USENIX Association

Proceedings of the 17th Large Installation Systems Administration Conference

San Diego, CA, USA
October 26–31, 2003

An Analysis of Database-Driven Mail Servers

Nick Elprin and Bryan Parno – Harvard College

ABSTRACT

This paper compares the performance of three different IMAP servers, each of which uses a different storage mechanism: Cyrus uses a database built on BerkeleyDB, Courier-IMAP uses maildirs, and UW-IMAP uses mbox files. We also use a mySQL database to simulate a relational-database-driven IMAP server. We find that Cyrus and mySQL outperform UW and Courier in most tests, often dramatically beating Courier. Cyrus is particularly efficient at scan operations such as retrieving headers, and it also does particularly well on searches on header fields. UW and Cyrus perform similarly on full-text searches, although Cyrus seems to scale slightly better as the size of the mailbox grows. mySQL excels at full-text searches and header retrieval, but it performs poorly when deleting messages. In general, we believe that a database system offers better email storage facilities than traditional file systems.

Introduction

Most IMAP and POP3 servers, even ones commonly used in environments with many users, store mail data in flat text files. This suffices when the quantity of email is small, but when an email application must sift through hundreds of megabytes of inefficiently stored data, server performance can suffer dramatically. A recent analysis of a typical university file system found "a population of users who spend the majority of their file system accesses reading email" [1]. The paper concludes by noting: "While it may have been obvious a priori, flat-file mailboxes are quite inefficient. Anecdotal evidence suggests that database-driven mail servers are faster and consume fewer resources than file-system based ones, and we have no reason to dispute this."

Despite the alleged a priori obviousness of the benefits of a database-driven mail storage system – or perhaps because of it – there has been little empirical exploration of the potential for database use in mail servers. Yet the problem of email storage seems perfectly suited for a database solution [2]. Databases allow for more advanced and efficient content queries (particularly sorting and searching data) than flat files and, moreover, a cleanly structured database schema for email would better mediate the transfer and exchange of mail data to other applications and formats.

Background And Related Work

To our knowledge, little work exists comparing the performance of traditional file-based email servers with that of a database-driven system, although several papers do propose and defend various types of file-based email systems. Sam Varshavchik, for example, compares mbox with maildir mail storage [3]. The mbox system concatenates every message together and stores the result in a single file, while the maildir format stores each message in a separate file. Even though Varshavchik concludes that maildirs will match or outperform mbox systems in all but a few isolated cases, he admits that maildirs still contain inherent flaws. Among them, he mentions that maildirs will perform poorly if used for specific content searches on large folders, particularly if the server runs on a machine with an inefficient file system.

Mark Crispin also highlights the shortcomings of file-based mail storage [4]. He notes that most operating systems synchronize file creation. As a result, any attempt to create or delete email runs into a narrow bottleneck at the level of the file system. Furthermore, a text search requires opening and closing every file in the mail directory, placing a considerable strain on the file system. In the end, Crispin concludes that a database-driven solution would result in superior performance, but he offers no arguments to support his position, other than his comments on the shortcomings of file-based systems.

Several companies offer email server products based on database storage. DBMail offers programs that "enable the possibility of storing and retrieving mail messages from a database" [5]. The site claims the system performs faster queries, scales better, and provides better flexibility than file-based storage, but it offers little reasoning or evidence to support these claims.

Openwave Email Mx (formerly Intermail), a product from Software.com, uses Berkeley DB, an embedded data store, to archive messages. But again, aside from its claim that Email MX sets "new standards of excellence with its massive scalability, unrivaled performance, and superior architecture," the company provides no information on the server's performance compared to traditional mail systems [6]. Similarly, Citadel also bases its mail storage on a database, but offers no evidence to support this decision [7]. In fact, the only argument advanced in

favor of using a database relies on creating a single instance of each email that enters the system, so that if multiple users receive the same email, the system stores only one copy. Without evidence to support the conjectured performance improvement, we learn little about the relative merits of database versus file-system storage.

Thus, although researchers have argued over the advantages and disadvantages of different file-based mail storage systems, they have neglected the subject of database-driven solutions. A few companies do offer such solutions, but as far as we could find, none provides any evidence to justify the alleged benefits of this alternative.

Experimental Setup

We tested IMAP servers with three different storage mechanisms. Courier IMAP (v. 1.4.2) uses maildirs, UW-IMAP-2002 uses mbox files, and Cyrus (v. 2.1.9) uses BerkeleyDB; additionally, we created an email database in mySQL (v. 3.23.49) and used it to model a relational-database-driven IMAP server. Other mailbox formats, such as the more performance-oriented mbx format, do exist, but we believe the popularity of mbox and maildirs justifies their use in these tests. And although other database-driven servers also exist – the few noted earlier, as well as more widely known applications from Microsoft and Sun Microsystems – we chose to include only non-commercial servers in our comparison. All servers were run on a machine running NetBSD 1.6 with a 200 Mhz Pentium processor, 128 MB of RAM, and a 1.5 GB IDE hard drive.

Our benchmark consists of seven basic requests, and we perform each on three different mailbox sizes:[1]

Test Mailbox Sizes			
	Small	Medium	Large
Size	5 MB	30 MB	100 MB
Number Messages	1,046	8,734	21,282

All users have a single folder named ‘INBOX’.

Our setup attempts to model email usage and configuration of users from three different categories. The small user represents a typical webmail user with a relatively small quota. The medium user represents the average size for a commercial or academic email account. The large mailbox typifies a high-volume account which regularly accumulates thousands of emails, such as a corporate account. Our basic configuration models the typical naïve user who saves every message in the inbox.

Our mySQL database schema consists of a table of messages for each account size, small, medium, and large. Each table stores header information and the message body in separate fields. We strategically defined indices on properties we expected to be particularly relevant to typical IMAP usage. For instance, we created a full-text index on subject, body, sender, and recipient. Thus the mySQL model and Cyrus used storage mechanisms designed to structure data intelligently, while UW and Courier used standard file system structures. Arguably, the playing field would have been more level with a standard IMAP server altered to use a mySQL database rather than a database model independent of the rest of the IMAP server. We chose the latter option for a number of reasons: we wanted to test publicly available software rather than customized versions; we believe overhead of accessing the data source will outweigh other factors necessary to run an IMAP server; and our time and resource constraints made it difficult to integrate mySQL storage into an existing IMAP server.

To generate a large collection of diverse email, we created an email alias (namely emagnet@hein.eecs.harvard.edu) to the three accounts on each server and then subscribed emagnet to a large variety of mailing lists. Thus, each account received exactly the same email. This approach allowed us to measure each server’s performance on real email, rather than artificially created messages. It also guaranteed diversity in the mail headers, so that we could accurately test the servers’ searching abilities.

The benchmark times the following requests:

1. *Retrieve all headers in the user’s INBOX.*
   All mail clients must routinely perform this operation to download new mail and refresh cached information, so the server’s response should be quick and efficient. This procedure is analogous to a scan operation that returns a single field’s value for every tuple in a relation.
2. 2a. *Retrieve a list of messages whose full text (subject and body) contains “happy” (chosen to match approximately 1 percent of the messages).*
   2b. *Retrieve a list of messages whose full text contains “free” (chosen to match approximately 25 percent of the messages).*
   The IMAP protocol allows users to retrieve mail that matches conditions a user can specify in a simple query. Performance of this operation must scale well with the size of the mail data and the number of successful matches.
3. 3a. *Retrieve a list of messages whose sender is “parno@fas.harvard.edu”*
   3b. *Retrieve a list of messages received on November 30, 2002*
   Searches on the header fields may be faster if the mail server does not need to open an entire message file to perform the search. A storage system that allows selective access to only those parts of the message that the search requires should outperform a flat-file system on tests like this.

[1] In addition to the three user accounts described above, we also tested a tiny account consisting of 105 messages (0.5 MB). In virtually every test, the results paralleled those of our other three accounts. However, the time differences between servers proved trivial, so we chose to omit them from our results for the sake of brevity.

4. 4a. *Expunge 1 percent of the user's mail.*
   4b. *Expunge 20 percent of the user's mail.*
   Deleting mail is routine maintenance that all email users must perform regularly. This test measures the time necessary to fully purge all mail flagged for deletion. The flagging of messages for deletion was done randomly.

We measured times from the perspective of the mail client application. That is, the time necessary for the request is the difference between the time at which the client sends the request and the time at which it receives data back from the server. We ran each of our tests cold, i.e., the server had not previously accessed the data we queried. To avoid network-related timing delays, our benchmark client runs on the server. We ran each test (except expunging deleted messages) three times; results are the average of the three numbers.

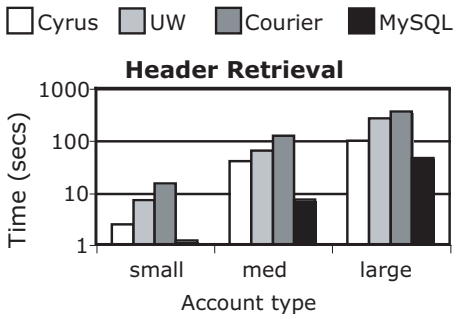
Results

Header Retrieval Times

First, the header retrieval times in tabular format:

Header Retrieval Times (in seconds)			
	Small	**Medium**	**Large**
Cyrus	2.36	39.3	98.2
UW	6.83	61.9	263.7
Courier	14.37	119.4	356.7
mySQL	1.16	7.08	44.79

All of the standard deviations were less than 1.0. More graphically, this looks like:



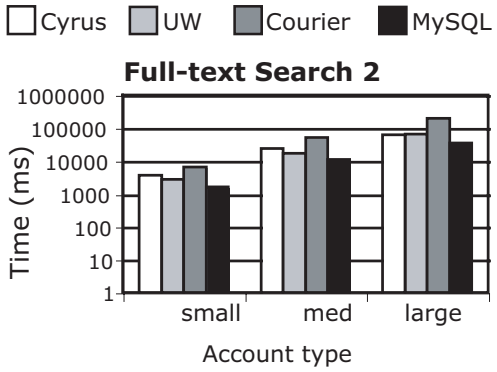
For this benchmark, a retrieval of all message headers in a user's inbox, both database-solutions outperform the file-based servers, with the performance differential increasing as the size of the mailbox grows. An intelligent index on the header field makes this a simple sequential scan for both Cyrus and mySQL. Furthermore, our SQL schema gives mySQL an advantage over Cyrus: mySQL isolates each user's messages into a unique table, while Cyrus must scan through all messages to select only those from the account requested. In contrast to both of these, UW must search through the entire mbox file. Worse still, Courier must open and close hundreds – thousands for medium and large – of files and then scan through each file to find the relevant information.

Full-Text Searches

Figure 1 show the tabular and graphic results of searching on "happy" and "free." All of the standard deviations less than 0.3 except as noted: † standard deviations less than 1; ‡ standard deviation = 26.8.

Search on "happy" (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	4.04	25.64	65.6†
UW	2.92	17.76	64.22
Courier	7.76†	52.2†	209.1‡
mySQL	0.71	5.49	15.28

Search on "free" (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	3.81	24.55	62.4†
UW	2.79	17.43	64.19
Courier	6.56†	50.5†	193.6†
mySQL	1.62	11.32	35.77



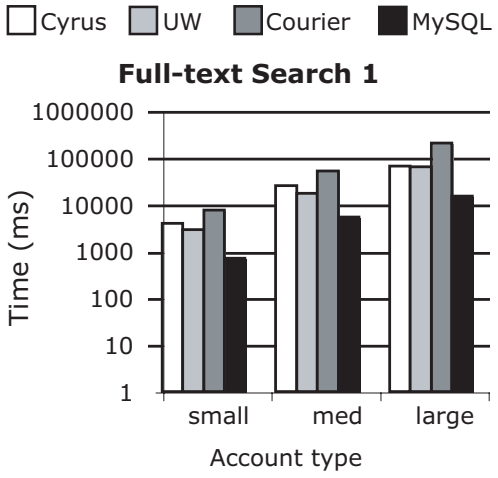


Figure 1: Retrieval times for searching on "happy" and "free".

This test measures the speed of a full-text search on messages' subjects and bodies. On the small and medium accounts, UW responds slightly faster than Cyrus, though both outperform Courier by a significant margin. Cyrus's somewhat disappointing statistics may result from the fact that it must first isolate the specific user's mail from all stored messages, and only then can it perform a full-text search. Meanwhile, UW performs a relatively simple text search through a single file. Simple file searches such as these can often outperform databases by avoiding the extra overhead dictated by a DBMS. As the account size grows however, Cyrus's database scales better than UW's flat files: Cyrus's times on the large-account search are faster than, or about equal to, UW's times.

Beating UW and Courier by a much wider margin, mySQL clearly demonstrates the advantage of intelligent indexing. Although it takes initial overhead to precompute[2], the full-text index allows mySQL to find text in message bodies much more quickly than all three other solutions. Overall, this suggests that a database solution will be most efficient when indexing is utilized or a user's mailbox constitutes a majority of the total mail stored on the server. The disadvantage of a database solution is that the combined size of all accounts affects its performance on each account, so the worst-case scenario is when a user's account is small but the server stores a large quantity of mail in other accounts.

[2]One could argue that the cost of maintaining a full-text index would also slow the speed with which the server could store incoming messages. Even if the cost of indexing a single message were significant, however, it would not be relevant to our benchmarks, which time queries from the perspective of the end user. It is unlikely that indexing incoming messages would affect the perceived speed of a user trying to search messages, for example.

Searching Specific Header Fields

See Figure 2 for the graphical results of searching the 'From' and 'Date' header fields. All standard deviations were less than 0.1, except as noted: † standard deviation = 3.6; ‡ standard deviation = 0.93 .

On a search over specific header fields, Cyrus and mySQL once again outperform both file-based servers on all three accounts, with the exception of UW's superior performance on the date search on the small account. Given that the database solutions still scale in a far superior manner, this exception seems negligible. An index on the header information makes this search an extremely simple database problem, though without the overhead of a traditional relational database, Cyrus can execute these queries more quickly than mySQL. UW, while still slower than Cyrus, outperforms Courier – UW merely performs the equivalent of a grep on a single file, while Courier must open and close every single message file.

Expunging

Figure 3 shows the times for expunging 1 and 20 percent of the box's mail.

Expunging messages flagged for deletion proved to be a particularly elucidating test. Cyrus dramatically beats UW when purging a relatively small number of messages but takes much longer than UW when the number of deleted messages grows to 20 percent. Opening the mbox file and rewriting it are the most costly disk operations UW must perform; identifying messages for deletion once the file is open should be relatively quick, and the more messages UW deletes, the less data it must write back to disk. In contrast, although Cyrus seems to have quicker access to the messages it must delete, it scales more slowly when it

Searching 'from' Header (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	0.03	0.25	0.55
UW	1.02	6.67	20.33†
Courier	5.37	45.4	164.9‡
mySQL	0.52	0.85	3.51

Searching 'date' Header (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	0.06	0.18	0.36
UW	0.026	0.39	1.21
Courier	0.35	3.78	56.1
mySQL	0.27	0.32	0.8

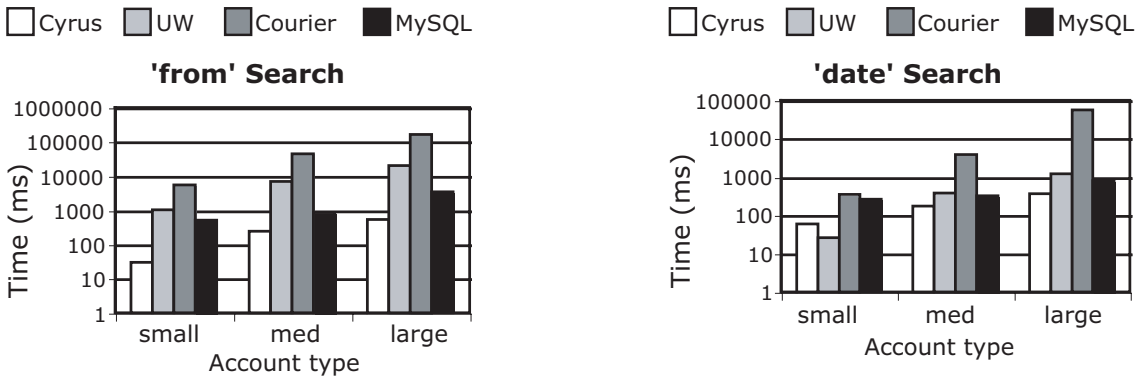


Figure 2: Timing for searching header fields.

deletes a large number of messages. Cyrus must also update its indices as it deletes, and this can be a costly operation for a deletion as large as 20 percent. Data for mySQL further demonstrates the cost of maintaining extensive indices: although our SQL schema allows mySQL to perform queries quickly, its performance suffers severely when it must delete messages. Courier's deletion times scale approximately linearly with the number of messages. Courier's deletion process requires no significant overhead, it is simply a matter of deleting one file for each message.

General Analysis

In general, the database solutions, Cyrus and mySQL, consistently outperform the file-based servers in both response time and scalability, often by orders of magnitude. Moreover, Cyrus and mySQL have buffer caches that enable even faster performance than our results indicate.[3] Executing queries on the same data sequentially yielded significantly faster results for both Cyrus and mySQL. For example, although Cyrus requires four seconds to perform the first-full-text search, running a second full-text search immediately after the first one requires only one second. Similarly, a search on recently queried data in mySQL often takes less than a tenth of a second.

For Cyrus, warm runs only yielded faster times on the full-text searches. It presumably has indices on header fields, so caching records would not help it perform searches on these data. Warm runs yielded a similar performance increase on the medium account but had no effect on Cyrus's large account, probably for the simple reason that the large account has too much data to cache. MySQL, in contrast, exhibited significant performance improvements on all accounts and all queries when using warm data. UW-IMAP and Courier's times remained unchanged on warm runs, however, so clearly caching endows database implementations with a significant advantage over file-based storage mechanisms in certain contexts.

Compared with UW and Courier, Cyrus does perform poorly on our twenty-percent deletion test, and mySQL performs embarrassingly on both deletions. We do not believe this is a significant problem, however, as users rarely delete such large quantities of mail at once; instead, we suspect the one-percent deletion occurs far more regularly. Furthermore, deleting a large amount of email can happen, by nature, only infrequently, while header retrieval takes place constantly, so it is far more important that a mail server handle the latter task efficiently.

Although mySQL outperforms the other solutions in header retrieval and full-text searches, it has the advantage of avoiding the overhead of an actual IMAP server implementation. For example, mySQL avoids network connections, message-wrapper data structures, and IMAP command parsing. In general, however, the majority of the time processing each benchmark consists of executing the query rather than performing quick operations like parsing the IMAP commands. Furthermore, since none of our benchmarks measured the time needed to connect to the servers, adding an actual IMAP server on top of the mySQL database would not add to the overhead of a connection. Thus, our simulation presents a reasonable approximation of a relational-database solution.

[3] Client-side caching is also important for performance from the end-user's point of view, but it is not germane to our current discussion. Please see Varshavchik's analysis [3] for a more thorough discussion of performance and client-side caching.

Expunge 1% of mail (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	0.68	4.52	12.13
UW	0.82	6.14	53.69
Courier	0.41	4.72	13.69
mySQL	2.84	29.47	120.56

Expunge 20% of mail (times in seconds)			
	Small	**Medium**	**Large**
Cyrus	6.17	64.63	156.50
UW	0.93	8.25	69.25
Courier	15.18	61.83	195.76
mySQL	7.71	155.14	1086.39

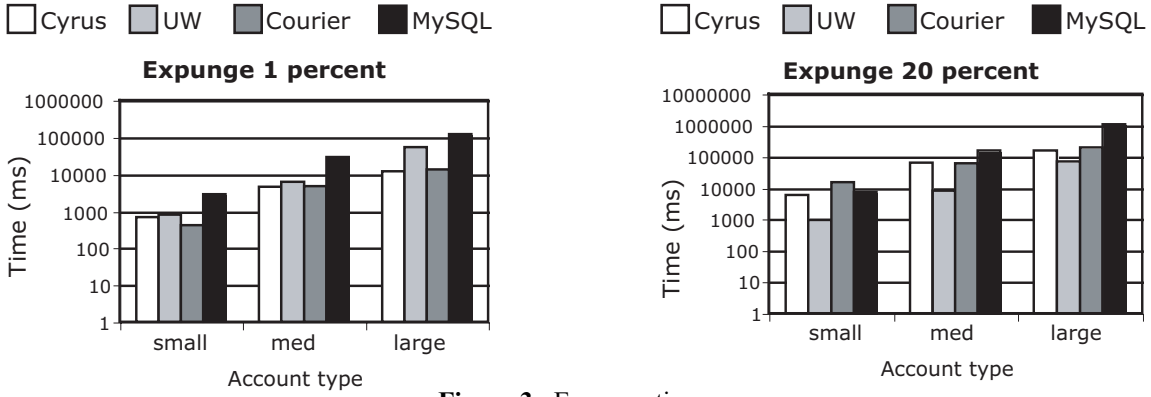


Figure 3: Expunge times.

Our results differed dramatically from Varshavchik's [3], the only other empirical data we have found. Varshavchik concludes that ''using maildirs will be just as fast – and in sometimes [sic] much faster – than mbox files,'' while our results indicate that Courier's maildirs scale terribly compared to the storage mechanisms of Cyrus and UW. We believe at least two factors contribute to this discrepancy. First, according to Varshavchik, ''Maildirs will not scale very well on servers that use old, slow, hardware.'' Indeed, his ''low-end'' configuration, on which UW did outperform Courier, is similar to our hardware configuration. Also, we tested accounts with both more messages and a greater aggregate size than the accounts in Varshavchik's benchmarks. Instead of upper limits of 10,000 messages and 40 MB, we tested accounts up to over 20,000 messages and 100 MB. Similarly, Varshavchik's benchmarks automatically generated batches of homogeneous messages while we used real email from an eclectic mix of sources; our email contained heterogeneous content and significant variance among message sizes.

Areas For Future Research

Varying some of the benchmark parameters could yield additional interesting and illustrative results. For example, no sane email user would keep 21,000 messages in one folder, so it would be more realistic to run tests with mail distributed among multiple folders. Since the IMAP protocol allows a user to work with only one folder at a time, we felt the results from such tests would not differ significantly from our other results, but this is certainly open to further investigation. In general, more advanced queries, such as a search over multiple header fields, would also offer room for more exploration.

With fewer hardware limitations it would be much easier to run our benchmarks on scales that more accurately resemble large corporate and academic computing environments. It would be interesting, for example, to see how well the different storage systems perform when the server has hundreds of users and must store over a gigabyte of email. Similarly, an investigation of server performance during concurrent access from multiple clients could reveal other advantages of a database storage system; unfortunately, limitations in our experimental setup prevented us from pursuing this inquiry.

Additionally, although our study examines email server performance from the client's perspective, a system administrator may care more about the memory, processor time or disk I/O usage of the mail server, all of which may vary based on the storage system selected.

Finally, we believe that database storage systems will allow mail servers to evolve more advanced features, and an analytic investigation of these possibilities could be a fruitful endeavor. For example, advanced queries could allow for server-side junk mail filtering. More generally, client email applications can be more compact and efficient if they can rely on the server for additional mail-management features, and this could stimulate the development of truly practical email clients on small mobile devices such as cellular phones.

Conclusion

Many features of a DBMS are highly advantageous from the point of view of an IMAP server. The obvious performance differential between the database options and both UW and Courier indicates that email storage is indeed a problem well suited for a database solution. Indexing capabilities give Cyrus and mySQL an advantage over Courier and UW when scanning headers and searching header fields. mySQL's full-text index provides a particularly expedient method for searching through message text, although it adds significant maintenance cost to operations such as adding and removing messages. A server-side buffer cache also improves performance by speeding up searches on recently accessed data. Although UW outperforms Cyrus by a small margin on some full-text searches, mySQL demonstrates clearly that a DBMS can search email much more quickly than a file-based solution. Most importantly, these results offer desperately needed empirical data comparing the performance of these three storage implementations.

Acknowledgements

We would like to thank David Holland and Noam Zeilberger for their technical assistance and UNIX troubleshooting, as well as Stuart Schechter for suggesting this line of research. Dan Ellard and Margo Seltzer were both indispensable for their thoughtful criticism and willingness to help.

Author Information

Nick Elprin (elprin@fas.harvard.edu) is a third-year computer science undergraduate at Harvard College. Bryan Parno (parno@fas.harvard.edu) is a fourth-year computer science undergraduate at Harvard College.

References

[1] Ellard, Daniel, Jonathan Ledlie, Pia Malkani, and Margo Seltzer, ''Passive NFS Tracing of Email and Research Workloads,'' *Second Annual USENIX File and Storage Technologies Conference*, pp. 203-216, San Francisco, CA, March, 2003.

[2] Silberschatz, Avi and Stan Zdonik, et al., ''Strategic Directions in Database Systems-Breaking Out of the Box,'' *ACM Computing Surveys,* Vol. 28, Num. 4, pp. 764-778, December, 1996.

[3] Varshavchik, S., http://courier-mta.org/mbox-vs-maildir/ , 2001.
[4] Crispin, M., *Mailbox Format Characteristics*, http://washington.edu/imap/documentation/formats.txt.html , 1999.
[5] *DBMail*, http://dbmail.org .
[6] *Openwave Email Mx*, http://openwave.com .
[7] *Citadel*, http://uncensored.citadel.org/citadel/ .