SIFT – A Tool for Wide-Area Information Dissemination *

Tak W. Yan Hector Garcia-Molina
Department of Computer Science
Stanford University
Stanford, CA 94305
{tyan, hector}@cs.stanford.edu

Abstract

The dissemination model is becoming increasingly important in wide-area information system. In this model, the user subscribes to an information dissemination service by submitting profiles that describe his interests. He then passively receives new, filtered information. The Stanford Information Filtering Tool (SIFT) is a tool to help provide such service. It supports full-text filtering using well-known information retrieval models. The SIFT filtering engine implements novel indexing techniques, capable of processing large volumes of information against a large number of profiles. It runs on several major Unix platforms and is freely available to the public. In this paper we present SIFT's approach to user interest modeling and user-server communication. We demonstrate the processing capability of SIFT by describing a running server that disseminates USENET News. We present an empirical study of SIFT's performance, examining its main memory requirement and ability to scale with information volume and user population.

1 Introduction

Technological advances have made wide-area information sharing commonplace. A suite of tools have emerged for network information finding and discovery; e.g., Wide-Area Information Servers (WAIS) [KM91], archie [ED92], World-Wide Web (WWW) [BLCGP92], and gopher [McC92]. However, these new tools have one important missing element. They provide a means to search for existing information, but lack a mechanism for continuously informing the user of new information. The exploding volume of digital information makes it difficult for the user, equipped with only search capability, to keep up with the fast pace of information generation. Instead of making the user go after the information, it is desirable to have information selectively flow to the user. In an *information dissemination* (aka. *alert, information filtering, selective dissemination of information*) service, the user expresses his interests in a number of long-term, continuously evaluated queries, called *profiles.* He will then passively receive documents filtered according to the profiles. Such a service will become increasingly important and form an indispensable tool for the dynamic environment of wide-area information systems.

A very simple kind of information dissemination service is already available on the Internet: mailing lists (see e.g., [Kro92]). Hundreds of mailing lists exist, covering a wide variety of topics. The user subscribes to lists of interest to him and receives messages on the topic via email. He may also send messages to the lists to reach other subscribers. LISTSERV is a software system for maintaining mailing lists. A problem with the mailing list mechanism as a tool for information dissemination is that it provides a crude granularity of interest matching. A user whose information need does not exactly match certain lists will either receive too many irrelevant or too few relevant messages. The USENET News (or Net-

*This research was sponsored by the Advanced Research Projects Agency (ARPA) of the Department of Defense under Grant No.MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of ARPA, the U.S. Government, or CNRI.

news) system [Kro92], an electronic bulletin board system on the Internet, is similar in nature to mailing lists. While Netnews is extremely successful with millions of users and megabytes of daily traffic, at the same time it often creates an information overload. Like mailing lists, the coarse classification of topics into newsgroups means that a user subscribing to certain newsgroups may not find all articles interesting, and also he will miss relevant articles posted in newsgroups that he does not subscribe to.

Recent research efforts on information filtering focus on the filtering *effectiveness*, attempting to provide more fine-grained filtering using relational, rule-based, information retrieval (IR), and artificial intelligence approaches. With few exceptions, they are often small scale (i.e., involving a small number of users or profiles) and thus the need to provide *efficient* filtering is not apparent. However, in a large-scale wide-area system where the number of information providers and seekers are large, efficiency in the dissemination process is an important issue and must be addressed.

The Stanford Information Filtering Tool (SIFT) is a tool for information providers to perform large-scale information dissemination. It can be used to set up a clearinghouse service that gathers large amount of information and selectively disseminates the information to a large population of users. It supports full-text filtering, using well-known and well-studied IR models. The SIFT filtering engine implements novel indexing techniques, capable of scaling to large number of documents and profiles. It runs on several major Unix platforms and is freely available to the public by anonymous ftp at URL:

```
ftp://db.stanford.edu/pub/sift/sift-1.0.tar.Z
```

In this paper we describe SIFT. We present its approach to user interest modeling and user-server communication. We demonstrate the processing capability of SIFT by describing a running server that disseminates tens of thousands of Netnews articles daily to some 13,000 subscriptions. We describe the implementation of SIFT, focusing on the filtering engine. Finally we present an empirical study of SIFT's performance, examining its main memory requirement and ability to scale with information volume and user population.

2 Other Previous Work

Boston Community Information System [GBBL85] is an experimental information dissemination system. Like SIFT, it allows a finer granularity of interest matching than mailing lists or Netnews. Users can express their interests with IR-style, keyword-based profiles. The system broadcasts new information via radio channel to all users, who then apply their own filters locally. While the radio communication channel makes broadcast inexpensive, local processing of mostly irrelevant information is very expensive. (For every user, a personal computer is dedicated for this purpose – this is cited as a major source of complaints from the users.)

Information Lens [MGT+87] provides categorization and filtering of semi-structured messages such as email. The user defines rules for filtering, and the processing is done at the user site. It provides effective filtering, but local processing is expensive for large-scale information dissemination. Similarly, the "kill file" mechanism in certain news reader programs allows the user to locally screen out irrelevant articles. A kill file only removes specified articles from newsgroups that a user subscribes to, but it does not discover relevant articles in other newsgroups. To provide the same kind of filtering power as SIFT would require much local processing. It is more cost-effective to pool profiles together to share the processing overhead.

The Tapestry system [GNOT92] is a research prototype that uses the relational model for matching user interests and documents; filtering computation is done not on the properties of individual documents, but rather on the entire append-only document database. Efficient query processing techniques are proposed for handling this kind of queries. Tapestry is built on top of a commercial relational database system. The Pasadena system [WF91] investigates the effectiveness of different IR techniques in filtering. It collects new documents from several Internet information sources, and periodically run profiles against them. Similar to SIFT, it uses a clearinghouse approach, but efficient filtering is not addressed.

In [YGM94b, YGM94c] we proposed a variety of indexing techniques for speeding up information filtering under IR models. There we evaluated these techniques using analysis and simulation. The SIFT filtering engine is a real implementation of one class of index structures that we found to be efficient.

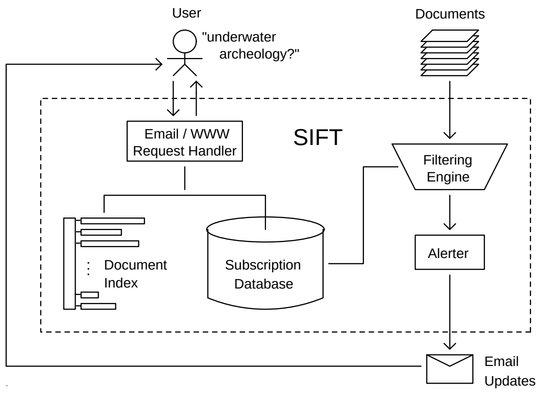

Figure 1: An overview of SIFT

3 SIFT

We begin our presentation of SIFT with an example. Suppose a user is interested in underwater archeology (Figure 1). He sends an email subscribe request to a SIFT server specifying the profile "underwater archeology," and optionally parameters that control, for example, how often he wants to be updated or how long his subscription is for. He may alternatively access the SIFT server via WWW, using a graphical WWW client interface to fill out a form with the subscription information. (Before he subscribes, he may test run his profile against an index of a sample document collection.) His subscription is stored in the subscription database. As the SIFT server receives new documents, the filtering engine will process them against the stored subscriptions, and notifications will be sent out based on the user-specified parameters.

In the following we detail the SIFT system from the perspectives of user interest modeling and communication protocol. We then illustrate SIFT using the Netnews SIFT server.

3.1 User Interest Modeling

A user subscribes to a SIFT server with one or more subscriptions, one for each topic of interest. A subscription includes an IR-style profile, and as mentioned additional parameters to control the frequency of updates, the amount of information to receive, and the length of the subscription. A subscription is identified by the email address of the user and a subscription identifier.

3.1.1 Filtering Model

The interest profile can be expressed in one of two IR models: *boolean* and *vector space* [Sal89]. We first focus on the vector space model.

In vector space model, queries and documents are identified by terms, usually words. If there are m terms for content identification, then a document D is conceptually represented as an m-dimensional vector $D = \langle w_1, w_2, \ldots, w_m \rangle$, where weight w_i for term t_i signifies its statistical importance, such as its frequency in the document. We may also write D as $\langle (t_{i_1}, w_{i_1}), \ldots, (t_{i_k}, w_{i_k}) \rangle$ where $w_{i_j} \neq 0$; e.g., $\langle$ (underwater, 60), (archeology, 60) $\rangle$. A query is similarly represented. For a document-query pair, a similarity measure (such as the dot product) can be computed to determine how "similar" the two are. In an IR environment, the top ranked documents are retrieved for a query.

In an information filtering setting, a key question is how many documents to return to the user. We could have allowed the user to receive a fixed number of top ranked documents per update period, e.g., as done in [FD92]. However, this is not very desirable: in a period when there are many relevant documents, he may miss some (low *recall*[1]); in a period when there are few interesting documents, he may receive irrelevant ones (low *precision*[1]). We instead allow the user to specify a *relevance threshold*, which is the minimum similarity score that a document must have against the profile for it to be delivered. A default value is supplied to the user for convenience.

Instead of using the vector space model, the user may use boolean profiles to specify words that he wants in documents received, and words to be excluded. For example, the boolean profile "fly fishing not underwater" is for documents that contain both words "fly" and "fishing" but not the word "underwater." The reader may note that the SIFT boolean model only allows conjunction and negation of words. However, the user may approximate disjunction semantics by submitting multiple subscriptions (though a document may match more than one subscriptions).

3.1.2 Profile Construction and Modification

To assist the user with the construction of a profile, a SIFT server provides a test run facility. The user may

[1]Recall is the proportion of relevant documents returned, and precision is the proportion of documents returned that are relevant.

run his initial profile against an existing, representative collection of documents to test the effectiveness of his profile. He may interactively change the profile and (for weighted profiles) adjust the threshold to the desired level of precision and recall. When he is satisfied with the performance of the filtering, he may then subscribe with the selected settings.

After the user receives some periodic updates, he may decide to modify his profile or change the threshold for vector space profiles. He may do this by accessing the SIFT server. Furthermore, for vector space profiles, *relevance feedback* [Sal89], a well-known technique in IR to improve retrieval effectiveness, can be used. The user simply gives SIFT the documents that he finds interesting; after examining them, the server adjusts the weights of the words in the user's profile accordingly.

3.2 Communication Protocol

There are two modes of communication between the user and a SIFT server. In the interactive mode, the user subscribes, test-runs a profile, views, updates, or cancels his subscriptions. In the passive mode, the user periodically receives information updates. Instead of developing a new communication protocol, we make use of current technologies: email [Cro82] and World-Wide Web HTTP [BLCGP92].

First we discuss the interactive communication mode. Email communication is the lowest common denominator of network connectivity. By having an email interface, a SIFT server is accessible from users with less powerful machines, with limited network capability, or behind Internet-access firewalls. We adopt the LISTSERV mailing list syntax wherever possible, to ensure that minimal learning is required. We also use default settings to reduce the complexity of email requests for novice users.

As the appeal of hypermedia navigation and the development of sophisticated client interfaces are making WWW the preferred tool for wide-area information sharing, we also developed a WWW access interface for SIFT. Using a WWW client program, the user interacts with the SIFT server through a user-friendly graphical interface. We believe the dual email and WWW access covers the tradeoff between wide availability and ease of use.

In the passive mode of user notification, a SIFT server sends out email messages that contain excerpts of new, potentially relevant documents (certain number of lines from the beginning, as specified by the user). After the user reads the excerpts, he may access the SIFT server to retrieve the entire documents. Currently the excerpts are formatted to be read from regular mail readers. We plan to offer the option of formatting them in HTML, so that the user may view the notifications from sophisticated WWW viewers, and then interactively retrieve interesting documents from SIFT or provide feedback via HTTP.

3.3 SIFTing Netnews

Using SIFT, we have set up a server for selectively disseminating Netnews articles (text articles only; binary ones are first screened out). Like any other SIFT server, the user accesses the Netnews SIFT server via email or WWW. The reader is encouraged to try it out: for email access, please send an electronic message to `netnews@db.stanford.edu`, with the word "help" in the body; for WWW access, please connect to `http://sift.stanford.edu`. In February 1994, we publicized the Netnews server in two newsgroups; within ten days of the announcement, we received well over a thousand profiles. The number of profiles keeps increasing and now (November 1994) exceeds 13,000, submitted by users from almost all continents. Table 1 shows some interesting statistics obtained from the server on the day of November 10, 1994. The average number of articles is over the week of November 4 - 10, 1994.

Number of subscriptions	13,381
Number of users	5,146
Daily average number of articles	45,127
Average notification period (in days)	1.4

Table 1: Netnews SIFT server statistics

Apparent from these numbers is that the load on the SIFT server is very high. It is necessary to match an average of over 45,000 articles against some 13,000 profiles and deliver most updates within a day. The efficient implementation of the SIFT filtering engine enables the job to be done on regular hardware, a DECstation 5000/240. Even though we believe SIFT is efficient, there is a limit to the load that it can handle. It is necessary to replicate the server; in fact, we have been in contact with several sites (in the U.S. and Europe) that expressed interests in providing the same service.

The Netnews SIFT server is not meant to be a replacement of the Netnews system, which is an invaluable tool for carrying on distributed discussions. Rather, it provides a complementary filtering service. Only the beginning lines of matched articles are sent out. The user may, after determining that an article is relevant, access his own local news host to access the article. In cases where he does not have access to a news host, he may request the whole article be sent from the SIFT server.

Sifting Netnews is just one application of SIFT. We have set up another SIFT server, disseminating Computer Science Technical Reports (email access: `elib@db.stanford.edu`, WWW access will soon be available). This server has close to 1,000 subscriptions (November 1994).

4 Implementation

SIFT is implemented in C and has been compiled on several Unix platforms: DEC Ultrix 4.2, HPUX 8.07, and SunOS 4.1. In the following we first give an overview of the components of the system and then focus on the implementation of the filtering engine.

4.1 System Components

The program `mailui` implements the email request handler shown in Figure 1. It parses email messages, assuming the format in [Cro82]. User requests are processed by accessing the subscription database and the document index (for test runs).

The program `wwwui` is a so-called "cgi-script" for use with the HTTP daemon released by National Center for Supercomputing Applications. It accepts requests submitted by users using a WWW client (with a form-filling graphical interface), and, similar to `mailui`, it accesses the subscription database and document index to process the requests.

The program `filter` implements the filtering engine. It accepts as input a list of documents (in the form of Unix path names) and matches them against the stored profiles. The implementation details are discussed in the next subsection. The output from `filter` is a file of matchings.

After the filtering is completed, the matchings are sorted by users and subscription identifiers. This is necessary because SIFT sends out updates on a per subscription basis. After sorting, the program `alert` can be used to send out the message one by one, using Unix `sendmail`. Included in the messages are excerpts from the matched documents (a few lines from the beginning of the document).

To provide the test run capability, we need to index a collection of documents. This can be done by some publicly available software, and we choose to use the index engine in the WAIS distribution (called `waisindex`). From our experience it is a very robust implementation. However, since we use a different scoring scheme than WAIS, we made slight modifications to the WAIS code to make it compatible with our filtering engine (i.e., giving the same results). The major changes are to support the use of relevance threshold and the processing of boolean queries. The modified `waisindex` code is included in our SIFT distribution.

4.2 Filtering Engine Implementation

In IR, to efficient process queries against a collection of documents, an *inverted index* [Sal89] of documents is built, which associates a word with its occurrences in the documents. In the information filtering scenario, we may use the same approach: an index is built of a batch of new documents, and profiles are treated as stored queries that are run periodically. This is the approach used in an earlier SIFT prototype. We refer to this as the document index approach. This approach may not be very appropriate with a high volume of new information. The document index is large and resides on disk. Running a large number of profiles against it requires a lot of disk I/Os.

In the current SIFT implementation, we use a different approach. The idea is to consider information filtering as the dual problem of IR, and treat profiles as documents and documents as queries. Instead of building a document index, we build a *profile index*. In [YGM94b,YGM94c] we propose and analyze a variety of profile indexing techniques for the vector space and boolean models respectively. In our SIFT implementation, we have chosen good indexing techniques for the two models that are compatible in nature and combined them together in the same index structure. Below we briefly present this combined index structure through an example. For a detailed description, as well as descriptions of other profile indexing techniques and analysis, the reader is referred to [YGM94b,YGM94c].

Referring to Figure 2, suppose P1 is a weighted profile ⟨(underwater, 60), (archeology, 60)⟩ and P2 is a boolean profile "fly fishing not underwater." For each word, we associate it with an *inverted list* of *postings* that reference profiles containing it. For example, in

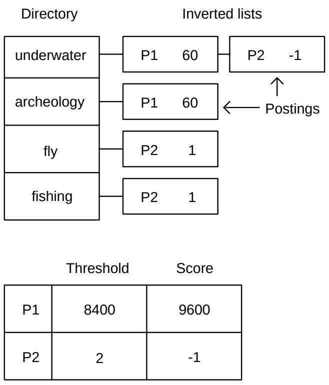

Figure 2: Profile index and auxiliary data structures

Figure 2 the inverted list for word "underwater" is made up of two postings referencing P1 and P2. In the posting, if the profile is weighted, we include the weight of the word in the profile; e.g., 60 for "underwater" in P1. If the profile is a boolean one, we assign a weight of 1 to a non-negated word, and -1 to a negated one.

In addition, we make use of two arrays called Threshold and Score; each profile has an entry in each array. For each profile, its Threshold entry stores the minimum score that a document must have against it to be a match. For a boolean profile, this minimum is equal to the number of non-negated words. For a weighted profile, the minimum is derived using the relevance threshold specified by the user. The Score entry is used to accumulate the score that a document has against the profile.

Now suppose a document D containing the words "underwater archeology" (and none of the other profile words) is being processed. Suppose the vector space representation for D is $\langle$(underwater, 80), (archeology, 80), $\ldots\rangle$; i.e., the words "underwater" and "archeology" are both assigned weights of 80. To process the word "underwater," we look up the *directory* and access its inverted list. For a weighted profile like P1, the document weight of "underwater" is multiplied with the weight in the posting (60), and the product is accumulated in P1's Score entry (note that we are calculating the dot product of the two vectors). For a boolean profile like P2, we simply add its posting weight (-1 for "underwater") to the Score entry. Similarly the word "archeology" is processed. Finally, profiles whose resulting Score entry is no less than the Threshold entry match the document, such as P1 in Figure 2.

5 Performance Study

A practical challenge for SIFT is to process a new batch of documents within the smallest time unit of notification. For example, in the Netnews SIFT server, it is critical that the filter and notify process is finished within 24 hours. Thus, it is important to understand and characterize the performance of the SIFT system, and measure how it scales with the number of profiles and volume of information.

First we study the performance of the processing time with respect to the number of documents and number of profiles. We then compare the performance of the filtering engine with and the alternative document index approach discussed in Section 4.2. Finally we investigate the main memory requirement of the filtering engine. All experiments are run on a dedicated DECstation 5000/240 running Ultrix 4.2 and all times reported are real (wall clock) time.

The study was performed in early July, 1994. The test data consisted of 38,000 articles received on the day of July 6, 1994 by our department's news host and 7,000 randomly selected subscriptions from the Netnews SIFT server at that time. The average article size was 269 words (a word is defined as an alphanumeric string longer than 2 characters). The subscriptions were stored on a local disk, while the news articles were stored on an NFS-mounted disk (from the news host). We repeated the experiments with test data from two other days and the results were within $\pm 10\%$ of those reported below.

In the result graphs that follow, we divide SIFT processing time into these four steps:

1. *Build Time* The profile index structure is built. Other auxiliary data structures are allocated and initialized.

2. *Filtering Time* Documents are run against the index one by one. Document-profile matchings are written into a file.

3. *Sorting Time* The document-profile matching file is sorted by user email address and subscription identifier, using Unix `sort` command.

4. *Notify Time* The sorted matching file is read. Excerpts of matchings for each subscription are prepared into an email message. Unix `sendmail` is invoked to send out each message.

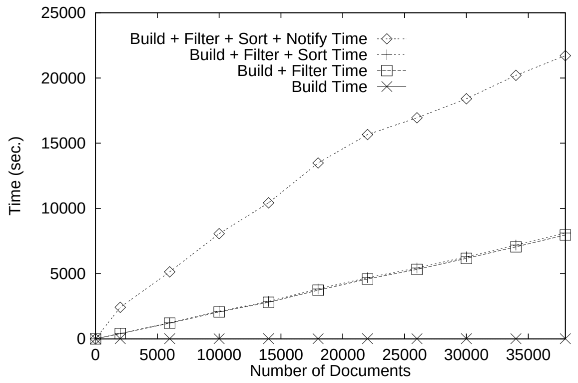

Figure 3: SIFT performance vs. # documents

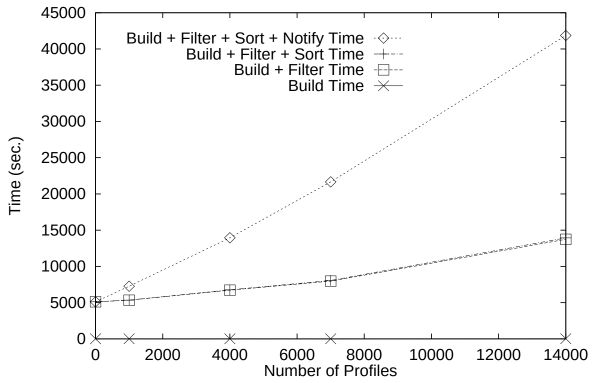

Figure 4: SIFT performance vs. # profiles

5.1 Coping with Information Volume

First we investigate the relationship between the processing time and the number of documents. Figure 3 shows the results of processing the 38,000 articles against the 7,000 subscriptions. The time it takes to build the profile index is very small and is as expected independent of the number of documents. The filtering time is linear with respect to the number of documents, with slope 0.21 sec./document (value obtained by curve-fitting using Mathematica). The time it takes to sort the matching file is negligible. The proportionality constant for the total running time is 0.63 sec./document. (Recall that the articles are stored on a remote news host, so some fraction of this time is spent in reading the articles via local network. This is confirmed by looking at the CPU utilization, which is found to be 42.8%.)

To measure the notify time, since it would be impossible to send the same updates repeatedly to our real users, the notify times shown in Figure 3 (except the rightmost data point) are interpolated results. We assume that the time it takes to send out notifications is proportional to the number of matchings in the matching file. We have verified this assumption with a separate experiment and derived the proportionality constant. Then in the experiment for Figure 3, we count the number of matchings in the matching file at the data points shown. We compute the notify time as the product of the number of matchings and the proportionality constant. The results show that a large fraction ($\approx$ 63% for 38,000 documents) of the total time is spent in sending out subscriptions.

5.2 Coping with Subscription Growth

In the second experiment, we look at the relationship between the processing time and the number of profiles. We repeat the process of filtering 38,000 documents against 1, 1,000, 4,000, 7,000, and 14,000 subscriptions. The last run is done simply by duplicating the 7,000 subscriptions in the database. Figure 4 shows the results. Again, the profile index build time is negligible. For the filtering time, we find that even for one profile, it takes 5,105 sec. to filter. This is the time spent reading in the articles via NFS. Beyond the initial start-up cost, filter time is apparently linear to the number of profiles, with slope 0.62 sec./profile. The notify time is similarly obtained as discussed above. We find the total running time to be linear with respect to the number of profiles, and the slope is 2.63 sec./profile.

5.3 Performance Improvement

To quantify the performance improvement obtained by using the filtering engine, we compare it with the earlier SIFT implementation which runs profiles against a document index to perform the filtering. This process consists of these three steps:

1. *Build Time* The document index structure is built.

2. *Filter Time* Profiles are run against the document index one by one. Document-profile matchings are written into a file.

3. *Notify Time* Update messages are sent out.

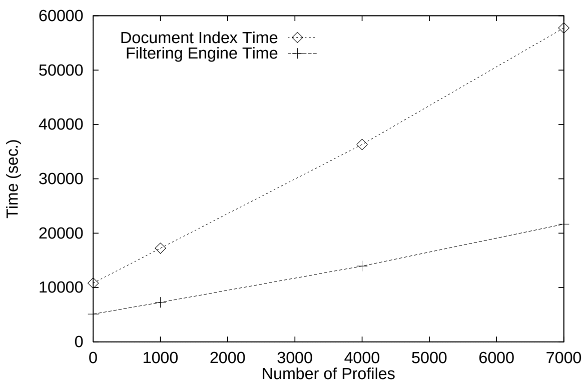

Figure 5: Comparing performances of document index and filtering engine

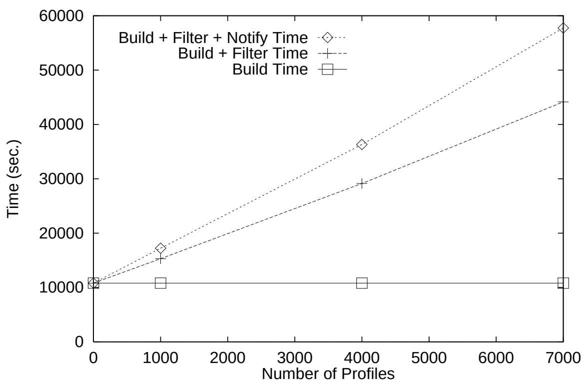

Figure 6: Breakdown of running time for filtering using document index

Comparing this with the filtering engine process, we note that although the first steps have similar names, they represent totally different work. The document index takes much longer to built since there are many more documents than profiles and a document is much larger than a profile. Also the sorting of matchings is not needed in this approach because the filtering is done on a per subscription basis. In fact, a notification may be sent out as soon as a subscription is processed. However, for comparison purposes we opt to separate the work into the shown sequence.

First we compare the total running times of the two approaches. Figure 5 shows the results of processing different number of profiles against 38,000 documents. The slope for the document index total running time is 6.68 sec./profile, compared with 2.63 sec./profile for the filtering engine. If we focus on the 7,000 profile runs, the total running time with the document index is 57,745 sec., compared with 21,652 sec. with the filtering engine. Thus, the SIFT filtering engine is more than twice as fast as the document index approach.

Figure 6 shows the time breakdown for the document index approach. We see that the document index build stage makes up a significant portion (10,803 sec.) of the whole process. The majority of the time is spent in the second stage; for the 7,000 profile run, the filter time is 33,345 sec. or 58% of the whole processing time. On the other hand, the notify time now takes up a smaller fraction of the running time.

It may be argued that the document index build time should also be included in computing the total running time for the filtering engine case, since it would be built in that case to provide the test run facility. However, if the document index is used strictly for test run purpose, we might simply build an index with any representative collection and not once for every batch. Even if we do include the document index build time, the total time for 7,000 profiles sums up to 32,455 sec., still just 56% of that for the document index approach.

5.4 Main Memory Requirement

By inspection of the filtering engine code, the formula for main memory requirement (in bytes) for the SIFT filtering engine is:

$$\begin{aligned}\text{Main memory requirement} =\ & \\ & \#\ \text{profiles} \times 132\ + \\ & \#\ \text{distinct profile words} \times 32\ + \\ & \#\ \text{profile word occurrences} \times 12\end{aligned}$$

The first term on the right-hand-side is for the auxiliary data structures (132 bytes per profile; besides keeping the Threshold and Score entries, we also need to keep the email address and other information in main memory to efficiently output the matchings). The second term is for the directory nodes (one per distinct profile word). The last term is for the posting nodes (one per profile word occurrence). Using the subscriptions to the Netnews SIFT server, we count the various numbers on the right-hand-side of the above formula and compute the main memory requirement for various number of subscriptions. The results are shown in Table 2.

# profiles	# distinct words	# total words	Main memory required (bytes)
1,000	1,595	2,334	211,048
2,000	2,806	4,756	410,864
3,000	3,746	6,893	598,588
4,000	4,657	9,244	787,952
5,000	5,535	11,735	977,940
6,000	6,278	14,102	1,162,120
7,000	7,001	16,489	1,345,900

Table 2: Filtering engine main memory requirement

Several interesting statistics can be obtained from these data. The total number of word occurrences is linear with respect to the number of profiles, and the slope is found by curve-fitting to be 2.34 words/profile. On the other hand, the number of distinct words increases more rapidly in the beginning, but at large number of profiles, seems to approach the slope 1 word/profile. Making the rough assumption that the number of distinct words is also linear with the number of profiles, we use curve-fitting to derive the ratio (main memory requirement / number of profiles) and find it to be 195 bytes/profile. This translates to a capacity of 5,128 profiles per megabyte. Thus, with today's large memories, SIFT can process hundreds of thousands of profiles.

6 Conclusion

The user population of our SIFT Netnews server has grown at a rate of approximately 1,400 subscriptions a month; this indicates that there is a recognized need for wide-area information dissemination. Further, since the server was made public, we have received a lot of feedback and comments, which are predominately positive. The server was reported in numerous magazines and newsletters (e.g., WIRED, Internet World, MicroTimes), resulting in the continued growth of its population. A number of sites have requested the SIFT code, which is now publicly available by anonymous ftp. Anticipating further subscription growth for the service, we are in our initial steps to replicate the server at other sites.

One of the main suggestions for improvement is to provide more expressive filtering models to screen out irrelevant information. Netnews is an extremely diverse source of information, with topics that are extremely disparate in nature. Thus the noise level of the filtered results is in some cases high. In the server for disseminating Computer Science Technical Reports, we found the noise level to be much lower. Still we have taken steps to provide more filtering capability, and the boolean model has been added as a result. Extensions to phrases and proximity matching are also planned.

Evident from the SIFT Netnews server, the efficiency of the filtering process is critical in providing such a service. We believe the indexing method implemented in SIFT scales to large number of users and high information generation rate. The time taken to process a batch of documents against a collection of subscriptions is linear with respect to to the number of documents and the number of subscriptions. The proportionality constants are roughly half of those based on a document index.

The sending out of email notifications consumes a major portion of the SIFT running time. To improve this, a more sophisticated document delivery scheme is needed. One idea is to group users geographically [YGM94a]. If a document matches one of the users in a group, a single copy of the document is sent to the group. A local distribution site for the group receives the copy and retransmits it to users locally. This is beneficial if users have overlapping interests, and is also useful in reducing wide-area network traffic. However, to install such a scheme, cooperation from the user side is needed.

Acknowledgements

Thanks to Anthony Tomasic for suggesting filtering Netnews as an application of SIFT and for providing comments to improve the readability of this paper.

References

[BLCGP92] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications, and Policy*, 1(2):52–8, 1992.

[Cro82] D. Crocker. *Standard for the Format of ARPA Internet Text Messages (RFC 822)*. Network Information Center, SRI International, Menlo Park, California, 1982.

[ED92] A. Emtage and P. Deutsch. Archie: An electronic directory service for the Internet. In *Proc. Usenix Winter 1992 Technical Conference*, pages 93–110, 1992.

[FD92] P.W. Foltz and S.T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Communication of the ACM*, 35(12):29–38, 1992.

[GBBL85] D. Gifford, R. Baldwin, S. Berlin, and J. Lucassen. An architecture for large scale information systems. In *Proc. Symposium on Operating System Principles*, pages 161–70, 1985.

[GNOT92] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communication of the ACM*, 35(12):61–70, 1992.

[KM91] B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. *Connexions – The Interoperability Report*, 5(11):2–9, 1991.

[Kro92] E. Krol. *The Whole Internet User's Guide & Catalog*. O'Reilly & Associates, Sebastopol, California, 1992.

[McC92] M. McCahill. The internet gopher protocol: A distributed server information system. *Connexions – The Interoperability Report*, 6(7):10–14, 1992.

[MGT+87] T. Malone, K. Grant, F. Turbak, S. Brobst, and M. Cohen. Intelligent information sharing systems. *Communications of the ACM*, 30(5):390–402, 1987.

[Sal89] G. Salton. *Automatic Text Processing*. Addison Wesley, Reading, Massachusetts, 1989.

[WF91] M.F. Wyle and H.P. Frei. Retrieval algorithm effectiveness in a wide area network information filter. In *Proc. ACM SIGIR Conference*, pages 114–22, 1991.

[YGM94a] T.W. Yan and H. Garcia-Molina. Distributed selective dissemination of information. In *Proc. Parallel and Distributed Information Systems*, pages 89–98, 1994.

[YGM94b] T.W. Yan and H. Garcia-Molina. Index structures for information filtering under the vector space model. In *Proc. International Conference on Data Engineering*, pages 337–47, 1994.

[YGM94c] T.W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Transactions on Database Systems*, 19(2):332–64, 1994.

Bibliography

Tak W. Yan is a Ph.D. student in the Department of Computer Science at Stanford University, Stanford, California. His research interests are in the areas of database systems, digital libraries, and information retrieval and filtering systems. He received a B.S. in Electrical Engineering and Computer Science from University of California, Berkeley in 1991. In 1993 he received a M.S. in Computer Science from Stanford University.

Hector Garcia-Molina is Professor in the Departments of Computer Science and Electrical Engineering at Stanford University, Stanford, California. His research interests include distributed computing systems, database systems, and digital libraries. He received a B.S. in Electrical Engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. From Stanford University he received in 1975 a M.S. in Electrical Engineering and a Ph.D. in Computer Science in 1979.