Mailing List Archive Tools

Sam Leffler and Melange Tortuba
Silicon Graphics and Tortuba Consulting

Mailing List Archive Tools

Sam Leffler, *Silicon Graphics, Inc.*
Melange Tortuba, *Tortuba Consulting*

Abstract

Electronic mailing lists are a common forum for discussion on the Internet. Each participant receives a copy of each message posted to the list with an additional copy typically archived at a well-known site for later retrieval. Busy lists and long-running lists can quickly accumulate many messages making it difficult to locate information. Few sites provide mechanisms for the efficient storage or retrieval of information from these archives. The Mailing List Archive (MLA) tools were designed to address these issues. Mail messages are stored in a space-efficient manner and the archive information is processed to create a database that is optimized for fast queries. The MLA tools are designed for use as Computer Gateway Interface (CGI) applications so that archives can be accessed through HTML forms on the World Wide Web. Concurrent update and retrieval are an integral part of the design so that up to date information is always returned in query results.

1.0 Introduction

UNIX mailing list archives are typically unstructured files that hold the concatenated mail messages stored in the order in which they were received. This simple format makes it easy to apply text-oriented UNIX tools such as grep and awk but fails to capture much of the structure of the discussions that transpire on a list. Furthermore, when mailing lists are especially active archives can grow very large, causing text-oriented tools to slow down noticeably. Bulletin board systems such as Usenet together with threaded reader programs attempt to address these problems but require nontrivial administration and the use of involved packages for transmitting and storing messages. For many discussion lists electronic mail is the preferred medium, and what is needed is a simple to use backend system for archiving messages. While it is possible to feed a mailing list into the normal Usenet software for the purposes of creating an archive, the MLA tools were created as a much simpler solution that is easier to maintain and more efficient.

The MLA tools consist of a program that stores mail messages in a mailing list archive database and several programs for doing queries from a database. Additional tools are provided for doing administrative tasks such as removing messages that are old (i.e. expired) and for optimally compressing a database (most compression is, however done on-line). User access to an archive is possible through command-line programs, electronic mail, or HTML forms on the World Wide Web. Message relationships (i.e. threads of discussion) are maintained and query and navigation tools preserve this threading structure even when navigation is done with an HTML browser.

The MLA tools differ from other packages in several important ways:

- They work with relatively unstructured mail messages (as opposed to news postings, which have structured information that simplifies cross-referencing messages). Programs such as hypermail [1] also work directly with mail messages but do not do as good a job of deducing message relationships, do not work on-line, and do not provide a query interface.
- They are intended to be simple to use in unprivileged environments. The tools require no special-purpose protocols or system registration of services.
- They are designed to efficiently support database updates without impacting concurrent queries. The MLA tools are designed with the express intent of doing on-line updates for each mail message.

- They are simple to use within the framework of the World Wide Web by way of a template-based facility for generating HTML documents.

The MLA tools might best be contrasted with hybrid systems built from an indexing engine and a CGI interface to a threaded news server. A system such as ni [4] has similar functionality but lacks the navigational support provided by the MLA tools.

The next sections of this paper present the tools and provide examples of how they are used. Subsequent sections discuss the design of the database used to store archives, issues in parsing and building threaded mailing list archives, and issues in building a useful navigation interface for the World Wide Web. These are followed by a section reporting performance results and, finally, a summary and discussion of future work.

1.1 Creating a Mailing List Archive

The mlaupdate tool is used to create and update (add messages to) an archive. Typically, an MLA is created from an existing archive of mail messages and then a feed is setup so that new traffic is directly entered into the database. Old message archives are assumed to come in three forms: mbox-style files in which mail messages are concatenated and the UNIX-style “From” message lines are still present, MH folders where each mail message is written to a separate file and tagged with a ``Delivery-Date’’ header line, and Usenet news articles where a ‘‘Newsgroups’’ line is treated as a ‘‘To’’ address. To convert an mbox-style file to an MLA, a command of the form:

```
mlaupdate foo-archive
```

might be used. This creates an MLA in the current working directory, if it does not already exist, and enters the mail messages found in the file foo-archive. mlaupdate can also read from standard input; for example,

```
zcat foo-archive.Z | mlaupdate -
```

can be used to supply a compressed archive to mlaupdate.

Alternatively, the following enters all the messages from an MH folder:

```
mlaupdate -y ~/Mail/inbox/*
```

(The -y option to mlaupdate forces it to not scan the input text for an initial message separator pattern, typically a UNIX-style “From line” as generated by the normal system mail delivery program. In this case the -y option is needed because MH removes UNIX-style “From lines” when mail messages are incorporated into folders.)

Messages are stored in a compacted form with the useful header information written to a table of contents database file that is optimized for searching and the body written to a message-body database file. Message bodies that exceed a certain threshold size are written to separate spillover files instead of the message body database to optimize space usage in the hashed message body database. All message bodies, whether they appear in the database file or in a spillover file, are stored in a compressed form using a PKZIP-style compression algorithm [8].

In addition to the message header information, the table of contents database also includes inter-message relationships such as whether a message is a reply to another message. This information is used to build thread relationships between messages: all messages that appear to be part of a single thread of conversation are identified and query results may provide messages organized as threads, as opposed to unrelated collections of messages.

1.2 Feeding an Archive

A key component of the MLA design is the ability to directly connect a live feed to the database. That is, mailing list traffic can be immediately entered into an archive with the thread-related information automatically updated and available for queries. The typical way to effect this is to setup a mail alias that invokes the mlaupdate program to enter each mail message in an archive. Alternatively, mail messages can be fed to mlaupdate through personal delivery tools such as the slocal [3] or procmail [5] programs. Each linkage mechanism has advantages and disadvantages; most importantly, delivery via a mail alias may require that the MLA files and containing directory be owned by the user that invokes mlaupdate. Invoking mlaupdate through personal delivery tools permits file protections to be setup in a more private manner. The sendmail delivery route can be overcome by using a setuid wrapper program that invokes mlaupdate. Note also that ownerships and protections must permit read access to the CGI applications invoked by the HTTP server.

The following .maildelivery entry might be used with the MH slocal program to feed incoming messages for a “flexfax” mailing list directly into an archive:

```
sender  owner-flexfax  |  R \
"mlaupdate -y -d arch/fax -"
```

(the -y option is supplied in case slocal removes the ``From’’ line; see above). An equivalent mail alias would be:

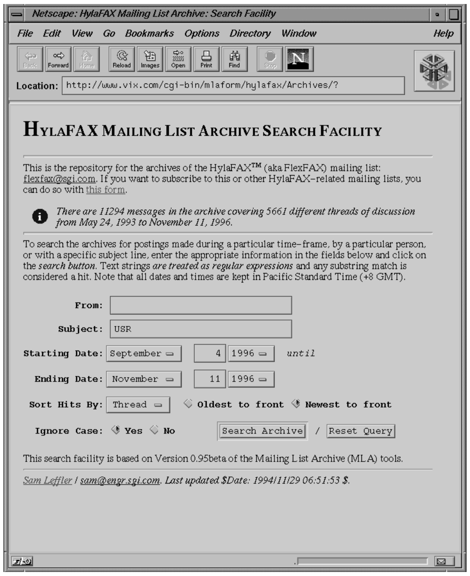


Figure 1. Form generated by mlaform.

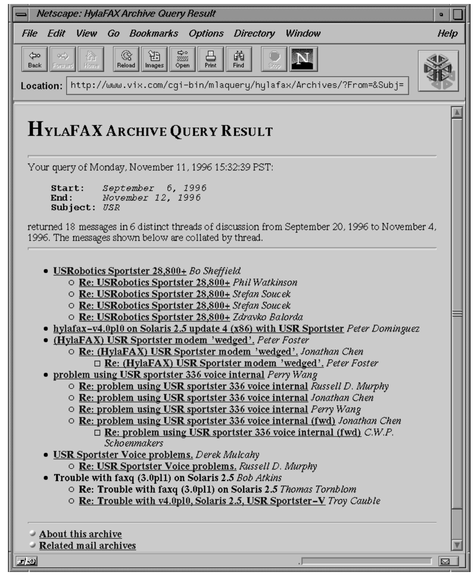


Figure 2. Query result generated by mlaquery.

```
flexfax-archive: "|/bin/mlaupdate \
-d /fax/Archives -"
```

Note that in this situation the -y option is not needed because the "From line" is known to be present in the message delivered to mlaupdate.

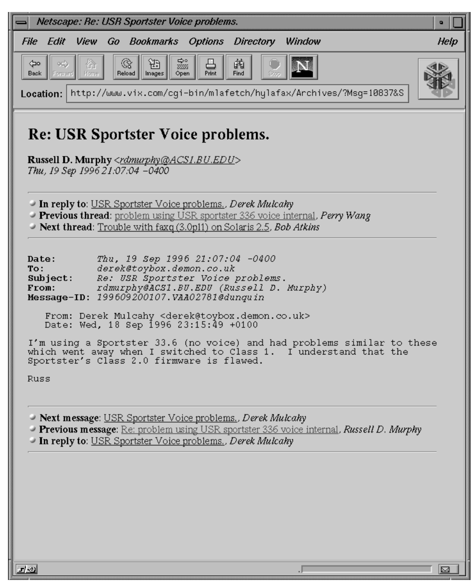
If a mailing list has a very high traffic rate and archive updates cause excessive locking of the database, then the mlabatch program can be used in place of mlaupdate. mlabatch works by holding onto mail messages for a period of time before entering them in a database. If multiple messages are received before an update is scheduled, then they are batched together and only a single update of the database is done (thereby reducing the time the archive is kept locked). mlabatch supports all the same command line options that mlaupdate does as well as two additional options for controlling the batching operation. Thus an archive feed can be converted to use batching simply by invoking mlabatch instead of mlaupdate.

1.3 Querying an Archive Through the World Wide Web

Three tools are provided for querying an archive through the World Wide Web: mlaform, mlaquery, and, mlafetch. These programs comply with the Computer Gateway Interface (CGI) protocol [12] and so may be invoked directly from other HTML pages. mlaform is used to generate an HTML page from which a query is constructed. mlaquery implements database queries returning a specially formulated HTML page that includes a hidden form of the query result in each HTML link. mlafetch retrieves a mail message from an archive, also encoding query results in links for navigational purposes. In normal use, a user is presented a query form generated by mlaform (see Figure 1), constructs a query that is performed by mlaquery (Figure 2), and then retrieves articles using mlafetch (Figure 3). As such the design of these three programs assumes that they will be used together.

mlaform is typically used as the top-level interface to an archive. It generates an HTML page based on a template file and the underlying MLA database. Template files are HTML documents that contain escape codes that are replaced by an MLA

Figure 3. Article retrieved by mlafetch.



tool. Interesting information such as the number of messages in the archive and the ages of messages in the archive are available through escape codes. Escape codes are also provided for formulating queries that are to be done by mlaquery. The fragment of the flexfax archive template shown in Figure 4 gives the general flavor of an MLA form template.

mlaquery supports HTML query-style forms; it takes its arguments from the command line or through shell environment variables as specified by the CGI specification. Searches are done over a single archive with queries constructed from phrases that constrain the returned messages:

- match a regular expression or boolean combination of expressions against mail message header lines (e.g., messages from “sam@.*sgi.com”),
- consider messages within a specific time frame, and
- ignore upper-lower case text distinction in matching strings.

The matching messages are returned collated according to thread, author, subject, or date. The maximum number of hits returned can be controlled as well as the maximum depth of the threading information returned when query results are collated by threads.

The HTML document generated by mlaquery includes an encoded form of the *query hit set*–the set of messages that satisfied the query. This information is critical to the navigational support provided by the tools and is described more in Section 2.3.

mlafetch returns a formatted message together with links to other messages in the query hit set and to other messages in the database that are related to the message; e.g., if a message is a reply to another message then the original message can be viewed even if it was not matched by the query. mlaform propagates the encoded information passed to it by mlaquery so that the context of the original query is always present. Like mlaform and mlaquery, the format and content of the HTML is defined by a template file that administrators can tailor to their needs.

Figure 4. Sample MLA form template.

```
<PRE>
<B>          From:</B>%+F
<B>       Subject:</B>%+S
<B>Starting Date:</B>%+< <I>until</I>
<B>  Ending Date:</B>%+>
<B> Sort Hits By:</B>%+C <INPUT TYPE=submit
VALUE="Search Archives">/ <INPUT TYPE=reset
VALUE="Reset Query">
</PRE>
```

1.4 Administering an Archive

Archives are for the most part self-maintaining. Data structures are designed to grow efficiently as new messages are added to the database. There are, however some components of the database that can be compacted if the database is manipulated off-line and tools are provided to do this. Otherwise, there are tools to delete messages from an archive and low-level tools for interrogating the internal data structures used in an archive–mainly for the purpose of debugging problems. Each MLA is stored in a separate directory and may be copied with the normal UNIX command line tools. However using programs such as cp or tar to copy an MLA may generate a copy that uses more disk space than the original because the hashed database package used by the MLA tools utilizes holey files. If this is an issue, the mladump tool can be used to emit an ASCII form of the archive that can then be fed back into mladump. Thus an archive can be copied by commands of the form:

```
    mladump -d foo.orig first-last |\
mlaupdate -d foo.copy -
```

2.0 MLA Design and Implementation

The MLA tools were designed to be efficient both for queries and for updates, but the expectation was that far more queries would be done than updates. It was very important however, that on-line updates be supported so that queries always return current information; otherwise, the tools would not be used in place of existing facilities.

An archive consists of at least two files: the table of contents database, and the message body database. Message bodies that are too large to fit directly in the message body database file are written to spillover files, with the entry for the message setup to reference this file. The table of contents file is the critical data structure in obtaining high performance for both queries and updates; it is managed with a special-purpose package. The message body database is implemented using a publicly-available hashed database package [9]. Message bodies are compressed with a PKZIP-style compression algorithm whether they are stored directly in the database file or in spillover files; this is done with the publicly available zlib package [10].

2.1 Table of Contents Database

The table of contents database (TOC) holds all the information about an MLA except for the mail message bodies. The data structures are designed so that programs that do queries can map the file contents into memory in a read-only fashion and do a lookup with minimal overhead. Specifically, the number of pages that must be touched is typically small and no parallel data structures or excessive calculations are required of the application as a result of the memory-mapped style of accessing the data. TOC updates also use a memory-mapped interface to the data and are designed to be done in-place for the vast majority of updates (when certain data structures overflow the database is grown to accommodate more information and this operation can be expensive to carry out). The efficiency of normal queries and updates is important in minimizing TOC lock contention when queries collide with updates.

Each TOC has a fixed size header followed by several tables and a *string pool*. The string pool is a data structure that holds all the ASCII strings retained from the headers of mail messages stored in the archive. Redundant strings and substrings are shared so that, for example, subject lines are trivially compressed. The other tables in the TOC hold the following information:

- a message descriptor table that has one entry for each message in the archive,
- a thread table that identifies the top-most message in each thread of discussion,
- a sorted message table that references entries in the message descriptor table sorted by message delivery time, and
- a reply spillover table that holds message reply information that does not fit in the normal message descriptor data structure.

Tables are initially sized according to a set of heuristics and configurable parameters; these parameters are stored in the TOC. From that point onward the tables are automatically grown whenever entering a new message would fill up a table. The revised table sizes are calculated to reflect the existing usage patterns, with each table guaranteed to be no more than 75% full. The initial table sizes are based on the following assumptions:

- the number of threads of discussion is at most 1/3rd the number of messages, i.e., on average there will be no more than two replies for any message;
- the size of the reply spillover table is 1/6th the number of messages, the expectation being that most messages get a single reply that can be stored directly in the message descriptor;
- the string pool gets an average size message (administrator defined) for 1/2th the maximum number of messages (since most messages are replies there is a significant sharing of strings for the subject, user names, mail addresses, etc.).

By default, mlaupdate creates new archives with space for 2500 mail messages, and assumes that an average message requires 128 bytes of space in the string pool. These numbers were chosen empirically. At an average of 100 mail messages per day, these initial sizes accommodate a mailing list for 25 days before the database must be expanded. After that, the database will need to grow about once a week if traffic patterns remains consistent.

Each mail message in an MLA is assigned a unique number that is used for all references (rather than file offsets that would need to be updated when data structures grow). Likewise each distinct thread of discussion is given a unique number that is used for all references. String references are stored as offsets into the string pool and must be updated if a string is moved as a result of the string pool being compacted.

The two important pieces of information that need to be tracked are the mail message contents (headers and body) and the message relationships that define the discussion threads. Each message descriptor contains references to the strings that make up the headers; the message body is looked up separately using the unique message number assigned when the message is added to the database. Message descriptors include references to the message's parent message (if it exists), top-level thread (as stored in the global thread table), and a list of messages that are considered replies to this message.

Message replies are handled specially. Each message may have a single reply that is stored directly in the message's descriptor. However, if multiple replies exist then the reply information in the message descriptor points to a list of reply blocks that are stored in the spillover reply table. Reply blocks are always allocated in the reply spillover table in groups of four. If all the replies for a message can be stored in the available space then they are; however if there are more replies than will fit, three replies are recorded and the fourth entry in the block is used to reference another block in the table where the remaining replies are stored. Figure 5 shows an example where a message has five replies.

The scheme used to manage replies provides an efficient mechanism for handling messages with a small number of replies, but can incur noticeable overhead when a single message has many replies since 25% of each reply block in the spillover table is spent on linking to the next block. Various schemes and reply block sizes were considered before choosing this design based on statistics collected from a variety of mail archives.

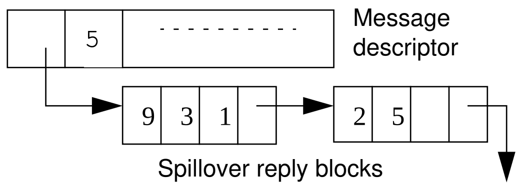


Figure 5. Message reply data structures. Message 31 has 5 replies so the in-descriptor reply points to a block in the reply spillover table. Two 4-element blocks in the spillover table are allocated but only five of the six possible entries are actually filled in/valid.

2.2 Message Parsing and Thread Construction

The MLA tools expect mail messages that conform to [6]. When a message is to be added to an archive, the header information is extracted and the body is stored in the message body database. Only the interesting header lines in a mail message are retained; all others are discarded. Mail message bodies can be stored preformatted as HTML or in their original form. Storing messages as HTML is most efficient if queries are to be done through the World Wide Web, as the overhead of parsing the body to insert HTML escapes (e.g., “>” for “<“) and for recognizing and inserting URLs can be done once (rather than each time the message is retrieved for display). It is also possible to retrieve message bodies with embedded HTML directives removed, but this can remove information if the original message included HTML directives.

The majority of the effort involved in entering a message into the archive is in establishing the relationship to other messages; specifically, determining if a message is a reply to another message and, most importantly, which one. This problem is difficult because there are no standard mechanisms used to identify the target message to which a message is a reply. Reliable reply information can usually be found in the “In-reply-to” and “References” lines in a header, but not always. It is not uncommon for users to use an existing mail message as a convenient means for formulating a new posting to a mailing list (to avoid having to enter the “To” address for the message); this can cause anomalous results in formulating message relationships. Also, some mail systems do not include either type of reply information, leaving only the hint of a relationship in the “Subject” line of a message (e.g., a leading “Re:” in the subject information).

Reply information is handled by the following scheme:

- Look for an “In-reply-to” field in the header.
- If no In-reply-to field is present, use any “References” header information.
- If no In-reply-to or References information is present, consider the “Subject” line.
- Extract identification information from the reply information (In-reply-to, References, or Subject). Note that doing this can be problematic because the format of these strings is not standardized.
- Search the archive for an existing mail message with matching identification and if the parent message was identified by something other than the Subject line, compare the subject information to weed out postings that are false relationships (see above).

This scheme can still link messages into a thread that do not belong, but such messages are usually impossible to deduce without actually interpreting the meaning of the message (e.g., someone that uses one posting to create an unrelated posting, leaves the subject line the same, but the content of the posting is totally unrelated to the original).

2.3 CGI Support and Navigation

The MLA tools are designed to be used through the Computer Gateway Interface (CGI) protocol that is used to connect static URLs to programs that execute on a server machine. Programs intended for use in this way all work as backends to query-style HTML forms (i.e., they take a collection of arguments and return an HTML document that is the result of applying the arguments to an MLA). The format and content of the information returned by a tool is defined by a template file that contains HTML directives and, optionally, escape codes that are replaced ‘‘on the fly’’ according to the contents of an archive or the result of a query.

An interesting aspect of the work done in this area is how the tools provide good navigational support in accessing the messages that comprise a query hit set. The basic problem is this: Given a query, how can the results be made available to clients across the World Wide Web in such a way that the context of the query result is maintained across multiple HTML pages. This problem is difficult because the underlying HyperText Transmission Protocol [7] is stateless and so any context that is to be maintained across multiple HTTP connections must be stored in the client, possibly in reference to some state on the server. In this case the state to maintain is the result of doing a query of the archive

11236 11243 11245 11248 11264 11237 11238 11239 11240 11253 ... 11271 11273 11272 11278 11279 11280 11281

11236 +0 +7 +9 +12 +28 +1 +2 +3 +4 +17 ... +35 +37 +36 +42 +43 +44 +45

NHits: 47 MinMsg: 11236 BitRange: 6 MsgDeltas: 0, 7, 9, 12, 28, 1, 2, 3, 4, 17, ...

`AAM5SmAAAC5OB0SeyJAdIOalDZAc...XcKdBgUaq2DgSF6UAAAebv`

Figure 6. Encoding a hit set. The set of message numbers is encoded as an ASCII string for embedding in URLs. First the set of numbers are converted to a base value and a set of positive offsets. Then the numbers are encoded as a bit string in which the offsets are bit-packed. Finally the bit string is converted to ASCII using a restricted alphabet.

database. This state can be stored in the client in one of several ways:

- as a query that must be redone each time the result is needed,
- as a reference to state cached on the server,
- as the result itself.

The first two options might be the same if query results are cached with the query string used as a key to reference cached results. This has the important characteristic that if the cached result on the server is no longer available then it can be recreated by redoing the query using the supplied key/query string. If query strings are large however then this scheme can be expensive. Also, caching state on the server potentially requires unlimited server resources to store results. Systems such as Alta Vista [11] and DejaNews [13] appear cache state on the server.

The last option, storing the query result in the client, works well only if the result is small enough that it does not cause a large HTML document to be generated. There is also the subtle requirement that results may be stored in a client for very long periods of time–months, maybe years–so references encoded in the result must work even if the database has changed.

The MLA tools use the third scheme to manage query results. The set of message numbers that comprise a query hit set are returned to the client in an encoded format. This is combined with some careful usage of the HTML language to minimize the size of HTML documents that are returned as a result of a query.

An encoded hit set is an array of message numbers. This information must be returned as an ASCII string comprised of a limited set of characters that are legal to pass as part of a URL. The set of hits is transformed to a 32-bit base message number and a set of offsets that are relative to the base message number. Offsets are expressed using the minimum number of bits required to represent the positive values. A fixed header is then prepended to this information and the entire result is encoded as ASCII using a restricted alphabet. This scheme works especially well when queries are constrained to reasonable time ranges since message numbers in the hit set are then closely packed. Figure 6 shows an example of this procedure.

The encoded hit set is used by programs like mlafetch to generate context-sensitive linkage such as the next message in a query result; linkage that could not otherwise be done without redoing the original query. Note that a side-effect of storing the result in the client is that if the contents of the database change while the client is navigating an HTML page, then the navigated view will remain unchanged even if redoing the query might yield a different set of hits. This is considered good from the standpoint of a user interface designer, though some people might prefer the alternative.

3.0 Performance

3.1 Space Efficiency

To save space in the archives, the MLA tools do the following when a new message is entered:

- unneeded mail headers are removed,
- the message body is stored compressed,
- common strings in the header are shared through the string pool.

This saved space is offset by the space required to store message relationship information and general overhead in the TOC database related to efficient operation. In addition there is overhead associated with the hashed database used to store message bodies.

Usage statistics for several archives are show in Table 1. The flexfax archive contains messages for about three years worth of postings to a fairly quiet mailing list (about 10 mail messages a day). The majordomo archive contains about four years of postings to the majordomo users list. The sgi.bad-attitude archive holds about four years of news article postings to an internal Silicon Graphics newsgroup. The www-vrml archive is about a year and a half of postings to the www-vrml mailing list.

Tests were run on a Silicon Graphics Indy workstation running IRIX 6.2 with the XFS filesystem (a filesystem that supports files with holes in them) and version 0.95 of the MLA tools. The

Archive	Messages		Threads		Replies		String Pool		Spillover Files	Space Usage	
	Total	Max	Total	Max	Total	Max	Total	Max		Data	MLA
flexfax	11265	14293	5645	7162	4264	5388	1.14MB	1.43MB	226	31.6MB	15.5MB
majordomo	10310	11481	5158	5890	4488	4840	1.08MB	1.43MB	79	23.7MB	13.9MB
sgi.bad-attitude	12317	14524	1571	1875	13136	15440	0.79MB	1.16MB	55	69.6MB	12.5MB
www-vrml	5470	5979	1975	2191	3896	4212	0.49MB	0.59MB	126	14.4MB	9.4MB

Table 1: Sample MLA statistics. The sizes of data structures are shown as the number of items in use (**Total**) versus the number of items to be filled before the TOC database must be expanded (**Max**). Spillover files are files created to hold message bodies that exceed the maximum size of a message that is stored directly in the message body database. Space usage gives the total amount of disk space used by the uncompressed input data (**Data**) and the final MLA (the TOC database, the message body database, and the spillover files).

hashed message body database files were created with version 1.85 of the Berkeley DB software. Other than a 3 kilobyte threshold for creating spillover files the default parameters were used in creating the archives.

The implementation uses a 56 byte message descriptor and 32-bit numbers for all thread and message numbers. The fixed size header at the front of the TOC is about 100 bytes. This results in TOC files that are very small in comparison to the space used to store message bodies; c.f. Table 2.

Two specific areas of space usage in the TOC were examined: the string pool and the reply spillover table. Table 3 summarizes statistics for the sample archives related to these two areas. The string pool was generally very effective in saving space; on average nearly half the space required to store the header strings was saved by sharing strings. This reflects the fact that most postings have at least one reply and so the subject strings in the messages can be shared. The more replies there are to a posting, the more sharing that will take place. Note however that optimal sharing requires periodic off-line compaction of the string pool since reply messages chronologically follow the initial posting so the subject string in the original posting cannot be compacted until after a reply is received.

The sample archives examined had very few unspecified header lines. As a result there were few null strings in the archive so the default string sharing for this special case was ineffective. This result would likely change if more header information were retained in the TOC database.

The overhead for the linked list scheme for storing reply information is more than 30%. That is, more than 30% of the space allocated to storing reply information is either used for links to following blocks in the spillover table or not used at all (i.e. unused entries in the last block of a list). This number indicates that most reply spillover blocks are fully populated since if every block were filled this figure would be approximately 25%. While the overhead may seem high, the actual amount of storage associated with this waste is relatively insignificant in the overall totals. An alternate scheme that allocates contiguous arrays in the spillover table might be more efficient, but would significantly increase the complexity of the software.

Archive	Input	TOC	MsgBodies		Total
			DB	Spill	
flexfax	31.6	2.06	12.0	1.44	15.5
majordomo	23.7	2.00	11.2	0.70	13.9
sgi.bad-attitude	69.6	1.91	10.2	0.44	12.5
www-vrml	14.4	0.92	7.6	0.88	9.4

Table 2: Sample MLA space usage. All numbers are megabytes of disk space allocated (no holes). Input refers to the raw input data. TOC is the table of contents database. Message body data is broken up into to the space used by the hashed database file and separate spillover files.

Archive	String Comp.	Null Strings	Reply Overhead
flexfax	1.88	13	32%
majordomo	1.87	55	35%
sg.bad-attitude	2.60	5	30%
www-vrml	1.97	2011	32%

Table 3: Sample MLA space efficiency. String Compression is the ratio of the space required to store strings without the string pool to storage with the string pool. Null Strings is the count of empty strings. Reply Overhead is the percentage of reply entries not used or used to link blocks in the spillover table.

3.2 Update Performance

Table 4 shows results for some sample update tests. The tests were all run on an SGI Indy system running IRIX 6.2 with the archive stored on an XFS filesystem. The machine had a 133MHZ R4600SC with 96MBytes of memory.

The flexfax, majordomo, and www-vrml archives were built from several gzip-compressed files that each contained multiple messages. The raw data for the sgi.bad-attitude archive had each message in a separate file; this is more indicative of normal usage where mlaupdate processes mail messages as they are received. In all test cases the average real time to process a message is less than a tenth of a second; this is consistent with the goal to keep update time small so that lock contention on the MLA is minimized for interactive users doing queries.

All the work that involves the table of contents database is done by directly manipulating a memory-mapped copy of the TOC file. The most expensive work is determining the thread relationship. If the message appears to be a reply then it may be necessary to search backwards chronologically over message headers to find a message's parent message. In normal operation adding a message to an archive involves touching only a few pages of the TOC file. In the worst case most of the string pool and message descriptor table must be touched. The sgi.bad-attitude test shows the most pronounced effect of this requirement; many messages in this archive are replies and the update time is noticeably higher than a similar-sized archive such as flexfax.

As noted in Section 2.1, allocating entries in the various TOC tables is typically inexpensive; just incrementing a counter in the TOC header. When the addition of a message would cause one of these tables to overflow, all the tables are expanded in-place; this operation can take several seconds depending on the characteristics of the underlying system (when there is sufficient memory to cache the contents of the TOC the grow operation can happen very quickly).

Archive	Total Time		
	User	System	Real
flexfax	163.86	51.22	259.75
majordomo	131.25	41.02	196.54
sgi.bad-attitude	217.19	61.02	423.52
www-vrml	82.02	23.62	135.88

Table 4: Build times for sample MLAs. All times are in seconds.

One thing to note about the operation of the mlaupdate program is that it is faster to process multiple messages together than to add each individually. This is because mlaupdate builds hash tables for message header strings to use in determining message relationships. When mlaupdate is given one message at a time to add to an archive it is likely these tables will be built multiple times (whether or not there is overlap depends on whether the messages are related).

3.3 Query Performance

MLA queries are done as a result of URL lookups that cause the MLA tools to be invoked as CGI applications. Consequently user perception of query performance is dominated by the overhead associated with the HTTP client and server programs and the underlying network performance.

Table 5 shows the average time for several exemplary query operations. These numbers were collected by invoking the MLA programs from the command line, using the same method by which they are invoked as CGI applications (i.e. passing arguments through environment variables). The results of the query were sent to the null device; e.g.,

```
mlaquery -h 1000 >/dev/null
```

Test runs were repeated 10 times each; the reported times are the average of these runs. No other queries were done in between so subsequent queries should find much of the information in the system buffer cache. Using a warm cache is representative of normal usage; if users do repeated queries they are typically done to narrow the hit set.

The test query operations are:

- 1000 items: return the most recent 1000 messages in the archive, no constraints are applied
- 5000 items: the same as 1000 items, but return five times as many messages

Test Query	Hits	Time		
		User	System	Real
1000 items	1000	0.85	0.07	0.95
5000 items	5000	16.80	0.42	17.06
date range	200	0.08	0.04	0.12
match 1 re	200	0.44	0.06	0.53
match 2 re	5	0.49	0.06	0.57

Table 5: Sample MLA query tests using the majordomo archive. All times are in seconds.

- date range: return the first 200 messages between 1/1/95 and 1/1/96
- match 1 re: constrain the date range query to messages with “BOUNCE” in the subject
- match 2 re: constrain the match 1 re query to messages sent by users with “Brent” in their name (case sensitive)

Constraining a query to a date range is fast because the TOC includes table of messages sorted by date. Any other constraints require mlaquery to apply a regular expression package to header strings that are taken from the string pool. This results in an increase in the query time that is typically proportional to the length of the strings that are checked. Note that mlaquery first applies any date range constraint before doing regular expression matching so any cheap culling of the hit set is done before the more expensive pattern matching.

One should note the 5000 items test result; the time to carry out this query is not a linear product of the number of items to return. This is because the HTML document generated by the query is not proportional in size because the encoded hit set returned in the URLs is significantly larger (due to the encoding scheme). The running time of mlaquery is dominated not by the time to do the query but instead by the time to generate the resulting HTML document. This is an example of the limitation of the scheme by which query context is stored in the client.

4.0 Summary and Future Work

The Mailing List Archive (MLA) tools provide an efficient system for storing and retrieving mailing lists. The ability to access and navigate archives via the World Wide Web is valuable; providing high quality graphical user interfaces basically for free. The underlying database design for the table of contents and message bodies has worked out well. The scheme for storing query context on the client side of an HTTP connection has proven successful so long as the hit set size is constrained to be reasonably small (several hundred).

The design and implementation of the MLA tools was done in early 1994. At that time, they were intended mainly to support a few private mailing lists; since then they have been used in a variety of settings with good results. The two main enhancements that have been requested are: support for searching multiple archives and searching the body of mail messages.

The first request–to search multiple archives– is contrary to the original design. In most cases, what people really want is not the MLA tools but the user interface and the ease of access that are provided by the tools.

The second request for searches over the body of mail messages has been added in an experimental version of the tools. Support was added to automatically construct inverted keyword indices from the entire text of each mail message. This new functionality permits queries to be formulated that search more than just the headers of a mail message. However, the indices are larger than desired (with only minimal filtering of uninteresting words they take up about equal space to the compacted message bodies) and compacting them opens up the usual set of issues: one can index fewer words by applying relevancy information to weed out unimportant information, or one can use data structures that support two-level queries such as those used in glimpse [2]. The indexing support was added in a way that makes it easy to evaluate alternate indexing techniques. This area is the subject of ongoing work.

5.0 Bibliography

[1] Kevin Hughes. Hypermail 1.02. http://www.eit.com/goodies/software/hypermail/hypermail.html.

[2] Udi Manber, Sun Wu. “GLIMPSE: A Tool to Search Through Entire File Systems”. TR 93-34. Department of Computer Science, University of Arizona. October 1993. http://glimpse.cs.arizona.edu:1994/.

[3] The RAND MH Message Handling System. UCI Version 6.8.3. October 29, 1995.

[4] NI, Usenet search gateway by Mike Burrows.

[5] Stephen R. van den Berg. Procmail & formail mail processing package. ftp://ftp.informatik.rwthaachen.de/pub/packages/procmail/procmail.tar.gz.

[6] D. Crocker, “Standard for the format of ARPA Internet text messages”, August 13, 1982. Updated by RFC1327, RFC0987. ftp://ds.internic.net/rfc/rfc822.txt.

[7] T. Berners-Lee, R. Fielding, H. Nielsen, “Hypertext Transfer Protocol -- HTTP/1.0”. May 17, 1996. ftp://ds.internic.net/rfc/rfc1945.txt.

[8] L. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3”. May 23, 1996. ftp://ds.internic.net/rfc/rfc1951.txt.

[9] Margo Seltzer, Ozan Yigit. “A New Hashing Package for UNIX”. USENIX Technical Conference. January 1991. Dallas, Texas. pp. 173-184.

[10] Jean-loup Gailly, Mark Adler. zlib data compression library.

ftp://ftp.uu.net/pub/archiving/zip/zlib/zlib0.99.tar.gz.
[11] Digital Equipment Corporation. "AltaVista Search". July 21, 1996. http://altavista.software.digital.com/products/search/whitpapr/.
[12] NCSA. "The CGI Specification, Version 1.1". http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.
[13] Deja News. http://www.dejanews.com/.

Sam Leffler is a member of the corporate research group at Silicon Graphics where he works on a variety of projects. His current focus is on processor scheduling and operating system support for high performance multi-threaded applications. Previous work has spanned the gamut from 3D graphics to computer networking to programming languages. He received an M.S in Computer Science and a B.S. in Mathematics from Case Western Reserve University, both in 1980.

Melange Tortuba is president of Tortuba Consulting, a small thinktank located in Berkeley California. She spends most of her time researching the needs and activities of overly-pampered domesticated felines. Her most well-known accomplishments to date have been in non-computer fields. She was educated at the University of California at Santa Barbara where she majored in mooching attention from students and faculty.