structure of **q'**. As **q'** has already been forwarded to **b**, **b** will send the document to the current broker. So, there is no need to forward **q** to **b**. Besides, all the structure_prefixes of **q** can be found from the starting state of the NFA to the accepting state of **q**, according to the second property of structure_prefix.

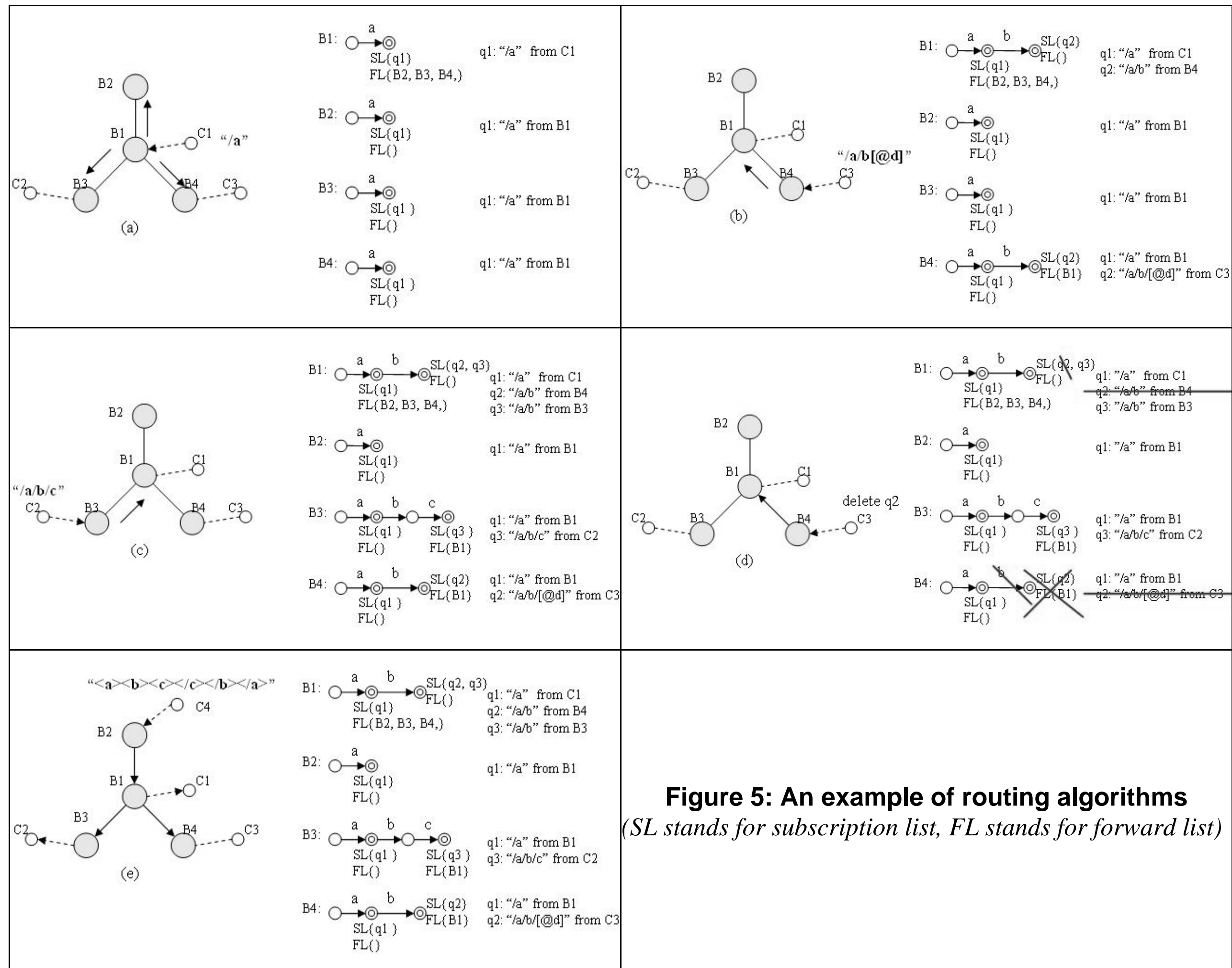


Figure 5: An example of routing algorithms
(SL stands for subscription list, FL stands for forward list)

Figure 5(a), 5(b) and 5(c) show an example of the subscription routing process. In the example, there are 4 brokers, B1 to B4, interconnected based on a global spanning tree. At the very beginning, there is no query in any broker. In Figure 5(a), the local client C1 of B1 submits a query **/a**. The query is combined in B1's local NFA, and propagated to its neighbors B2, B3 and B4. After that, the NFA of each broker is shown on the right side of Figure 5(a). In Figure 5(b), C3 submits a query **/a/b[@d]** to B4. This subscription is forwarded to B1, but as the *forward list* of q2 in B1 ends up with an empty set, B1 does not propagate it to the other neighbors. Another subscription **/a/b/c** is issued at B3 in Figure 5(c); again it is only forwarded to B1 without further propagation.

Deleting a subscription. When deleting a subscription, the NFA is traversed to locate the accepting state of the subscription and the subscription is deleted from the subscription list of the accepting state. After that, if the accepting state has no outgoing transition and the subscription list is empty, all the states from the starting state to the accepting state that are not shared by any other subscriptions are removed from the NFA, and for all the brokers in the forward List of the accepting state, the deletion is propagated. Figure 5(d) shows an example when C3 unsubscribes q2 after Figure 5(c). q2 is deleted from B4's NFA and the deletion is propagated to B1.

4.3.2. An improved algorithm

The above subscription routing strategy works well when there is a lot of queries satisfying structure_prefix relation. However, the algorithm will potentially forward any incoming subscription when few queries have structure_prefix relations. There is an important observation that although two XPath queries may not satisfy structure_prefix relation, they may share a common prefix at a relatively higher

probability. For example **/a/b/c/e/f** and **/a/b/c/f/e** do not satisfy the structure_prefix relation, but they do share the common prefix **/a/b/c**. Based on this observation, an improved subscription routing algorithm is developed as follows.

A global parameter called *InterestedDepth* is defined to indicate the maximum depth of an XPath query we are interested in. (See Figure 6) If adding a subscription doesn't change the first *InterestedDepth* of the NFA structure, the subscription is not forwarded. Otherwise, a modified subscription is forwarded. The modified subscription is achieved by discarding any value-based predicates and extracting the first *InterestedDepth* portion (the first *InterestedDepth* location steps of the XPath query) of the subscription. For example, if the *InterestedDepth* is 3, we forward **/a/b/c** instead of the original subscription **/a/b/c/e/f**. So, actually the modified subscription is a structure_prefix of the original subscription, which matches any document satisfying the original subscription. When deleting a subscription, we delete the subscription form the subscription list of corresponding accepting state. The InterestedDepth can be tuned to obtain a desired performance. The effect of InterestedDepth on performance is discussed in Section 6.

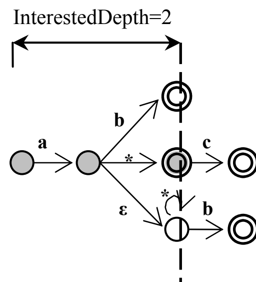


Figure 6. InterestedDepth

Our basic subscription routing algorithm, which exploits the structure_prefix relation among subscriptions, resembles the covering-based routing in Siena and Rebeca; whereas the improved algorithm, in which only common prefixes for multiple subscriptions are forwarded, is similar to the merging-based routing in Rebeca.

4.4. Publication filtering and routing

In YAK, publication filtering and routing are implemented in one algorithm. In the subscription routing described above, forwarded subscriptions are also combined in the NFA, thus the filtering algorithm automatically decides which neighbor brokers should receive the incoming documents. When a published XML document comes, the broker parses it and traverses it against the NFA using a stack-based approach as described in [7]. When an accepting state is reached, the subscriptions in the subscription list of that accepting state are added to the result list. After the structure matching, a list of subscriptions is obtained. The entire document is forwarded to the neighbor brokers who have subscriptions in the list. Note that if the document is from one neighbor to the current broker, the document should not be forwarded to this neighbor again. For the subscriptions from local clients in this list, value-based predicates and nested paths are further evaluated against the document. The incoming document is only sent to those subscribers that satisfy all the predicates in the subscriptions. In some sense, publication routing is a pre-processing of the publication filtering for the descendants along the spanning tree. It decides in a coarse granularity whether the descendent brokers have potential interested subscribers for an XML document.

For example, assume that a document that has the structure **<a><c></c>** is published at B2 in Figure 5(e). The publication matches a query from B1 and is delivered to B1. It is then checked at B1 and delivered to C1, B3 and B4. At B3 it is decided to be sent to C2, but at B4 it doesn't match any subscription (although it matches the structure of C3, it doesn't contain the attribute d).

5. Communication infrastructure

Communication protocols determine the message flow and type of connection between different constituents of the YAK system. There are three communication protocols:

-Broker – client communication protocol
-Broker - broker communication protocol
-Broker - RV server communication protocol

5.1. Broker – client communication protocol

The broker-client protocol is built on top of TCP. A client must first register with a broker before submitting a subscription or a publication. A global unique ID, consisting of the broker ID and a sequence number, is assigned to the client, if the registration succeeds. Later, this global ID is used by the client to submit a subscription or a publication. Matched publications are forwarded to the subscribers by their respective access point brokers. When a subscription is submitted, a listening thread

is spawned in the middleware at the subscriber side to handle publication notification. After receiving a publication notification, the YAK middleware delivers the publication to the application through callback.

5.2. Broker – broker communication protocol

The broker-broker protocol is also built on top of TCP, to ensure reliability of communication between brokers. As is mentioned before, the interconnection between brokers is a general cyclic graphical topology. So, to avoid repeated message deliveries, a global spanning tree is deduced from the topology by the PRIM algorithm. A point-to-point communication connection is established between each pair of neighbors according to the global spanning tree. When the global spanning tree is changed, new communication connections between brokers must be established accordingly.

5.3. Broker – RV server communication protocol

The broker-RV server protocol is essentially a group multicast protocol. All the brokers and the RV server join a group and the RV server acts as the coordinator for maintaining the global spanning tree and the group membership. The first startup broker in the system is designated as the RV Server. When a broker joins the group, the RV server assigns a unique ID to the broker. To improve fault tolerance of the YAK system, the RV server periodically multicasts heartbeat messages to brokers. If a heartbeat message is not answered for a period of time, the corresponding broker is judged as a crash failure. Whenever the group membership changes, due to a broker joining, or voluntarily leaving the group, or broker failure, the RV server establishes a new spanning tree and a new view of the group membership and then multicasts them to the group of brokers. After the new network topology is established, the NFA indexes are rebuilt on the new global spanning tree. The RV server can crash too. In the case of RV server crash failure, an election algorithm must be adopted in order to elect a new RV server. Right now, we only handle the broker crash failure; the handling of RV server crash failure is left as a future work.

6. Performance evaluations

In this section, we examine the performance of YAK through performance tests. As the filtering algorithm has been evaluated thoroughly in [7], in this paper, we focus on the evaluation of the routing algorithms.

6.1. Experimental topology

In order to simulate the YAK model in large wide-area networks, we use the Georgia Tech Internetwork Topology Models [12] to generate the network topology. More specifically, the transit-stub topology model is made use of, as it generates topologies that reflect the typical properties of real internetworks. We conducted the performance tests on several overlay network topologies and the conclusion remains unchanged. So, in this paper, we base our tests on one topology. Here is how we establish the YAK overlay network. We first generate a network topology with 4400 nodes using the transit-stub model. Then we arbitrarily choose 20 nodes out of them as the brokers, and choose another 200 nodes as the clients. After that, each client is assigned to its closest broker. To simplify our topology, we let each broker have equal number of clients. Therefore, when a client's nearest broker already has 10 clients, this client is assigned to the next nearest broker; if the next nearest broker is also full, it is assigned to the third nearest neighbor, etc. Practically, we utilize a heap to store the links between clients and brokers. Each time we pop up the shortest link from the heap, and decide whether the corresponding client can be assigned to the broker linked to, according to whether the broker has already enough clients. In this way, we can ensure the locality of clients, which is a fundamental hypothesis in our model. Figure 7 shows the global spanning tree of the experimental broker network.

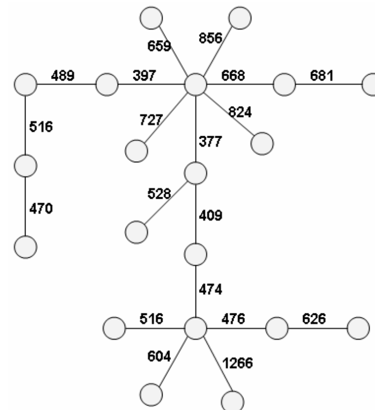


Figure 7. The global minimum spanning tree of the experimental broker network

6.2. Workload generation

In order to test the functionality and efficiency of YAK, we need to generate various workloads and see how our system works under these workloads.

As publications are expressed as XML documents in YAK, we need a generator to generate arbitrary documents to test our system. Without any existing tools that satisfy our needs, we implemented our own XML Document Generator. The generator takes a DTD (Document Type Definition, which specifies the format of XML documents) as one input and can generate a specified number of arbitrary XML documents that conform to the DTD.

User subscriptions are expressed as XPath queries in YAK, therefore we need a tool to generate arbitrary number of XPath queries. We utilize the YFilter Path Generator and adapt it to meet our needs. Given a DTD, the YFiler Path Generator can generate a number of XPath queries that comply with the structure defined in the DTD.

6.3. Performance metrics

As we are interested in evaluating the efficiency of routing algorithm, the network resource usage is a good metric. The network resource usage can be defined as the summation of network link costs consumed in routing a subscription or a publication among the broker network, like in [18]. If we assume the link weight generated by the transit-stub model is the link latency, we can use this value as the cost measure. For example, when a publication is routed along part of the minimum spanning tree, we use the summation of link latency along the subtree as the network resource usage for this subscription.

6.4. Simulation

The performance tests are conducted through simulation in a centralized machine. In all the tests, we hold the following assumptions. Clients for each broker submit equal number of subscriptions and equal number of publications. And clients for each broker submit subscriptions or publications in a round-robin way. There are 10 types of interesting objects (10 DTDs), which subscribed queries and published documents may conform to. But in order to simulate the workload in real applications, we also create subscriptions and publications that conform to DTDs beyond the 10 types. Some of these DTDs are similar to the 10 types, so that there are subscriptions interested in similar but slightly different objects and publications that look like but are not exactly what the clients want. We create subscriptions and publications that conform to DTDs completely distinct from the 10 types as well, to simulate the cases of odd user interests and unpopular publications. We can control the proportion of these kinds of subscriptions and publications to change the subscription match ratio in the system. In the following experiments, the maximum depth (number of location steps) of a query is 10 and average depth is about 4.4. For each location step, the wildcard probability is 0.1 and "//" probability is 0.1. The path generator uses uniform distribution for choosing child elements. The average number of value-based predicates per query is 1, and for simplicity, no query contains nested paths. For the publications, the average size is about 14.4KB, and maximum depth is 13 while the average depth is 7.1. Publications of different types conform to uniform distribution.

6.5. Test results

In this section, we conduct 3 tests to evaluate the YAK performance. In the first test, we show the scalability of YAK model when the number of subscriptions increases. The effect of InterestedDepth on subscription routing and publication routing costs are explored in test 2. At last, we examine the impact of subscription selectivity and regionalism on publication routing cost, like in [19]. The selectivity of subscriptions captures the match ratio of subscriptions, i.e. the ratio of the number of matched subscriptions to the number of total subscriptions. The higher the selectivity is, the smaller the match ratio is. The subscription regionalism describes the correlation between a client's location and its interests. In a system with high regionalism, clients for one broker tend to be interested in a subset of all types of interesting objects.

6.5.1. Scalability in terms of the number of subscriptions. This test shows how YAK scales as the number of subscriptions increases. We assume that there is no correlation between the location of a client and its interests, i.e. each client is interested in all the 10 types of the objects. Here, we use the average network resource usage per subscription as the metric. In the worst case, in which every subscription is forwarded along the whole spanning tree, the average network resource usage per subscription is the summation of the link latency along the global spanning tree, which is also the cost for the simple subscription routing in Gryphon. Therefore, this value serves as a baseline for

comparison. As shown in Figure 8, as the number of subscriptions increases, the average network resource usage decrease dramatically, especially at the beginning, then the curves shows more steady decrease afterwards. This result is consistent with our expectation. As more subscriptions are submitted, the chances that the later submitted subscriptions have the similar structure as the former ones increases, therefore, fewer subscriptions are needed to be forwarded. This figure also shows the effect of different InterestedDepth on the subscription routing cost. Clearly, the smaller InterestedDepth is, the more chances a later subscription shares the first interestedDepth structure with the former subscriptions, hence fewer subscriptions are forwarded. In the next section, we will further explore the effect of interestedDepth.

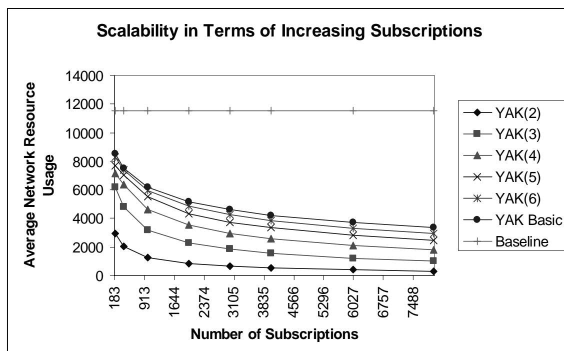


Figure 8. Scalability of YAK model in terms of increasing number of subscriptions (YAK(i) means YAK model with the InterestedDepth to be i)

6.5.2. Effect of InterestedDepth. This experiment will show the effect of InterestedDepth on the subscription routing and publication routing performance. In the previous experiment, the effect of the different interestedDepth on the subscription routing is shown. In this section, we will further thoroughly explore the impact of InterestedDepth. As in the last experiment, we assume that there is neither subscription regionalism nor publication regionalism. In this experiment, the average subscription match ratio is 6.97%, meaning there are 6.97% subscriptions match one publication on average. In Figure 9, each client submits 10 subscriptions. As the InterestedDepth grows, subscription routing cost decreases dramatically. This is consistent with the conclusion in the previous test. Now that broker contains 100 local subscriptions, and let each client of each broker submits 10 publications. Figure 10 demonstrates that as the interestedDepth decreases, the average publication cost increases. This is not surprising, as the smaller the interestedDepth is, the less accurate the publication routing is and more unrelated similar publications are forwarded.

Through the above experiments, we get the following conclusion that smaller InterestedDepth is beneficial for subscription routing while it increases the cost of publication routing. Then what is a reasonable value of InterestedDepth? This answer depends on the specific application. First, the InterestedDepth is highly dependent on the average subscription length and maximum subscription length. Besides, subscription submission rate and publication submission rate also affect the InterestedDepth. For an application, in which subscription submission rate is much higher than publication submission rate, a smaller InterestedDepth may be preferred, and vice versa. At last, the feature of subscriptions and publications can influence the InterestedDepth as well. In an application, if subscriptions or publications are highly diverse, i.e. share very little common structure, then the InterestedDepth has little effect on performance. Whereas, in a system with high subscription regionalism or publication regionalism, i.e. clients for a broker tend to have similar subscriptions or similar publications, the different choices of interestedDepth deserve more concern. In fact, the latter applications are more common, and are what we target at.

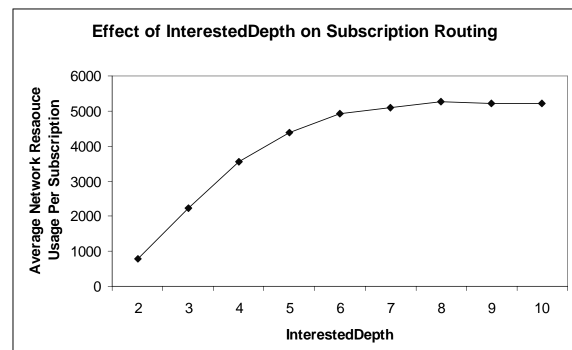


Figure 9. Effect of InterestedDepth on subscription routing (As the maximum subscription depth is 10, so InterestedDepth=10 actually means the basic YAK subscription routing)

6.5.3. Effect of subscription selectivity and regionalism. In the following 2 experiments, each broker contains 100 local subscriptions, and each client submits 10 publications. We also fix the InterestedDepth to be 4 in the 2 tests.

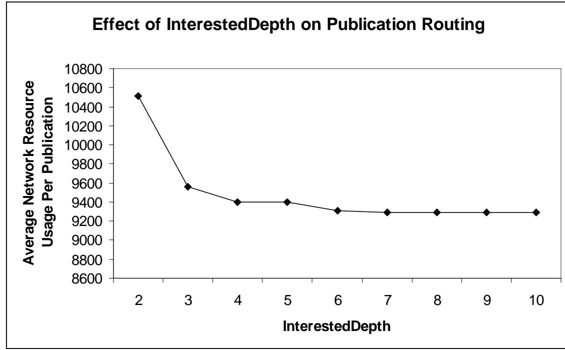


Figure 10. Effect of InterestedDepth on publication routing

In order to show the scalability of YAK, we compare the YAK model with flooding model, in which brokers are also organized in an overlay network, but there is neither subscription routing nor publication routing. Each broker organizes the subscriptions from local clients in an NFA-based index. When a publication is submitted to one broker, this broker simply floods it to the other brokers along the global spanning tree, and then each broker matches the publication against the local subscriptions. So, it is not hard to see that the average network resource usage per publication is summation of like cost along the spanning tree, which we use as a baseline.

We first evaluate the effect of selectivity of subscriptions on publication routing cost. In this experiment, there is no subscription regionalism. Figure 11 shows that the publication routing cost decreases as the subscription selectivity increases. And when the match ratio falls bellow 3% in our case, the YAK model only consumes half of the network cost of that in flooding model.

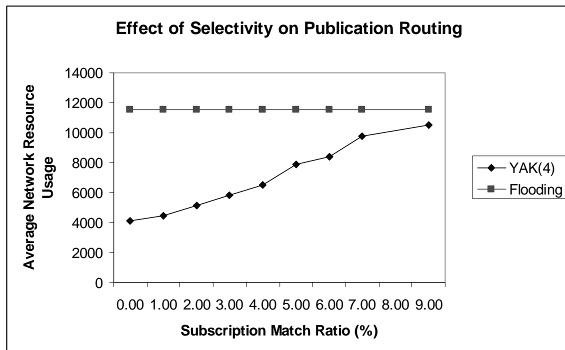


Figure 11. Effect of subscription selectivity on publication routing

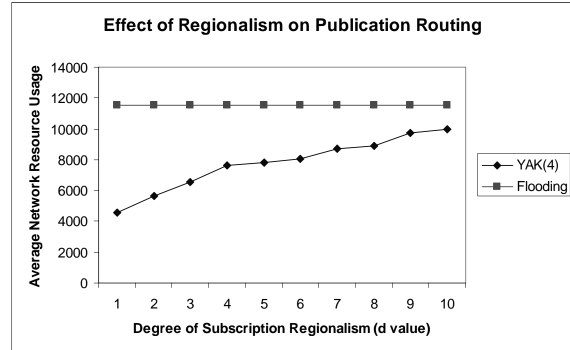


Figure 12. Effect of subscription regionalism on publication routing

Now, we evaluate the effect of subscription regionalism on publication routing cost. In order to control subscription regionalism, we let clients for each broker only interested in a subset of the 10 types of objects. To simplify our experiment, each broker contains subscriptions of equal number of types. Let the number be denoted as d. For example, if d=3, then each broker contains 3 types of subscriptions, although the specific types is different from broker to broker. We randomly choose the types for each broker. To describe it in detail, let T be the set of 10 types and t_i be type i where $0<i\leq 10$. Then we define $T_d = \{ \{t_{i1}, t_{i2},\ldots, t_{id}\} \mid t_{ij} \in T \text{ and } 0<j\leq d \}$, actually T_d is the set containing all the subsets of T which have d elements. It is not hard to derive that $|T_d|=C(10,d)$. So, in our example, we will choose one element from T_d for each broker, we allow replication in the choosing, i.e. there is chance that different brokers have same set of types. In the following experiment, we fix the selectivity of subscriptions to be 8.6%, and change the value of d to control the degree of regionalism. The result is shown in Figure 12. As can be seen, increasing the degree of regionalism (decreasing d), the publication routing cost decreases gradually.

Through the above experiments, we can easily conclude that YAK works very well when subsections are selective and have regionalism. In fact, most real application have the above 2 features. Our result is consistent with the conclusions in [19]. The work in [20] also evaluates the impact of advertisements routing on the cost of publication routing. Advertisements are filters submitted by publishers to indicate their intention of certain types of publications. Their conclusion is that the usage of advertisements can dramatically reduce the publication routing cost. Although, we haven't implement advertisement in YAK, the DTD is instinctive implementation of advertisement.

The above series of performance tests have shown the efficiency of the YAK publish-subscribe model and strongly suggest that we have achieved our goal of developing a high scalable publish-subscribe system.

7. Conclusion and future work

This paper proposes a scalable content-based publish-subscribe prototype – YAK. To achieve expressiveness of subscriptions and publications, XML document and XPath query are used as the publication model and subscription model respectively. YAK is a highly distributed model, in which the filtering and routing strategies exploit the locality of clients and dramatically reduce the communication costs. The simulations performed confirm our intuition about the scalability of YAK.

In the future we plan to extend YAK to support mobile and peer-to-peer environment. We also intend to implement the merging-based content routing and include advertisement in YAK. Besides, improving the fault tolerance of YAK and exploring security of communication are important follow-up work as well.

8. Acknowledgements

We thank Ahsan Vaqar Hundal and Kavita Sukerkar for their great effort on the design and implementation of communication infrastructure of YAK system. Thank Wenjie Wang on his help with the simulation tests.

9. References

[1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", at http://www.w3.org/TR/REC-xml, 1998.

[2] J. Clark, and S. Derose, "XML Path Language(XPath) version 1.0", at http://www.w3.org/TR/xpath, 1998.

[3] K. K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, and T.D Chandra, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems", *Proc. of 19th IEEE ICDCS Conf.*, Austin, TX, 1999.

[4] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", *ACM TOCS*, August, 2001.

[5] J.R Erenkrantz, "Handling Hierarchical Events In An Internet-Scale Event Service", at http://www.ucf.ics.uci.edu/~jerenk/siena-xml/SienaPaper.html, March 2001.

[6] M. Altinel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information", *Proc. of the 26th VLDB Conf.*, Cairo, Egypt, 2000.

[7] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer, "Path Sharing and Predicate Evaluation for High-Performance XML Filtering", *ACM TODS*, December, 2003.

[8] C. Chan, P. Felber, M. Garofalakis, and R. Rastogi, "Efficient Filtering of XML Documents with XPath Expressions", *Proc. of 18th ICDE Conf.*, San Jose, CA, 2002.

[9] J. Chen, D. Dewitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases", *Proc. ACM SIGMOD Conf.*, Dallas, TX, 2000.

[10] D. Carney, U. tintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Monitoring Streams - A New Class of Data Management Applications", *Proc. of 28th VLDB Conf.*, Hong Kong, China, 2002.

[11] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable Distributed Stream Processing", *Proc. of the 1st CIDR Conf.*, Asilomar, CA, 2003.

[12] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet Topology", *IEEE Communications Magazine*, 1997.

[13] L. Fiege, G. Mühl, and F. C. Gärtner, "Modular Event-based Systems", *The Knowledge Engineering Review*, 17(4), 2003.

[14] M. Caporuscio, A. Carzaniga, and A. L. Wolf, "Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications", *IEEE Transaction on Software Engineering*, 2003.

[15] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann, "A Peer-to-Peer Approach to Content-Based Publish/Subscribe", *Proc. of the 2nd DEBS*, San Diego, CA, 2003.

[16] G. Cugola and E. Di Nitto, "Using a Publish/Subscribe Middleware to Support Mobile Computing", *Proc. of the Workshop on Middleware for Mobile Computing*, Heidelberg, Germany, 2001.

[17] H. A. Jacobsen, "Middleware Services for Selective and Location-based Information Dissemination in Mobile Wireless Networks", *Proc. of the Workshop on Middleware for Mobile Computing*, Heidelberg, Germany, 2001.

[18] Y. H. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast", *ACM TOCS*, 2001.

[19] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Storm, and D. Sturman, "Exploiting IP Multicast in Content-based Publish-Subscribe System", *Proc. of Middleware 2000 Conf.*, Hudson River Valley, NY, 2000.

[20] G. Bricconi, E. D. Nitto, A. Fugetta, and E. Tracanella, "Analyzing the Behavior of Event Dispatching Systems", *Proc. of the 19th IEEE ICDCS*, Austin, TX, 1999.

[21] F. Tian, B. Reinwald, H. Pirahesh, T. Mayr, and J. Myllymaki, "Implementing A Scalable XML Publish/Subscribe System Using Relational Database Systems", *Proc. of ACM SIGMOD*, Paris, France, 2004.

[22] L. Fiege, F. Gärtner, O. Kasten, and A. Zeidler, "Supporting Mobility in Content-Based Publish/Subscribe Middleware", *Proc. of the 2nd ACM/IFIP/USENIX International Middleware Conf.*, Rio de Janeiro, Brazil, 2003.