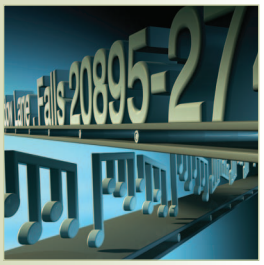

Architectural Alternatives for Information Filtering in Structured Overlays

Today's content providers are naturally distributed and produce large amounts of new information every day. Peer-to-peer information filtering is a promising approach that offers scalability, adaptivity to high dynamics, and failure resilience. The authors developed two approaches that utilize the Chord distributed hash table as the routing substrate, but one stresses retrieval effectiveness, whereas the other relaxes recall guarantees to achieve lower message traffic and thus better scalability. This article highlights the two approaches' main characteristics, presents the issues and trade-offs involved in their design, and compares them in terms of scalability, efficiency, and filtering effectiveness.

Christos Tryfonopoulos, Christian Zimmer, and Gerhard Weikum
Max-Planck Institute for Informatics

Manolis Koubarakis
National and Kapodistrian University of Athens

In recent years, dynamic information dissemination applications, such as news alerts, weather monitoring, and stock quotes, have gained popularity for large, open, and dynamic user communities. With news alerts, digital libraries, or RSS feeds — where the data of interest is mostly textual, and users express their needs using information retrieval languages (for example, keywords or pieces of text) — peer-to-peer (P2P) structured overlay networks are well-suited as an implementation technology because they can handle huge amounts of information in a highly distributed, self-organizing way. These characteristics offer enormous potential benefits for information dissemination capabilities in terms of openness, scalability, efficiency, and resilience to failures.

In this article, we consider *information filtering* (IF), also known as *publish–subscribe* (pub–sub). Users, or services that act on users' behalf, specify continuous queries, thus subscribing to newly appearing documents that satisfy the query conditions. The system will then notify the user automatically whenever a new matching document is published. Publishers can be news feeds, digital libraries, or users who post new items to blogs or other Internet communities.

Information filtering is very different from information retrieval (as in search engines), in which a user poses a query

 1089-7801/07/$25.00 © 2007 IEEE

and the search engine executes it only once to retrieve the currently matching documents. In fact, the preferred approach for centralized pub–sub applications is to index queries rather than data and evaluate newly published data items against these existing subscriptions. Depending on the expressiveness of the query language, such IF services are technically already difficult to implement in a centralized setting; in a widely distributed setting with several subscribers and potential publishers, efficiently routing newly published documents to match them against existing subscriptions poses additional complexity and technical challenges.

Building on our experience with designing IF systems on top of structured overlay networks, we present architectural alternatives and considerations that you should take into account if you want to build efficient and scalable distributed pub–sub systems. Although, distributed hash tables (DHTs) can prove useful as the routing substrate for higher-level services such as IF, extensions to support expressive query languages are necessary (see the "Filtering in Structured Overlays" sidebar for related work in this area). We built our filtering functionality on top of the Chord[1] DHT by focusing on two different architectures that we designed, DHTrie and Minerva Approximate Publish–Subscribe (MAPS). We aim to demonstrate the feasibility of P2P IF, highlight the different trade-offs that designing such systems requires, and provide pointers for further reading and exploration.

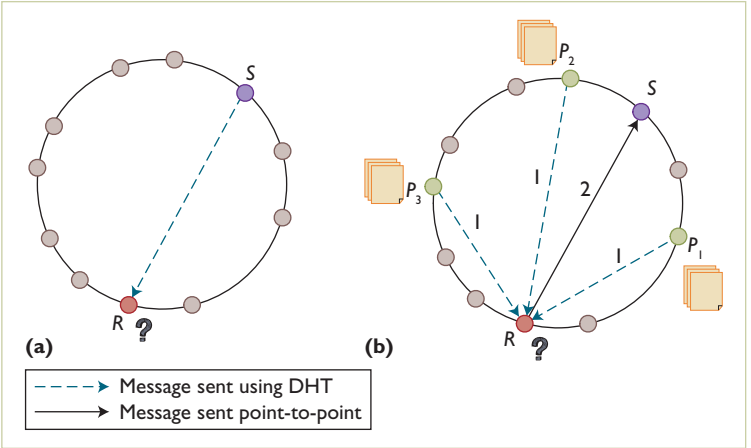

Figure 1. DHTrie subscription, publication, and notification procedures. (a) The subscriber (S) chooses a peer in the network using the distributed hash table (DHT) mapping function and then uses the DHT to send the query to this peer. This peer (R) will store the query and notify the subscriber when a publication that matches the query arrives at any network node. (b) The dashed arrows show publishers (P_1, P_2, P_3) using the DHT to contact a peer (R) that's storing a possibly matching query. Peer R storing the query, matches the new documents against its local database and notifies the subscriber (S) about documents that match its query. The solid arrow shows a point-to-point communication.

The DHTrie Architecture

The DHTrie architecture[2] builds on the Chord DHT to provide efficient IF functionality. It extends the Chord protocols to support content-based multicasting and introduces new routing tables that provide fast access to nodes contacted often. As we will show, these optimizations are necessary for supporting large-scale IF services efficiently.

Services

In the DHTrie network, nodes can implement a basic *router service* and either (or both) a *subscriber service* or a *publisher service*. To facilitate efficient messaging between nodes, we developed extensions to the Chord[1] DHT, which provides the basic routing infrastructure for exact matching. Figure 1 shows how DHTrie realizes the subscription, publication, and notification protocols.

Router service. All nodes in the DHTrie system form the network's message routing layer by implementing the router service. Each node that implements this service runs a DHT protocol, which is an extension of Chord. The DHT's role in DHTrie is to

- act as a rendezvous point between information producers and consumers;
- serve as a robust, fault-tolerant, and scalable routing infrastructure — when the network grows in size, the DHT provides a scalable means of locating other nodes; and
- serve as a global metadata index that is partitioned among nodes.

Subscriber service. Nodes implementing only the subscriber service are called *subscribers*; these are information consumers — they can subscribe to resource publications and receive notifications about published resources that match their interests. The publisher node that owns the resource directly handles these resource requests.

Publisher service. This service is implemented by information producers (for example, digital libraries or users) that want to expose their content to the

rest of the network. A node implementing only the publisher service creates metadata for the resources it stores using the *attribute word-pattern with similarity* (AWPS) data model,[3] and uses the router service to connect to the rest of the network.

Content-Based Multicasting

In DHTrie, we use AWPS, which is a well-understood attribute-value data model.[3] AWPS's query language allows equality, containment, and similarity operators on attribute values. The containment operator offers conjunction, disjunction, word negation, and distance between words. For simplicity, we consider only keyword queries and describe how DHTrie indexes them.

Let's assume that a subscriber node wants to submit a continuous query q, consisting of k distinct terms. It uses the DHT hash function to obtain k identifiers, and then uses the DHT to route q to the nodes responsible for these identifiers. In this way, q is stored at k nodes in the network, and these nodes are responsible for matching the query against new publications and notifying the subscriber. This query-indexing mechanism doesn't affect recall because the whole query q is indexed at k different nodes. Thus, at publication time, DHTrie matches the full query against the incoming document and returns matches to the users.

When a node wants to publish a document d, it identifies all the distinct words in it and uses the DHT hash function to obtain, at most, as many identifiers as the number of distinct words. Then, it contacts all nodes responsible for these words, given that these nodes might store queries that will match d. Each peer utilizes appropriate local filtering algorithms to perform local matching of incoming documents and stored queries.[4] The subscription procedure clearly demonstrates that query placement in DHTrie is deterministic and depends on the query terms. Publication is also deterministic, and depends on the words contained in the incoming document. The DHT partitions the term space and acts as a rendezvous point for queries and documents. This query-indexing strategy lets DHTrie achieve the recall of a centralized system. We describe the DHTrie protocols in detail elsewhere.[4]

To facilitate message sending between nodes, we use the function `send`(msg,I) to send message msg from some node to the node responsible for identifier I. The function `send()` is similar to the Chord function `lookup`(I) and costs $O(logN)$ overlay hops for a network of N nodes.

The subscription and publication procedure just described requires a node to send the same message to a group of other nodes to either locate the nodes that will store a query or to locate nodes storing queries that are possible matches for a publication. The creation of this group is dynamic: the message originator specifies it based on the resource publication or query subscription that takes place. In this way, group members are implicitly subscribed to groups depending on the keys they're responsible for. Thus, our multicast groups aren't known by the message originator a priori; this is the major difference between our content-based multicast scheme and the standard enrollment-based schemes presented in systems such as Scribe.[5] In those schemes, receivers explicitly subscribe to multicast groups, which maintain special data structures (such as multicast trees) to be able to notify subscribers.

To facilitate message sending, we designed and implemented three different methods that show the trade-offs between message traffic and publication latency:

- *The iterative method*. The obvious way to handle content-based multicasting over Chord is to create k different `send()` messages, where k is the number of different nodes to be contacted, and then locate the message's recipients in an iterative fashion using $O(k \log N)$ messages. The iterative method optimizes latency because messages are sent in parallel.
- *The recursive method*. Using the iterative method requires asking the same node the same routing question multiple times, which leads to an increase in message traffic. The idea behind the recursive method is to pack messages together and exploit other nodes' routing tables to reduce network traffic (see Figure 2a). DHTrie achieves this by using a recipients list, which it sends along with the actual message. This list contains all intended recipients' identifiers sorted in ascending order, starting from the identifier of the message originator. The message originator then sends the message to the node responsible for the first identifier on the list. Once the message reaches the node responsible for this identifier, this node copies it for matching against its query database; then the node removes the identifier from the list, and forwards the message in the same recursive way to the rest of the list's recipients. If the

recipient list is long, then the last recipient must wait for a long time until he or she is notified about the publication, which means higher publication latency (see Figure 2b).

- *The hybrid method*. The idea behind the hybrid method is to design a tunable alternative that will provide fast message delivery at low network cost. Thus, the hybrid method combines the techniques used in the two previous methods, and can trade network traffic for publication latency. All intended recipients' identifiers are split into several small recipient lists, and the message originator sends the messages containing the lists iteratively, while the intended recipients within the lists are reached in a recursive way. This splitting of lists uses a system parameter called *maximum list size* and is based on the message originator's finger table entries. Each of the finger table entries is associated with several lists so as to better exploit the routing information in the finger tables. In the experiments we present in the next section, we use a maximum list size of 30. In general, the hybrid method provides a versatile algorithm that combines the benefits of the two previous methods to minimize both message traffic and latency.

In combination with these methods, we also introduce a simple routing table (called FCache) that uses only local information and manages to further reduce network traffic and publication latency. Each node updates, with minimal bookkeeping, the FCache entries with the IP addresses of other nodes that it contacts often. In this way, message originators can bypass the routing infrastructure and thus relieve it from a significant load. This caching scheme provides a mapping between often-used keys and the nodes responsible for them (similar to value proxying) and differs from other caching mechanisms,[6] in which cached copies of the actual data are left along the path from the querying to the data provider peer. By design, FCache is self-maintained (outdated entries are repaired when needed, no extra maintenance cost in terms of messages), self-tunable (frequently used entries need less repairs), and self-adaptive (entries are dynamic), and it increases the system's scalability and efficiency.

Note that although our query-partitioning scheme is by keyword, the setting isn't similar to the one Jinyang Li and colleagues describe[7] because no posting lists of any sort need to be sent over the network to match a query against an incoming document. The only information the message originator sends across the network is a link to the document publisher, which can be used by nodes storing possibly matching queries. Additionally, the optimizations we just discussed greatly improve the proposed architecture's scalability, which, as Figure 2 shows, follows Chord's scalability guarantees. Our goal of further efficiency improvements was a main impetus for developing MAPS, which we describe in the next section.

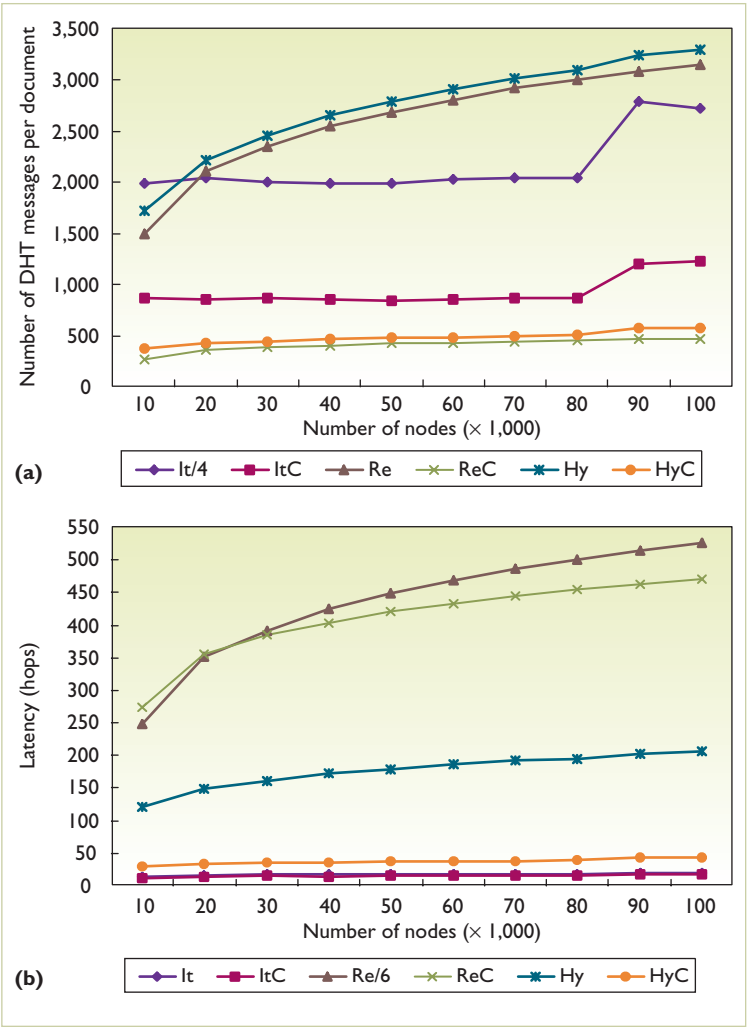

Figure 2. Performance measurements. (a) Message traffic performance. We measure the average number of messages needed to publish a document of about 5,000 words in networks of various sizes. (b) Latency performance. We measure latency as the longest chain of messages needed until the publication reaches all its intended recipients.

Experimental Evaluation

To show DHTrie's scalability, we implemented and experimented with six variations of the protocols. Algorithms *It*, *Re*, and *Hy* use the iterative, recursive, and hybrid methods, respectively, in the publication protocol and don't use FCache, whereas algorithms *ItC*, *ReC*, and *HyC* use an FCache with 10,000 entries. Figures 2a and 2b show the message traffic and latency per document observed when publishing 10,000 documents with an average of 5,415 words at networks varying from 10,000 nodes to 100,000 nodes. Note that both message traffic and latency grow at a logarithmic scale compared to network size, due to the DHT usage. Additionally, using FCache results in a sixfold reduction in both message traffic and latency.

In general, our experiments[4] showed that algorithm *ReC* performs best when message traffic is in question. It remains relatively unaffected by network size, shows small sensitivity to increasing document sizes, and needs only a few valid FCache entries to achieve a big performance improvement. On the other hand, when publication latency is in question, algorithm *ItC* is a good candidate. It remains unaffected by network size, FCache's level of training and size, or publication size; however, it incurs higher network traffic in the routing infrastructure. Finally, *HyC* is a tunable alternative that manages to strike a balance between optimizing network traffic and publication latency. It's slightly more expensive than *ReC* in terms of network traffic but still significantly lower than *ItC*, whereas it performs well in terms of latency.

The MAPS Architecture

In the previous section, we looked at an architecture that uses a DHT as the routing infrastructure to support IF and provides the recall of a centralized system. Here, we present MAPS,[8] a complementary approach that shows how to improve the previous solution's scalability by trading reduced recall for lower message traffic. Again, we use a DHT as an underlying routing mechanism, but its purpose is to disseminate *statistics* about the publications rather than the publications themselves (as in DHTrie). In the MAPS approach, only a few carefully selected, specialized, and promising peers store the user query and are monitored for new publications. This approximate filtering relaxes the assumption — which holds in most pub–sub systems — of potentially delivering notifications from every producer and amplifies scalability.

Service Types

The MAPS architecture is based on the P2P search engine Minerva.[9] Each participating peer implements three types of services. Figure 3 shows how MAPS realizes the subscription, publication, and notification protocols.

Directory service. All nodes that participate in the MAPS network implement the directory service, which provides the DHT-based routing infrastructure and maintains a distributed statistics index for document terms. This index both forms a conceptually global but physically distributed directory, which is layered on top of a Chord-style DHT, and manages aggregated information about each peer's local knowledge in compact form. The DHT partitions the term space, such that every peer is responsible for the statistics of a randomized subset of terms within the global directory. To keep the information retrieval (IR) statistics up to date, each peer distributes per-term summaries of its local index along with its contact information to the global directory. For efficiency, peers piggyback these messages on DHT maintenance messages and use message batching. Additionally, to reduce the term space and improve recall efficiency, MAPS exploits correlations among query terms (for example, the word Aguilera is almost always combined with the word Christina in a query) and considers multiword terms.

Publication service. Users can utilize the publication service when they want to publish their own documents to the MAPS network, or digital libraries can use it when they want to expose their content. A node implementing the publisher service uses the directory service to update statistics about the terms contained in the documents it publishes. Publisher nodes also store continuous queries submitted by the users and match them against their own publications.

Subscription service. Users implement the subscription service when they want to monitor specific data sources. The subscription service is critical to the recall the system will achieve at filtering time because it's responsible for selecting the peers that will index the query. This node-selection procedure uses the directory service to discover and retrieve node statistics that will facilitate query indexing. Once the querying peer retrieves these statistics, it ranks the potential

sources and sends the user query to the k top-ranked publishers. In this way, the querying peer will monitor only these publishers for new publications, and because their publication behavior might not be consistent over time, query repositioning is necessary to achieve higher recall.

Continuous Query Forwarding

A peer P receiving a keyword query q with t distinct terms must determine which network peers are promising candidates for satisfying q with documents published in the future. Thus, for each term t_i contained in q, the peer requests per-peer statistics from the directory service. Then, the querying peer computes a peer score based on a combination of *resource selection* and *peer behavior prediction* formulas, as demonstrated by the following equation:

$$score(P,q) = \alpha \cdot \underbrace{sel(P,q)}_{selection} + (1-\alpha) \cdot \underbrace{pred(P,q)}_{prediction}.$$

The tunable parameter α affects the balance between authorities (high $sel(P, q)$ scores) and promising peers (high $pred(P, q)$ scores) in the final ranking. Based on the scores calculated for each peer, the querying peer determines a peer ranking and forwards q to the highest-ranked peers. Only peers storing q will match it against their new publications, and the matching documents will produce appropriate notifications to subscribers. Given that publishing is a dynamic process, the querying peer might need to reposition q periodically to achieve higher recall.

We implement *resource selection* using standard algorithms[10] from the IR literature (such as the collection retrieval with inference [CORI]). These techniques rely on a node's collection size and on statistical information about terms' occurrence within documents and within a node's collection, to estimate which node is likely to provide relevant answers to a query. Using $sel(p, q)$, we identify authorities specialized in a topic, but as we show later, this isn't enough in an IF setting.

Peer behavior prediction computes a score for a peer P and a query q that represents how likely peer P is to publish documents containing terms found in q in the future. This prediction mechanism is based on statistical analysis of appropriate IR metrics such as the number of documents published in the last interval and containing a specific term. These statistics are available through appropriate requests from the directory service, and are treated as *time series data*. Then, querying peers use double exponential smoothing to model a peer's publishing behavior and predict future publications.

The main idea behind predicting peer behavior is to view the IR statistics as time series data and use statistical analysis tools to model peer behavior. Time series analysis accounts for the fact that the data points taken over time have some sort of internal structure (trend, periodicity, and so on), and uses this observation to analyze older values and predict future ones. Prediction is complementary to resource selection; the following example demonstrates its necessity in a filtering setting. Assume two peers, where the first is specialized in soccer, although it's no longer publishing new documents. The second isn't specialized in soccer, but its crawler is currently crawling a big soccer portal. If a user subscribes for documents with the continuous query "soccer world cup 2010," for example, a ranking function based only on resource selection algorithms would always choose the first peer. To get a higher ranking score, and thus get selected for indexing the query, the second peer must specialize in soccer, a long procedure that is inapplicable in a filtering setting, which is by definition dynamic.

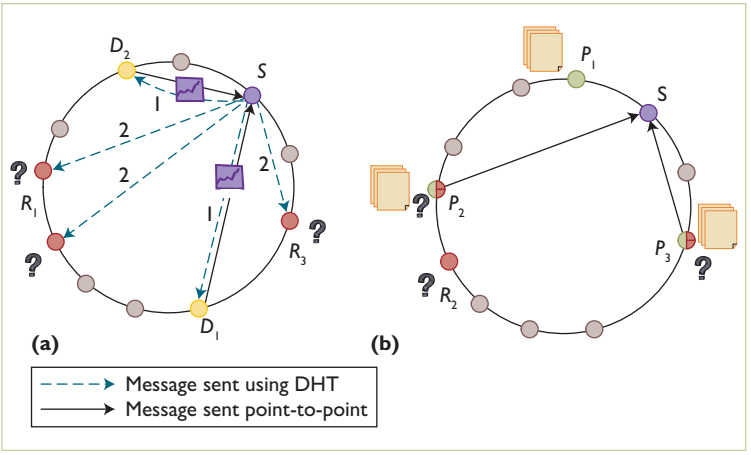

Figure 3. Subscription, publication, and notification procedure in MAPS. (a) In the subscription procedure, the subscriber (S) collects term statistics from directory nodes (D_1, D_2) using the distributed hash table (DHT) and selects the peers it will monitor. S forwards its query to the selected peers (R_1, R_2, R_3) using the DHT. (b) Only publisher peers that store the subscriber's query (P_2, P_3) and produce matching documents notify the subscriber (S). Publishers P_2 and P_3 use point-to-point links to notify S.

Resource Publication

When a peer p publishes a new document, the

Table 1. A comparison between DHTrie and MAPS.

	DHTrie	MAPS
Objective	Exact publish–subscribe functionality (queries are indexed, all publishers are monitored)	Approximate publish–subscribe functionality (only selected publishers are monitored)
Advantages (+) and Disadvantages (–)	+Retrieval effectiveness (recall of a centralized system) –Dependent on publication rate –Relatively high message traffic	+Low network traffic, scalability +Independent of publication rate –Lower recall (missing potentially interesting publications)
Routing	Chord DHT (to index queries)	Chord DHT (to maintain statistics)
DHT optimizations	Message grouping Extra routing table	Batch messaging Piggy-backing on DHT messages Term-space reduction
Query placement	Deterministic indexing (depends on query terms)	On selected nodes (depends on publishing behavior)
Terms statistics	Implicit (due to indexing) Needed for matching	Explicit (peers post and collect statistics) Needed for peer ranking and matching
Load balancing	Explicit (algorithm to address imbalances) More sensitive to load imbalances	Implicit (due to query placement) Less sensitive to load imbalances

document is matched against its local query index using appropriate local filtering algorithms[3] and triggers notifications to all subscribers. Only peers with their continuous query indexed in p will be notified about the new publication because the document isn't distributed to any other peer in the network. This subscription scheme makes query placement a crucial decision. The publication and notification process itself doesn't require any additional communication costs.

Optimizations

Because Chord's stabilization functionality ensures correct routing within the directory, peers leave and join the distributed directory at any time. To avoid data loss, especially of directory entries, MAPS uses standard replication and caching strategies. We chose to replicate directory entries at successor peers in order to address peer failures transparently to the querying peer.

To improve resource selection, MAPS can exploit statistics about correlated terms. Similar to a previous approach,[11] the directory learns correlated terms by looking at queries. Once a directory peer recognizes a correlation between terms, the directory also stores the peer statistics concerning this term set. Thus, resource selection utilizes the correlated term statistics to monitor peers publishing documents with all requested terms. This approach avoids monitoring peers that publish many documents containing the single terms, but no or only a few documents containing all query terms.

Experimental Evaluation

We conducted experiments with 1,000 peers specialized in 10 categories (such as "Sports" or "Travel"); in this way, roughly 100 peers store documents from the same category. Our data collection consists of 2 million categorized documents from a focused Web crawl, and we used continuous queries with two, three, and four query terms that strongly represented the categories. Example queries are "modern art" or "music instrument." Subscribers submitted the queries and monitored a fraction ρ of the publisher population, while they periodically repositioned their queries according to the scoring function. We measured recall as the ratio of the total number of notifications the subscriber received to the total number of published documents matching the subscriptions.

Figure 4 shows the experimental results for a scenario in which publishers shift their interest to a different topic. We used different values for α to investigate the influence of peer behavior prediction and resource selection in our scoring function. On the x-axis, parameter ρ varies from 1 to 25, meaning that the subscriber monitored up to 25 percent of the publishers. In this setting, we observe that resource selection only ($\alpha = 1$) isn't enough to reach a satisfying recall, whereas when α is small (for example $\alpha = 0$ or $\alpha = 0.25$), then the emphasis is on peer behavior prediction, and MAPS reaches a recall value of 0.6 by monitoring only 10 percent of the publishers. In our scalability experiments, we observed a reduction in the message overhead incurred by MAPS of as much

as eight times compared to DHTrie, by giving up roughly 10 to 20 percent of recall.

Comparison

In the previous sections, we presented our DHTrie and MAPS approaches and highlighted their differences. Here, we compare the two architectures in terms of design decisions made; Table 1 summarizes this comparison.

Routing Infrastructure

A crucial design decision in both approaches is to use a DHT as the underlying routing infrastructure. Content-based filtering requires an efficient object-location mechanism to support expressive query languages that capture the user's specific interests. This renders Gnutella-style networks an inefficient solution and DHTs' simple keyword functionality an insufficient one. To overcome this limitation of exact lookup, both approaches extend the DHT functionality to support richer data models and more expressive queries. However, to efficiently support this functionality, DHT protocols and data structures require some changes and extensions. DHTrie uses message grouping and extra routing tables to overcome inefficiencies, whereas MAPS batch-posts term summaries, piggy-backs post messages on directory maintenance messages, and decreases the term space by using correlated multiword terms.

Query Placement

Distributed pub–sub systems involve peer-selection techniques to decide where to place a user query. This selection is critical because future publications that might match this query should also be able to reach the node storing it to trigger a notification in case of a match. Query placement in DHTrie is deterministic and depends on the terms contained in the query and the hash function the DHT provides. To decide where to place a keyword query, the query originator hashes all query terms and forwards the query to the nodes responsible for these identifiers to ensure correctness at publication time. These query placement protocols lead to filtering performance that's exactly the same as that of a centralized system. On the other hand, in MAPS, only the most specialized and promising nodes store a user query and are thus monitored. Publications that each node produces are matched against the node's local query database because no publication forwarding is used for scalability reasons. In this case, MAPS' recall is lower than DHTrie's, but it avoids document-granularity network dissemination.

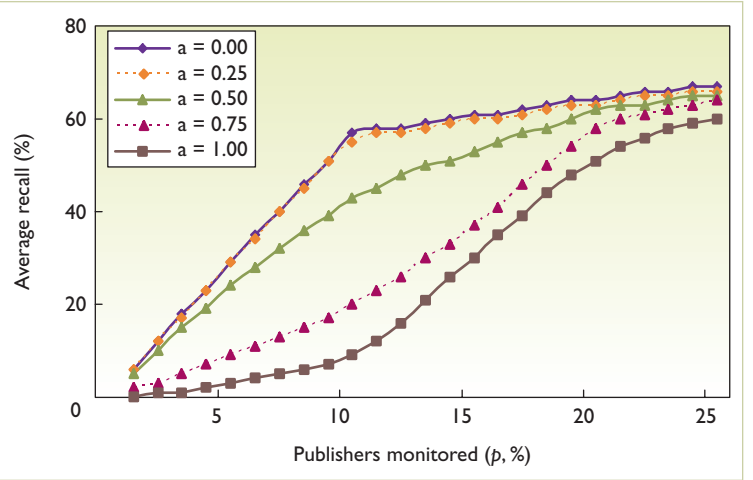

Figure 4. Experimental results. High recall is possible with peer behavior prediction even though we monitor only a few publishers.

Statistical Information

Matching incoming documents with stored subscriptions involves maintaining and estimating important global statistics, such as a certain term's document frequency — that is, the number of distinct documents seen in the last interval that contain that specific keyword. In DHTrie, this functionality is implicit; every time some node publishes a new document, it reaches the nodes responsible for the distinct keywords contained in that document. Such nodes are candidates for storing continuous queries that will potentially match the document. These nodes do all the necessary bookkeeping for the statistical information, and other nodes can query them for this information when they need to compute a similarity score. On the other hand, MAPS explicitly addresses this issue by maintaining a directory. With DHTrie, the statistics maintenance is a byproduct of the filtering process and is only necessary for computing similarity scores; with MAPS, it's the cornerstone of the filtering process because both peer selection (and thus query routing) and similarity computation utilize it.

Load Balancing

The effect of load imbalances between nodes in the two approaches is also different. DHTrie nodes are more susceptible to load imbalances due to the nature of the subscription and publication protocol. Mapping the DHT hash function means that

Information Filtering in Structured Overlays

We can typically categorize peer-to-peer (P2P) networks in three different classes according to topology: unstructured, hierarchical, and structured networks. Distributed hash tables (DHTs)[1–3] are a prominent class of structured overlay networks devised to efficiently solve the object lookup problem — that is, locate a node that holds a specific data item. With their invention, distributed information management emerged as an interesting research area that could benefit from the P2P paradigm. One category of information management applications focuses on supporting content retrieval on top of structured overlays using various data models and query languages,[4,5] whereas another focuses on the publish–subscribe (pub–sub) paradigm. Here, we emphasize surveying work from the pub–sub domain, and don't further explore retrieval approaches because they aren't directly relevant to the main article's scope.

Over the past few years, a new wave of pub–sub systems that use structured overlays as the routing substrate has appeared. Scribe[6] was one of the first topic-based approaches built using the Pastry DHT.[3] Hermes[7] is similar to Scribe because it uses the same underlying DHT (Pastry), but it allows more expressive subscriptions by supporting the notion of an event type with attributes. Each event type in Hermes is managed by an *event broker*, which is a rendezvous node for subscriptions and publications related to this event. PeerCQ[8] is another notable pub–sub system implemented on top of a DHT infrastructure. PeerCQ's most important contribution is that it takes into account peer heterogeneity and extends consistent hashing with simple load-balancing techniques based on appropriate assignment of peer identifiers to network nodes. Meghdoot[9] is a recent proposal implemented on top of the Content Addressable Network (CAN) DHT[2]; it supports an attribute-value data model and offers new ideas for processing subscriptions with range predicates (for example, if a price is between 20 and 40 euros) and load balancing. Mercury[10] is a P2P system with a similar attribute-value data model which is utilized in the implementation of a publish–subscribe system for network games. All the aforementioned approaches support Boolean queries with arithmetic operators and don't consider information retrieval (IR) based data models and languages.

Recently, researchers have deployed systems that employ an IR-based query language to support information filtering on top of structured overlay networks. pFilter[11] uses a hierarchical extension of the CAN DHT to store user queries and relies on multicast trees to notify subscribers. LibraRing[12] and MinervaDL[13] present two frameworks for providing information retrieval and filtering services for future digital libraries in super-peer environments. Finally, PastryStrings[14] demonstrates how to implement a DHT-agnostic solution to support prefix and suffix operations over string attributes in a pub–sub environment. Compared to these approaches, the solutions we pres-

continued on p. 25

an overloaded node or a node with small processing capabilities might become responsible for a popular term, and thus be forced to store high volumes of user queries and process high numbers of publications. DHTrie load-balancing mechanisms are based on load shedding and prove efficient even for highly skewed distributions. On the other hand, MAPS has an intrinsic load-balancing mechanism to cope with imbalances. Assuming that nodes are willing to offer some of their resources to the community, a node with resources to share soon becomes a hub node for some topic and receives more subscriptions, whereas a node with few resources will be forced to specialize more and thus reduce the number of users interested in its publications. MAPS is more sensitive to filtering load balancing (that is, the load imposed by the filtering requests that need to be processed) because a directory node responsible for a popular term will get high numbers of statistics retrieval requests. Finally, routing load (that is, the load imposed by the messages a node has to forward due to the overlay maintenance protocols) is similar in both systems, and mainly depends on DHT usage.

Supporting full-fledged IR and IF functionality on top of overlay networks is an important research direction that showcases the potential of the P2P paradigm. Our two IF architectures utilize a DHT in different ways, showing the diversity of solutions available. This diversity spans not only on the underlying routing infrastructure support, but also on the query languages used and the load-balancing techniques utilized. Thus, concrete applications are necessary to drive our understanding of these trade-offs.

Acknowledgments

This work has been partly supported by the DELOS Network of Excellence, and the EU projects AEOLUS and Evergrow.

References

1. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM Conference on*

Information Filtering in Structured Overlays, continued

ent in the main text use a more expressive data model and query language, don't need to maintain multicast data structures to notify subscribers, and don't utilize a hierarchical (two-tier) architecture.

References

1. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (SIGCOMM 2001), ACM Press, 2001, pp. 149–160.
2. S. Shenker et al., "A Scalable Content-addressable Network," *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication* (SIGCOMM 2001), AMC Press, 2001, pp. 161–172.
3. A. Rowstron and P. Druschel, "Pastry: Scalable, "Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms* (Middleware 2001), LNCS 2218, Springer-Verlag, 2001, pp. 329–350.
4. R. Huebsch et al., "Querying the Internet with PIER," *Proc. 1st Int'l Workshop Peer-to-Peer Systems* (IPTPS 2002), LNCS 2429, Springer-Verlag, 2002, pp. 242–259.
5. J. Stribling et al., "OverCite: A Cooperative Digital Research Library," *Proc. 4th Int'l Workshop Peer-to-Peer Systems* (IPTPS 2005), LNCS 3640 Springer-Verlag, 2005, pp. 69–75.
6. A. Rowstron et al., "Scribe: The Design of a Large-Scale Event Notification Infrastructure," *Proc. 3rd Int'l COST264 Workshop Networked Group Communication*, LNCS 2233, Springer-Verlag, 2001, pp. 30–43.
7. D. Faensen et al., "Hermes—A Notification Service for Digital Libraries," *Proc. ACM/IEEE Joint Conf. Digital Libraries* (JCDL 2001), ACM Press, 2001, pp. 373–380.
8. B. Gedik and L. Liu, "PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System," *Proc. 23rd Int'l Conf. Distributed Computing Systems* (ICDCS 2003), IEEE CS Press, 2003, pp. 490–499.
9. A. Gupta et al., "Meghdoot: Content-Based Publish/Subscribe over P2P Networks," *Proc. ACM/IFIP/Usenix Int'l Middleware Conf.* (Middleware 2004), LNCS 3231, Springer-Verlag, 2004, pp. 254–273.
10. A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-attribute Range Queries," *Proc. ACM SIGCOMM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM Press, 2004, pp. 353–366.
11. C. Tang and Z. Xu. "pFilter: Global Information Filtering and Dissemination Using Structured Overlays," *Proc. 9th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems* (FTDCS 2003), IEEE CS Press, 2003, pp. 24–30.
12. C. Tryfonopoulos, S. Idreos, and M. Koubarakis, "LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs," *Proc. 9th European Conf. Research and Advanced Technology for Digital Libraries* (ECDL 2005, LNCS 3652, Springer-Verlag, 2005, pp. 25–36.
13. C. Zimmer, C. Tryfonopoulos, and G. Weikum, "MinervaDL: An Architecture for Information Retrieval and Filtering in Distributed Digital Libraries," *Proc. 11th European Conf. Research and Advanced Technology for Digital Libraries* (ECDL), to be published, 2007.
14. I. Aekaterinidis and P. Triantafillou, "PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network," *Proc. 26th IEEE Int'l Conf. Distributed Computing Systems* (ICDCS 2006), IEEE CS Press, 2006, pp. 4–7.

Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2001), ACM Press, 2001, pp. 149–160.

2. C. Tryfonopoulos, S. Idreos, and M. Koubarakis, "Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks," *Proc. 28th Ann. Int'l ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Press, 2005, pp. 322–329.
3. M. Koubarakis et al., "Information Alert in Distributed Digital Libraries: The Models, Languages, and Architecture of DIAS," *Proc. 6th European Conf. Research and Advanced Technology for Digital Libraries* (ECDL 2002), LNCS 2458, Springer-Verlag, 2002, pp. 527–542.
4. C. Tryfonopoulos, M. Koubarakis, and Y. Drougas, "Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators," *Proc. 27th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* ACM Press, 2004, pp. 313–320.
5. A. Rowstron et al., "Scribe: The Design of a Large-Scale Event Notification Infrastructure," *Proc. 3rd Int'l COST264 Workshop Networked Group Communication*, LNCS 2233, Springer-Verlag, 2001, pp. 30–43.
6. F. Dabek et al., "Wide Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating System Principles*, Operating System Review, vol. 35, no. 5, ACM Press, 2001, pp. 202–215.
7. J. Li et al., "On the Feasibility of Peer-to-Peer Web Indexing and Search," *Proc. 2nd Int'l Workshop Peer-to-Peer Systems* (IPTPS 2003), LNCS 2735, Springer-Verlag, 2003, pp. 207–215.
8. K. Berberich et al., "MAPS: Approximate Publish/Subscribe Functionality in Peer-to-Peer Networks," *Proc. 1st Int'l Workshop Advanced Data Processing in Ubiquitous Computing* (ADPUC), ACM Press, 2006, pp. 1–6.
9. M. Bender et al., "Improving Collection Selection with Overlap-Awareness," *Proc. 28th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM Press, 2005, pp. 67–74.
10. J. Callan, "Distributed Information Retrieval," *Advances in Information Retrieval*, W.B. Croft, ed., Kluwer Academic Publishers, 2000, chapter 5.
11. N. Ntarmos et al., "Discovering and Exploiting Keyword and Attribute-Value Co-Occurrences to Improve P2P Routing Indices," *Proc. 2006 Int'l Conference on Information and Knowledge Management* (CIKM), ACM Press, 2006, pp. 172–181.

Christos Tryfonopoulos is a post doctoral researcher in the Databases and Information Systems Group at the Max-Planck Institute for Informatics in Saarbruecken, Germany. His research interests include information retrieval and filtering over wide-area networks, P2P and grid computing, publish–subscribe systems, and multiagent systems. Tryfonopoulos has a BSc in computer science from the University of Crete, and an MSc and a PhD in computer engineering from the Technical University of Crete. He has received two scholarships from the Greek Ministry of Education and won a best student paper award at the European Conference on Digital Libraries in 2005. Contact him at trifon@mpi-inf.mpg.de; www.mpi-inf.mpg.de/~trifon/.

Christian Zimmer is a PhD student in the Databases and Information Systems Group at the Max-Planck Institute for Informatics in Saarbruecken, Germany. His main research interests include information retrieval and information filtering in a distributed P2P environment; his main focus is on approximate publish–subscribe approaches. Zimmer has a diploma in computer science from the University of Saarland. Contact him at czimmer@mpi-sb.mpg.de ; www.mpi-inf.mpg.de/~czimmer.

Manolis Koubarakis is an associate professor in the Department of Informatics and Telecommunications, National and Kapodistrian University of Athens. His research interests include database and knowledge-base systems, temporal and spatial reasoning, constraint programming, intelligent agents, Semantic Web, peer-to-peer, and grid computing. Koubarakis has a degree in mathematics from the University of Crete, an MSc in computer science from the University of Toronto, and a PhD in computer science from the National Technical University of Athens. Contact him at koubarak@di.uoa.gr; www.di.uoa.gr/~koubarak.

Gerhard Weikum is a scientific director at the Max-Planck Institute for Informatics in Saarbruecken, Germany, where he leads the Databases and Information Systems research group. His interests include distributed information systems such as P2P systems, and intelligent search and organization of semistructured data on the Web and in digital libraries. Weikum received his diploma and doctoral degrees from the University of Darmstadt, Germany. He has received several best paper awards, including the VLDB 2002 10-year award, and is an ACM fellow. Contact him at weikum@mpi-inf.mpg.de; www.mpi-inf.mpg.de/~weikum.