

pFilter: Global Information Filtering and Dissemination

Chunqiang Tang^{*}
Dept. of Computer Science
Univ. of Rochester
Rochester, NY 14627-0226
sarrmor@cs.rochester.edu

Zhichen Xu
HP Laboratories
1501 Page Mill Rd., MLS 1177
Palo Alto, CA 94304
zhichen@hpl.hp.com

ABSTRACT

Due to the overwhelming amount of information on the Internet, it is becoming increasingly difficult for people to find relevant information in a timely fashion. Information filtering and dissemination systems allow user to register persistent queries called *user profiles*. They detect new contents, match them against the profiles, and continuously notify users when relevant information becomes available. Existing systems, however, either are not scalable; or do not support matching of unstructured documents. Unstructured documents such as text, HTML or multimedia files, account for a significant percentage of contents on the Internet.

To address the limits of the existing systems, we describe pFilter, a global-scale decentralized information filtering and dissemination system for unstructured documents. To handle potentially billions of documents for millions of subscribers, pFilter connects potentially millions of computers in national (and international) computing Grids or ordinary desktops into a structured peer-to-peer overlay network. Nodes in the overlay collectively publish/collect documents, build index, register profiles, and filter and disseminate information. To enable efficient and accurate match between profiles and documents without flooding either documents or profiles, profiles in the overlay are organized around their vector representations (based on modern information retrieval algorithms) such that the searching space of a new document is organized around related profiles. In pFilter, we introduce a new application-level multicast algorithm that allows documents to be efficiently disseminated to a large number of interested parties.

1. INTRODUCTION

Today, Internet has become the single most important information source that affects our everyday lives. With huge amount of information at our finger tips is certainly an advantage, but it also poses a big challenge with respect to getting *desired* information in a *timely* fashion.

Search engines such as Google only partially solve this problem by collecting documents on the Internet and using information retrieval (IR) algorithms [10] to rank them. Search engines, however, cannot guarantee *timely* access to relevant information. Because of the size of the Internet and large quantity of Internet contents, it usually takes search engines months to crawl the Web and update indices, and they typically can index only 1/10 or 1/1000 of the entire Internet contents [1]. Furthermore, users who concern about latest information have to frequently revisit search engines for possible updates.

^{*}This work was done when the author worked at HP Labs in fall 2002.

Instead of requiring users constantly checking for new contents, information filtering and dissemination systems [16] take over this responsibility. These systems allow user to register persistent queries called *user profiles*. They detect new contents, match them against the profiles, and continuously notify users when relevant information becomes available. We illustrate the use of such a system using a science project as an example.

At the beginning of a project, scientists use search engines to find related work and manually select out the most relevant papers. Besides adding these papers to their bibliography, they also register user profiles to have the system notify them when documents similar to those papers show up. As a result, they are constantly informed about relevant papers, discussions in news groups, etc., always staying at the cutting edge of the field. Later, after writing up a paper, they can register another profile to ask the system to notify them when their paper is cited.

The above ideal scenario, unfortunately, is still far from the reality. The majority of existing event-notification systems support only subject- or schema-based publish/subscribe model, thereby cannot be used to filter unstructured documents such as text, HTML or multimedia files. Unstructured documents account for a significant percentage of contents on the Internet. Several systems do filter unstructured documents (e.g., SIFT [16]), but they run at a centralized site, and are unable to filter or disseminate information at a large scale.

To address these limitations, we are developing a planetary-scale decentralized information filtering and dissemination system *pFilter*, aiming to processing billions of documents for millions of users. There are three major challenges for pFilter: (1) scalability—how to keep pace with the astonishing growth rate of the Internet content; (2) matching—how to accurately match documents against profiles without flooding potentially billions of documents or profiles to every node in the system; and (3) dissemination—how to efficiently disseminate documents to a large number of subscribers.

To achieve good scalability in pFilter, computers in national (and international) computing Grids or ordinary desktops are connected into a structured peer-to-peer (P2P) overlay network [7]. Nodes in the overlay collectively publish/collect documents, build index, register profiles, and filter and disseminate information. The advantage of adopting a P2P model is that the capability of the system scales when the user population increases.

To support accurate full-text searches, pFilter adopts state-of-the-art IR algorithms such as vector space model (VSM) and latent se-

mantic indexing (LSI) [2]. These algorithms represent documents and queries as vectors, and measure the similarity between a query and a document as the cosine of the angle between their vector representations. To avoid the flooding of either documents or profiles, profiles in the P2P overlay are organized around their vector representations such that the searching space of a new document is organized around related profiles, achieving both efficiency and accuracy.

To efficiently deliver documents to a large number of interested parties, pFilter uses application-level multicast to disseminate documents. Several features distinguish pFilter’s multicast algorithm from other publish/subscribe systems. First, based on semantics, similar subscriptions are fused to reduce the number of individual profiles. Second, to support potentially billions of multicast trees, the resources devoted to a multicast tree is a tunable continuum depending on the importance of each tree. Last, unlike existing multicast tree construction algorithms that only utilize limited local proximity information, our *landmark+latency measurement* algorithm can generate an accurate global proximity map of the system in a decentralized fashion, which is used to guide the construction of multicast trees.

The remainder of the paper is organized as follows. Section 2 provides background information about peer-to-peer overlay networks and information retrieval algorithms. We describe profile matching and document dissemination in Section 3 and Section 4 respectively. A discussion is provided in Section 5. Related work is given in Section 6. Section 7 concludes the paper.

2. BACKGROUND

pFilter is built on top of eCAN [15] (a hierarchical version of CAN) and uses extensions to VSM and LSI [2]. The Cartesian space abstraction of CAN makes it particularly attractive when used to store vector representations of documents generated by IR algorithms such as VSM and LSI. We describe these basic components below.

2.1 DHT Systems and eCAN

Recent DHT systems, represented by CAN, Chord, and Pastry, offer an administration-free and fault-tolerant application-level overlay network. The basic functionality these system provide is a distributed hash table (DHT). In these systems, there is a consistent binding between objects and nodes. Locating an object is reduced to the problem of routing to the node that hosts the object from the node where the query is submitted.

CAN stands for content-addressable network. It organizes the logical space as a d -dimensional Cartesian space (a d -torus) and partitions it into *zones*. One or more nodes serve(s) as owner(s) of a zone. An object key is a point in the space, and the object is stored at the node whose zone contains the point. Locating an object is reduced to the problem of routing to the node that hosts the object. Routing from a source node to a destination node is equivalent to routing from one zone to another in the Cartesian space. A node join corresponds to picking a random point in the Cartesian space, routing to the zone that contains the point, and splitting the zone with its current owner(s).

An example of CAN is shown in Figure 1. There are five nodes A to E in the overlay network. Each node owns a zone in the Cartesian space. Initially C owns the entire zone at the upper-right corner. When D joins, the zone owned by C splits and part of the zone is given to D. When D wants to retrieve the object with key $(0.4, 0.1)$,

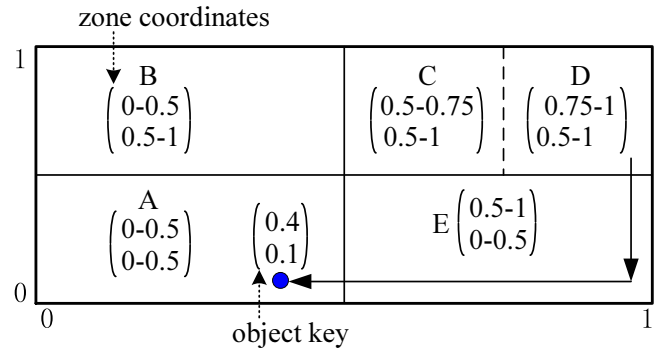


Figure 1: A 2-dimensional CAN.

it sends the request to E because E’s coordinates is closer to the object key. E forwards the request to A and A sends the object back to D directly.

eCAN improves CAN’s logical routing cost to $O(\log(n))$, and takes only routes that closely approximate the underlying Internet topology.

2.2 Vector Space Model

VSM represents documents and queries as *term vectors*. Each element of the vector represents the importance of a word (*term*) in the document or query. The weight of an element is often computed using the *term frequency * inverse document frequency* (TF*IDF) scheme [2]. The intuition behind it is that two factors decide the importance of a term in a document: the frequency of the term in the document and the frequency of the term in other documents. If a term appears in a document with a high frequency, there is a good chance that the term could be used to differentiate the document from others. However, if the term also appears in a lot of other documents, e.g., *computer*, the importance of the term should be penalized. VSM usually normalizes vectors to unit Euclidean norm to compensate for differences in document length. During a retrieval operation, the query vector is compared to document vectors. Those closest to the query vector are considered to be similar and are returned. A common measure of similarity is the cosine of the angle between vectors.

2.3 Latent Semantic Indexing

Literal matching schemes such as VSM suffer from synonymy, polysemy and noise in documents. LSI has been proposed to address these problems. It uses singular value decomposition (SVD) to transform and truncate a matrix of term vectors computed from VSM to discover the semantics of terms and documents. For instance, although *car*, *vehicle*, and *automobile* are different terms, LSI may be able to discover that they are related in semantics. Intuitively, LSI transforms a high-dimensional term vector into a medium-dimensional *semantic vector* by projecting the former into a medium-dimensional semantic subspace. The basis of the semantic subspace is computed using SVD. Semantic vectors are normalized and their similarity is measured as in VSM.

In our P2P search engine *pSearch* [12], we describe how to extend VSM and LSI to work in P2P networks. In this paper, our focus is on LSI since it is more powerful. Some techniques developed for LSI here can be easily extended to work with VSM. We will describe profile matching using LSI and document dissemination

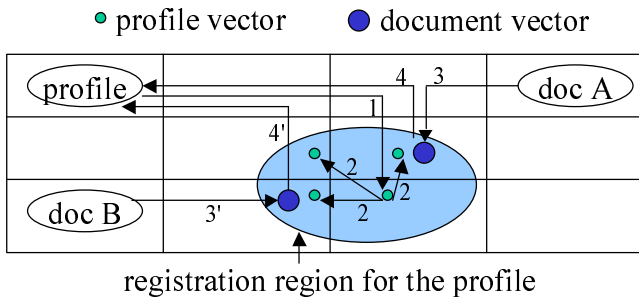


Figure 2: Profile matching in a 2-dimensional CAN.

in pFilter starting from the next section.

3. PROFILE MATCHING

As is mentioned earlier, pFilter connects a large number of computers into a structured P2P overlay network using an extension to CAN [8]. The advantage of adopting a completely decentralized P2P model is that the capability of the system scales with the amount of resources the system aggregates (in this case, the number of users). But it also poses a significant challenge with respect to efficient and accurate matching between profiles and documents in a decentralized environment. A naive solution that sends every document or profile to every node in the system is obviously not an option for systems running at pFilter’s scale.

3.1 The Basic Idea

In pFilter, through a careful combination of profile placement and document routing, a new document only needs to be matched against relevant profiles stored at a rendezvous node, which is identified by overlay routing based on the semantics of the document. In particular, pFilter uses the vector representations (*semantic vectors*) of profiles and documents, generated by the LSI algorithm, as the key to store them in CAN’s DHT. This has the effect that profiles or documents stored logically close in the overlay are also close in semantics, reducing the problem of semantic-based searches to routing in overlay.

Figure 2 illustrates this with an example. In the figure, each zone is owned by a node. Given a profile, it is routed in CAN using its semantic vector as the key (*step 1*), and registered at zones within a small radius based on a similarity threshold set by the user (*step 2*). We call these zones the *registration region* for the profile. Given a new document A, it is routed using its semantic vector as the key (*step 3*), and matched against profiles stored at the target zone z . Then a notification is sent to all matching subscribers (*step 4*). Because profiles similar to the document above certain threshold must have been registered at z , pFilter will not miss any matching profiles. In Figure 2, the same profile can also match with document B that is routed to another zone.

The efficiency of the above algorithm closely depends on the number of nodes whose zones intersect with the registration region of a profile, because every such node has to store a copy of the profile. If a large number of nodes are involved in registering a single profile, pFilter’s performance would degrade to a simple system where profiles are flooded to every node. Unfortunately, although the radius of the registration region is typically small (users usually prefer queries with tight selectivity), the registration region intersects with almost all zones in the system when the dimensionality

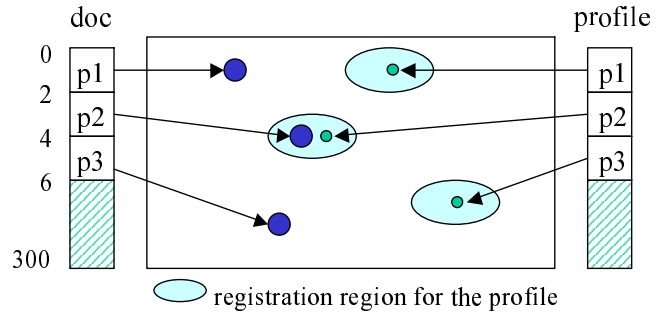


Figure 3: Multi-plane in a 2-dimensional CAN.

of the semantic vectors is high (100-300). This is the well-known *curse of dimensionality* [5].

3.2 Dispelling the Curse of Dimensionality

The curse of dimensionality is the holly grail in many fields such as database and data mining. Hundreds of papers [5] have been published on this subject, using data structures such as tree or grid to prune the search space. These algorithms, however, usually do not work well at hundreds of dimensions, and require changing CAN’s DHT abstraction. Our own experience in using space filling curve or distance-based dimensionality reduction techniques confirms that they are not effective at handling more than 10-20 dimensions.

In pFilter, domain-specific knowledge allows us to attack this problem at a lower cost. As a result of the singular value decomposition used in LSI, the elements appearing earlier in a semantic vector are more important than those appearing later. By aggressive cutoff, a semantic vector can be truncated to only a few dimensions to reduce the number of nodes involved in a profile registration, at the cost of precision.

We use a multi-plane scheme to reduce the dimensionality while keeping good precision. We partition lower elements of a semantic vector into multiple low-dimensional subvectors, with one subvector on each *plane*. The dimensionality of CAN is set to that of an individual plane. Each of the subvectors is used as the DHT key for routing. A profile is registered to nodes on each plane and a document is also routed on each plane. Subscribers found on each plane are notified. Note that the local matching between a document and a profile still uses the full-length semantic vector to keep precision. Local matching can take advantage of advanced algorithms that use data structures such as trees or grids to speed up the searching process [5].

A multi-plane example is shown in Figure 3. The first 6 elements of the vectors are partitioned into 2-dimensional subvectors on 3 planes. Both the document and profile are routed on every plane. In this example, they match on the second plane. Note that only a single CAN overlay is needed to support multiple planes.

The profile placement and matching algorithm is closely related to the pLSI algorithm used in our P2P search engine *pSearch* [12]. In pSearch, we find that using 4-6 planes with about 10-12 elements on each plane can be very effective. Figure 4 shows the result of a pLSI experiment that matches 50 queries against the 528,543 documents in the Text Retrieval Conference-8 (TREC-8) corpus [13]. The X axis is the number of nodes in the P2P overlay, the left Y

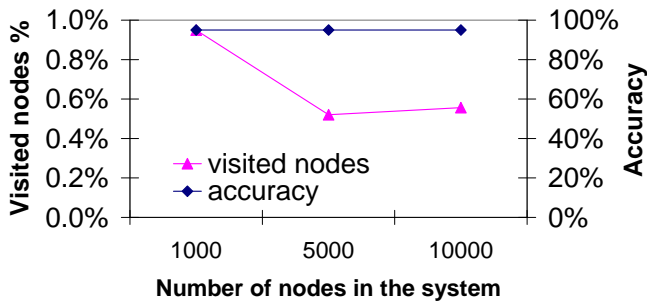


Figure 4: Performance of pLSI, using TREC corpus.

axis is the percentage of visited nodes, and the right Y axis is the percentage of shared documents between the documents returned by pLSI and LSI. We can see that pLSI is efficient as well as accurate. It only needs to visit about 0.4-1.0% nodes to achieve an accuracy of 95%. Translating this result into pFilter, it means that pFilter only needs to register a profile at about 0.4-1.0% nodes to achieve an accuracy nearly as good as a centralized information filtering system.

4. DOCUMENT DISSEMINATION

When a new document arrives, pFilter needs to deliver it to subscribers of all matching profiles identified by the algorithm described in Section 3. An efficient way to do this is to disseminate the document through an application-level multicast tree. To filter Internet-scale contents, pFilter employs three major optimizations in multicast tree construction:

- Utilizing a global view of the system to build and maintain efficient multicast trees;
- Reducing the number of multicast trees by fusing similar profiles;
- Reducing the routing states through discriminative treatment of the trees.

We describe each of them in the sections that follow.

4.1 Tree Construction Using Global States

The performance of a multicast tree, to a great extent, depends on how close the structure of the tree approximates the underlying Internet topology. The simplest way to construct an optimal multicast tree without router support is to measure the latencies between each pair of nodes in the system, build a graph with nodes as vertices and latencies as the weights of edges, and then run a spanning tree algorithm over the graph. Though simple, this centralized algorithm is not suitable for a distributed environment due to its excessive latency measurements.

4.1.1 Tree Construction Using the Closest Neighbor

The basic idea of the naive algorithm, however, is still valid. That is, it is crucial for the tree construction algorithm to have a *global* view of the system in order to build efficient multicast trees. We propose a more realistic variant of the naive algorithm, *pCast*. In pCast, when a node joins a multicast group, it attaches to a node that is closest among all existing nodes in the multicast tree. The global information pCast requires is the knowledge of the closest

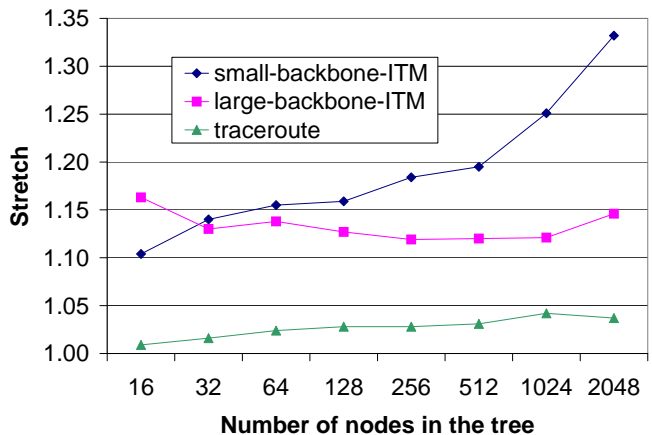


Figure 5: Performance of the pCast algorithm with optimal neighbor selection.

neighbor. For the time being, we assume that it is possible to find the closest neighbor in a decentralized fashion. We will describe such an algorithm in Section 4.1.2.

To give the readers a flavor of the quality of trees constructed by pCast, we compare the cost of a pCast tree to that of the minimal spanning tree. Here tree cost is the summation of the weights of all edges in the tree. The metric we use is *stretch*, defined as the ratio of tree cost for a tree constructed by pCast to that of the minimal spanning tree.

Three network topologies are used in the experiment. The first two topologies are of 10,000 nodes each, generated from GT-ITM [4]: one with a large backbone (228 transit domains) and the other with a small backbone (25 transit domains). The third topology is sampled from the Internet, using *traceroute* to measure latencies among 85,007 nodes.¹ For each topology, some nodes are randomly selected to join a multicast group. The results are shown in Figure 5, where the X axis is the number of nodes in the multicast tree, and the Y axis is the stretch. In the figure, with as many as 2048 nodes in the tree, pCast introduces less than 15% overhead for the *large-backbone-ITM* and *traceroute* topologies. The performance for the *small-backbone-ITM* topology is worse because its edge network is denser than the other two. In such a network, when a nodes N joins, it is more likely that some nodes already in the tree can benefit from choosing N as their parent (Section 4.1.4 and 4.1.5), as opposed to simply attaching N to an existing node. On the other hand, *stretch* only measures the relative performance. The absolute performance overhead of pCast in a small backbone network is not as significant as that in a big backbone network.

4.1.2 Finding the Closest Neighbor

The need to efficiently generate proximity information among nodes is crucial to all multicast tree construction algorithms. In our previous work [14], we show that just using local knowledge to generate proximity information is not effective, e.g., expanding-ring search. We propose a *landmark+latency measurement* algorithm *pNeighbor* to find close-by neighbors for a node in a decentralized fashion.

¹We thank Zhiheng Wang at University of Michigan for providing us with this topology.

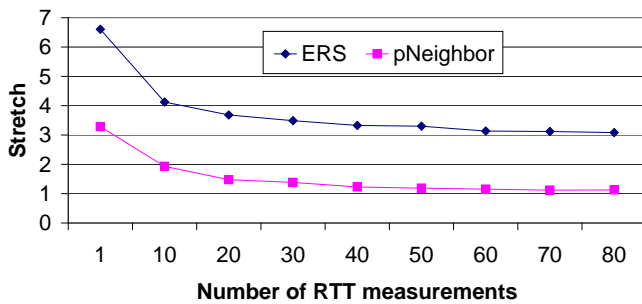


Figure 6: Performance of the pNeighbor algorithm.

The intuition behind the landmark technique is that nodes physically close to each other in the Internet are likely to have similar latencies to some selected *landmark* nodes. Suppose there are l landmark nodes L_i , $i = 1, \dots, l$ in the system, then each node has a *landmark vector* (d_1, d_2, \dots, d_l) associated with it, where d_i is the node's round trip time (RTT) to landmark L_i . To find a physically close-by neighbor in the overlay, a node ranks other nodes according to the similarity in landmark vectors, and measure RTTs to top k candidates to locate the actually closest one.

We evaluate the performance of pNeighbor using the same graph used in Section 4.1.1. Given a node A, we use pNeighbor to find its closest neighbor among all 10,000 nodes in the system. The metric we use is still *stretch*, but defined here as the ratio of the distance between a node A and its nearest neighbor found by pNeighbor to the distance between A and its actual nearest neighbor. We randomly select 1000 nodes in the graph and report their average stretch in Figure 6, where ERS is expanding-ring search, the X axis is the number of RTT measurements to top k candidates, and the Y axis is the stretch. For ERS, it measures k direct or indirect neighbors in the overlay. Two conclusions can be drawn from the figure. First, algorithms that only utilize local information such as expanding-ring search is not effective in finding the closest neighbor. Second, pNeighbor's performance improves quickly while the number of RTT measurements increases. It only needs to measure RTTs to about 20-40 nodes to achieve a stretch nearly as good as the ideal case.

4.1.3 Peer-to-Peer Tree Construction

In Section 4.1.1 and 4.1.2 we demonstrate the effectiveness of the pCast and pNeighbor algorithms through experiments. In this section, we describe how to extend pCast to work in a P2P fashion. The basic idea is to have each node independently discover a close-by node that subscribes to the same query, using the landmark vector of the node or the semantic vector of the profile as keys to access CAN's DHT. In the following, we first give the steps of the decentralized pCast algorithm and then illustrate it with an example.

When a node N registers a profile, it carries out the following steps:

1. N measures RTTs to landmark nodes and include its landmark vector, CPU/storage capacity, and current load in the profile;
2. N uses the *landmark vector* as the DHT key to store the profile in a registration region \mathcal{L} of the overlay.² This process

²The number of landmarks may not match with the dimensionality of CAN. Solutions for this problem is described in [14].

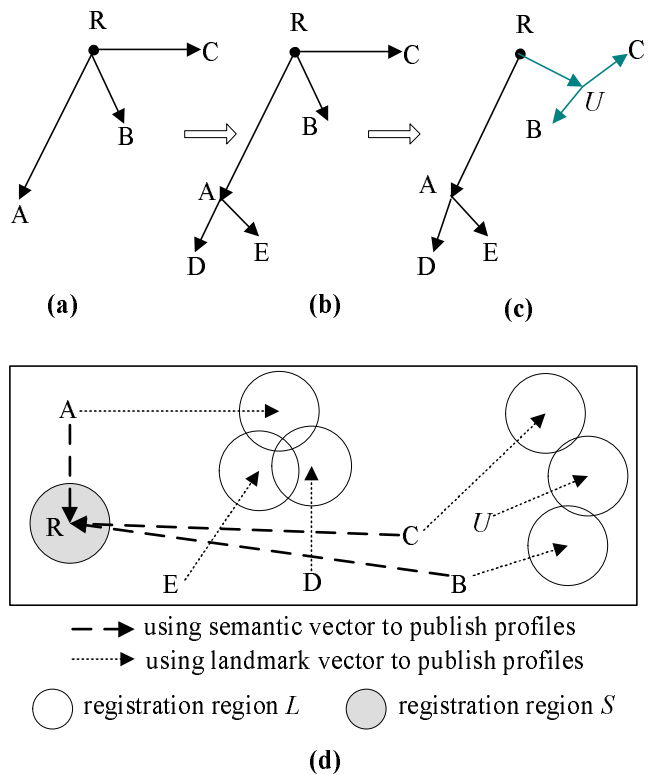


Figure 7: Publish a query under both semantic vector and landmark vector.

is similar to the step 1 and 2 in Figure 2 but the routing is guided by the landmark vector instead of the semantic vector.

3. At nodes within the registration region \mathcal{L} , it searches for profiles of close-by nodes that already subscribe to the same query. If such nodes exist, it uses the pNeighbor algorithm to find the closest one, attach to it, and the registration process terminates.
4. If no close-by nodes are found in step 3, N uses the *semantic vector* as the DHT key to store the profile in another registration region \mathcal{S} of the overlay. This process is the step 1 and 2 in Figure 2.
5. At nodes within the registration region \mathcal{S} , it again uses the pNeighbor algorithm to find the closest node that already subscribes to the same query and attaches to it. If no such node is found, N must be the first node that subscribes to this query. It simply attaches to the root.

We illustrate this algorithm with an example in Figure 7. To make the example intuitive, we draw the distances in Figure 7 (a)-(c) proportional to physical distances among nodes in the Internet. In Figure 7 (d), the positions of nodes in the graph corresponds to their positions in CAN's logical Cartesian space, and the circles represent the registration regions for profiles.

In Figure 7 (a), three subscribers A, B, and C already join the multicast tree. In Figure 7 (d), they use the semantic vector and landmark vector as the DHT key to store their profiles in registration region \mathcal{L} and \mathcal{S} , respectively. In Figure 7 (b), subscriber D and E join. In

step 3 of the algorithm, D and E notice the profile A left there and directly attach to A, skipping step 4 and 5 of the algorithm. (The use of node U in Figure 7 will be explained in Section 4.3.)

Comparing with traditional schemes, our tree construction algorithm offers the following advantages: (1) Better proximity approximation: the parent selection for a new node is not constrained by the structure of the tree; (2) Faster tree construction: the parent selection process avoids searching the tree level by level, sequentially; (3) No bottleneck at the rendezvous node R in Figure 7, which is typically a problem for existing publish/subscribe systems. It has been shown that 25 most popular queries account for over 1% of all queries submitted to search engines [6]. For these hot queries, a new subscriber is likely to find a close-by node that already subscribe to the same query in step 3 of the algorithm, avoiding going to R in step 4.

To increase the probability that a subscriber is satisfied by a close-by neighbor in step 3 of the algorithm, we can condense the registration region by using only a subset of nodes in the overlay to store proximity information. More details are available in [14].

Except its direct children, the rendezvous node R does not need to remember every registered profile to function properly. When its load increases due to the excessive publishing of profiles, it can ignore incoming profiles whose landmark vectors are similar to the landmark vectors in known profiles, after directing the owners of those profiles to attach to an appropriate place. However, we expect it is not a problem for a node to record several thousand profiles.

4.1.4 Basic Tree Maintenance Operations

In Figure 5, the performance of the tree constructed by pCast is not as good as the minimal spanning tree, and the performance gap widens as the number of nodes in the tree increases. This is because pCast only attaches a node to its closest neighbor without fixing the edges that already exist in the tree. When the network conditions flux, maintenance of the tree also needs to be done to adapt the tree structure to the changing environments. We refine the pCast algorithm to address both problems.

When node N joins a multicast group and finds the closest node X in the current tree, it establishes links to both X and X 's parent P . The distances between (P, N) , (N, X) , and (P, X) are measured, and the link with the largest distance is cut. (In reality, the decision can be made also taking QoS requirements into consideration.) For each of X 's children and P 's children, we incrementally exam them to see if each of them is better to list it as a child of N .

We illustrate our algorithm with an example. Similar to Figure 7 (a)-(c), we draw the distances in Figure 8 proportional to physical distances among nodes in the Internet.

In the very beginning, A joins and the closest node to A is R , and hence a link is established between R and A . When B joins, it again find that R is the closest node and connects to R directly. (see Figure 8 (a).)

When C joins, it finds that the closest node is A , and links are established between R and C and C and A . The distances between (R, A) , (R, C) and (C, A) are compared, and the link with the largest distance (R, A) are broken. (see Figure 8 (b) and (c)).

Then node D joins, and finds out that B is the closest node. Two

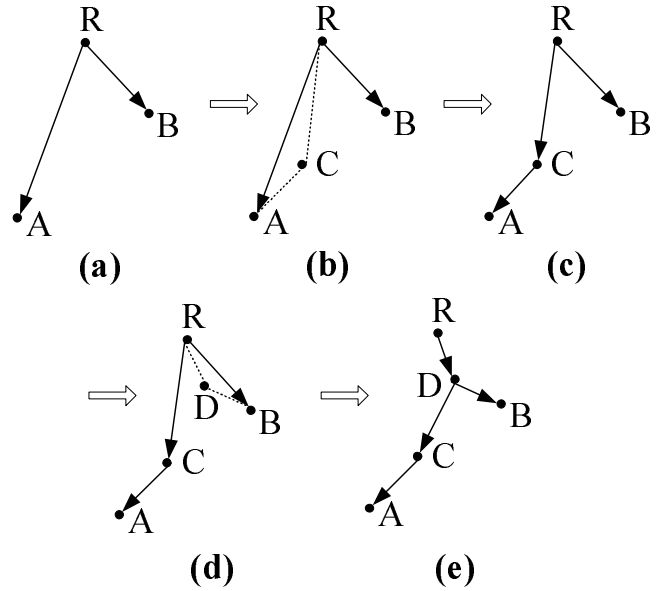


Figure 8: Tree maintenance operations.

links (R, D) and (D, B) are established and the link with the largest distance (R, B) is then broken. We now exam the children of R and B and find that it is beneficial to link C as a child of D . The result is shown in Figure 8 (e).

In pFilter, nodes in the overlay replicate their neighbors' contents. If a node A 's parent in the multicast tree fails, A restarts the profile registration process to attach to a close-by node. A 's descendants could be ignorant of this repair. If the root R of a tree fails, one of R 's neighbors Q will notice the failure and take over the zone owned by R , along with the tree rooted at R . Q uses the profile registration process to find the closest node in the tree, attaches to it, and announces that it is the new root of the tree. This announcement is propagated throughout the tree to change the directions of edges properly.

4.1.5 Advanced Tree Maintenance Operations

The capability of the tree maintenance algorithm described in the previous section is still limited. We describe a more advanced algorithm in this section. This algorithm can be carried out optionally and independently by nodes to improve performance.

Either at join or periodically, a node N use pCast's registration process to find a set of nodes that subscribe to the same query and have landmark vectors similar to that of N . N measures RTTs to top k candidates and identifies l nodes that are actually close. Both k and l are tunable parameters based on the efforts N is willing to spend. We use \mathcal{W} to denote the closest l nodes, their parents, N and N 's parent. N locally builds a graph consisting of nodes in \mathcal{W} and run a spanning tree algorithm over the graph to find a tree structure that can improve performance.

We illustrate the details of the algorithm using Figure 9. At the beginning of the algorithm, N finds three closest nodes A , B and C and builds a graph as shown in Figure 9 (a), where P_x is X 's parent and R is a conceptual root of the tree. Since N does not know the structure of the rest of the tree, it can only add three *virtual links* between (R, P_a) , (R, P_b) and (R, P_c) to annotate that these

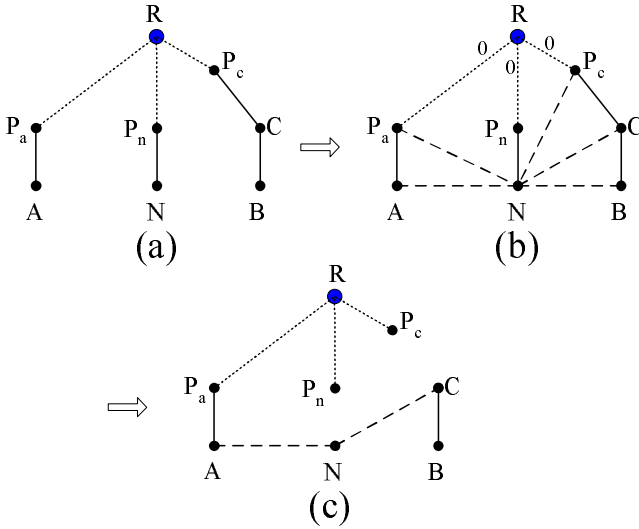


Figure 9: Advanced tree maintenance operations.

nodes are somehow connected through the root. In Figure 9 (b), N adds a link to every node in \mathcal{W} and annotate the links with measured latency. These latencies are either measured by N directly or reported by other nodes upon N 's request. The latency of virtual links is set to 0. N runs a spanning tree algorithm over the graph to find the best tree structure and informs other nodes to adapt accordingly. The result is shown in Figure 9 (c). Note that the virtual links are always included in the minimal spanning tree since their weights are 0.

To guarantee that the result of this algorithm is still a tree that connects all nodes in the original tree, a virtual link is added between R and a node X if and only if X is in \mathcal{W} and it is not a descendant of any other nodes in \mathcal{W} . For instance, although C is B 's parent, there is no virtual link between R and C . To discover the ancestor-descendant relationship, the root periodically propagates a *hello* message throughout the tree and each node adds itself into the message. Each node can learn its ancestors from the message independently. The message can also be used to detect loops in the tree due to rare race conditions in the tree construction process.

4.2 Fusion of Profiles

To handle potentially billions of registered profiles, it is not affordable to maintain one ordinary multicast tree for each unique profile. For queries that are similar enough, pFilter builds a single multicast tree for them. Assume that each profile is represented by (q, r_q) , where q is the semantic vector of the query, and r_q is the similarity threshold set by the user. Any document whose semantic vector is within r_q distance to q are considered as a match to query (q, r_q) .

There are two cases that queries can be fused, and we illustrate them in Figure 10. In Figure 10 (a), query (a, r_a) covers query (b, r_b) . We can simply add subscribers for (b, r_b) to the (a, r_a) multicast tree, with the guarantee that nobody will miss interesting documents. In Figure 10 (b), query (c, r_c) and (d, r_d) do not cover each other, but we can build an artificial query (e, r_e) to cover both. In Figure 10 (c), the multicast tree is dedicated for query (c, r_c) . In Figure 10 (d), when node D registers its query in CAN's DHT using d as the key, it notices that there exists a multicast tree for similar query (c, r_c) . Instead of building a new multicast tree, it tries to

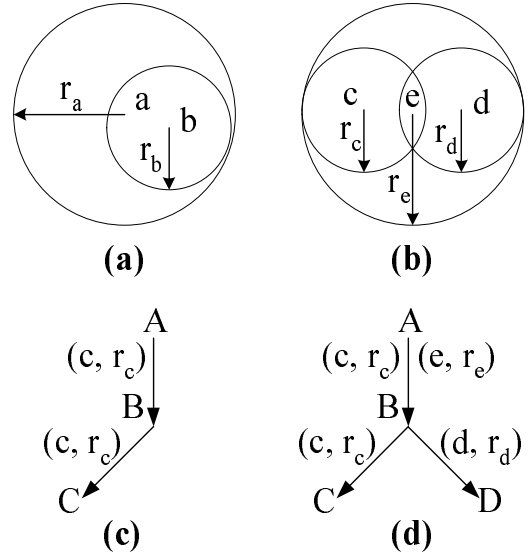


Figure 10: Two cases that profiles can be fused.

attach to the existing multicast tree by informing a physically close-by node B that is already in the multicast tree (how to find such a node is described in Section 4.1.3). B then informs A to increase the query region covered by this multicast tree. A in turn propagates this notification to its parent.

In a multicast tree for fused queries, a node must compare an incoming document with each of its children's interests to decide whether to forward it along certain links. For instance, in Figure 10 (d), when B receives a document only matches with (c, r_c) , it forward the document to C , ignoring D .

4.3 Discriminative Treatment of the Trees

Existing multicast algorithms maintain too many routing states for each individual multicast tree, because of the lack of discrimination among subscriptions. (It has been shown that the popularities of queries submitted to search engines are highly biased [6].) For instance, in Scribe [11], every node on the route connecting a subscriber to the root of the topic has to maintain some states for the subscription.

In addition to profiles fusion, in pFilter, the states devoted for a multicast tree is a tunable continuum depending on the popularity of the profile. There are two extremes. For a very popular profile, pFilter selects nodes from all nodes in the system to build the most efficient multicast tree, whereas for unpopular profiles, which are the majority, matching documents can be sent directly the corresponding subscribers using point-to-point communication. If systems like Scribe is used, a lot of multicast trees would have dozens of internal nodes that are not interested in the subscriptions, with only several or even a single real subscriber(s) at leaves.

In Figure 7 (c), as the traffic in the multicast tree increases, B wants to pull in a close-by high-bandwidth node to improve performance. It will discover U because they have similar landmark vectors and their registration regions intersect (see Figure 8 (d)). If no close-by nodes can help improving performance, B computes the landmark vector of an ideal node V such that the summation of distance between (R, V) , (V, B) and (V, C) is likely to be significantly less than

the summation of distance between (R, B) and (R, C). B then uses the ideal landmark vector as the key to access the DHT to find such a node. For very important (high traffic) trees, every node in the overlay can be pulled in to build the most efficient tree.

5. DISCUSSION

We discuss some issues not covered in Sections 3 and 4 in this section.

Audio/Video. pFilter works by representing contents as vectors and organizing profiles around these vector representations. This method can be applied to any media that can be abstracted as vectors and have its object similarity measured as some kind of distance in the vector space. A lot of pattern recognition problems fall into this category. For instance, people also employ singular value decomposition to extract algebraic features from images, and use various extractors to derive frequency, amplitude, and tempo feature vectors from music data.

Information crawler and gateway. The ultimate goal of pFilter is to involve millions of machines to build a large-scale information filtering system. Users of pFilter can voluntarily publish their contents as people do for Web today. But we can imagine that not all content sources are able to (or willing to) participate in pFilter due to reasons such as firewall segregation or security concerns. In addition to voluntary content publishing, pFilter can also have some nodes act as information crawler to collect contents and publish them in pFilter, as Web crawlers do for search engines. Some nodes can also act as information gateway, forwarding information they receive from various channels to pFilter, e.g., discussions in mailing list, stock prices, and so forth.

Duplicate elimination. There are several reasons that a single document may be delivered to a node multiple times. A document may be injected into the system by different users that think it is interesting. Even a document injected by a single user can travel through multicast trees on different planes (Section 3.2) and a node sits at the conjunction of these trees will receive it more than once. To eliminate duplicates, which is both annoying and inefficient, we can ask each node to cache the digital signature of documents that it forwarded recently and discard documents that appear to be duplicates judging from the signatures.

Advanced matching. Besides LSI, many other searching techniques can also be adopted in pFilter. For instance, a profile can specify that only contents injected by certain persons, collected from certain Web sites, or created within certain time frame are allowed to be delivered to the user. More advanced matching includes the example described in Section 1, detecting if a document is a citation to a given paper. These advanced matching conditions can be checked at every forwarding step in the multicast tree, or after it reaches a subscriber. Executing the check in the multicast tree may require associating mobile codes with profiles. We think that adding a final filtering step at the subscriber may be more practical.

Searching structured documents. We have concentrated on filtering and dissemination of unstructured documents. Many of our techniques can be used for structured document as well. In database systems, data are organized according to well-defined schemas. In pFilter, we can use a *schema vector* to abstract a schema, mapping an attribute in a schema to an element in the schema vector. Because there are potentially many different schemas, different schema vectors can be of different dimensions and the elements in

a schema vector can be of different types. Publishing, matching, and fusion of relational data can be done in a similar manner as we do for the semantic vectors. For XML documents, matching is typically performed based on DTDs or Xpaths. This typically involves XML structure (tree) matching. Besides introducing distributed tree matching algorithms, one way to take advantage of the techniques proposed in this paper is to first use pFilter to retrieve matching documents from distributed sources based on contents, and then run a conventional XML structure matching algorithm locally to refine the results.

6. RELATED WORK

There is an enormous body of related work—far too many to enumerate in this paper. Here we give only references to surveys of related fields, and focus on those most related. Surveys could be found on P2P [7], information retrieval and filtering [10], and publish/subscribe [9].

Information retrieval and filtering are well-studied fields. VSM and LSI [2] are among the most successful IR algorithms, and are adopted by major search engines [6]. But publications are mainly focused on the precision of IR algorithms, rather than their efficiency and scalability. SIFT [16] does pay attention to efficiency but it runs at small scale, only filter messages in news groups, and do not use multicast for efficient document delivery.

Existing content-based publish/subscribe systems only need to match several well-structured attributes against values, while pFilter must match unstructured documents and profiles expressed in natural languages, using vectors of hundreds of dimensions.

P2P publish/subscribe systems such as Scribe [11] only support simple subject-based subscriptions. Their performance is even more questionable than conventional publish/subscribe systems, in our opinion, because the edges of the multicast tree must be routes in the P2P overlay. As demonstrated in [14], the performance of P2P routing is inherently inferior to IP routing, because of the constraints of the DHT abstraction.

7. CONCLUSION

In this paper, we describe pFilter, a global-scale decentralized information filtering and dissemination system for unstructured information, such as text, HTML and multimedia files. pFilter connects potentially millions of computers in national (and/or international) computing Grids or user desktops into a structured peer-to-peer overlay network. Nodes in the overlay collectively publish/collect documents, build index, register profiles, and filter and disseminate information.

To our knowledge, pFilter is the first large-scale decentralized information filtering and dissemination system. In pFilter, we made the following contributions:

- Avoid the flooding of either documents or profiles through a careful combination of profile placement and document routing, achieving both efficiency and accuracy;
- Experiment with various searching algorithms for high-dimensional data, identify their weakness and devise the multi-plane dimensionality reduction algorithm that takes advantage of domain-specific knowledge;

- Utilize a global view of the system to build and maintain efficient multicast trees;
- Reduce the number of multicast trees by fusing similar profiles;
- Reduce the routing states through discriminative treatment of the trees.

We already built pFilter's basic components separately: a LSI-based information filtering system that extends the SMART package developed at Cornell [3], a topology-aware P2P overlay and its simulator [14], and a simulator for the multicast tree construction algorithm pCast. Currently we are in the process of integrating these pieces together to build a pFilter prototype.

REFERENCES

- [1] M. K. Bergman. The deep web: Surfacing hidden value. <http://www.brightplanet.com/deepcontent>.
- [2] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.
- [3] C. Buckley. Implementation of the smart information retrieval system. Technical Report TR85-686, Department of Computer Science, Cornell University, Ithaca, NY 14853, May 1985. Source codes are available at <ftp://ftp.cs.cornell.edu/pub/smart>.
- [4] K. Calvert, M. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, June 1997.
- [5] V. Gaede and O. Gunther. Survey on multidimensional access methods. Technical Report ISS-16, Institut für Wirtschaftsinformatik, Humboldt Universität zu Berlin, August 1995.
- [6] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. In *VLDB*, 2001.
- [7] D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Lab, 2002.
- [8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM'01*, August 2001.
- [9] A. Rifkin and R. Khare. A survey of event systems. <http://www.cs.caltech.edu/~adam/isen/event-systems.html>.
- [10] C. O. Riordan and H. Sorensen. Information filtering and retrieval: An overview.
- [11] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.
- [12] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *HotNets-I*, Princeton, NJ, October 2002. Available at <http://www.cs.washington.edu/hotnets>.
- [13] Text Retrieval Conference (TREC). <http://trec.nist.gov>.
- [14] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. Technical Report HPL-2002-281, HP Labs, September 2002. Submitted for publication, available at <http://www.cs.rochester.edu/u/sarrmor>.
- [15] Z. Xu and Z. Zhang. Building low-maintenance expressways for p2p systems. Technical Report HPL-2002-41, HP Laboratories Palo Alto, 2002.
- [16] T. Yan and H. Garcia-Molina. SIFT—A tool for wide-area information dissemination. In *Proc. 1995 USENIX Technical Conference*, pages 177–186, New Orleans, 1995.