Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables*

David Tam, Reza Azimi, and Hans-Arno Jacobsen

Department of Electrical and Computer Engineering,
University of Toronto, Toronto ON M5S 3G4, Canada
{tamda, azimi, jacobsen}@eecg.toronto.edu

Abstract. Building distributed content–based publish/subscribe systems has remained a challenge. Existing solutions typically use a relatively small set of trusted computers as brokers, which may lead to scalability concerns for large Internet–scale workloads. Moreover, since each broker maintains state for a large number of users, it may be difficult to tolerate faults at each broker. In this paper we propose an approach to building content–based publish/subscribe systems on top of distributed hash table (DHT) systems. DHT systems have been effectively used for scalable and fault–tolerant resource lookup in large peer–to–peer networks. Our approach provides predicate–based query semantics and supports constrained range queries. Experimental evaluation shows that our approach is scalable to thousands of brokers, although proper tuning is required.

1 Introduction

Publish/subscribe systems are becoming increasingly popular in building large distributed information systems. In such systems, subscribers specify their interests to the system using a set of subscriptions. Publishers submit new information into the system using a set of publications. Upon receiving a publication, the system searches for matching subscriptions and notifies the interested subscribers. Unlike the client/server model, the publish/subscribe model decouples time, space, and flow between publishers and subscribers, which may lead to benefits such as reduced program complexity and resource consumption.

There are at least two major classes of publish/subscribe systems: (i) topic–based and (ii) content–based. In topic–based systems, subscribers join a group containing a topic of interest. Publications that belong to the topic are broadcasted to all members of the group. Therefore, publishers and subscribers must explicitly specify the group they wish to join. Topic–based systems are similar to the earlier group communication and event–notification systems (e.g. in newsgroups).

* To appear in (i) the International Workshop on Databases, Information Systems and Peer–to–Peer Computing, September 7–8, 2003, Humboldt University, Berlin, Germany, and (ii) Lecture Notes in Computer Science, © Springer-Verlag.

In content–based publish/subscribe systems, the matching of subscriptions and publications is based on content and no prior knowledge is needed (e.g. the set of available topics). Therefore, these systems are more flexible and useful since subscribers can specify their interests more accurately using a set of predicates. The main challenge in building such systems is to develop an efficient matching algorithm that scales to millions of publications and subscriptions.

Publish/subscribe systems can be implemented centrally or in a distributed manner. Centralized systems have the advantage of retaining a global image of the system at all times, enabling intelligent optimizations during the matching process. Examples of intelligent matching algorithms can be found in [1], [2], [3], [4], and [5]. Major disadvantages of centralized systems are the lack of scalability and fault–tolerance. Distributed publish/subscribe systems have been introduced to address these problems [6] [7]. However, the main difficulty in building distributed content–based systems is the design of an efficient distributed matching algorithm. Existing distributed content–based systems such as [6] typically rely on a small number of trusted brokers that are inter–connected using a high–bandwidth network. In some scenarios, such configurations may not offer adequate scalability. As well, they do not provide a satisfactory level of fault–tolerance since crashing a single broker may result in a large number of state transfer operations during recovery.

Recently, distributed hash tables (DHTs) [8] [9] [10] [11] have emerged as an infrastructure for efficient, scalable resource lookup in large peer–to–peer distributed networks. Such systems are decentralized, scalable, and self–organizing (i.e. often as well, they automatically adapt to the arrival, departure and failure of nodes in the network). Such characteristics make DHTs attractive for building distributed applications. In fact, DHTs have successfully been used in several application domains, such as distributed file systems [12] [13] [14] [15]. It has also been shown that topic–based publish/subscribe systems can be built on top of DHTs [16]. Although there have been several attempts in building content–based publish/subscribe systems on top of peer–to–peer systems [17] [18], there remains much to be explored. Compared to our system, the systems described in [17] and [18] exploit the underlying DHT infrastructure to different degrees.

In this paper, we present a simple approach to building a distributed content–based publish/subscribe system on top of a DHT. More specifically, we use a topic–based system (Scribe) [16] that is implemented using a DHT (Pastry) [10]. In our approach, topics are automatically detected from the content of subscriptions and publications through the use of a *schema*, which is a set of guidelines for selecting topics. The schema is application–specific and can be provided by the application designer after some statistical analysis. Schemas are similar to database schemas used in RDBMS. With this approach we can significantly increase the expressiveness of subscriptions compared to purely topic–based systems. However, our scheme does not fully provide the query semantics of a traditional content–based system. Queries are not completely free–form but must adhere to a predefined template. Moreover, issues of fault–tolerance in subscrip-

tion storage have yet to be explored in our system, although fault–tolerance in DHT routing and multicast routing can be transparently handled by Pastry and Scribe, respectively. We implement our scheme on top of the existing DHT simulator included in Pastry.[1] Our evaluation shows that with a carefully designed schema, it is possible to achieve accurate, efficient and scalable matching.

The remainder of the paper is organized as follows. In Section 2, we review related work, including a brief overview of DHT systems. In Sections 3 and 4, we describe the key features of our design. In Section 5, the experimental platform and our results are presented and discussed. In Section 6, we conclude and suggest directions for future work.

2 Related Work

In a typical DHT system, each node has a unique identifier (*nodeId*). Also, each message can be associated with a *key* of the same type as the nodeIds. Keys and nodeIds are typically uniformly distributed. Given a message and its corresponding key, a DHT system routes the message to the node whose nodeId is numerically closest to the message key (*home node*). Given a uniform distribution of the nodeIds and keys, the routing task is evenly distributed among the nodes of the network. The message routing algorithm works based on the key and nodeId digits (in any base). Therefore, routing usually requires $O(logN)$ hops, where N is the total number of nodes in the network. In order to tolerate node failures or network disconnections, several methods are used to replicate messages to a set of neighboring nodes of the home node. Also, in order to increase locality, a proximity metric may be defined to reflect the latency and bandwidth of the connection between any pair of nodes. Such a proximity metric along with keys are used in finding the optimal route between nodes.

Scribe [16] is a topic–based publish/subscribe system (a.k.a. a multicast infrastructure) that is built on top of Pastry [10], a DHT system developed at Rice University. Subscribers join topics of interest, where each topic is identified with a Pastry–level key. Therefore, for each topic there is a Pastry node whose nodeId is numerically closest to the topic key (topic root). Publications are submitted to the corresponding topics. Each publication is then multicasted to all subscribers of the topic. Scribe is a simple, well–structured topic–based system. However, it does not support content–based subscriptions and publications.

SIENA [6] is a distributed content–based event notification system which can be used to implement a publish/subscribe system. Routing is performed in the overlay network based on the content of messages. However, SIENA is not based on a DHT networking substrate. Therefore, it cannot take advantage of the inherent scalability and fault–tolerance of such an infrastructure.

In content–based publish/subscribe systems, the handling of range queries is an important capability to possess. It allows for higher expressiveness and a more elegant interface to the system. A number of projects, as described below, have attempted to address this issue in peer–to–peer systems.

[1] We used FreePastry, which is an open–source implementation of Pastry.

Extensions to CAN [9] have been proposed and evaluated in [19] to enable range queries. The DHT hash function is modified to use a Hilbert space filling curve to map an attribute value to a location in the CAN key space. This curve enables proximity in the attribute value space to correspond to proximity in the key space. With such a configuration, routing and searching in the key space corresponds to routing and searching in the attribute value space. While the work addresses range queries of a single attribute, extending the technique to handle range queries of multiple attributes has yet to be addressed. It is not clear whether handling multiple attributes can be accomplished by simply applying the technique multiple times. As explained by Andrzejak et al., the extension of this technique to handle multiple attributes in a single enhanced DHT system is an interesting problem and remains to be completed.

SkipNet [20] offers an alternative and complimentary technology to DHTs. Using a distributed form of the skip lists data structure, SkipNet is able to control both routing and data placement in an overlay network. Harvey et al. briefly suggest that the use of skip lists enables SkipNet to inherit the ability to perform range queries in an efficient and flexible manner. The implementation and evaluation of range queries in SkipNet has yet to be reported. P–Grid [21] is yet another alternative to DHTs, which uses a distributed binary tree rather than a hash table.

Finally, in [22] Gupta et al. develop a peer–to–peer data sharing architecture for providing approximate answers to range queries by finding similar ranges. They use a locality–sensitive hashing scheme to co–locate similar ranges. Although the contribution of the work is important in providing DBMS–like query processing in peer–to–peer systems, for practical content–based publish/subscribe the hash function error rate of this approach may become too high. In [23], Sahin et al. extend CAN to approximately process basic range queries as well.

3 Design

Our approach bridges the gap between topic–based and content–based systems by automatically organizing the content into several topics. For each publication and subscription we build a set of topics for submission to a topic–based system. Automatically building such topics requires that the content provided by the user application follows certain rules and constraints. We call such rules and constraints the *schema*. Each application domain may have a different schema, and the system is capable of handling multiple domain schemas simultaneously.

3.1 Domain Schema

For each application domain, we define a schema to describe the general format of subscriptions and publications that belong to the domain. The idea of a domain schema is similar to a DBMS–style schema. The purpose of using a domain schema is to limit the number of possible combinations of messages that must

Table 1. An example of a schema table for "COMPUTERS" with three indices. The first index is useful to subscribers concerned mainly with price and visible quality. The second index is useful to subscribers concerned mainly with processing power. The third index is useful to subscribers concerned mainly with storage capacity combined with processing power

Order	Type/Unit	Name	Values	Index 1	Index 2	Index 3
1	USD	Price	1000 .. 5000	√		
2	String	CPU	PII, PIII, P4, Celeron		√	√
3	MHz	Clock	200 .. 3000		√	
4	Mbyte	RAM	64 .. 1024		√	
5	Gbyte	HDD	10.. 200			√
6	Inch	Monitor	14 .. 20	√		
7	String	CD	CDROM, CDRW, DVD			√
8	String	Quality	New, Used, Demo	√		

be generated and sent out in the DHT system. This technique enables us to feasibly transform a topic–based publish/subscribe system into a content–based system. The schema must be broadcasted to all nodes that are running relevant applications prior to the operation of such applications. Each domain is identified with a unique ID (name) so that the system can handle multiple application domains simultaneously. For instance, the publish/subscribe system may be used by a stock market and an auction network simultaneously.

Each schema consists of several tables, each with a standard name. For each table, we maintain information about a set of attributes, including their type, name, and constraints on possible values. We assume the table attributes are ordered. Also for each table, there is a set of *indices* that are used for the actual lookup in the network. Each index is an ordered collection of strategically selected attributes. Selecting the optimal set of attributes for indices is essential to achieving acceptable performance. Similar to indices in database systems, it is imperative to choose attributes that are more of a user's concern, and hence, more likely to be used by the users for lookup. Table 1 shows a simple example of a schema table. It contains three indices corresponding to users with various interests. Currently, we require application designers to intelligently specify domain schemas manually. Designers could be assisted by profiles from a centralized content–based system to identify important attributes and their values.

3.2 Basic System Operation

When a request (publication or subscription) is submitted to the system, it is inspected to extract several *index digests*. Each index digest is a string of characters that is formed by concatenating the attribute type, name, and value of each attribute in the index. For the example schema in Table 1, some possible topic digests would be [*USD : Price : 1000 : Inch: Monitor : 19 : String : Quality : Used*] and [*String : CPU : PIII : MHz : Clock : 650 : Mbyte : RAM : 512*]. In the

simple case, we assume that subscribers provide exact values for each attribute. Handling range queries is discussed in Section 3.5. The use of schemas is a key technique in significantly reducing the number of topic digests and corresponding messages. Without a schema, the above technique would require a digest for every possible combination of predicates in every subscription. The maximum number of such digests would be 2^N, where N is the number of predicates in the subscription. Such requirements would generate an extremely large number of messages and render the system infeasible.

A given subscription can be submitted to the system only if it specifies all attribute values for at least one of the indices in the corresponding schema table. In such a case, the composed index digest is translated to a DHT *hash key*. The subscription is then sent to the key's home node by the DHT system.[2] When a publication with attribute values that match the subscription is submitted to the system, the same hash key is generated and therefore the publication is delivered to the same node in the network. Such a matching is partial since only a subset of the subscription predicates are matched in this way. The key's home node is responsible for completing the matching process by comparing the publication with all submitted subscriptions to the node. A standard centralized matching algorithm can be used in the home node for this purpose. Nodes with subscriptions that completely match the publication are notified by the hash key home node.

3.3 Event Notification Multicast Tree

A problem with the basic scheme is that home nodes may become overloaded with the task of processing publications and subscriptions. An improvement to the scheme is to build a multicast tree structure to enable better distribution of these tasks. A multicast tree can be constructed for all nodes that have subscribed to a certain hash key. The *root* of the tree is the hash key's home node, and its branches are formed along the routes from the subscriber nodes to the root. Similar to the root node, the internal nodes in the tree may or may not have subscribed to the index key.

The Scribe multicast tree infrastructure is exploited by our system to achieve scalable performance. To implement efficient multicasting, Scribe builds a multicast tree rooted at each topic root node as follows. When a subscription finds its route to the topic root, all intermediate nodes along the route are added to the multicast notification tree unless they already belong to it. Under this particular scenario, routing stops upon encountering the edge of the corresponding multicast tree. This optimization enables the subscription operation to be efficient and completely distributed. The multicast tree structure significantly reduces the total number of messages generated by subscribers by allowing subscriptions to be submitted to lower levels of the tree, obviating the need to traverse the entire route to the root node. Moreover, the lower level nodes absorb most of the

[2] The home node itself may or may not have subscribed to this hash key.

work targeted at the root node and therefore prevent the root node from being a bottleneck in the matching process.

For multicasting, the publication is first sent to the topic root, from which the publication is sent all the way down the multicast tree. Such an infrastructure can significantly reduce (i) the number of messages generated by publishers and (ii) notification latency. Event notification is accomplished in $O(logN)$ time, where N is the number of nodes. Further details about the Scribe multicast infrastructure can be found in [16]. In Section 3.4 and Section 3.5 we describe how the Scribe multicast tree is exploited to improve the basic matching scheme.

3.4 Handling False Positives

Since the predicates of a subscription may be used in several indices, some of the publications that are sent to an index home node may not match all predicates of a subscription. In fact, they match only the subset of subscription predicates that were specified in the index. We use the term *miss* to refer to a received publication that partially matches a submitted subscription but does not fully match it. We use the term *hit* to refer to a received publication that fully matches a submitted subscription. To filter out the misses, exact matching is performed at the receiver node by using conventional matching algorithms. Since subscriptions and publications are uniformly distributed in the network, such a matching process occurs in a distributed manner and therefore scales well with the number of publications. Moreover, the use of multicast trees for event notification allows for further distribution of the matching process among the nodes of the multicast tree rather than concentrating it solely on the root node. For index values that are frequently used, the multicast tree grows proportionately to the number of users that use the index value, and therefore the matching load is automatically balanced. The impact of false positives is examined in our experiments and is illustrated by a variety of *hit rate* curves (see Fig. 1 and Fig. 2).

3.5 Range Queries and Disjunctive Forms

A potential complexity in content–based subscriptions is that the constraint on some attributes may be specified as a range rather than an exact value. A naive approach in handling such queries is to build hash keys only for indices that do not include range predicates. The disadvantage of this approach is two–fold. First, for some subscriptions, range predicates may be present in all possible indices so that no indices can be used for submission. This solution suggests dropping such subscriptions and hence, results in a less practical system. Second, this solution limits the index selection policy, and therefore might result in less accurate partial matching.

Another method of handling range predicates is to build a separate index hash key for every attribute value in the specified range. This solution is only adequate for attributes whose value domains are not large. For instance, in Table 1 there may be a handful of possible values for the *Monitor* attribute. However, for

other attributes such as *Price*, a value range might include thousands of possible values.

Our solution for these types of attributes is to divide the range of values into intervals and use interval indicators as values for building hash keys. For instance, for the RAM attribute, one may suggest that there are four possible intervals: *less than 128*, *128–256*, *256–512*, and *greater than 512*. Therefore, the predicate $RAM > 384\,Mbyte$ would belong to two keys: $RAM = 256–512$ and $RAM = greater\ than$ 512. Of course, the matching that occurs with this scheme is partial, and the actual complete matching must be performed at one of the nodes in the multicast tree. Given a probability distribution function for an attribute, it is possible to improve the division of value intervals by choosing interval boundaries so that the collective probability of each interval is equal. This means that for value subranges that are queried frequently, the intervals are finer–grained. For instance in Table 1, the density of values for the *Clock* attribute may be higher near main–stream processors frequencies (2.0 GHz – 2.5 GHz) than those of older processors (less than 1.0 GHz) or high performance processors (greater than 2.5 GHz).

By having a handful of values for each attribute, we may be able to confine the total number of hash keys for a subscription to the order of tens. As we will show in Section 5.4, it is possible to maintain acceptable performance with this solution since the cost of a subscription in a multicast tree–based system is relatively low.

Similar to range queries, disjunctive form subscriptions may be treated as a set of separate conjunctive subscriptions, where hash keys are generated for each separate subscription.

3.6 Duplicate Notifications

Another problem with the basic scheme is that it is possible that a publication is sent several times to a single subscriber if the subscriber has used several indices. One solution to this problem is for the subscriber to maintain a list of publications it has seen (given that each publication has a unique identifier), and discard duplicate publication messages. Such a list may be relatively small, since the average propagation time for a publication is small. Therefore, it is expected that publication duplicates are received by the subscriber within a relatively short interval. Thus, the subscriber may be able to set a short deadline for each publication and purge its copy from the list after the deadline.

A better solution is to prevent a subscriber from submitting more than one index digest, rather than to allow it to submit an index digest for every applicable index. In this case, the subscriber must submit the index digest corresponding to the index that is the most important. This solution would only work if publications specify values for all attributes, which we believe will be the common case since publishers are information sources. If a publication does not specify a value for an attribute in an index, the publication would be missed by subscribers that used only that index. In our experiments, publishers specify values for all attributes.

4 Implementation

We have implemented our approach on top of Scribe/Pastry. Our choice of platform was not fundamental to our design and we could have built our system on top of any DHT system. We found it advantageous to exploit the multicast tree infrastructure of Scribe, although it could have been manually implemented if it did not exist. The Scribe/Pastry platform is implemented in Java. It simulates a network of nodes with arbitrary size. Networks are associated with a proximity metric to improve locality. However, such optimizations are transparent to our system. The platform allows us to bind a Java object to each node as the *application*. The object must implement an interface that provides at least two functions: *receiveHandler()* and *forwardHandler()*. The former is called by the Pastry system when a message is delivered to a node, whereas the latter is called when a node is located along a route and is forwarding a message to some other node. With this programmability, we have been able to embed the functionality of a content–based publish/subscribe system in the application object. Moreover, we instrumented the calls to these methods to collect statistics and understand the behavior of the system.

5 Evaluation

As a preliminary DHT content–based publish/subscribe system, the main performance goal is scalability. We want to ensure that as the size of the system grows, performance remains relatively reasonable. In this section, we evaluate the feasibility of the system.

The metric we used was the number of messages exchanged in the system. This metric is a reasonable, first–order measurement of the system. Since distributed systems deal with communication and computation of physically separate nodes, measuring traffic characteristics is important. Detailed metrics concerning communication characteristics, such as latency and bandwidth consumption have yet to be gathered. This area is a subject of future work.

Measuring the number of messages offers a first–order evaluation of the system performance. Therefore, we examine the impact of changes in various system parameters on the number of messages. Such parameters include the number of publications and subscriptions, the number of nodes in the system, and the number of range predicates in a subscription.

5.1 Experimental Setup

Pastry/Scribe includes a network simulator, enabling us to simulate 1000s of nodes within one physical computer. We used the ToPSS [2] workload generator to generate synthetic workloads.

The workload followed the schema table described in Table 2. This schema table defines possible values of the attributes. For instance, *Price* can have a value of {*1, 2, 3, 4, 5*}. Although some of the value ranges seem inappropriate,

Table 2. Workload schema table

Order	Type	Name	Values	Index 1	Index 2	Index 3	Index 4
1	Integer	Price	1 to 5	√	√	√	√
2	Integer	Volume	1 to 5	√	√	√	√
3	Integer	Color	1 to 5	√	√	√	√
4	Integer	Size	1 to 5	√	√	√	
5	Integer	Temperature	1 to 2	√	√		
6	Integer	Circumference	1 to 2	√			

we used them for simplicity. In reality, the DHT is not concerned with whether the possible values of *Color* are {*red, green, blue, cyan, magenta*} or whether the possible values are {*1, 2, 3, 4, 5*}. These values are used as input into a hash function that generates a fixed–size key. For any particular attribute, integer values are randomly generated by the workload generator with a uniform distribution within the specified range. For this particular workload, there are $5 \times 5 \times 5 \times 5 \times 2 \times 2 = 2500$ unique hash key values.

We recognize that this workload does not necessarily represent a real–world application. However, it provides a simple example of what a fairly typical subscription or publication may look like. Results produced by this workload can be easily understood and analyzed. Moreover, it is difficult to find a real–world content–based publish/subscribe system from which we could obtain workload traces.

Although the chosen schema size in Table 2 is small, an examination of Table 1 may help predict the impact of schema growth. For example, if the number of attributes in Table 1 were to increase dramatically, we would expect only a moderate increase in the number of indices in the application domain. This pattern appears plausible due to the inherent nature of indices. Since an index is formed for a set of commonly used attributes, the number of indices is largely independent of the number of attributes available. Therefore, schema size should have limited impact on performance.

As for the size of value ranges (the number of intervals for a given attribute), this parameter may also have a limited impact on performance. A more important characteristic is the distribution of range queries across the intervals. For example, in a particular application domain, range queries on a particular attribute may often intersect only two intervals, regardless of the number of available intervals. In this case, only two corresponding hash keys are submitted. Our experiments can be viewed as examining the performance impact of up to five intervals of intersection.

5.2 Event Scalability

Figure 1 shows the scalability of the system in terms of the number of events. That is, the impact of the number of subscriptions and publications on the number of messages exchanged. The number of nodes was fixed at 1000 nodes. On the

x–axis of the graph, 10000 subscriptions and publications refers to 10000 random subscriptions followed by 10000 random publications.[3] We used an equal number of subscriptions and publications for simplicity. We also show how various hit rates affected the number of messages generated. Hit rate is the proportion of delivered publications that fully match the subscriptions maintained by a node.

To enforce a 100 % hit rate, the schema index consisted of all attributes in Table 2 (Index 1). To enforce other hit rates, we exploit the fact that the workload generator randomly chooses attribute values in a uniformly distributed manner. To attain the 50 % hit rate, the schema index consisted of the first 5 attributes (Index 2). To attain the 25 % hit rate, the schema index consisted of the first 4 attributes (Index 3). Finally, to attain the 5 % hit rate, the schema index consisted of the first 3 attributes (Index 4).

The first observation from Fig. 1 is that message traffic for any particular hit rate grows approximately quadratically with the number of subscriptions and publications. This growth is reasonable since both subscriptions and publications are increased simultaneously.

The second observation is on the incremental impact of the various hit rates shown by the separate curves in the graph. One might expect that a 50 % hit rate means that a node receives twice as many publication messages than were really destined for the node. Similarly, at a 25 % hit rate four times as many publication messages would have been generated compared to the 100 % hit rate configuration. Interestingly, the results showed lower than expected increases.

Our experiments indicated that 72 % of the total messages shown in the y-axis of Fig. 1 were publication messages. This composition is for 40000 subscriptions and publications, with a 100 % hit rate. Therefore, at a hit rate of 50 %, we expected a 72 % increase in total messages but observed only a 46 % increase. At a 25 % hit rate, we expected an increase of $(72 + 72 + 72) = 216$ % in total messages but observed only a 123 % increase.

These lower than expected increases were due to the use of multicast notification trees. When publication notifications are triggered, only a few additional messages are needed to reach an additional subscribing node. This phenomenon leads to the smaller observed increase in message traffic between the various hit rate curves.

A low hit rate, such as the 5 % curve, shows relatively steep traffic increase compared to higher hit rates, such as 25 %, 50 %, or 100 %. These results suggest that it is important to have a well–designed schema such that the indices used incur hit rates greater than 25 %. Other complementary techniques, such as subscription summaries [7], may be applicable in reducing message size.

In our current implementation, exact matching of publications to subscriptions is done at the leaves of the multicast notification trees. The results in Fig. 1 could be further improved by implementing exact matching higher up in the multicast tree. However, such reductions in message traffic must be carefully balanced against increased processor load on the nodes performing the exact

[3] There is no correlation between subscriptions and publications. That is, there is no guarantee that every subscription will be matched by at least one publication.

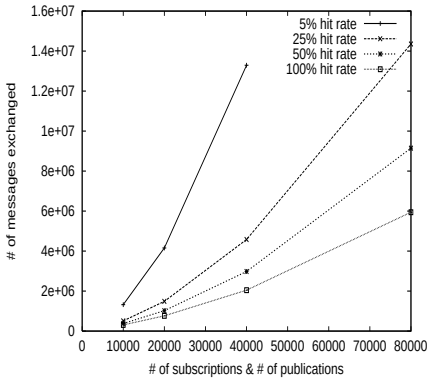

Fig. 1. Scalability in the number of events. 1000 nodes

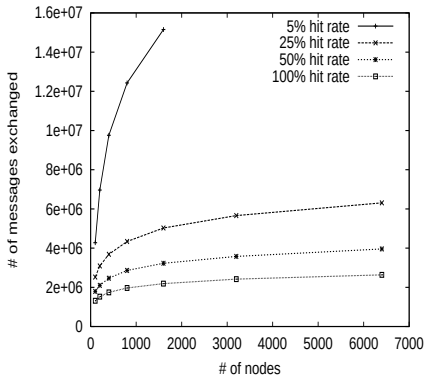

Fig. 2. Scalability in the number of nodes. 40000 subscriptions, 40000 publications

matching. In summary, there is a trade–off between message traffic and node processor workload.

5.3 Node Scalability

Figure 2 shows the scalability of the system in terms of the number of nodes. The system used a fixed number of events consisting of 40000 random subscriptions followed by 40000 random publications. The value 40000 was arbitrarily chosen since it represented a middle point in the range of values used in Fig. 1. The results show that the system scales well for most hit rates (except for perhaps the 5 % hit rate). Again, these results further suggest that is it important to have a well–designed schema such that the indices used incur hit rates greater than 25 %.

5.4 Impact of Range Queries

Figure 3 shows the impact of range queries. In order to isolate this impact, we use Index 1 (Table 2). The "0 range" curve represents a workload without any range queries. For the "1 range" curve, the first attribute (*Price*) is allowed to specify a range. It may have specifications such as $Price = x$, or $Price < y$, or $Price > z$. The choice among $\{<,=,>\}$ is chosen randomly using a uniform distribution. All other attributes specify exact values. Similarly, for the "2 range" curve, the first and second attributes (*Price* and *Volume*) are allowed to specify ranges.

These results suggest that range queries have a moderate and acceptable impact on the system. In the "1 range" curve, the *Price* attribute has 5 possible values. For each range query, an average of 3 index digests are submitted, given a uniformly random distribution. The selection of the *operator* $\{<,=,>\}$ is also

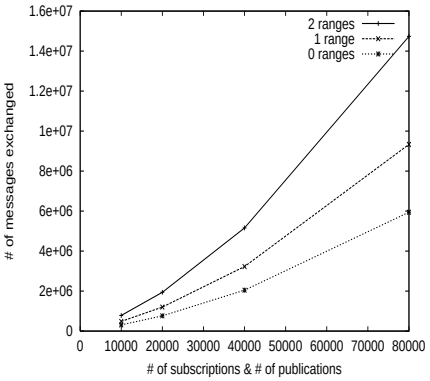

Fig. 3. Impact of range queries. 1000 nodes

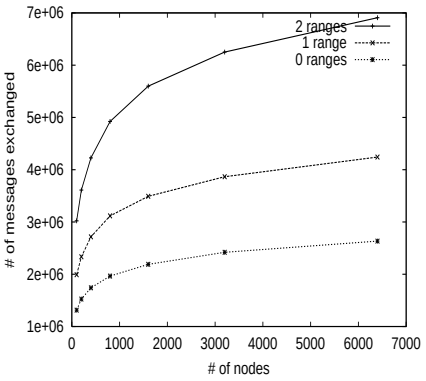

Fig. 4. Scalability in the number of nodes with range queries. 40000 subscriptions, 40000 publications

uniformly distributed. Given that $P(o)$ is the probability of a particular operator o, and $E(o)$ is the expected number of index digest submissions for that operator, then the expected number of index digest submissions per subscription is

$$\begin{array}{lcl} E(s) = P(<)E(<) + P(=)E(=) + P(>)E(>) & & \\ E(s) = \quad \frac{1}{3} \times 3 \quad + \quad \frac{1}{3} \times 1 \quad + \quad \frac{1}{3} \times 3 & & (1) \\ E(s) = \quad 2.33 \ . & & \end{array}$$

Similarly, we expected publications to require 2.33 times as many messages since there are roughly 2.33 times as many interested subscribers. Due to these expected increases in both the subscription and publication components, we therefore expected 2.33 times as many total messages as the "0 range" curve. In contrast, the results show only 1.6 times as many messages (at 40000 subscriptions and publications). Similarly for the "2 range" curve, we expected 5.43 times as many messages as the "0 range" curve. In contrast, the results show only 2.5 times as many messages. These beneficial results are due to the use of multicast notification trees. Subscriptions are registered efficiently and additional publication notifications incur low incremental costs.

Figure 4 shows the impact of range queries on node scalability. Similar to Section 5.3, the system used a fixed number of events consisting of 40000 random subscriptions followed by 40000 random publications. The results show that the system scales well while supporting range queries.

6 Conclusion and Future Work

We have developed a technique to implement a content–based distributed peer–to–peer DHT–based publish/subscribe system on top of an existing topic–based system. We found that the use of a multicast tree infrastructure was critical

to achieving good performance. Our design offers an interesting and unexplored point in the design space of publish/subscribe systems. Our design point exists somewhere between a fully centralized content–based system and a fully distributed topic–based system. As a compromise, our system is a fully distributed, content–based system with some restrictions on the expression of content. The content must follow and fit within a well–defined schema for that particular application domain.

Since this content–based publish/subscribe system is an early prototype, there is plenty of future work to be done. Our next step is to perform a more detailed examination of the benefits of the multicast tree infrastructure. Other tasks include (i) adding more features to enable execution of real–world workloads, (ii) performing detailed modeling of the peer–to–peer network, and (iii) examining fault–tolerance. This paper represents on–going research conducted under the p2p–ToPSS project (peer–to–peer–based Toronto Publish/Subscribe System).

We recognize that an endless number of experiments can be run, with many possible combinations of workload parameters. In particular, using locality–sensitive distributions in the workload generator, rather than a uniform distribution, should produce an interesting set of comparative results. However, to achieve some level of workload validity, guidelines on how these parameters vary in relation to each other need to be researched. We are currently not aware of any well–accepted, standard set of workloads for content–based publish/subscribe systems.

Guidelines for good schema design for an application domain remain an important open research question. Determining the optimal set of indices for a particular application domain may require intimate knowledge of the application domain, such as the typical query behavior of users. Achieving an optimal index may require selecting attributes that are commonly specified but can uniquely identify a subscription.

References

1. Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., Shasha, D.: Filtering algorithms and implementation for very fast publish/subscribe systems. ACM SIGMOD Record **30** (2001) 115–126
2. Ashayer, G., Leung, H.K.Y., Jacobsen, H.A.: Predicate matching and subscription matching in publish/subscribe systems. In: Proc. of Workshop on Distributed Event-Based Systems (DEBS), Vienna, Austria (2002) 539–546
3. Petrovic, M., Burcea, I., Jacobsen, H.A.: S-ToPSS: Semantic Toronto publish/subscribe system. In: Proc. of Conf. on Very Large Data Bases, Berlin, Germany (2003) 1101–1104
4. Liu, H., Jacobsen, H.A.: Modeling uncertainties in publish/subscribe. In: Conf. on Data Engineering (to appear). (2004)
5. Burcea, I., Muthusamy, V., Petrovic, M., Jacobsen, H.A., de Lara, E.: Disconnected operations in publish/subscribe. In: IEEE Mobile Data Management (to appear). (2004)

6. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Achieving scalability and expressiveness in an Internet-scale event notification service. In: Proc. of ACM Symp. on Principles of Distributed Computing (PODC), Portland, OR (2000) 219–227
7. Triantafillou, P., Economides, A.: Subscription summaries for scalability and efficiency in publish/subscribe. In: Proc. of Workshop on Distributed Event-Based Systems, Vienna, Austria (2002) 619–624
8. Kaashoek, F.: Distributed hash tables: Building large-scale, robust distributed applications. Presentation: ACM Symp. on PODC (2002)
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content–addressable network. In: Proc. of ACM SIGCOMM, San Diego, CA (2001) 161–172
10. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Proc. of IFIP/ACM Conf. on Distributed Systems Platforms, Heidelberg, Germany (2001) 329–350
11. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. of ACM SIGCOMM, San Diego, CA (2001) 149–160
12. Adya, A., Bolosky, W.J., Castro, M., Cermak, G., Chaiken, R., Douceur, J.R., Howell, J., Lorch, J.R., Theimer, M., Wattenhofer, R.P.: FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In: Proc. of USENIX Symp. on Operating Systems Design and Implementation (OSDI), Boston, MA (2002) 1–14
13. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: Proc. of ACM Symp. on Operating Systems Principles (SOSP), Banff, Canada (2001) 202–215
14. Muthitacharoen, A., Morris, R., Gil, T.M., Chen, B.: Ivy: A read/write peer-to-peer file system. In: Proc. of USENIX Symp. on OSDI, Boston, MA (2002) 31–44
15. Rowstron, A., Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: Proc. of ACM SOSP, Banff, Canada (2001) 188–201
16. Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. In: IEEE Journal on Selected Areas in Communication. Volume 20. (2002) 1489–1499
17. Pietzuch, P.R., Bacon, J.: Peer-to-peer overlay broker networks in an event-based middleware. In: Proc. of Workshop on DEBS, San Diego, CA (2003)
18. Terpstra, W.W., Behnel, S., Fiege, L., Zeidler, A., Buchmann, A.P.: A peer-to-peer approach to content-based publish/subscribe. In: Proc. of Workshop on DEBS, San Diego, CA (2003)
19. Andrzejak, A., Xu, Z.: Scalable, efficient range queries for grid information services. In: Proc. of IEEE Conf. on Peer-to-Peer Computing, Linköping, Sweden (2002) 33–40
20. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: A scalable overlay network with practical locality properties. In: Proc. of USENIX Symp. on Internet Technologies and Systems, Seattle, WA (2003)
21. Aberer, K., Hauswirth, M., Punceva, M., Schmidt, R.: Improving data access in P2P systems. IEEE Internet Computing **6** (2002) 58–67
22. Gupta, A., Agrawal, D., El Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: Proc. of Conf. on Innovative Data Systems Research, Asilomar, CA (2003)
23. Sahin, O.D., Gupta, A., Agrawal, D., El Abbadi, A.: Query processing over peer–to–peer data sharing systems. Technical Report UCSB/CSD-2002-28, University of California at Santa Barbara, Department of Computer Science (2002)