, , 1–19 ()

Rama: An Architecture for Internet Information Filtering

JIM BINKLEY jrb@cs.pdx.edu
LESLIE YOUNG leslie@cs.pdx.edu
Computer Science Department, Portland State University, Portland, OR 97207

Received May 1, 1991

Abstract. This paper describes Rama, a first generation experimental information retrieval and filtering system that attempts to recover useful information from various Internet sources including USENIX news and anonymous FTP servers. The focus of the Rama system to date has been on building a distributed query and information retrieval system, which provides an interface to heterogeneous information services. A user of Rama sends one or more asynchronous queries to a Rama server using existing SMTP e-mail clients. The server periodically searches local and remote Internet services. Searches are prefiltered with the use of timestamps. Data objects which are newer than the timestamp are then searched via a query mechanism which relies on a combination of vector-distance and pattern matching operands, and boolean operators. Results are weighted according to how closely they match queries and are posted via e-mail to the user. Input to the e-mail client can be further filtered – one can use the MH mail system and sort input by weight. Results indicate that the current system is useful and extensible. So far we have assumed that existing e-mail systems will be used for input and output and have not attempted to construct special client interfaces. Efforts are underway to extend the system with WWW searching capabilities and construct a special WWW oriented user-interface.

Keywords: agent, e-mail, FTP, HTTP, information filtering, information retrieval, MIME, mobility, news, NNTP, pattern matching, vector-distance

1. Introduction

Our goal is to develop an active information retrieval system that is useful for searching and filtering information from many different Internet information services in a relatively transparent way. Much like Arthur C. Clark's alien space probe Rama (Clarke, 1973), which looks for intelligent life in the galaxy, our Rama probes the Internet looking for information.[1] In a loose sense, Rama resembles a knowbot (Cerf and Kahn, 1988), where processes move across the network finding information of interest, eventually returning the information to the user. However, the original knowbot system as described by Droms (Droms, 1989), and Michael Schwartz's Netfind (Pu and Schwartz, 1993) are both white page systems; that is, they are focused on recovering names and addresses. They also integrate multiple information services behind a common interface. Rama resembles these systems in that it attempts to integrate searching for disparate information sources behind one interface. Unlike them, it is searching for general information and is not providing a directory service. SIFT (Garcia-Molina and Yan, 1994), the Stanford

Information Filtering Tool, has a number of common attributes with Rama. Both are asynchronous and use e-mail for input and output, focus on USENET news (Kantor and Lapsley, 1986), search for new news, and employ Salton (Salton, 1993) vector-distance queries for producing weighted output. Rama, unlike SIFT, is investigating information heterogeneity and can search more than one source at a time. In addition to boolean and vector-distance mechanisms, Rama's query mechanism also includes pattern matching which may be more useful in choosing what not to deliver and may be more flexible when talking to remote indexing systems that take patterns as input. For example, we might try to talk to the Glimpse index mechanism running under Harvest (Bowman et. al., 1994). Another important difference in filtering is that Rama forces a user to choose a set of news groups to search up front, whereas SIFT searches everything. We feel that it is important to make the user choose a specific area (although it can be made fuzzy with the use of wildcards) in order to cut down on the signal to noise ratio inherent in USENET news. We also allow users to search 'subjects only' as opposed to the 'entire text,' which is another way of filtering – albeit a simple one.

A taxonomy of agent systems might be categorized as follows:

1. User-interface systems that directly interact with the user and attempt to infer suitable information queries from user behavior. We might call such systems *user-interface agents*. Maes' (Kozierok and Maes, 1993) (Sheth and Maes, 1993) systems at MIT are an example.

2. Distributed query systems where a server must act as a client in order to search existing remote information systems. Rama is an example of such a system. Another example is the CERN WWW proxy server (Luotonen and Altis, 1994) that unifies client WWW searches behind a firewall. Harvest also has this characteristic.

3. Distributed query systems that can route queries (or more generally, programs) to remote servers that have local access to information.

Certainly the problem of building an Internet information system from scratch is daunting and the solution must by definition be heterogeneous and incremental. Architectural components including intelligent user-interfaces, distributed query protocols, and back-end server systems must be built in an extensible and open fashion. Currently, there is limited support on the Internet for such mechanisms. Users are expected to directly manipulate heterogeneous user interfaces while searching for information, because protocols for automatic information gathering are in their infancy. The size and format of information also present complex problems. Information is scattered and does not compare in terms of organization to data provided in traditional databases.

We view the system we have developed as a testbed for architectural experimentation in information filtering and retrieval and as a platform for development of any or all of the above kinds of agent systems. Our efforts to date have primarily focused on building a category 2 system. We are interested in learning how we might build a query protocol that would closely interface with other information systems. We are also interested in integrating a smart user interface with our distributed information search system.

Rama queries are asynchronous. A user does not typically make a query and then wait for the results. Queries are typically serviced once a day in the early morning hours. Results show up in the user's mailbox, generally when the user first logs in. As a result, we are able to take advantage of local compute power that is typically unused at night. More important, we have a day of compute time to devote to searching.

Rama queries are stored in a simple ASCII file known as the *Personal Information Profile* (or PIP file) (Clarke, 1978). The PIP or query file is mailed to the Rama e-mail server and stored there. Query files are compiled and submitted to the scheduler for execution. Logically, a Rama query of USENET news looks like this:

> *Give me everything in news group G matching key K that appeared since yesterday.*

Each search request has a timestamp associated with it. After the query is submitted for searching, its associated timestamp is updated to the current time. As a result, the next time the query is processed, Rama will search only for new information. Searching is carried out by the server according to the information resource, such as news, a local file system, anonymous FTP, etc. Requests are prefiltered by the timestamp and postfiltered by a runtime query engine that combines elements of Salton vector-distance queries, pattern matching, and boolean operators. All data items that are newer than the timestamp are evaluated and weighted between 0.0 and 1.0. If the evaluation is greater than 0, or an optional user supplied low-water cutoff value, the data object is returned to the user via e-mail.

In the next section of this paper, we present an overview of Rama's architecture and protocol. Section 3 provides some examples and a discussion of experience gained. Section 4 discusses test results and measurements. Section 5 reveals Rama's future. In section 6, we present some conclusions and final discussion.

2. Rama Architecture

The Rama architecture consists of an e-mail input system, a query protocol, a master scheduler, *infosphere* drivers[2], and an output system.

One can loosely use the phrase client-server to describe the current architecture. However, the component parts of the Rama server (UNIX processes) are designed to either run on the same machine or be distributed over a number of machines. Components may in turn play both a client and server function. Queries arrive via e-mail and are stored in a database local to the server. The query compiler (or profile reader) is invoked once a day. (Please see Figure 1, which shows the server architecture but does not include the e-mail input system). The reader compiles the queries into a binary form and transmits them over a TCP connection

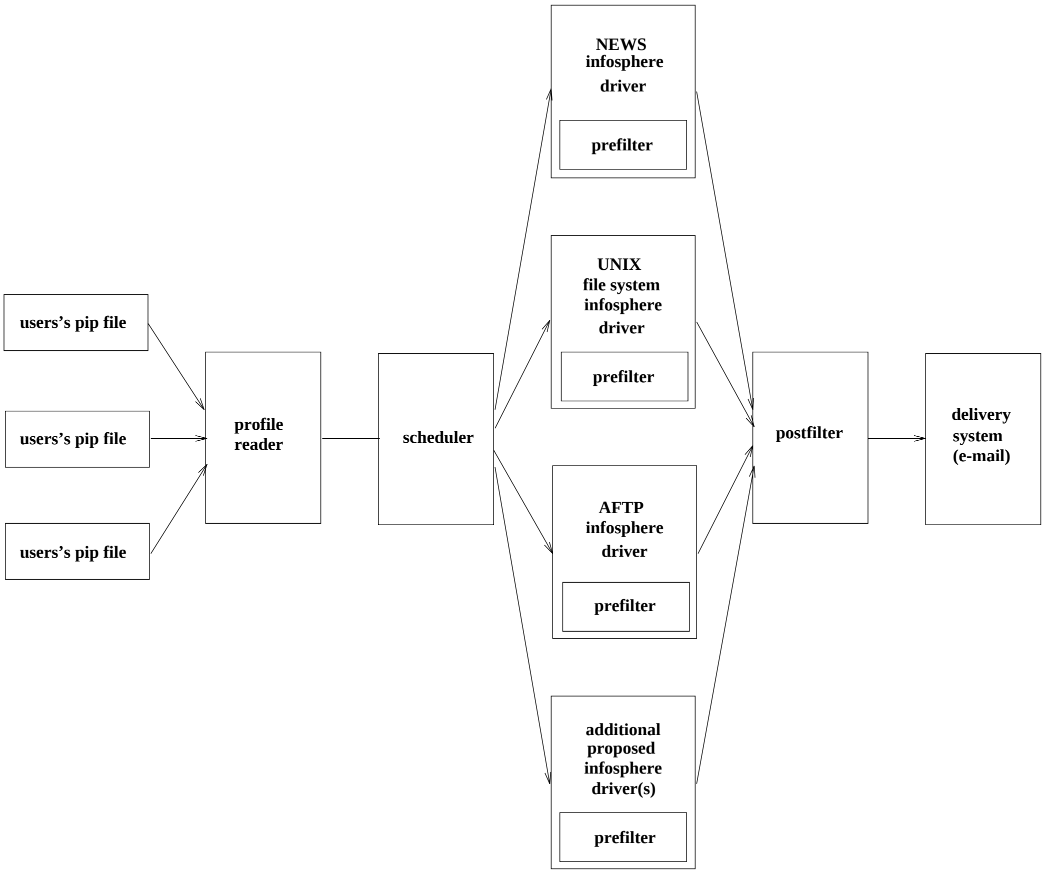

Figure 1. Rama Architecture

to the scheduler. The scheduler receives the list of binary records and feeds them as a client to the infosphere drivers that actually do the dirty work of digging for information in a particular infosphere. It may be useful to regard the component parts as a pipeline, where the pipe in each case is a TCP connection or a traditional UNIX pipe. Other components may be added to the pipeline. For example, an e-mail client at the user's system might sort the e-mail. We have given particular attention to developing an architecture that is flexible and extensible, because it is our belief that we will always need to integrate new search engines, new clients, and new delivery paradigms in order to successfully adapt to the ever-changing landscape of information retrieval on the Internet. To that end, components can easily be added to or removed from the input and output ends of the Rama system.

2.1. The Input System

Rama's input system consists of an e-mail list-server based on the Majordomo (Chapman, 1992) perl (Schwartz and Wall, 1990) program. A file consisting of a set of query items is kept at a remote client system. Each query item causes a remote resource like a news group to be searched. When the user wants to make changes, a new file can be submitted to the Rama server with an e-mail command called `put`. The query file is stored at the server in a user database for daily processing. The user can retrieve the query file from the server with a `get` command. Commands for subscribing and unsubscribing to the service are also available. Users can also temporarily suspend their subscriptions. Various help files including a Rama user guide are available via e-mail. In effect, the default Rama user interface is any SMTP e-mail client.

2.2. The Query Language and Protocol

The ASCII query files are compiled and syntactically verified by a compiler (named *pip*). The compiler is activated once a day by the UNIX *cron* utility. All records are submitted to the scheduler over a TCP connection as a continuous stream – thus giving the scheduler an opportunity to optimize the queries.

The contents of the query file are written in a query language that can be thought of as having either an ASCII form or a binary form. The ASCII form is suitable for e-mail transmission. The binary form must be used to talk to the scheduler and is also used between the scheduler and the back-end infosphere search processes.

Figure 2 shows a sample query file with 3 different kinds of queries, including searches of USENET news, a local UNIX file system, and a remote FTP site. The query file can contain any number of search requests. A number of different keywords are used per query. Syntactically each query begins with a left brace, and ends with a right brace. The query record contains a number of keyword/value pairs. The more important keywords are:

TITLE: The TITLE string is a user label for an individual query. It helps users match e-mail output to an individual query. The TITLE is included in the e-mail header.

KEY: The KEY specifies an expression used for searching. The grammar for an expression contains the traditional boolean operators AND, OR, and NOT that can operate on vector-distance and/or pattern matching operands. Vector-distance query terms are indicated with single quotes; for example, `'agent search FAQ'`. (See Figure 2, for examples in the context of a full query). Pattern matching is indicated with double quotes; for example, `"agents"`. It is implemented with the University of Arizona's agrep (Manber and Wu, 1992), a new form of grep that can deal with spelling errors. Multiple KEY lines may be used and are implicitly ANDed together. Typically this feature can be used to produce a kill list. For example, NOT sub-expressions using pattern matching can be appended to the original master KEY to prevent certain information items from showing up again.

```
# The following entry will search USENET news for all articles in all
# comp.infosystem news groups that contain the terms "agent," "search,"
# or "FAQ" in either the article body or subject.
# Any matched object found with the pattern "wais" in it is discarded.
# The search will be discontinued after 10 hits.  Any search items that
# match but have a weight less than .2 are discarded.

[
 TITLE = "news from comp.infosystems";
 KEY = 'agent agent search FAQ';
 KEY = NOT "wais";
 SEARCH_TYPE = ALL;
 INFOSPHERE = NEWS;
 ROOT = "comp.infosystems.*";
 HITS = 10;
 RFW = "0.2";
]

# The following entry will search for the string "agents" in all the
# files in the directory and subdirectories of /usr/local/db/papers.
# Since no date is listed, all files in the directory tree are searched.
# The search will be discontinued after 5 hits.

[
 TITLE = "local papers directory";
 KEY = "agents";
 SEARCH_TYPE = BODY;
 INFOSPHERE = FILE;
 ROOT = "/usr/local/db/papers";
 HITS = 5;
]

# The following entry will search our anonymous FTP site to inform us
# if any new files have shown up in the blackadder directory.  This
# search is only done monthly.

[
 TITLE="ftp and blackadder files";
 KEY = ".";
 SEARCH_TYPE = MAP;
 INFOSPHERE = AFTP;
 PERIOD = MONTHLY;
 ROOT = "ftp://ftp.cs.pdx.edu/pub/blackadder";
]
```

Figure 2. A Sample Pip File

Compilation (ASCII to binary) turns the KEY expression into an executable postfix format. At runtime, the infosphere drivers fetch objects and execute the query engine on each object. More details will be supplied on this mechanism below, but the reader should understand that the eventual result of a KEY expression applied to a search object is a weight between 0.0 and 1.0, where 0.0 means no match, and 1.0 means a perfect match.

Here are some example KEYS and explanation as needed:

```
KEY = ".";  #match everything
KEY = 'agent book mobile wireless';  #vector-distance query
KEY = NOT "misc.job.resumes";  #not this news group
#combination of vector-distance and pattern matching
KEY = 'Binkley Newton Cameron Hoke' OR "english#family";
```

SEARCH_TYPE: Search types specify whether the search is to be made on an object header, or a body, or both. How header and body are interpreted depends on the back-end information resource mechanism. For example, if one is searching a file system, the header is a filename, the body is the contents of the file. Current search types include SUBJECT, BODY, ALL (both SUBJECT and BODY), and MAP. The exact semantics of MAP is very dependent on the infosphere. The idea is to allow the driver to return some overview of the search space, like a directory listing or list of news groups. An FTP driver returns a remote directory listing (filtered by timestamp). Thus a user can determine if new files have shown up on a remote resource. A hypothetical WAIS (Kahle and Medlar, 1991) directory server might return a list of directories. MAP information may be of interest to a human user or to a user-interface agent that might be attempting to broaden its background search area.

INFOSPHERE: The infosphere specifies the search domain; for example, AFTP (anonymous FTP (Postel, 1985)), FILE (UNIX file system), NEWS (NNTP), etc.

PERIOD: The default search period is daily. However, a longer period may be specified. Current parameters include: NEVER, ONCE, DAILY, WEEKLY, FORTNIGHT and MONTHLY.

ROOT: The ROOT keyword specifies an infosphere directory or sub-area for searching. For NEWS, this is the news group. For FTP, we use a HTTP URL . Of course, for the WWW (Berners-Lee et. al., 1992), we will use a HTTP URL too. An URL, or Uniform Resource Locator is typically a combination of a protocol type (FTP or HTTP), a remote system name, and a file pathname on that system. If the pathname is a directory, we search the directory tree.

DATE: The DATE keyword may be specified by the user to indicate that searching should proceed for items newer than the DATE specified. However, it is typically set only by the system itself. The compiler updates all DATE keywords to the current time after processing a request. Thus searching can proceed for new items day by day. The DATE keyword constitutes a prefilter timestamp for searching. It is used by an infosphere driver to simply ignore old objects. There is no need to fetch a file if it is older than the current DATE.

HITS: The user may specify a maximum number of matches per query per day. Once the HITS are exhausted, an infosphere driver will stop searching on that query.

RFW: The RFW keyword sets a low-water mark for matched weights on any query object. If a query returns a weight less than the RFW on an object, the object is discarded. This mechanism allows a user to set a much needed pain threshold when the system is returning too much output. The RFW value is applied to the result of the entire set of KEY expressions in a query.

2.3. The Binary Query Protocol

All scheduler input and output uses the compiled binary format of the Rama query protocol. The compiler, which can be either local or remote, translates the ASCII queries into the binary format and sends them to the scheduler over a TCP connection. The scheduler in turn routes the protocol queries to the back-end drivers that understand a particular remote information type. The drivers may be on the local system or may be on remote systems. In its binary form, the query is called a *ramagram*. The format of a ramagram is made more complex by the fact that the KEY expression can contain arbitrary strings. Figure 3 shows the layout of a ramagram. Each ramagram starts with a fixed header which can be followed by any number of TLV fields. TLV stands for "Tag Length Value" and is a common protocol encoding technique for dealing with variable length data.

```
Ramagram header:
int version;  /* version number */
int totallength;  /* total length of entire record */
int infosphere;  /* infosphere for this search item */
int search_type;  /* search type */
int filter_type; /* filter type */
int timeout; /* how long to search */
int hits; /* how many itmes max to find */
unsigned long date; /* time filter for search */
TLVS:
short tag, short length, bytes values.
short tag, short length, bytes values.
short tag, short length, bytes values.
...
```

Figure 3. Ramagram Header Declaration

The TLV fields are used for encoding per query strings such as the remote user name, hostname, and all of the operators and operands used with KEY expressions. The TAG field indicates the kind of tag to be used, such as AND or OR in the KEY expression. The length field indicates the length of the data followed by the data itself. Use of the TLVs makes the binary format almost as flexible as the ASCII format. As a result, it is possible that new fields can be added without unduly modifying the code for back-end infosphere drivers.

2.4. The Scheduler

The scheduler (named *ramad*) reads records posted to it by the compiler. It is responsible for scheduling and dispatching tasks to back-end infosphere drivers. It uses the INFOSPHERE value to determine what kind of driver should be invoked. Using runtime switches, a system administrator can decide the parallelism of query execution. The administrator may choose between the logical poles of one infosphere process per query searched, or one process maximum for a given infosphere (all the received jobs in that area). Or she may choose some ratio of processes to work items. The control over parallelism is intended to to allow an administrator to tune the system to match the compute capabilities of the server. In general, more parallelism helps to make the system more scalable.

Although we are not currently working in this area, it would also be useful to investigate other scheduling issues. For example, one could schedule according to quality of service desired and give one infosphere precedence over others (local news is better than remote FTP, or an indexed news server is better than a non-indexed one). One could take advantage of opportunities to optimize searches. There is little point in making the same search twice if two users ask for the same search. It may be useful to optimize work according to previous history.

2.5. The Infosphere Drivers

An infosphere driver is an amorphous term used to describe a process that takes query records as input, searches a back-end information resource and forwards any hits to the user.

For each record posted by the scheduler, the infosphere driver must:

1. search the ROOT information area.
2. perform a search according to the semantics of SEARCH_TYPE, which can have the value SUBJECT, BODY, ALL or MAP.
3. use the DATE value to prefilter any search item. In other words, ignore remote objects that are too old.
4. pass each potential match through the KEY query engine. The output of the query engine is a weight.
5. according to the weight and the semantics of the found object, mail any resulting match to the user.

Three current driver types exist (with an HTTP driver underway). These driver types are FILE (UNIX file system), NEWS (NNTP access), and AFTP (anonymous FTP).

The FILE infosphere driver is the default driver and searches a local or NFS-mounted UNIX file system for matches. Because each directory entry has a timestamp associated with it, file systems allow prefiltering and easily avoid unnecessary searching. As with any driver, results are postfiltered and mailed to the user. The contents of binary files are not searched. The driver will mail matched ASCII files, but only sends the path for a matched (by subject) binary file. For security reasons, the FILE driver may not be used by remote e-mail users. However, an administrator might set it up in a restricted fashion – analogous to anonymous ftp directories. It could be used to search news on a local disk.

The NEWS infosphere driver uses the NNTP protocol in conjunction with a NNTP news server to search news. The NNTP NEWNEWS command is used to obtain a list of articles more recent than the current DATE. Since the NEWNEWS command takes a timestamp, the prefilter work here is actually done by the NNTP server. Also the ROOT is passed via the NEWNEWS command to the server. As a result, because NEWNEWS can take limited wildcards, it is possible to have the news server search more than one news group at a time. For example, one can pass it "comp.infosystems.*" and thus search a number of component news groups at once. HEAD, BODY, SUBJECT and ARTICLE commands are used to fetch component data, which is stored in temporary files, and then postfiltered. The NEWS infosphere has a number of efficiency advantages, since the news server itself does a large part of the work. Also typically a news server is nearby (news is a heavily distributed database). All matches are mailed to the user.

The AFTP (anonymous FTP) infosphere driver can search remote FTP sites and make SUBJECT, BODY, or MAP searches as requested. The current driver does not allow BODY searches of binary files and makes no attempt to unwrap remote compressed tar archives. We are assuming that in general this driver will not be used to aggressively search remote sites on a daily basis, and when combined with the PERIOD keyword will be used even more infrequently. For example, the MAP function could be used to inform the user that new files have shown up at a particular site in a particular directory. This driver does not mail files. Instead it sends an indirect MIME (Berenstein and Freed, 1993) mail pointer that will allow the user to fetch the file via a MIME-capable mail interface.

The system can be extended by adding new drivers that implement the semantics of the query record interface. Drivers can be internal to the scheduler (forked), or external (exec'ed). External drivers may be local or remote. They can be coded in C or in a scripting language such as perl which allows rapid prototyping. Ramad writes queries to internal drivers via function calls. Queries to external drivers are sent over the same TCP stream mechanism used between the compiler and scheduler.[3]

2.6. The Query Engine

The query engine is a shared facility used by all infosphere drivers to determine if a new data item is of interest to the user. A data object and all KEY expressions in a ramagram are passed to the query engine. The KEY operators and operands are in postfix order and are executed with intermediate results pushed on a runtime stack. Each operand evaluates to a weight. Each KEY operator takes one or more weights and outputs another weight. The final result is always a weight.

In the computer science community it is customary to give a precise semantics for the operations in a proposed computational language. However, the semantics of the operators used in matching for information retrieval are a matter of some debate in the research and development community. As a result, the semantics for Rama's operators reflects this fuzziness. Depending on the paradigm currently in vogue, the details of the semantics might be changed. Here we will only give a sketch of the semantics we are currently using.

The current semantics are as follows:

- `AND`: - Assume two operands `lhs AND rhs`. If either evaluate to zero, the resulting weight is zero. If they evaluate to a non-zero result, the resulting weight is slightly increased; that is, the returned weight is max(lhs, rhs) + a reward.
- `OR`: - Assume two operands `lhs OR rhs`. The traditional probability equation is used here: `w = lhs + rhs - (lhs * rhs)`.
- `NOT`: `NOT` is a binary operator. If the operand is vector-distance, a probability function is used, `w = 1.0 - op`. If the operand is a pattern, the decision is binary; that is, the resulting weight is 1.0 or 0.0.
- `vector-distance`: The traditional weight mechanism is used here including stemming, stopping and term weight calculations. (We refer you to Salton (Salton, 1993)). Vector-distance works by converting the query and the document both to sets of *terms*; i.e., root words with the endings discarded. Terms are counted and a function of similarity between the query and document is computed based on term counts. For example, in the query in Figure 2, *'agent agent search FAQ'*, by repeating the term 'agent', we increase its count in the query. Therefore we emphasize the relative importance of that term over other query terms and any matches of that term result in a higher overall weight for a document.
- `pattern matching`: To determine a weight, we run agrep over the file to get the number of matched lines and then divide that number by a crudely derived estimate of the number of lines in the file. Thus the pattern weight is roughly: `w = matched lines / total lines + S`. `S` is a scaling factor. Agrep is faster than normal grep, tolerates spelling errors, and also has some simple boolean operators of its own which extend its flexibility. For example, the Rama KEY expression ``` ``a'' AND ``b'' ``` would run agrep over a file twice and only match if both patterns were found anywhere in the file. The agrep expression ``` ``a,b'' ``` would match only if particular lines have both patterns in them. As a result, the entire matching mechanism is more versatile.

Other semantics lurk in the runtime query expression engine. For example, we reward matches found at the head of a document by giving a slightly higher weight to matches of query and document terms found in the first N document terms. As another example, we felt that implementing the AND operator as a probabilistic set function does not make sense. For example, if `w1 = .3`, and `w2 = .4`, then of course `w1 * w2 == .12`. But if you find a document that matches "agent" AND "distributed", you do not really want to reduce its output weight. Instead we increase the weight slightly by adding a small bonus to the maximum of the two weight operands and thus reward a successful AND. NOT is typically used as a second KEY to explicitly weed out unwanted information. NOT has two flavors. Use of vector-distance as the operand results in a weak NOT. Pattern matching gives a strong NOT – "I mean what I say and I don't want to see *barney* ever again."

2.7. The Output System

Drivers return results to users via e-mail by delivering a file with an SMTP (Postel, 1982) e-mail header. Rama-specific e-mail tags are added to the header including the TITLE, weight, etc. (see Figure 5, a sample Rama e-mail header, in the next section). Originally, due to Rama's heavy news orientation, the system returned ASCII files of limited size. One direction that we have begun to exploit is to take more advantage of MIME capabilities for dealing with file types and for encoding binary files.

Any e-mail client can be used for mail delivery. If a client has filtering capabilities, mail can be postfiltered to advantage. We will see an example of that in the next section.

3. Examples and Experience

What does Rama look like to the current users? Mail seems to be a natural mechanism for dealing with heavy amounts of input whether from Rama or from mailing lists. The user can choose her preferred mail system. The mail system itself may be able to provide an additional client filtering system. For example, MH mail has the mhook filtering mechanism, which allows incoming e-mail to be placed in folders. We slightly modified MH mail so that we can sort the mail by weight.

Our Rama server runs at 4:00 a.m. E-mail is ready for a user when she logs in. Please refer to Figure 4 that shows a typical Rama e-mail input directory. The Ramad server runs as user "rama" so that users will receive mail from "rama" (as opposed to other entities). In the case of USENET news, the news subject line becomes the mail subject line. It is useful to show as much as possible of the subject line. A user may typically choose to look at an item based on that piece of information alone. In addition, a user might choose to sort by weight or by query item or both. User experience has shown that receiving too much mail can be stressful and in fact can be an important drawback in using Rama. If a user goes away and forgets to stop e-mail delivery, quite a pile of unimportant e-mail

```
% scan +ramainput

1 10/26 rama@cs.pdx.edu [0.35] Re: Man to Html (or Cprolog) <<News groups:
2 10/26 rama@cs.pdx.edu [0.30] Re: CGI access to remote (client) user id
3 10/26 rama@cs.pdx.edu [0.28] ORACLE Contract Programmer - Portland, Or
4 10/26 rama@cs.pdx.edu [0.27] DB2 Contract Programmer - Portland, Orego
5 10/26 rama@cs.pdx.edu [0.25] PORTLAND, OR: S/W/DEV Engineer - C, PVC
6 10/26 rama@cs.pdx.edu [0.24] Viewing Unix mail folders via WWW<<Path:
7 10/26 rama@cs.pdx.edu [0.24] PORTLAND, OR: SR S/W Engineer - Embedded
8 10/26 rama@cs.pdx.edu [0.24] PORTLAND, OR: Embedded Network Protocols
9 10/26 rama@cs.pdx.edu [0.24] FAQ - ASCII Art Questions & Answers (4.5
10 10/26 rama@cs.pdx.edu [0.23] mobile IP - searching 4 infor<<Newgroups
11 10/26 rama@cs.pdx.edu [0.23] ANNOUNCING: Mailto-1.2 release (CGI Gatew
12 10/26 rama@cs.pdx.edu [0.22] Senior UNIX development - #65, Chicago<<N
13 10/26 rama@cs.pdx.edu [0.22] comp.databases.sybase Frequently Asked Qu
14 10/26 rama@cs.pdx.edu [0.21] ANNOUNCE: Katsu Custom Works Catalog<<Pat
15 10/26 rama@cs.pdx.edu [0.18] Britt<<Path: usenet.ee.pdx.edu!fastrac.ll
16 10/26 rama@cs.pdx.edu [0.16] Internals of TCP/IP implementation under
17 10/26 rama@cs.pdx.edu [0.12] Determining Task Priorities<<Path: usenet
```

Figure 4. Typical Rama Inbox Listing

can build up. As a result, we allow e-mail users to suspend their mail temporarily. Also one MH mail filter sorts the mail by weight, and can restrict the growth of the mail box past a fixed number of items. This has a positive aspect. Users can ignore their Rama e-mail for a while, with the idea that the Rama query result weights will improve over time.

A small Rama header is added that provides more information about why the e-mail arrived. Figure 5 shows a sample e-mail message that has been trimmed of non-essential fields. Note that Rama appends X-Rama flags to the e-mail header to provide useful information such as which query was used, and what infosphere was searched.

Figure 6 shows part of a query file that is in actual use on our server. The first entry searches Portland State's local message group. All results are matched and returned. A NEWS match that always succeeds turns Rama into an asynchronous mail-based news reader.

In the second entry, we are searching the NEWS jobs group for jobs available in Oregon. Repetition of the term Oregon increases the weight of a matched article. Certain data items and subjects have been negated (resumes and Windows-oriented jobs).

In the third entry, we are searching the "comp.answers" news group for FAQs (frequently asked questions) that may be of interest. The RFW field here has trimmed the noise a bit.

The last entry represents a use of the anonymous FTP MAP search mechanisms to allow the user to monitor a remote FTP site for new files.

```
(Message todo:33)
Date: Thu, 6 Oct 94 04:38:08 PDT
From: rama@cs.pdx.edu
Subject: [0.48] ANNOUNCING: Gloss--A Scalable Text-Database Discovery Tool
X-Ramawgt: 0.48
X-RamaTitle: "infosystems news"
X-RamaRoot: NEWS://sirius.cs.pdx.edu/comp.infosystems.wais
X-Ramainfo: search type=subject, hits left=9
Followup-To: comp.infosystems.wais
Data: 5 Oct 1994 20:55:07 GMT
Organization: Computer Science Department, Stanford University

-------------------------------------------------------------------------
Gloss: A Scalable Text-Database Discovery Tool
-------------------------------------------------------------------------

WHAT IS GLOSS

Gloss is a system being developed at Stanford University that helps
you find data sources that are relevant to your queries.
```

Figure 5. Sample E-mail Message

4. Measurements

We feel that the current Rama system shows signs of promise, even in a harsh environment such as USENET news. However, it is hard to determine what success means in an environment where there is so much noise. The attempt to filter information on a long-term daily basis implies that monitoring user interactions with the Rama system is a long-term process. Recently, a user claimed that the system had by serendipity provided information even before the user asked. Still we do not yet know how to quantify user interactions with the system other than by a "seat of the pants" feel.

We collected performance measurements on Rama for a period of 1.5 weeks in early February 1995.[4] The metrics are for USENET news only. Our local news server is an overburdened SparcStation II handling approximately 60000 new articles per day. Roughly 1.5 gigabytes total disk space is used for storage of news. Rama runs on a Sun Sparc 10 with 4 Ross HyperSparc 625 modules. Both Sun systems are running SunOS 4.1.3.

During the measurement period, about 40 queries from 10 users were processed per day. The number of news articles mailed per day averaged 109 and varied from 69 to 172. The number of articles searched per day averaged 13000 and varied from approximately 8000 to 18000. The ratio of mail sent to articles searched is very low (2 orders of magnitude difference) and indicates that a high degree of filtering is taking place. Of course, this is no guarantee that interesting mail is being sent.

Typically, half of the queries used NNTP wildcards, which allow for more news groups to be searched in a particular area.

A small number of benchmark tests were run to explore the efficiency of various parts of Rama. The vector-distance query filter was compared to agrep by running both on large ASCII file sets. The vector-distance filter is about 2 orders of magnitude faster. In turn, agrep is much faster than the stock grep available on SunOS.

The speed of the NNTP driver was compared to the UNIX file system driver in reading news articles. BODY searches (the news body as opposed to the header) were performed with a vector-distance query that failed to match any article, thus producing no e-mail output. We discovered that the news driver can search around 6000 articles per hour. Searching "alt.*" for new articles took about four hours (22k articles). This suggests that with NNTP, we could read all new news in around half a day. If we ignore timestamps, we certainly could not use the news driver to search all available news on the server. We are currently reading 13000 articles per day, So even with our current limited user base, we are reading %21 percent of new news available (13000/60000).

Interesting enough, if news articles are on the local file system, the UNIX file system driver can search about 60000 articles per hour. This is an order of magnitude faster than NNTP. Local searching is certainly more scalable than remote searching, but it may not be an option at a particular site.

As mentioned previously, the Rama server offers a certain limited parameterization of parallelism. We conducted a few tests with NNTP searching done in parallel, and found that searching the same news group with two drivers running in parallel results in roughly twice as many articles searched. This suggests that the parallel mechanism is useful and makes the NNTP search mechanism (or any other network driver) more scalable. However, in actual practice, if two news infosphere drivers are used, the results are not twice as good because the scheduler currently makes no attempt to predict actual job load. One driver might receive heavy tasks and one might receive light tasks. However actual use of the Rama server with a modicum of parallelism still shows some improvement.

5. Future Work

The typical Rama user will be concerned with long-term information goals or interests and will use the system over an extended period. That is, in addition to ad hoc isolated information seeking episodes, a Rama user may want to know all retrievable information related to a given subject on a daily basis. A desirable feature in an information filtering system is that it recommend new information which might possibly be of interest to the user. The system should also be able to help a user more precisely focus on a particular subject area. When the user's interests change, the system should be able to unlearn previously learned knowledge. So learning and adaptation are issues of primary importance to filtering systems. We feel that because our current distributed query mechanism already is capable of handling relevance feedback, we should be able to construct a smart user-interface that will be able to map user feedback to more precise queries. Also, the client should be able to more closely coordinate with the output from the Rama server. For example, there is no reason why the client cannot sort output by query and/or by weight. We intend to integrate a smart Rama client capability with a WWW client and to research how best to divide the learning responsibilities between a smart client and the Rama server.

In the short-term we plan on teaching our server how to search remote HHTP sites. We are also investigating ways to optimize and cache remote searches, as well as how Rama might interface with a remote index-service like Harvest.

6. Discussion and Conclusions

As everyone knows, the Internet is rapidly expanding in both users and data volume. As the volume of information continues to grow, user-driven data browsing alone may fail as a primary means of information discovery. Rama is an attempt to automate that task. Our current architecture shows evidence of being a useful mechanism for at least tackling the problems of gleaning useful information from the increasing volume of USENET news. In this section, we will present a few conclusions about the Rama system. However before we present those conclusions, we would like to briefly discuss the issues of scalability and heterogeneity of information sources and how they might relate to information systems like Rama.

Scalability is not a simple one-dimensional issue, it has several aspects. A system may be scalable to a certain number of users and may deal with a certain amount of data. How data is fetched and what kind of data is searched are also important issues. Even user psychology may be an issue.

An important difference between SIFT and Rama is that SIFT builds an inverted index for user profiles only (and not for input) and then passes news serially through the index. This allows SIFT to be much more scalable than Rama in terms of users. SIFT is intended to be available to an Internet-wide set of users (in a sense like Veronica (Barrie and Foster, 1993) or the limited set of archie (Deutsch and Emtage, 1992) servers).[5] Rama was designed to be available to only a limited set of local users that could form a workgroup. There might be one local Rama for every news server.[6]

We must also ask how well an information service scales to the volume of Internet data. Neither Rama nor SIFT dare to search all of news (we must limit ourselves to new news). It is not reasonable to search all of a remote FTP site on a daily basis. In general, searching Internet-wide even at selected servers is a fanciful notion at the moment and probably will only become more absurd as the Internet grows. We question the notion of launching knowbots that somehow search every node on the Internet. Rama and certain other systems have employed a lesser entity known as a *web-walker* that will search all or part of a specific site (some web-walkers will build a local cache or index for multiple searches). We suspect that it is useful and necessary to help users focus on a specific sub-area for searching and that this should be a primary function for agents. Use of web-walkers in a sense, is an admission of frustration with the chaotic and ever-expanding Internet data set. Some have suggested that web-walkers are a replacement for archie, since archie's availability is limited due to "overuse". We disagree and think that on the contrary, archie is wildly successful, and its limited availability proves that we need more index-based servers (e.g., Harvest). Information servers need machine-searchable indices. In addition to first-level index services, there should be more systems like archie that provide a directory service. Directory service servers should provide a searchable index of sites. Both primary index servers and directory servers should be searchable by remote agents like Rama using the same query protocol.

It may be just as important to scale a system down to local resource constraints as to scale it up to try and deal with more data. In the current Rama system, while it is true that searching local files is faster than searching remote news via NNTP, you may not have enough local storage for news. NNTP access may be your only access mechanism.

Regarding data heterogeneity, neither SIFT nor Rama try to do much with non-ASCII data, although Rama can at least return binary files in some cases via MIME. In contrast, the Essence (Hardy and Schwartz, 1993) system (also part of Harvest), provides a per file type summarizer that allows it to deal with heterogeneous file types.

Another aspect of scalability is "how much information is enough"? Some users are cowering in fear at this point due to the daily volume of e-mail. Something useful or something interesting may be enough to satisfy a user. Of course, ideally we would like to provide the user with the best information possible within certain search limits. Current filtering techniques are based on statistical or other relatively crude techniques. We could certainly use better methods here.

In conclusion, we would like to mention some aspects of Rama that we consider important. Rama is a heterogeneous information service. It allows new information services to be added to the back-end in such a way that minimal changes are needed in the user interface. It currently supports USENET news, anonymous FTP and UNIX file system searching. At the discretion of an administrator, news can be searched either via an NNTP driver or on a local UNIX file system.

Rama is an asynchronous e-mail service. An asynchronous service gives us more time to search and allows us to use parallelism where possible to improve scalability. In general, e-mail delivery makes a lot of sense. E-mail systems are already equipped to deal with a flood of information and as a result, e-mail clients often have their own filtering systems. They may also have security features (PEM (Linn, 1993)) and may be able to deal with multimedia file formats (MIME). Why not deliver news via e-mail? Furthermore, since we already have e-mail, we can take advantage of a multitude of mail clients that already exist. Finally, e-mail is a store and forward service. It deals well with problems of disconnection caused by mobility. Queries and results can be shipped whenever a mobile user reconnects.

Work with Rama suggests that timestamps associated with search objects are very useful. Timestamps are a necessity for finding new objects. When it is possible to associate a timestamp directly with an index entry for an object, (for example, with its entry in a file directory), an agent can prefilter searches, thereby speeding up processing.

Rama's query protocol is flexible. It supports both pattern matching and vector-distance word lists. The query protocol could be used to send queries to either pattern matching based index systems or systems like SIFT that use vector-distance. The e-mail output produced by the server carries various tags that suggest to clients how they might display data; for example, all results returned have a weight. Dumb clients can still be used, but smarter clients could sort by weight and by user profile. Rama's e-mail output tags serve as an early suggestion as to what such a protocol should look like. The tags could be used to allow Rama-aware e-mail clients to more closely interact with information servers.

An alpha release of source and binary (SunOS 4.1 only) of Rama is available at: *ftp://ftp.cs.pdx.edu/faculty/jrb/rama*. We believe that the system can be used for local USENET news filtering but that the remote FTP facilities (and any other remote drivers) should be regarded as experimental and used with caution. Suggestions are welcome.

Notes

1. We will settle for useful information; locating intelligence on the Internet is not something we are promising.
2. An *infosphere* is a body of information presented to the world with a uniform interface, such as anonymous FTP servers, HTTP servers, a UNIX file system, etc. An infosphere driver searches a particular information area according to the query interface, attempting to hide the infosphere's peculiarities from both the front-end Rama system and users.
3. Since the query distribution mechanism is symmetric, note that one scheduler could act as an "external driver" for another scheduler.
4. Due to space limitations we must neglect details and focus on results. Also our statistics should be viewed with some caution. We do not have the resources to exercise Rama on unloaded systems and networks.
5. Veronica is an indexing system that indexes filenames from gopher servers. Archie is an indexing system that indexes filenames from ftp servers. There are not very many Veronica or Archie servers.
6. We are in the process of optimizing queries so that remote sources are only searched once. This will allow our system to scale to more users.

References

Barrie, F., and Foster, S., "Veronica Frequently Asked Questions (FAQ)," 1993, ftp://veronica.scs.unr.edu/veronica-docs/veronica.faq.

Berenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," 1993, Internet Engineering Task Force, RFC 1521.

Berners-Lee, T., Cailliau, R., Groff, J., and Pollermann, B., "World-Wide Web: The Information Universe," 1992, Electronic Networking: Research, Applications and Policy, Vol. 2(1), pp.52-58.

Bowman, Danzig, Hardy, Manber, Schwartz, "Harvest: A Scalable, Customizable Discovery and Access System," 1994, University of Colorado, Technical Report CU-CS-73294.

Cerf, V., and Kahn, R., "The Digital Library Project," 1988, Corporation for National Research Initiatives, Technical Report.

Chapman, B. D., "Majordomo: How I Manage 17 Lists Without Answering '-request' mail," 1992, LISA VI Conference Proceedings.

Clarke, A. C., *Rendezvous with Rama*, 1973, Harcour Brace Jovanovice, New York, N.Y.

Clarke, A. C., *The Fountains of Paradise*, 1978, Harcourt Brace Jovanovich, New York, N. Y.

Deutsch, P., and Emtage, A., "An Electronic Directory Service for the Internet," Proc. USENIX Winter Conference, pp. 93-110.

Droms, R., "The Knowbot Information Services," 1989, Internet Engineering Task Force, Draft.

Garcia-Milina, H., and Yan, T. W., "SIFT - A Tool for Wide-Area Information Dissemination," 1994, Stanford University, ftp://db.stanford.edu/public_html/sift/sift.ps.

Hardy, D., and Schwartz, M. F., "Essence: A Resource Discovery System Based on Semantic File Indexing," 1993, Proc. USENIX Winter Conference, pp. 361-374.

Kahle, B., and Medlar, A., "An Information System for Corporate Users: Wide Area Information Servers," 1991, ConneXions - The Interoperability Report, Vol. 5(11), pp. 2-9.

Kantor, B., and Lapsley, P., "Network News Transport Protocol (NNTP): A Proposed Standard for the Stream-Based Transmission of News," 1986, Internet Engineering Task Force, RFC 977.

Kozierok, R., and Maes, P., "Learning Interface Agents," 1993, Proc. of AAAI.

Linn, J., "Privacy Enhancement for Internet Electronic Mail. Part One: Message Encryption and Authentication Procedures," 1993, Internet Engineering Task Force, RFC 1421.

Luotonen, A., and Altis, K., "World-Wide Web Proxies," presented at the 1st International Conference on the World-Wide Web, May 1994, Geneva, Switzerland, http://www.w3.org/hypertext/WWW/Proxies/.

Manber, U., and Wu, S., "Agrep – A Fast Approximate Pattern-Matching Tool," 1992, Proc. USENIX Winter Conference.

Postel, J. B., "Simple Mail Transfer Protocol (SMTP)," 1982, Internet Engineering Task Force, RFC 821.

Postel, J. B., "File Transfer Protocol (FTP)," 1985, Internet Engineering Task Force, RFC 959.

Pu, C., and Schwartz, M. F., "Applying an Information Gathering Architecture to Netfind: A White Pages Tool for a Changing and Growing Internet," 1993, University of Colorado, Technical Report CU-CS-656-93.

Salton, G., *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, 1993, Addison Wesley Publishing.

Schwartz, R. L., and Wall, L., *Programming Perl*, 1990, O'Reilly and Associates, Inc.

Sheth, B., and Maes, P., "Evolving Agents for Personalized Information Filtering," 1993, Proc. of the 9th IEEE Conference on Artificial Intelligence for Applications.

```
#
# search psu.msgs for all news
#
[
 TITLE="psu news";
 KEY = ".";
 SEARCH_TYPE = SUBJECT;
 INFOSPHERE = NEWS;
 ROOT = "psu.msgs";
 HITS = 100;
 DATE = 10/27/94 04:00;
]
#
# search misc.jobs for local mention
#
[
 TITLE="job news";
 KEY = 'oregon oregon beaverton';
 KEY = NOT "objective";
 KEY = NOT "misc.job.resumes";
 KEY = NOT "windows" AND NOT "pick";
 SEARCH_TYPE = ALL;
 INFOSPHERE = NEWS;
 ROOT = "misc.jobs.*";
 HITS = 5;
 DATE = 10/27/94 04:00;
]
#
# just the faqs
[
 TITLE="news faqs";
 KEY = 'faq bsd network os operating mail www world wide web';
 KEY = NOT "python";
 KEY = NOT "apple";
 KEY = NOT "amiga";
 KEY = NOT "mac";
 KEY = NOT "Typing";
 SEARCH_TYPE = SUBJECT;
 INFOSPHERE = NEWS;
 ROOT = "comp.answers";
 HITS = 5;
 RFW = "0.2";
 DATE = 10/27/94 04:00;
]
# FTP MAP search of freebsd incoming directory
#
[
 TITLE="ftp bsd archive incoming";
 KEY = ".";
 SEARCH_TYPE = MAP;
 INFOSPHERE = FTP;
 ROOT = "ftp://wcarchive.cdrom.com/pub/FreeBSD/incoming";
 PERIOD = WEEKLY;
 DATE = 10/28/94 04:00;
]
```

Figure 6. Sample User Pip File