Peer-to-Peer Result Dissemination in High-Volume Data Filtering

Paul Burstein  Shariq Rizvi

Computer Science Division
University of California, Berkeley
{burst, rizvi}@cs.berkeley.edu

Abstract

Data filtering is the problem of matching high-volume document streams, against a set of client profiles, often represented by queries. Recent work in high-volume data filtering, like the YFilter project at Berkeley, has focused on efficiently indexing client queries, by exploiting the similarity between queries. However, the problem of result dissemination in data filtering has not been received enough attention. Each filtered document has to be delivered to a unique, and typically large set of clients, making this a problem of highly dynamic multicast. We observe fundamental limitations in the client-server delivery model for the high-volume data filtering problem and describe a peer-to-peer scheme to solve it. It exploits the bandwidths of participating clients for data dissemination by building an unstructured overlay network. Our deployment on the PlanetLab testbed shows that the approach proposed is scalable and offers acceptable delivery delays and network economy.

1 Introduction

With the growth of the internet, information is abundant. Clearly, not all the information is of interest to all clients. There are two ways a client can access relevant information on the internet - using “pull” or “push” of information. The former happens when the client visits a website and accesses the information she wants. On the other hand, the “push” paradigm refers to a model where an information provider sends “relevant” information to the client - e.g. a newsgroup message delivered to an e-mail box. Roughly, there are two possible granularities at which the “push” paradigm can operate:

1. **Coarse-grained:** When the client has only limited ways to express her interests. The best example of this is newsgroups, where the client subscribes to groups of her interest and sources publish to relevant newsgroups. Then, a client receives only the information posted on the newsgroups she has subscribed to.

2. **Fine-grained:** This includes systems that give the clients fine-grained means of expressing their interests, e.g. by queries over data objects.

In this paper, we address issues that a fine-grained data filtering system faces, in the presence of heavy continuous data flow and a huge client base. We borrow workload and client characteristics from YFilter [4] which is an XML data filtering engine developed at Berkeley. We show that under a heavy data flow rate and a huge client base, YFilter faces two kinds of bottlenecks - processing and result dissemination. We then focus on the second of these bottleneck and argue that it raises the interesting problem of highly dynamic multicast. We show that with the numbers we are trying to target, a client-server model of data delivery fails to deliver. Then, we propose an application-level multicast scheme that exploits the bandwidths of clients for data delivery. The approach works by building an unstructured overlay network over the clients, and then performs result dissemination in a peer-to-peer fashion.

This paper is organized as follows. Section 2 presents a sketch of YFilter, and describes the problem of result dissemination. Then, in Section 3,
we discuss content-based routing, which represents a possible approach for addressing the bottlenecks we cite in a high-volume data filtering system. However, we discuss why such approaches are not very scalable and efficient. Section 4 discusses some current approaches for application-level multicast, which is closely related to our proposal. Section 5 presents our proposed solution for the problem, which is an application-level multicast scheme but goes beyond the concept of traditional multicast, which uses “multicast groups”. Section 6 evaluates the proposed scheme by means of a deployment over PlanetLab. Sections 7 and 8 present future work and conclusions respectively.

2 Data Filtering and the Two Bottlenecks

The problem of data filtering (Figure 1)\(^1\) deals with matching data objects produced by data sources against profiles of clients and delivering the clients with all the objects that are relevant to their interests. In current day scenarios, XML has become the “wire format” for information exchange between services and clients and between applications. Prominent XML-based services include RSS news feeds from various websites. YFilter [4] addresses the problem of filtering a typically high-volume stream of XML documents based on the interests of clients. Client interests are expressed using queries issued by the client. YFilter tries to minimize the time taken for filtering a document against a set of XPath queries by efficiently indexing these queries. It also exploits the similarity between queries by doing shared work over them (In the context of XML, this corresponds to exploiting shared paths in the XPath queries).

Currently, to filter a 20KB document against an index of 100,000 simple XPath queries, YFilter takes about 50ms. This limits the throughput of YFilter to 20 documents per second. Thus, processing will be a bottleneck for a YFilter installation that targets higher throughputs. Interestingly, while data filtering has focused on efficient representation and indexing of client queries for high performance, the problem of result dissemination has been ignored. With an incoming document rate of 20 documents per second, and an average document size of 20KB, if the YFilter has to deliver each document to an average of 10,000 clients, a very practical number in the presence of a million clients, the bandwidth needed is of the order of several Giga bits per second! \((20 \text{ docs/sec} \times 20\text{KB/doc} \times 10,000 \times 8 = 32 \text{ GbPS})\) Thus, result dissemination represents the second bottleneck for YFilter under high throughput requirements.

In this paper, we address the second bottleneck cited above. Clearly, the requirement of an impossibly high bandwidth makes result dissemination a more compelling bottleneck than processing itself. Our model works with the following assumptions:

1. The clients don’t want any irrelevant information. In other words, any approach that pushes the filtering functionality to the client end is unacceptable.

2. A possible way to remove the result dissemination bottleneck at the filter is to filter documents but publish only the links pointing to document sources, to the client. This will require much less data to be sent out by the filter. However, unless the number of sources publishing to the filter is high, this results in pushing the bottleneck to the sources. Hence, the problem at hand has not been solved. We assume that such a solution is unacceptable.

Under the above two assumptions, the problem reduces to filtering documents against client queries and disseminating the same to the clients straight from the filter. Given the bandwidth requirements described above, it is not possible to perform dissemination in a point-to-point manner using the client-server model. Hence, we need solutions that can “distribute” the load of result dis-

\(^1\)Borrowed from [4]
Fig. 2: Mesh-Based Content Routing

The dissemination over other nodes of the network, rather than just the filter. In the next section, we survey some interesting work on content-based routing that solves this problem by distributing the filtering functionality over several nodes in the network. Then we describe some issues in the scaling and maintainability of this approach.

3 Content-based Routing

Content-based routing (CBR) refers to routing of messages in a network, in a way that is quite different from IP routing. In CBR, messages are routed based on the content of the message and not the IP address of the message. A given message does not know its final destination. Instead of matching an IP address with an entry in a routing table to find the next hop towards the destination, the content-based router matches the message’s contents with some application-level information (like client queries) in order to find the next hop. This has been described as “semantic routing”.

Content-based routing has been exploited for data filtering [1]. The idea is to build an overlay network that uses the current internet as a substrate network. This overlay network holds query summaries at various router nodes and documents get filtered as they flow over this overlay.

A relevant example of such an approach is mesh-based content routing described in [7]. Their data streams are comprised of a sequence of XML packets and are forwarded by application-level XML routers. An XML router node performs content-based routing of individual XML packets to other routers or clients based upon queries that describe the needs of downstream nodes. They describe a protocol for delivery of redundant XML streams which assembles incoming packets based on the first copy of the packet delivered by any stream. This makes the system resilient to packet loss up to a certain degree. The approach is illustrated by Figure 2. Here R1 and R2 are root routers while I1, I2, I3 are inner XML routers. C1, C2 and C3 are client nodes.

However, content-based routing methods suffer from the following drawbacks:

1. Efficient query aggregation for building XML routing tables is an issue. The method described in mesh-based content routing [7] is to make every router hold a summary that is a disjunction of the queries on the child node downstream. This causes redundant work in parsing and matching each message several times against a query as the message flows over the mesh. An algorithm for efficient tree pattern aggregation has been proposed in [3] which considers the problem of generating summaries for queries under restrictions of space constraints and a maximum loss in precision due to aggregation. It makes use of document distribution statistics. However, even with such algorithms used for query aggregation, each message will have to be parsed at each router it goes through.

2. In scenarios where client queries change often, the state on each router may have to be changed too many times. This may lead to an unacceptable load on the system.

3. Similar queries are spread all across the network. This goes against the basic theme of keeping similar queries close to do shared work across them (as in YFilter).

4. The approach requires control over several router-hosting machines across the internet. This makes it expensive.

In contrast to the above, our scheme permits result dissemination in high-volume data filtering within the centralized filter framework. We make

\[^2\text{Borrowed from [7]}\]
only the result dissemination distributed, and not the filtering. This ensures that query similarity can be exploited by generating efficient summaries, and documents don’t have to be parsed multiple times. In the next section, we discuss some current application-level multicast issues, which are closely related to our work.

4 Application-level Multicast

Multicast is a mode of communication when one source needs to send some content to several clients. This can be performed at the IP level, which perhaps builds the multicast tree in the most network economic way. However, IP multicast has some disadvantages, when applied to the global scale:

1. Each router in the world has to be made multicast-aware. These routers should also be able to relate all multicast groups with the subscribers to these groups.

2. It is a rigid approach, compared to application-level multicast, which can be set in place by just downloading and running a piece of software.

However, application-level multicast performs poorly on the network economy metric as each hop at the application-level is multiple hops at the IP level. This means that, two application-level hops may be traversing the same IP-level hop redundantly. However, the flexibility and ease of application-level multicasting has lead the proposals like Overcast, SplitStream, and Bullet, which we describe next.

Overcast [5] provides a protocol for the building up of an application-level multicast tree. An assumption here is that all clients want the same data. The tree can adjust shape based on network congestion conditions. Additional features include archiving, to allow newly joining nodes to catchup on data, with the rest.

SplitStream [2] addresses the issue of the inner nodes in the multicast tree being too congested when compared to the leaf nodes, in Overcast-like approaches. It splits a data stream into redundant streams and each stream is multicast using a different multicast tree. The idea is to make sure that no one node is playing an inner node in lots of trees.

Finally, Bullet [6] argues that the best overlay organization for multicast is a mesh, rather than a tree.

However, our problem is more complicated than these systems, as not all clients want the same data. So, building a static tree or mesh will not help. Effectively, each document at the filter demands its own multicast group, as it goes to a specific set of clients out of the total client population. We call this “dynamic multicast”. In the next few sections, we will propose and evaluate a peer-to-peer scheme for result dissemination in high-volume data filtering systems. It works by building an unstructured overlay over the participating clients.

5 Proposed solution

Like some previous approaches, we use application-level multicast. What the existing overlay multicast systems provide are statically formed dissemination trees that include the intended recipients. Usually there are optimizations introduced to address the problems of restricted bandwidth, extended load and reliability. The problem we have at hand requires a different kind of a multicast structure, one that is dynamically formed. Since the dissemination group is produced on the fly at the data filter, predefinition of a routing structure is not possible.

We propose a scheme in which the filter, knowing all the recipients of a document, forms the tree structure on the fly just prior to sending the document. The tree is formed based on some heuristic and the tree structure is forwarded together with the document (Figure 3). At the filter, we wrap each document with information that contains the identities of all its intended recipients. Upon receiving a document, each client extracts the IP addresses of its intended child-clients in the dynamic multicast tree. Then it delivers the document to each of its children along with suitable identity lists for the clients downstream (we implement this by splitting the remaining IP addresses equally among the children). Thus, a client receives IP address information about all clients which belong to the subtree rooted at it. It should be noted that this IP address overhead decreases exponentially as the document gets delivered down a tree.

Every client offers a fanout based on its avail-
Figure 3: The Proposed Scheme

able outgoing bandwidth. This represents how much data a client can forward if a document is sent to it. We utilize this metric in tree construction. Clients with higher fanouts are placed higher up in the tree if we want to produce trees with small heights. Producing dissemination trees of small heights is desirable as it reduces the average document delivery delays experienced by the clients. The strategy that the filter implements involves ordering the dissemination list based on the recipient fanouts, breaking up the list and sending the document to the first-level clients, who happen to be high-bandwidth nodes. As the document propagates down the tree, the individual subtrees are formed from the dissemination list.

5.1 Scaling

Including every recipient’s address information with the document seems to induce a lot of overhead in our protocol. We need to keep the following information for every client: the IP address and the port number on which the recipient is running. This accumulates to six bytes of data per client. With expected document size of 20KB, the protocol overhead is 60KB with ten thousand receivers for a document. However, this is true only at the filter. Further down the tree the protocol overhead reduces exponentially with the current depth. With such scaling, for 100 million clients, with a binary dissemination tree, where half of the client base needs to be able to support a fanout of two, the worst case protocol overhead is less than 1% percent (relative to the document sizes). We feel that for an application-level solution to this problem, such an overhead is acceptable.

5.2 Client Failures

We are being highly reliant on all of the document recipients to be stable as they receive and forward these documents. This scheme can fall prey to churn. Two things may happen when disseminating a document through a dynamic multicast tree.

1. A client may fail to respond when it parent tries to connect to it. We remove such a client from the dissemination list, on the fly.

2. The second type of failure that might occur is when a forwarder itself fails after it receives a document but before it has forwarded it. Such a failure results in document loss for the whole subtree rooted at this client.

To handle the first kind of client failure, where a client does not respond when its parent tries to connect to it, we incorporate two schemes. When a forwarding client realizes that the next level recipient has failed, it simply removes the client from the
dissemination list. The ability to do this comes as a benefit of having the full dissemination list at the forwarding client. However, this scheme increases the overall delivery delay, since the discovery of a failed receiver usually involves some timeout. This tradeoff is explored in the following section, where we discuss the implementation and experimental results in more detail. The second scheme that can be employed is one of setting TTLs (time to live) on the client queries held at the filter. The clients may be required to periodically re-register, to make sure that the filter does not continuously include clients that have departed, as part of multicast trees.

The second kind of failure described above, when a client fails after receiving a document, but before forwarding it, is expected to be rare. We choose not to address this. A possible way to extend our system, is to bring in acknowledgements or retransmissions to deal with this problem.

5.3 Other Heuristics

Till this point we have described a solution that relies solely on one heuristic - based on the fanouts of clients. However, our scheme can be extended to use other heuristics on which we may base our tree-construction decisions. The optimal routing decisions would be based on the underlying network topology and would be highly similar to IP multicast. However, this is not easy to implement in the application layer and we do not attempt this. What we attempt is to make trees based on coarse-grained network topology. This particular heuristic is evaluated in the experimental section.

6 Evaluation

In this section we discuss our system implementation, explain our experimental setup and present our results. We implemented our solution as an application running over TCP. This relieves us from the burden of worrying about the reliable delivery of documents from client to client. The following are the possible metrics for evaluating our solution.

1. Document Delivery Delay: This represents the time taken for the document to reach a particular client, after being filtered.

2. Network Economy: The outgoing bandwidth consumed at each client during result dissemination.

3. Document Loss: A measure of how many documents are delivered successfully to the recipients.

4. Out-of-order Delivery: Given a scheme like ours, two documents being sent to the same client may reach her out of order. For some application semantics, this may be an issue.

In our evaluation, we pay attention to only the first three metrics. The dissemination of a document through a tree structure over TCP may produce long delivery delays, which is one of the metrics that we seek to minimize. We have to be careful about timing out when sending to a client which may take longer than expected. This delays delivery of the document to the clients in the subtree. Here we have to make a tradeoff between reliable delivery and reasonable delivery delay. We explore this tradeoff in our experimental section.

6.1 Experimental Setup

We deploy our system on the PlanetLab testbed. PlanetLab provides us with over two hundred hosts geographically distributed across the globe. It should be noted that the problem we are attempting to solve targets hundreds of thousands of clients and yet we are reducing our experiments to only a few hundred hosts. We are limited by the resources available to us.

We use the following setup in our experiments. The numbers are modeled over those from YFilter. The document size is set to 20KB, which seems to be realistic for an XML news message, wrapped in SOAP headers. The average query selectivity is 10%. We model a client’s query selectivity as an exponentially distributed random variable, with mean 10%. We set our document generation rate at 1 document per second. This allows us to generate a substantially large data flow without overloading the testbed. For the filter we set the fanout to 2, this is a low value, as we want to put minimum load on the filter’s bandwidth. We chose a PlanetLab host at MIT to run the filter.
A tougher challenge is to model client fanouts. We expect the client base to be composed mostly of end users connected to the Internet through home connections via modem, DSL or cable modem. With the considered document generation rate, document size and query selectivity, the incoming data rates at a client are expected to be around 16Kbps (20KB/doc x 1Doc/Sec x 0.1 (selectivity)). With the protocol overhead included, we conservatively assume that a 56Kbps dial-up client will be capable of forwarding a document to only one other client. Likewise, we conservatively set the fanouts for DSL and cable modem clients to 2 and 4 respectively. We model the client population as being composed of 20% dial-up users, 40% DSL users and 40% cable modem users. To show scalability, we run experiments by varying the number of virtual clients running on each PlanetLab node from 1 to 10.

6.2 Experiments and Observations

We perform four different runs on PlanetLab varying the number of clients from 200, to 400, 1000, and 2000. The first thing we observe is the average delivery delay that clients experience when receiving documents. The delivery delay is strongly correlated with the height of the multicast tree. The expected tree heights vary from 3 to 5 for all of our experiments. We expect the average delay of a single hop to be somewhat less than one second, where the average RTT between two hosts is about 100-200ms and we expect several RTTs to deliver a 20KB document. Figure 4 shows the cumulative distribution function for the delivery delays in the four runs mentioned above. We can observe average delays ranging from less than 1 second to about 3 seconds. A tenfold increase in the number of clients has three fold effects on the delay values. These delay values are less than what we had expected. This can be attributed to the fact that these runs were made during a period of low network traffic on PlanetLab. However, previous runs had produced similar distributions with slightly larger delay values. These differences can be attributed to the varying network conditions and the stability of PlanetLab nodes.

The main reason for choosing a distributed dissemination model was the presence of bandwidth constraints resulting from high-volume data streams. We now want to observe how well our solution addresses the issue of constrained bandwidth and how effective it is in distributing the bandwidth intensive workload among the participating clients. Figure 5 displays the CDF for average outgoing bandwidths experienced by clients during the four runs. On an average, about 50% of the clients are required to forward very little or nothing at all. This
draws from the fact that our dissemination structure is a tree and on an average about half the nodes are the leaves and perform no forwarding at all. The fact that the tree is organized based on the fanout heuristic makes sure that it’s the low bandwidth nodes that are usually the leaves and makes it very improbable that they are required to utilize their outgoing bandwidth. To put some more perspective on the obtained numbers, we can observe that no client ends up forwarding more than 8KB in a second. This is constrained by the nodes with the largest fanout of four, which can only forward up to 8KBps when the documents come in at a rate of 0.1 per second (1 document per second x a selectivity of 10%). Another interesting observation that can be made is the effect that increasing the number of clients has on the average outgoing bandwidth. In the 2000 clients curve we can observe that clients end up forwarding less data on an average. This can be explained by the fact that with an increased client base a client has larger variance in its placement within the dissemination tree. Thus, a client with a high fanout has a higher chance of ending up on a lower tree level.

To achieve a lower delivery delay, we had to play with the connection timeouts for the TCP connections between clients. From initial runs, we were able to observe huge delays in transfer rates between clients - perhaps due to network congestion. This let us to believe that we should set a very low timeout constraint on the connections. The runs described here were made with a timeouts set to 1 second. The concern now is that such a low timeout will force us to drop many good but slightly delayed connections. To justify our decision, we observed the successful delivery rates for the four runs. Figure 6 shows document receipt rates of over 90% for all of the runs. Considering the nature of the PlanetLab testbed, these values seem reasonable and acceptable.
6.3 Topological Optimization

In the previous schemes we do not follow any heuristic apart from one based on client fanout. This can potentially produce trees that bounce a document across the same physical links, without any knowledge of network topology. As we have already mentioned, it is hard to take advantage of the network topology at the application level, especially when every tree generated is potentially different from any other. We take a very coarse-grained approach. We observe that most of the PlanetLab nodes are within North America, with some in Asia and roughly 20% in Europe. With such a distribution, it is highly likely that a document will travel across the Atlantic several times before it reaches all of its intended recipients. We revise our tree-construction heuristics to take into account the topological position of the clients together with their fanout. The filter divides the dissemination list topologically on the first hop. It uses one of its fanout to send the document to all receivers in Europe and the other fanout for all other receivers.

Figure 7 shows two runs with 2000 clients, one is performed based on only the fanout heuristic, while the other splits the client list topologically, at the filter. The topological heuristic run shows better overall delivery delays and seems to have separated the clients into two segments, those that get their documents in less than 2 seconds and those that take more time. However it is unclear from this data which clients specifically benefit from this optimization. Figure 8 shows the average delays for these two runs for only the European nodes. They all seem to experience an improvement in delivery delay and are close to the 2 second mark.

7 Future Work

We have proposed a result dissemination scheme which builds an overlay network using only the par-
participating clients. What we can explore further is the establishment of a reliable dedicated infrastructure with high network capabilities to act as first level delivery nodes. Presumably these dedicated nodes would be able to sustain a higher fanout and could be strategically placed to optimize for locality. As PlanetLab was envisioned, it’s not only a testbed but also a deployment facility. We feel that using PlanetLab as a dedicated framework could be a starting point for deploying this dissemination service.

The problem of highly dynamic multicast is exciting in its own right. It needs to be explored to a greater depth than we have done with our simple heuristics.

8 Conclusions

We have addressed the problem of result dissemination in high-volume data filtering. We described that, for such a filter, bandwidth is a more pressing bottleneck than processing power. We showed that the problem boils down to highly dynamic multicast, where each filtered document has to be delivered to a potentially large set of clients. Then, we gave a peer-to-peer scheme for result dissemination, which builds an unstructured overlay over the participating clients, on the fly. We used some simple heuristics for constructing dynamic multicast trees. Our experiments on the Planetlab testbed show that the proposed scheme scales well with the size of the client base. We hit a tradeoff between delivery delays and document loss, the latter coming in because of timeouts induced to maintain low delivery delays.

References


