Scaling up Cambridge University's email service

David Carter <dpc22@cam.ac.uk>

Tony Finch <fanf2@cam.ac.uk> <dot@dotat.at>

University of Cambridge Computing Service

ABSTRACT

One of the central services provided by the University of Cambridge Computing Service is the email system, Hermes. Since its introduction in 1993 it was based on UW-IMAP and a shared Unix filestore. This architecture has a number of inherent limitations in performance, scalability, resilience, and cost-effectiveness. In order to address these problems, and particularly users' desire for larger messages and disk quotas, the system is being replaced with a mostly-compatible reimplementation based on multiple CMU Cyrus message stores.

This paper explains the effects of the limitations of the old system, including the restrictions imposed on our users, and the implementation of our webmail software, Prayer. We describe how Cyrus has been modified to make the transition to the new system easier, and to address the anticipated reliability and administration problems caused by using more inexpensive hardware. Finally, after covering the changing role of our central email relay in the new architecture, we discuss some possible development projects for the future.

1. Introduction

Hermes is the University of Cambridge's central email system. Almost 32,000 students and staff have Hermes accounts, and most of them use it at least once in the course of a week. Hermes is now ten years old, and it has grown exponentially through most of that time. The number of users and the load they impose is now fairly steady (though it is changing in character as people move from the terminal interface to webmail); however email volumes are still increasing, both in terms of message count and message size.

Protocol	Active users	Concurrent	Logins/hour
Webmail	17500	2000	4500
IMAP	7800	5000	24000
POP	6700	50	40000
Terminal	5000	500	950
Total	24700	7500	70000

	messages	megabytes
daily volume	300,000	10,000

Fig. 1: Typical load on Hermes

For a few years we have been considering a significant change to Hermes's architecture that will allow us to improve the quality of the service it provides. The goals of this project are:

- Larger quotas and message size limits;
- Better data recovery options;
- Better disaster recovery options.

We have not addressed high availability at this point.

The constraints on the project are cost and manpower. The budget for Hermes, including staff costs, hardware purchasing and maintenance, and software licensing and support, is £10 per user per year. The number of staff developing, maintaining, and running the system is two.

This paper was presented at the UK Unix User Group Large Installation System Administration Winter Conference on the 25th & 26th February 2004 in Bournemouth, and is published in the conference proceedings and on the Web at `http://www.cus.cam.ac.uk/~fanf2/hermes/doc/talks/2004-02-ukuug/`

The rest of this section contains some general background information as an introduction to the ideas used later on. We also cover some of the historical background, since some context helps with understanding the technical decisions that we have made.

The second section describes the old Hermes system in more detail, explaining its limitations in performance, scalability, resilience, and cost-effectiveness. These limitations mean we have had to impose tight restrictions on our users, and we have even implemented our own webmail system in order to avoid performance problems.

The third section describes the implementation of a replacement Hermes system based on the CMU Cyrus IMAP server. The following section describes the changes we have made to improve Cyrus's performance and reliability. Then we explain how we are managing the transition between the old and new systems to minimize disruption.

Finally, we describe some plans for the future. These include further reliability improvements, and support for more features.

1.1. Large-scale data storage

The basis of a large service like Hermes is a scalable and reliable data storage infrastructure. As well as scaling up for data size, performance is also important. While the storage capacity of disks is increasing rapidly, their speed is not. It is now possible to obtain ten times the storage of the old Hermes system at a fraction of the cost and occupying a fraction of the space; however it would not provide adequate performance because of the reduced number of disks.

An important performance-related figure is the number of users per disk, since the number of transactions per second required varies with the number of concurrent logins. This must be considered alongside the total storage required. On old Hermes there are about 1,000 users per disk, and on new Hermes about 300. This means a system designed to use lots of relatively small disks, rather than a few large ones. The difficulty of obtaining small disks means there is plenty of extra space to use creatively.

1.1.1. Block-level distribution

Storage-area networks use Fibre Channel[1] to connect computers to disk storage via a switched network. Logical Volume Management (LVM) is used to combine and/or divide the available storage into useful chunks. Special file systems such as the Global File System[2][3] or the VERITAS Cluster File system[4] are required in order for multiple front-end computers to be able to mount the same logical volume at the same time in order to scale up the front end. A SAN-based system can be scaled horizontally by adding more front-end computers, back-end disk storage units, and intermediate network switches. Since all the computers can access all the storage, any of them can satisfy requests from users.

Although SAN hardware is becoming more affordable (and indeed new Hermes uses it in the backup server), the main problem from our point of view is the commercial nature of the software. We prefer open source solutions both for reasons of cost and because of the quality of support[1]. Although Linux is generally supported as a SAN platform, versions before 2.6 limit the file system size to 2TB which makes it a less than ideal choice for this kind of storage architecture.

1.1.2. Filesystem-level distribution

Network-attached storage refers to file systems accessed over an Ethernet network using a protocol such as NFS or CIFS. This technology is older and more familiar than SAN technology, but doesn't have such good performance. Although the layer of abstraction is slightly higher, NAS-based systems are scaled in a similar way to SANs. There are significant problems with the semantics of NFS, especially when it comes to locking. This makes it a poor choice for applications that make heavy demands of the filesystem, including more advanced email server software.

The old Hermes system uses a pair of NetApp[5] file servers. These have very good performance and reliability, plus some nice features like file system snapshots[6] which are useful for recovering accidentally-deleted data. However they are very expensive—a significant fraction of our total budget. They are also a single point of failure: if one of them fails the whole of Hermes must be shut down. Fortunately they don't fail very often.

[1] At one point in the history of Hermes, the system was suffering horribly from a memory leak in the Solaris kernel. Although we have enough local programming expertise to fix this kind of problem, the only solution we could use was a daily reboot of the servers. See also the discussion of the problems with `rpc.yppasswdd` on old Hermes elsewhere in this paper.

1.1.3. Application-level distribution

This method of scaling can take a number of approaches which may be more or less appropriate depending on the application. In applications where the data is mostly read-only, caches can be used to keep copies of data closer to users and reduce the load on the back-end server(s). Examples include the DNS and web caches (whether for outgoing connections from a site, or wide-area server distribution such as Akamai[7]). Where the data is more volatile, as in the case of email, it is more common to partition the back-end server and put in front of it a mechanism to direct user requests to the correct partition. The redirector preserves the appearance of a single system. Examples include `syslogd`'s log message routing functionality, HTTP reverse proxies[8] and load-balancing layer 4 switches like the F5 Networks BIG-IP[9].

This method of scaling a system depends on appropriate application-specific technology being available. However when it is, it is often available as open source software. In our situation, a significant advantage of using an email redirector proxy is that it allows transparent migration of users between different back-end servers[10], and provides a way of implementing shared mailboxes in the future[11]. This is the approach used by the new Hermes system.

1.1.4. User-level distribution

If redirection technology isn't available, it may be possible to make the partitioning of the server visible to users by using some appropriate naming scheme. In the case of HTTP, different server names may be used for static and dynamic content, e.g. `www.example.com` vs. `cgi.example.com`. In the world of academic email, it is relatively common for staff and students to be separated between different mail domains, e.g. `staff.cam.ac.uk` vs. `student.cam.ac.uk`; however this does not give much opportunity for further scaling. Alternatively the division might be by department, e.g. `phy.cam.ac.uk` vs. `eng.cam.ac.uk`; however the baroque complexity of people's affiliations in the University of Cambridge makes this unattractive to us. The most fine-grained possibility is to have a different name for the service for each user, e.g. `imap.dpc22.hermes.cam.ac.uk` vs. `imap.fanf2.hermes.cam.ac.uk`, and use the DNS to direct users to the right server; this is the approach used by Oxford University for their email system, Herald[12]. However we don't like the idea of getting tens of thousands of users to reconfigure their email software.

This approach also has some wider problems. Client software must be configured specifically for each user, which makes shared workstations less convenient. It also makes it difficult to support sharing of data between different users. In our case, although there is a system in IMAP for mailbox referrals[13] the support for it in IMAP clients is too patchy for it to be usable.

1.2. Mailbox formats

Even if an email server has a fast and scalable disk storage system, its performance is greatly affected by the format it uses for its mailboxes. There are two principal choices: whether to use one file for the whole mailbox or to use a directory containing each message in a separate file; and whether or not to keep an index of the mailbox. The file/directory choice has a side-effect: directory mailboxes are "dual-use", i.e. they can have subordinate mailboxes which is not the case for flat files.

The following table illustrates the principal mailbox formats that this paper is concerned with. The UW-IMAP distribution includes a document with some details of other mailbox formats not mentioned here[14].

	raw	indexed
flat file	Unix mailbox	Cambridge idx
directory	qmail maildir	Cyrus mailstore

Fig. 2: Characteristics of common mailbox formats.

Traditional Berkeley Unix mailboxes consist of a series of messages concatenated together, separated by marker lines which start "`From `". Their main advantage is that they are a very old and widely supported standard, which makes them useful for transporting email between different systems. They have a number of problems:

- Instances of "`From `" in messages are escaped with a ">" in order to avoid confusion with message separator lines. Unfortunately lines starting with "`>From `" are not also escaped, so this minor corruption is not reversible.
- When listing the mailbox the whole file must be parsed in order to find each message, and to extract metadata such as header information and status flags (e.g. "read" or "deleted").
- Removing a message from the mailbox causes the suffix of the file following the message to be rewritten. Some POP clients abuse the protocol in order to keep a copy of the user's email on the server for a certain number of days. This causes the mailbox to be repeatedly rewritten as messages are removed one-by-one. IMAP clients, on the other hand, usually remove several messages in one "expunge" operation.
- A locking protocol is required to mediate access to the mailbox by multiple programs (e.g. the Mail Delivery Agent and the Mail User Agent). This is particularly troublesome if the mailbox happens to be stored on an NFS partition.
- It's very difficult to implement concurrent access to a mailbox by multiple clients, even if they are both running on the same computer. If the mailbox is shared over NFS then cache consistency problems make this effectively impossible with adequate performance.

Other single-file formats include that used by MMDF[15] and the UW-IMAP MBX format[16]. The principal advantage of MBX over the Berkeley format is that it allows concurrent access by multiple clients to the same mailbox.

Bernstein's maildir mailboxes[17] are designed to allow certain operations (including delivery, status changes, and message removal) to be performed without user-level locks. This makes it much simpler to use on an NFS file store, and particularly good for POP clients. However it also has downsides:

- The lack of user-level locks is made possible by the implicit locking on filesystem metadata inside the operating system. This can cause performance problems from increased contention between processes and can be very slow on filesystems with synchronous metadata updates.
- Unix filesystems are traditionally very bad at dealing with directories containing large numbers of files. More modern filesystems (such as ReiserFS[18] and XFS[19]) make this less of a problem, but they may not always be available.
- When listing the mailbox, every file must be opened, read, and parsed. This is even more expensive than for Unix mailboxes, because of the increased number of system calls and context switches.
- It's difficult to efficiently implement per-mailbox quotas. These are useful for limiting the performance problems caused by large mailboxes. Summing the file size of every message in a maildir is another expensive operation involving many system calls.

The Courier email system[20] uses maildirs, and the developers are happy with their performance[21], though with the caveat that good performance requires a cacheing IMAP client. The RAND MH[22] format is comparable to maildir, though has a less efficient file naming scheme.

Adding indexes to a mailbox format can significantly improve its performance. An index reduces the amount of data that needs to be read and parsed when opening a mailbox, which can be especially important if the filesystem is NFS. In the case of directory-based mailbox formats they also greatly reduce the number of system calls that is required. Later in this paper we'll describe how the old Hermes system adds indexes to Berkeley mailboxes and how CMU Cyrus gains efficiency from its indexes.

It's possible to go even further along the indexing route, and put the message store in a database system[23]. This is the approach taken by Microsoft Exchange. The problem with going this far is it becomes more difficult to diagnose and repair problems, since custom tools are required to manage the database. In addition to that, the view of a message store presented by IMAP is simpler than that provided by a relational database, so a simpler data store can provide adequate performance.

1.3. History of email in Cambridge

We'll briefly describe some of the systems run by the Computing Service that are or were relevant to email in Cambridge. This is partly to introduce some of our local terminology, and partly to explain how the old Hermes system came to work as it does.

1.3.1. Phoenix

From the early 1970s until the early 1990s the CS was dominated by an IBM mainframe called Phoenix. It ran locally-developed software initially on top of MVT and from the early 1980s MVS. Network connectivity was provided by the JANET X.25 service using the ''Coloured Book'' protocols for applications including email from 1986. TCP/IP was supported in the 1990s, but Internet email still had to be transferred via gateway systems. Phoenix was shut down on the 1st September 1995[24].

1.3.2. The Central Unix Service

Unix came to the CS relatively late. Though there were a number of installations of it around the University, including in the CS's sister institution the Computer Laboratory, it had not been considered reliable or friendly enough to be the basis of a large timesharing service. Eventually demand from users for Unix application software became insistent enough that CUS was introduced in 1990[25]. This service is generally reserved for staff and graduate students; undergraduates only get accounts in exceptional circumstances. CUS was the first system to run Exim[26], and is still used for pre-release testing.

1.3.3. The PP switch

Cambridge's central email hub is called `ppsw`, after the software it originally ran[27]. PP[2] could handle both Internet and ''Grey Book'' email, and perform protocol translation between the two. After the closure of Phoenix, the use of Grey Book email in the University declined rapidly. At about the same time, Philip Hazel was writing Exim as a replacement for Smail[28] which was the MTA running on CUS and Hermes. Exim included features needed by `ppsw` such as virtual domains and policy controls, and omitted support for non-Internet addressing such as Grey Book and UUCP. Exim is now used on all CS systems. The central switch was introduced in 1992[29], so predates and was originally separate from Hermes. Over time the two systems have grown closer, and as well as performing email routing between the various parts of the University, `ppsw` now also handles routing within the new Hermes system.

1.3.4. Hermes

The end of Phoenix meant that a system was required to provide email for those without CUS accounts. The view at the time was that computing was moving from central timesharing facilities to workstations, so the email system would be principally a message store accessed remotely from PCs. Hermes came into service in 1993[30], and its early years were characterized by very rapid and painful increase in load as email became more and more popular. It took rather longer than anticipated for terminal access usage to decline, and it was displaced more by webmail than by pure IMAP. Now that everyone has a Hermes account who wants one, the pressure is no longer for more users and more messages, but instead larger messages, larger quotas, better performance, and better resilience.

Apart from scaling up the hardware and upgrading the software, the overall design is the same as 10 years ago. The new Hermes system is a very different architecture designed to serve the University's email needs for the next 10 years. The rest of this paper examines the drawbacks of the old Hermes system, and explains how they are addressed by the new system. We'll describe the modifications we made to Cyrus to get the features we wanted, and to smooth the transition from old to new.

2. Inside the old Hermes system

Until last year[3], Hermes was a 2x scaled-up version of CUS with software tuned for email rather than general-purpose Unix timesharing, so that it was capable of supporting 10x the number of users.

The data storage is provided by a pair of NetApp[5] F740 file servers, each with 14 17GB SCSI disks. This provides a total of 280GB of storage after allowing for RAID and filesystem overheads. The four front-end servers comprise two Sun[31] Enterprise 450s (4 x UltraSPARC-II 296MHz / 99 MHz) and two E220Rs (2 x UltraSPARC-II 450MHz / 113 MHz), all four with 2GB RAM each.

The front-end machines are connected to the file servers with trunked 100Mb/s Ethernet, each Sun having two point-to-point connections to each NetApp. There is also a Sun Ultra 1 which serves as the cluster's admin machine, and an admin network which is used for private connectivity between the admin box, the file servers, the front end Hermes machines, and also `ppsw`.

The four front-end machines have roughly equivalent performance, so they have the same software configuration and each take a quarter of the load via round-robin DNS. Together with the file servers they effectively behave as one large Unix system. This system runs a strictly limited collection of software focussed on email.

[2] PP is not an abbreviation, and is not short for ''Postman Pat'' or ''Perfectly Painful''.

[3] Since then, the CUS file server has been upgraded and Hermes has gone off in an entirely different direction.

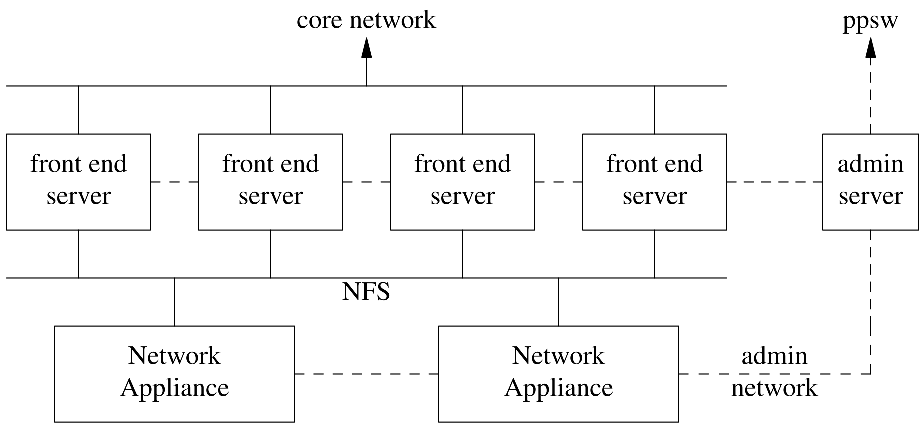

Fig. 3: Hardware architecture of the old Hermes system

Incoming email from the Internet and from remote users is received by Exim[26] and delivered to users' home directories on the NetApps. Keeping inboxes in home directories makes it easier to enforce quotas than the traditional location of `/var/mail/`. The queues of incoming and outgoing email are kept on local disk in the front-end machines; this includes email for users that are over their quotas.

Support for reading email is provided by the University of Washington's IMAP toolkit[32] for remote users, and their Pine user agent for network terminal users. These all access the message store via the `c-client` library. When terminal users log in over the network they are presented with a Menu System which allows them to run Pine and perform various administrative functions. These include some basic file management operations, a simple user-interface for configuring their email filtering options, and interfaces for managing the mailing lists and virtual domains that run on `ppsw`.

The admin machine does most of the behind-the-scenes work. It maintains the master configuration for the front-end servers and the `ppsw` machines, and manages network installs of new machines. Each night it takes tape backups of the NetApps, the Exim spools on the front-end servers and `ppsw`, and the master configuration data on the admin box. Each morning the list of users is obtained from the CS's central database and any necessary account creations and cancellations are performed. Account information is distributed to the front-end machines using NIS. Mailing list and virtual domain configurations are periodically distributed to `ppsw`.

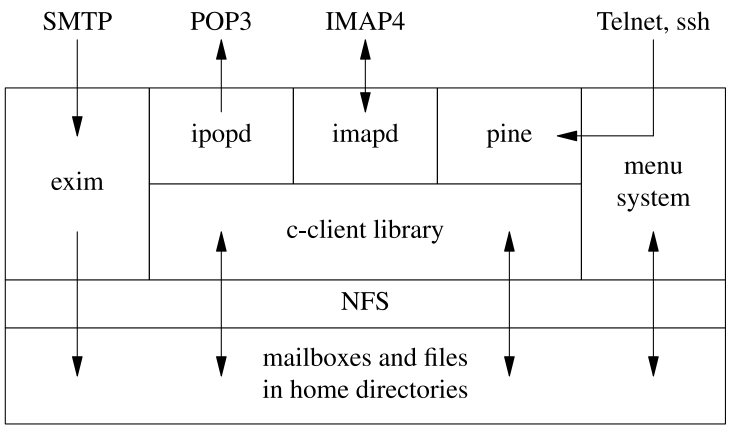

Fig. 4: Software architecture of the old Hermes system

2.1. Better performance

Strict quotas. The principal way that we maintain adequate performance on the old Hermes system is by enforcing strict quotas. The default is 10MB for each user, and we are willing to increase that to 50MB. In addition to that, each mailbox is limited to 10MB regardless of the user's total quota. There is a global limit of 4MB on the size of an individual message. This limits the load that is caused by parsing mailboxes and re-writing them when messages are expunged.

Indexed mailboxes. We have modified[4] c−client to augment the standard Unix mailbox format with a short index containing the IMAP FETCH FAST information, and a long index containing the information that Pine requests when listing a mailbox. This means these two very common cases no longer involve reading and parsing the whole mailbox. The indexes are created when the folder grows beyond 64K and they reduce the amount of data that needs to be read when Pine lists a mailbox by about a factor of 10. They are robust against problems such as write failures caused by quota limits or mailboxes unexpectedly changing because of access that bypasses c−client (e.g. file transfers). In these cases the code falls back to the standard Unix mailbox driver.

Avoid the automounter. CUS uses the automounter heavily for home directories and locally-managed software[33]. In its early days, Hermes did the same but this turned out to be a mistake. A user's home directory had to be mounted each time a message was received, and the effort of doing so placed considerable load on the system. In order to avoid this expense, all references to /home/<*userid*> in configuration files and source code were replaced with password file lookups to find home directories. This made it possible to turn off the automounter.

Centralized account management. Another problem was caused by bugs in rpc.yppasswdd which is the daemon that handles changes to the master password database from over the network. Hermes has a very high password changing load at the start of the academic year, when several thousand new users have to change their initial random password, and a large number of other users have their passwords reset because they forgot them over the long vacation. Because of races in the implementation of rpc.yppasswdd it had a habit of trashing the whole password database, which would have a fairly damaging effect on the operation of Hermes. This problem was worked around by moving the part of the Menu System that deals with password changes so that it runs on the admin server. When a user enters the relevant sub-menu they are transparently logged in to the admin box using rsh.

2.2. The Prayer webmail system

In 1999 the CS started a project to create a webmail user interface to Hermes, since the terminal-based Menu System was becoming more and more of a turn-off for our users. A particular problem with webmail systems is the mismatch between a login session and HTTP's relatively disconnected stateless requests. Most of them avoid the issue by assuming that the back-end IMAP server has enough capacity to cope with an IMAP login for each HTTP request. The less well-written ones may even do multiple logins per request. This would cause an unacceptably high increase to the load on Hermes, which did not have enough spare capacity.

At about the same time, Malcolm Beattie, formerly of Oxford University Computing Services, released a Webmail package named WING[34][5] based on Apache mod_perl[35]. The particularly interesting features from our point of view were that it is designed for use with UW-IMAP and it maintains persistent IMAP connections throughout a user's login session. This means it doesn't excessively stress the IMAP server. However, we came to the conclusion that the WING code that was available then was not sufficiently mature to use for our own Webmail system. Consequently (and with some reluctance) we decided to start writing our own package.

Prayer[36][6] is written entirely in C. It uses the UW-IMAP c−client library to talk to the back-end servers, and its front end is a custom HTTP server. It is extremely efficient, easily supporting our full webmail load (4,000,000 HTTP requests and 56,000 logins per day) on one PC[7]. Most of the pages it serves are dynamically generated and encrypted. If necessary it can easily be scaled horizontally for even higher loads. Prayer gains a lot from very aggressive cacheing, using persistent HTTP and IMAP connections, TLS session caches, and folder and directory index caches. As well as having a light-weight implementation, its user interface is also very frugal: it does not use frames or JavaScript and does not require cookies. Most of the pages it serves are also small—less than 1KB compressed.

The webmail system is not particularly closely integrated with the rest of Hermes: from that point of view it is just another IMAP client. However it does provide a replacement for some parts of the old Menu System, which relies on a support daemon running on the Hermes admin box. Its functionality is focussed on the message store so misses out the mailing list and virtual domain management features, which we expect to have their own separate web interfaces in due course.

---

[4] These modifications have a number of Cambridge-specific hacks, so they have not been published.

[5] "Web IMAP/NNTP Gateway"

[6] Prayer was the in house development name for this package, a simple play on words to acknowledge the strong influence that WING had over Prayer development, especially in the early days. But at the end of the day, everything needs a name.

[7] Some of the most amusing feedback we have received was from a user who successfully ran Prayer on his Apple Macintosh SE30 running NetBSD. The machine was too small to cope with other webmail software.

All is not sweetness and light, though. Prayer has a number of awkward limitations that will require significant work to address. It is not internationalized, and can only use the ISO 8859-1 character set. It has only a very simple user preferences system, and doesn't have the kind of templating mechanism that would allow extensive overhaul of the user interface. It makes the opposite error than is warned against by the fourth commandment of IMAP clients[37]: it assumes a UW-IMAP-style folder hierarchy that distinguishes mailboxes and directories, and is confused by a Cyrus-style hierarchy in which email folders may have sub-folders.

2.3. Limitations of old Hermes

The major limit on performance comes from the use of traditional Unix mailboxes, however this has been mitigated by the addition of indexes and the use of an efficient webmail system. The system can be scaled up by the addition of more and bigger file servers and front-end machines, however it quickly becomes difficult to do so within our budget.

In addition to this, each file server is a single point of failure: the whole of Hermes must be shut down if either of them fails. More file servers would mean more things to go wrong. It is possible to avoid this problem by replicating data between NetApps, but this does nothing to help with cost-effectiveness.

The old system also has some feature limitations. Concurrent access to mailboxes is not supported, which makes shared accounts less useful than they might otherwise be, and (combined with the limitations of Unix filesystem permissions) it makes shared mailboxes impossible. Checking for new email cannot be done with IMAP because it would disrupt any existing session, so we have to support various hacks based on the traditional filesystem-based Unix new email check.

3. Architecture of the new Hermes system

The most visible feature of the new Hermes system from the point of view of someone standing in our machine room is that is uses a large number of commodity PCs running Open Source software[38]. The most visible feature from the point of view of our users is that the standard quota is 100MB, and we are willing to increase it to 1GB.

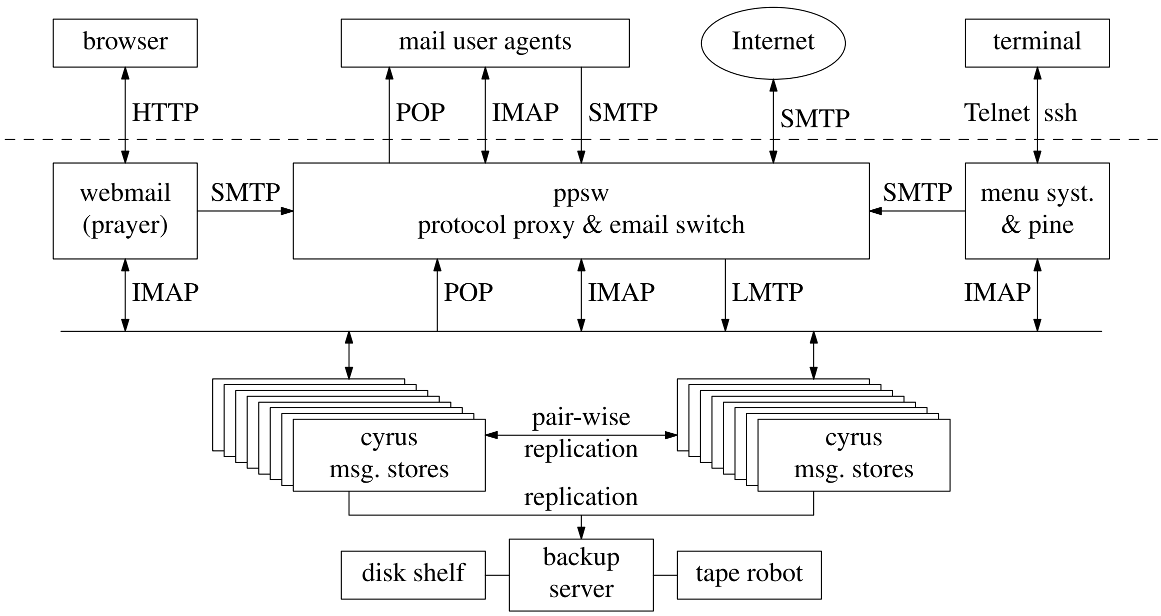

Fig. 5: Architecture of the new Hermes system.

The system's back end Cyrus message store comprises 16 single-CPU PCs, each with 3GB RAM and 7 72GB SCSI disks providing 350GB of available storage after RAID and filesystem overheads. The front end redirector `ppsw` is 6 dual-CPU PCs each with 1GB RAM and 4 18GB SCSI disks providing 50GB of available storage. The webmail system is another PC with the same configuration. The terminal-based Menu System is still running on the old Hermes systems at the moment, until migration to the new system is complete. It will then move onto a PC with plenty of RAM and CPU. The backup system is attached to a disk shelf containing 16 250GB SATA disks providing 3.2TB of usable storage, and a tape robot containing a 17 cartridge magazine and two LTO drives.

All the machines are running Linux of various kinds[8]. The `ppsw` machines use the relatively conservative `ext3` filesystem, the main requirement being that they can be dumped for backups. The Cyrus machines use ReiserFS because of its good large directory and small file performance which Cyrus makes good use of; dump support is not necessary here because we use our own data replication system instead. The backup server uses XFS because of its large directory and large file performance and support for dumping; the replicas of the Cyrus machines are dumped to disk then spooled to tape—the machine can't keep up with the tape when dumping directly.

The machines are connected to three[9] gigabit Ethernet networks by 100MB/s network interfaces. The public interfaces to the system (marked by the dotted line in fig. 5 above) are connected to the CS's core service network. Between the `ppsw` systems and the Cyrus systems is a private network used for proxied traffic. Finally, there is a network between the Cyrus servers and the backup server for replication traffic.

3.1. The Cyrus message store

The new Hermes storage infrastructure presents a much higher-level interface to the rest of the system. This allows it to use much more efficient data structures internally, and support more flexible usage patterns. This means that standard Unix tools are less useful for managing the message store; however Cyrus comes with a rich set of tools of its own.

Black-box server. Access to the message store is via email-specific protocols: IMAP[39], LMTP[40], and POP[41]. The store is managed with Perl scripts that speak an extended version of IMAP to the server. The server implements its own user management, quota restrictions, and access control—it does not rely on Unix for these features, and in fact the whole thing runs as a single Unix user. This makes things like shared folders and the single-instance store possible, and allows it to use Apache-style pre-forking to reduce the cost of accepting new connections. The only people with Unix logins on the machines are the system administrators.

Indexed file-per-message store. Cyrus stores each folder in a directory with one file per message, stored in wire format (i.e. CRLF line endings instead of Unix-style newlines). In addition to that the directory includes some index files: `cyrus.header` contains some brief folder metadata including its access control list and unique ID; `cyrus.index` contains the IMAP `FAST` information similar to the short index on old Hermes; `cyrus.cache` contains pre-rendered IMAP `ENVELOPE` and `BODYSTRUCTURE` data, and various message headers; and `cyrus.squat` contains an index for speeding up the IMAP `SEARCH TEXT` command. The `cyrus.cache` index contains more information than the old Hermes long indexes, in particular the `BODYSTRUCTURE` cache makes extracting individual attachments much more efficient. Like the message files, it is in wire format. The `cyrus.squat` index makes it possible to do a free text search at the rate of 200,000 messages per second[10].

Ancillary databases. Cyrus has a duplicate suppression system which makes it particularly good with mailing lists: only one copy of each message is stored with multiple hard links from each folder in which it appears. Each user also has a folder subscription list, and a list of the messages they have seen in each folder (for shared folder support).

Sieve filtering. Users are able to configure the disposition of the messages they receive using the standard Sieve email filtering language[42]. This is simplified and restricted so that it can be used inside black-box servers, unlike the Exim filtering language[43] which generally assumes it has lots of Unix facilities at its disposal.

Concurrent access. Cyrus fully supports concurrent access by more than one IMAP client to the same mailbox. This should reduce one of the old sources of confusion for our users. It also means that shared accounts on Hermes (used for role addresses etc.) are more useful than they were. We can use this feature to check for new email in a very efficient way: make an IMAP connection, `SELECT` the mailbox of interest, then use the `IDLE` command[44] and wait for notifications to come in; this does not require polling by either the client or the server. In the future this feature we allow us to support shared mailboxes properly.

Partitioned store. The message store is scaled up by having several separate machines running Cyrus. As on old Hermes, users are distributed between the various back-end stores in an ad-hoc manner intended to balance the load. On the old system a unified view of the store is created using NFS filesystem mounts on the

[8] Our distribution of choice has recently changed from Red Hat 9 to SUSE 9.

[9] Actually there are physically only two networks, so some of the logically distinct traffic is actually sharing the same wires.

[10] This is a good number to mention to users who are unenthusiastic about black-box IMAP message stores because they like being able to `grep` their email archive.

front-end machines. On the new system the same job is done by higher-level functionality on `ppsw`. There is a user location table which specifies which Cyrus machine a user is on so that connections can be routed appropriately; if a user is not in this table they are assumed to still be on the old Hermes system. (This latter function will obviously go away in due course.)

3.2. The central email switch, `ppsw`

Hermes and `ppsw` have until recently been run as separate but related systems. They share administrative infrastructure and Hermes provides the user interface for many of `ppsw`'s functions, but Hermes did its own handling of incoming and outgoing email. This has changed for two reasons: the spam and virus scanning facility on `ppsw` means that all email to and from our users should pass through it, and `ppsw` is now running the IMAP and POP redirecting proxy.

Central email router. The long-standing job of `ppsw` has been to handle the email for about 150 virtual domains (which are effectively just a list of email address aliases), 70 hubbed departmental servers (which send and receive email via `ppsw` rather than directly to and from the Internet) and over 3500 mailing lists. More recently it has taken on the task of filtering out viruses from all the email passing through and scoring email coming in from the Internet for spamminess.

Smart host for outgoing email. User agents require an SMTP server to handle outgoing email, routing it and queueing it as necessary. This job is moving from the old Hermes systems to `ppsw`. In order to support roaming users, `ppsw` also needs a copy of the Hermes password database so that it can authorize users to use it for relaying wherever they are on the Internet.

Incoming queues for the Cyrus store. Delivery of email to the Cyrus message store is over LMTP, which is a variant of SMTP that allows the receiving system to state in more detail to which recipients the message has been successfully delivered. This means that Cyrus doesn't have to maintain a queue since a message can be held on `ppsw` for users that are over their quotas without affecting the other recipients. The user location table is used by Exim to route messages to the appropriate Cyrus machine or to old Hermes as necessary.

Redirecting IMAP and POP proxy. We have developed our own IMAP and POP proxy which has similar functionality to Perdition[10]. This tracks a connection until a user logs in then uses the user location table to decide which back-end server to connect to. After login the proxy becomes a simple pass-through[11]. It includes a TLS session cache (based on code from Cyrus) to make negotiating encrypted connections more efficient. The connection from the proxy to the back end is always unencrypted, since it is on a private network. Note that the webmail system and Pine on the terminal login system connect directly instead of via the proxy; since they need to look at the user location table anyway to know if the user is on the old or new system this optimisation comes for free.

4. Improving Cyrus's reliability and performance

The standard version of Cyrus cannot by itself satisfy all the requirements we had for the new Hermes system, so we have made extensive improvements as described in this section. These improvements have been contributed back to the Andrew Systems Group at Carnegie-Mellon University and we expect them to be integrated into future versions of the software.

4.1. Two-phase expunge

One of the features of the old system's NetApps is filesystem snapshots, which we use heavily for recovering email that users have accidentally deleted. Although Linux has some support for similar functionality it is less easy to use, and it doesn't fit in with our general approach of replacing filesystem-level cleverness with application-level cleverness. In this case our replacement is to change Cyrus so that it doesn't immediately delete email when the users tell it to.

In theory, user agents are supposed to protect users from deleting messages by accident: IMAP uses a model where users mark messages for deletion then later expunge them, after explicit confirmation from the user. This also reduces the work that the server must do by batching up delete operations. Unfortunately:

- Many user agents hide this from the end user and expunge automatically.
- There is no similar protection against people deleting entire mail folders by accident.
- Sometimes people accidentally download their email to a random workstation using an unfamiliar POP client.

[11] This means that fully general shared folders cannot be supported since users on different back-ends cannot see each others' data.

Other IMAP user agents and Webmail clients follow the desktop metaphor and implement a Trash folder, though this isn't a natural IMAP concept, and is slightly awkward to implement.

In our modified Cyrus, when a user expunges a folder the deleted messages are "moved" to a shadow folder rather than being removed from disk. The message files themselves aren't actually moved: the shadow folder is implemented as a second set of index files in the same directory as the un-expunged version of the folder. Similarly, when a user asks for a folder to be deleted it is instead moved into a reserved part of the namespace with a timestamp appended to the name in order to avoid clashes. Overnight, or on demand when a user's expunged data gets too voluminous, an expire job (similar to that on an NNTP server) is run to remove old expunged data from the disk.

The expunged data lives in a couple of directories that are usually hidden to avoid confusing the users, and which have restricted ACLs in order to avoid abuse. Messages that have been expunged from `folder` can be found in `.EXPUNGED/folder`. When `folder` is deleted it is moved to `.DELETED/folder-20040226-12:34:56`. Messages that had been expunged before `folder` was deleted can be found in `.EXPUNGED/.DELETED/folder-20040226-12:34:56`. As well as allowing users to access their "lost" email via these hidden paths, we have `unexpunge` and `undelete` commands for use when we have to help a user recover it.

This scheme requires that the quota system is extended to keep track and limit the amount of expunged data, in addition to the 100MB default quota for visible email. The new configuration variables include `expunge_vol_min` and `expunge_vol_max`, which provide the normal bounds on the quantity of expunged data, `expunge_vol_overflow`, which is the point at which an emergency expire is run, and `expunge_timeout`, which is the maximum age of expired data. We have these set to 50MB, 75MB, 100MB, and 28 days, respectively.

The two-phase expunge system does of course increase the amount of disk space required. Our experience so far suggests that that overhead is around 30-50% with the above settings. However, modern disks are large, typically too large given the limit on the number of users per disk needed to control the high load associated with email applications. Expunged data occupies space, but doesn't create additional disk I/O. In fact two-phase expunge *improves* performance since `unlink` is an expensive operation, so it is better to do it in bulk during an expire job that runs when the system isn't busy. It also improves concurrent access to folders, since one session can still retrieve messages that have recently been expunged by another, an operation which would normally cause an error. We use this to good advantage in the replication system described below.

It is worth comparing our system with the Berkeley Undoable Email Store[45].

4.2. Data replication

One of the advantages of not using NFS in new Hermes is that the failure of one of the store machines has less of a knock-on effect for the rest of the system. However because we are using lots of commodity hardware we expect this event to be more frequent than before. Our recovery plans are based on an email replication engine, which we use for synchronizing data between pairs of machines that act as hot spares for each other, and for transferring data to the tape backup server. We also use it to move users between Cyrus servers in order to balance the load, and as part of the system for migrating users from old Hermes. The replication system is the largest part of our Cyrus customizations.

4.2.1. Replication overview

Our replication engine is a transaction based system built using the Cyrus `libimap` folder access library. This provides better flexibility and robustness than a replication scheme which transfers data by raw disk block or file without any understanding of the underlying content. It is also more efficient, as it doesn't replicate extraneous traffic generated by filesystem metadata activity. It guarantees that the replica system is in a self consistent state at the end of each transaction. This is true even if the source system suffers from file system corruption and is trying to replicate a corrupt database.

The lack of checksums in index files and other databases at the source end means we aren't 100% safe: it is just conceivable that the system would replicate "delete all folders" as an action, despite various safeguards. In practice sanity checks in `libimap` mean that the replication engine bails out rapidly in the event of corrupt input. Because the replication system is so important to Hermes (the hot spares and tape backups depend on it) we also have a sanity checking system based on databases of MD5 checksums. Each machine's database is updated incrementally and a proportion of old checksums is regenerated each night; the databases are then compared to ensure that everything is working as expected.

The replication system is asynchronous: it does not require that replica systems exactly match all the time. Synchronous replication, where all accesses to the message store are replicated before the user is notified that they have completed, is safer but would reduce the perceived performance of the system. It is also harder to retro-fit into existing software. Asynchronous replication reproduces events after the fact, so it is able to gain performance by merging transactions to create a smaller set sufficient to bring a replica in line with the master copy. The replication system can therefore be allowed to accumulate a backlog of updates (e.g. because it has not been running) and it is able to catch up later very efficiently. As a concrete example, a pair of our back-end Cyrus servers take about a minute to resynchronize after several hours' activity with replication turned off.

The protocol used for replication is a simple line-based text protocol similar in style to IMAP or POP. The reason that we cannot use IMAP itself is that we need to preserve a number of message and folder attributes, such as timestamps, message unique IDs, and folder unique ID validity information. The protocol attempts to minimise the number of round trips between client and server to improve efficiency and throughput. It is designed to be robust: it can recover from errors and inconsistencies which are caused if the master and replica lose synchronisation because someone has made unexpected changes at the replica end. It relies on cluster wide message globally unique identifiers (UUIDs) in order to resolve conflicts and maintain the single instance store when copying messages between systems. It is possible to run several instances of the replication system concurrently, which is particularly useful for checking the integrity of a replica system while rolling replication is also in operation.

The replication engine depends to some extent on the two-phase expunge system described in the previous section. This allows it to create snapshots of individual mail folders for the few seconds that it typically takes to synchronise a single folder: by locking out the asynchronous expire job we guarantee that no messages will disappear from the master end while we are updating a specific mailbox, which makes error recovery easier. The replication engine also has the ability to lock out all updates to the folder list for a specific user for a few seconds so that we have a guaranteed consistent set of mail folders to work with if any ambiguity arises there.

The server end of the replication system, `sync_server`, runs on the replica system and just follows instructions from the client, reporting errors and problems as needed. The client end, `sync_client`, does all the hard work: it asks the server for the current status of all the objects of interest in a given account, then works out a set of transactions that will transform the server into a clone of the client system, and finally executes them with whatever error recovery is necessary. A wrapper script starts the client and server programs with an `ssh` link over which they communicate.

4.2.2. Replication protocol

The replication protocol is based on a set of events and actions which are recorded by the Cyrus IMAP, POP, and LMTP servers during their normal operation. This list of actions is picked up by `sync_client` which runs through the queue asynchronously and pushes changes to `sync_server`, resolving any conflicts in the process. The actions which are currently defined and the consequence of these events are as follows:

- **`APPEND`** Upload all messages in a mailbox that exist on the client and have UIDs greater than the last UID at the server end. In case of problems, we fall back to `MAILBOX`.
- **`SEEN`** Update the user's database of which messages in the given folder have been seen. In case of problems, we fall back to `MAILBOX`.
- **`MAILBOX`** Perform a full update of a mailbox: upload missing messages, resolve UUID conflicts, expunge messages on server which have been expunged on client, update flags and ACLs as required. The replication engine uses message UUID values to track messages which have already been uploaded to the server and which can be moved or linked rather than uploaded again, improving efficiency and maintaining the single instance store.

  `MAILBOX` actions are also used to replicate IMAP `CREATE`, `RENAME` and `DELETE` commands. The IMAP server records the names of all the mailboxes involved, and the replication engine works out the minimum set of transactions required to synchronise the server by tracking the Cyrus unique ID values of mailboxes. Therefore we don't normally have to upload messages a second time.

  In case of problems, we fall back to `USER`.
- **`META`** Update a user's account meta-information: subscriptions, sieve filters, and quota limits. In case of problems, we fall back to `USER`.
- **`USER`** Synchronise everything: all mailboxes and account meta information. If the given account doesn't exist on the replica system it is created. In practice this only happens if the list of mailboxes changes under our feet.

If the first attempt at USER synchronisation fails, we lock the user out of the folder list for a few seconds and have another go: this deals with most cases where the user is renaming or deleting large numbers of folders at the time.

If the UID validity of a user's inbox on the server doesn't match the UID validity on the client, we are probably looking at an obsolete account. The system can either clear the replica account and replicate everything or bail out and demand intervention from the system administrator (the safer course of action).

The replication engine combines and promotes actions as required: if it saw 6 `APPEND` events, 3 `SEEN` events, and 2 `MAILBOX` events for a single mailbox in a single pass these are converted into a single `MAILBOX` action. Similarly the replication engine groups `MAILBOX` actions to give it a better chance of tracking messages which move between folders.

One amusing case is the way in which accounts are first replicated in the absence of any other instructions. Typically a message arrives, causing an `APPEND` action; this gets promoted to a `MAILBOX` action, which in turn is promoted to a `USER` action, and the entire account gets replicated. When we migrate a user from old Hermes to new Hermes, we send them a message to tell them that this has happened, so they are immediately replicated.

4.3. Performance improvements

As well as the big changes described in the previous two sections, we have made some more modest changes to help Cyrus cope with badly-behaved IMAP and POP clients.

Extended cache contents. A standard Cyrus installation stores only a limited set of message headers in a folder's `cyrus.cache` file. However most IMAP user agents request headers which are not in this set, causing an expensive cache miss which requires a large number of message files to be opened instead, reducing Cyrus to maildir performance. We store much more information in the `cyrus.cache` files, in particular all message headers except for things like `Received:` which are clearly a waste of time. This is a trade-off: we have far fewer cache misses (hopefully none at all in normal use), but the cache files become larger. This increased size would pose a substantial problem if it wasn't for the next modification on this list.

Lazy cache updates. Standard Cyrus rewrites the entire cache file at just about any update apart from a simple append caused by mail delivery. This is something of a problem given the larger caches that we have created, especially when POP keep-mail-on-server mode is taken into account—it reduces Cyrus to Berkeley mailbox performance. Our solution is to observe that there is no need to remove cache entries immediately when an expunge happens. Instead we leave the data in the cache and garbage collect it either overnight or when a certain threshold is reached. A complication is that standard Cyrus does not store the size of individual cache entries in the folder's index file: it just stores the offset to each messages and uses the difference between adjacent offsets to determine the size of a cache entry. Consequently we have extended the `cyrus.index` file to contain an additional entry for the size of the cache entry.

Fast folder renames. Standard Cyrus renames folders in a very conservative manner. The contents of the folder are copied to the target location one message at a time, the when this step has completed, the messages in the source folder are removed. This cautious approach is particularly useful when the source and target folders are on different partitions. However it assumes that folder rename operations are fairly rare events. Most of the time this is true; however on the first of each month we have thousands of Pine and Mulberry users who have the monthly folder rotation feature switched on. We have replaced this copy and then delete approach with a simple rename system call when it seems appropriate. This isn't quite as safe, but experimentation suggests that the Cyrus `reconstruct` utility can be used to deal with most of the obvious disasters which can take place as a consequence of a machine losing power when the folder list and filesystems are out of step with each other.

5. Smoothing the change from old to new

The first part of the new Hermes system that was deployed was the IMAP and POP proxy on `ppsw`. This was partly for testing purposes but also because it is an important part of our migration plan. As we described above, the proxy knows which users are on old and new Hermes and directs their connections accordingly. However there are significant differences between UW-IMAP and Cyrus which mean that migration would not be as transparent for our users as we would like. Therefore we have made some further changes to Cyrus to make it behave in a more familiar manner.

5.1. Cyrus compatibility

The namespace used by Cyrus is similar to Usenet's, which is very different from the Unix namespace used by UW-IMAP. It uses a "." separator instead of "/", and allows a folder to contain other sub-folders as well as just messages—i.e. they are "dual-use" mailboxes and directories. Cyrus keeps a user's folders in a hierarchy under the root `user.fanf2`, e.g. `user.fanf2.sent-mail` and `user.fanf2.saved-messages`, and it treats `INBOX` as an alias for `user.fanf2` so `INBOX.saved-messages` is an alias for `user.fanf2.saved-messages`.

The standard version of Cyrus has a couple of options that help to make it look more familiar from our users' point of view. The `unixhierarchysep` option changes the pathname separator to "/", and permits "." to be used within names like any other character (though it is translated to "^" behind the scenes). The `altnamespace` option makes a user's subsidiary folders appear at the top of the hierarchy, alongside their `INBOX` instead of below it.

We have extended this with with a `unixnamespace` option, which adds some aliases for the root of a user's folder hierarchy for compatibility with configurations that they typically use on the old Hermes systems. These aliases include `~/`, `~fanf2/`, `~fanf2/mail/`, `~/mail/`, and `mail/`.

We also have a large number of user agents that cope badly with dual-use folders. These include older versions of Pine and our own webmail software, Prayer. In order to reduce the problems that this would cause, we have disabled dual-use folders by forcing mailboxes to be leaves in the hierarchy, and hacking in support for temporary directory nodes. Cyrus doesn't really understand directories, so this patch is rather evil and we hope to be able to get rid of it in the future.

We have a few other patches to improve interoperability. We modified Cyrus's POP daemon to behave more like the UW daemon, so that keep-mail-on-server users don't get a fresh copy of all of their email every time they connect[12]. We have arranged for concurrent IMAP connections to the same folder to keep their view of the folder more closely in sync, at a slight performance cost. This is because Outlook Express opens two connections and gets confused when they disagree; however it copes fine when one of them is forced into read-only mode as occurs on old Hermes.

5.2. Migrating users

The system for migrating users is based on our replication software. A program called `sync_upload` uses `c-client` on the old Hermes system to read users' email and transfer it to a Cyrus machine via `sync_server`. Users are sent an email when they are added to the migration schedule, and another when they have been migrated. A number of things may prevent a user from being migrated:

- Their user agent is mis-configured so that it speaks to `hermes.cam.ac.uk` (which refers to the old Hermes front-end machines) rather than `imap.hermes.cam.ac.uk` or `pop.hermes.cam.ac.uk` (which refer to the proxy service on `ppsw`).
- They have written their own Exim filter, which cannot be automatically translated to Sieve.
- They are logged in when we run the migration script.

In the first two cases, our support staff must contact the user and help them to reconfigure as necessary. (Users who have set up filtering using webmail or the Menu System are not a problem, because our friendly user interfaces use a simplified representation of the filter which can be translated into the Exim filter language or into Sieve as required.) In the latter case we will try to migrate the user again at a later date.

There are very few obvious differences to the user interface after migration:

- The options for configuring email filtering are no longer present in the Menu System, since we have not added Sieve support to it. Users must configure it via webmail. An "advanced" filtering option appears on the webmail system allowing users to write their own Sieve scripts.
- A different version of Pine is provided for terminal users. As well as being more recent, it is configured to connect to the appropriate Cyrus server rather than access email on the local filesystem. This means that users must type in their password an extra time when Pine starts.

6. Plans for the future

There remain a few weaknesses with the new Hermes system as it is currently implemented. They include: the poor data-recovery user interface; the lack of support for shared mailboxes; some remaining situations where data is not replicated; the dependence on a single machine room; and the lack of high-availability

[12] Even if that would serve them right for abusing the protocol.

features.

6.1. A user interface for undelete and unexpunge

One of the more frequent requests we receive (several times a week) that require administrator privilege is to recover a user's accidentally-deleted email. On both the old and new systems users are able to use special paths to access the copies of their data that we retain (`~/.snapshot/*/inbox` on old Hermes, `.EXPUNGED/INBOX` on the Cyrus message store). On old Hermes the Menu System makes it especially difficult for users to access the snapshots; it's much easier to access deleted email on new Hermes using the webmail system, however the maze of similarly-named folders can be confusing and is too intricate for the majority of our users.

The `unexpunge` and `undelete` commands have made it much quicker for the system administrators to deal with data recovery requests, since we no longer have to dig through the various snapshots to find the one before the user fat-fingered the delete command. However, if a user interface to the `unexpunge` and `undelete` operations is added to the webmail system, or a friendlier way of browsing the `.EXPUNGED` or `.DELETED` hierarchies, these requests would be greatly reduced in number, and those that remain could be dealt with by the Help Desk.

6.2. Shared mailboxes

At the moment the IMAP proxy we are using requires that all of the mailboxes which a user may access are on the same back-end message store machine. The replication engine also has built in assumptions in the implementation of its `SEEN` action which limit the mailboxes that user can see to their home directory. This means that although Cyrus supports shared mailboxes, we cannot use them on the new Hermes system. This is a shame, because they would be useful for a number of purposes:

- A Professor could delegate responsibility for dealing with email to their personal assistant.
- Mailing lists can be delivered to shared mailboxes, which can be used as list archives. Users could access an archive mailbox instead of subscribing to the list, which would reduce the resources needed for list distribution.
- We could enable Cyrus's NNTP support and use it to merge the CS's email and Usenet services.
- At the moment we use shared accounts on Hermes to provide some of the functionality that would ideally be provided by mailing lists and shared mailboxes. Before these accounts are phased out, we can improve our audit trails by requiring users to log in as themselves and access the shared account as a set of shared mailboxes, rather than logging in to the shared account directly which hides their identity.

The Cyrus Murder[11][13] uses a more intelligent proxy which instead of mapping accounts to back-end servers maps mailboxes to back-end servers. The proxy is able to transparently re-connect to a different back-end server as necessary when the user `SELECT`s a different mailbox. However this system has the big disadvantage that it introduces a single point of failure in the form of the MUPDATE[46] server, which keeps track of the location of all the mailboxes on all the back-end servers. It may be possible to work around this problem by mapping parts of the mailbox namespace to different back-end servers, with the restriction that mailboxes cannot be renamed out of their partition. This would make the mapping used by the proxies sufficiently static that they would not need to be informed of activity on the back-end servers in real time.

6.3. Re-examining the terminal login service

Although the Menu System interface to Hermes remains in the new system in a smaller form, it is only going to become less important. For example, we expect that its role as the user interface for mailing lists and managed mail domains will be taken over by a web service. This would leave it providing only a place to run Pine and a nugatory file store for temporary storage of attachments, which seems to be a pointless duplication of effort given that one of the CS's other central services (the Public Workstation Facility) is a large general-purpose file store which can support a remote-access Unix service. The main disadvantage making users use the PWF is that they would have to authenticate first with their PWF password then with their Hermes password when they run Pine. However this concern may go away with the general moves towards a unified single-password authentication scheme, and it may be outweighed by the reduced need to copy attachments between different systems.

[13] Named that way "because it sounded cool for crows"

6.4. Data replication on `ppsw`

Although there is a very high degree of data redundancy in the message store, this does not come into effect until after a message has been delivered. Between reception of the message and the replication engine picking it up, we depend on the RAID system in the `ppsw` machine or the Cyrus machine (depending on whether the message has been delivered or not) for reliable storage. Although this can cope with a single disk failure, more serious failures[47] may cause data loss.

Loss of a `ppsw` machine will affect queued email, especially delayed deliveries to to unavailable departmental mail servers and external sites, and to users who have exceeded their quota. Loss of a Cyrus machine will affect messages that have been delivered or uploaded via IMAP `APPEND` but not yet replicated.

The impact of such a disaster would be reduced if `ppsw` had a data replication system. As an alternative to using more advanced storage technology, it is worth also considering what application-level techniques can be used. When a message is received, as well as storing it on the local queue the MTA could replicate it to a backup system before confirming to the sending system that the message has been received successfully. The replica message can be kept until the primary copy has either been delivered to a Cyrus system and replicated there, or delivered to an external email system. The IMAP proxy also needs to keep a replica of uploaded messages in case a Cyrus machine fails. The difficulty with this idea is that it adds complexity and failure modes to an important point in the process of email delivery, so a great deal of care is required to implement it.

6.5. Wide-area distribution and high availability

Highly redundant data storage in a system with no single points of failure is all very nice, but no good at all if the system has some external dependency that is a single point of failure. At the moment we only have one machine room, and a lot of our system failures in the past have been due to failure of its power supply[48]; fire is also a concern[47]. If we ever get a second machine room, the new Hermes system is well placed to benefit from it. We can install half of the machines in one location and half in the other, and do the pairwise replication between the two sites.

Human intervention is still required in the event of a failure of any component, in order to reconfigure the rest of the system so that it no longer depends on the failed part. A particular problem is when one of the front-end systems fails, because we use the DNS to distribute the load between them. Some clients will cache a particular IP address rather than cycling between the addresses in the list, and so are very bad at coping when the machine they have fixated on is no longer available. In addition to that, the larger number of cheaper machines that we are now running means that the mean time between failure of any part of Hermes is now much lower than it was. Deploying high-availability technology to automate failure recovery may make system management a significantly less fraught business, however we are concerned about the additional complexity and failure modes introduced by things like STONITH[14].

7. Conclusion

Returning to the introduction of this paper to review the list of goals for the project, we can now see how they have been met.

- *Larger quotas and message size limits.* The standard quota on the old Hermes system is 10MB, and we are willing to increase that to 50MB. There is a maximum folder size of 10MB, and a maximum message size of 4MB. The new system has a standard quota of 100MB, which may be increased up to 1GB. The folder size limit is 200MB and the message size limit is 10MB, and both of these can be increased with much less damage to performance than before.
- *Better data recovery options.* The old Hermes system uses filesystem snapshots to periodically make copies of the data that can easily be recovered in case a user accidentally deletes their email. However, snapshots are not useful for recovering data that is only stored on the server for a short period of time before being deleted (which is especially common for POP users). The new Hermes system does not delete anything when asked to by a user; instead it is kept on disk for easy recovery until an expiry process removes it up to a month later.
- *Better disaster recovery options.* If one of the file servers in the old Hermes system fails, the whole system is unavailable until it is fixed. The new system has pairs of back-end message store systems that replicate their data to each other. If one fails, its pair can take over with minimal fuss. For more serious problems, old Hermes has off-site tape backup. The new system improves on this with the ability to be distributed between multiple sites in the future, so that service can continue in the event of power loss or

[14] "Shoot The Other Node In The Head" e.g. using a network-attached power switch.

fire.

The new Hermes system is still in the process of being rolled out, so it is difficult to make a complete assessment of the project. So far the data replication and integrity checking software has worked well, and it has proved to be straight-forward to fix problems that have been caused by bugs.

Hermes is not a monopoly: departments of the University are free to run their own email services, and users are free to use off-site email services. However, Hermes continues to gain users as it provides new features such as webmail and spam filtering and now bigger quotas, which we think is a fair indication of success.

References

1. Fibre Channel Industry Association, *Executive Summary.* `http://www.fibrechannel.org/OVERVIEW/`
2. Kenneth W. Preslan et al., “A 64-bit, Shared Disk File System for Linux,” *Proceedings of the Sixteenth IEEE Mass Storage Systems Symposium* (1999). `http://storageconference.org/1999/1999/papers/03presla.pdf`
3. Sistina, *GFS datasheet.* `http://www.sistina.com/downloads/datasheets/GFS_datasheet.pdf`
4. *VERITAS SANPoint Foundation Suite: New VERITAS Technology for Cluster Environments* (Sep 2001). `http://eval.veritas.com/downloads/pro/spfs_new_tech_wp.pdf`
5. *Network Appliance Inc.* `http://www.netapp.com/`
6. Dave Hitz, James Lau, and Michael Malcolm, “File System Design for an NFS File Server Appliance,” TR3002, Network Appliance, Inc. (Mar 1995). `http://www.netapp.com/tech_library/3002.html`
7. *Akamai Technologies, Inc.* `http://www.akamai.com/`
8. *Apache HTTP server version 2.0: mod_proxy.* `http://httpd.apache.org/docs-2.0/mod/mod_proxy.html`
9. *F5 Networks BIG-IP.* `http://www.f5labs.com/f5products/bigip/`
10. Simon Horman, *Perdition: Mail Retrieval Proxy* (Jan 2003). `http://www.vergenet.net/linux/perdition/perdition_paper/`
11. Carnegie-Mellon University Andrew Systems Group, *Cyrus IMAP Aggregator.* `http://asg.web.cmu.edu/cyrus/ag.html`
12. Malcolm Beattie, “The Design and Implementation of a Large Scalable Mail Server,” *UKUUG Linux ’99 Conference.* `http://www.clueful.co.uk/mbeattie/herald-ukuug99.ps`
13. Mike Gahrns, “IMAP4 Mailbox Referrals,” RFC 2193 (Sep 1997). `http://www.ietf.org/rfc/rfc2193.txt`
14. Mark Crispin, *Mailbox Format Characteristics* (5 Jun 1999). `http://www.washington.edu/imap/documentation/formats.txt.html`
15. SCO, *Mail and Messaging Guide.* `http://docsrv.sco.com:507/en/MailMsgG/CONTENTS.html`
16. Mark Crispin, “MBX Format Description,” *IMAP mailing list* (19 May 2000). `http://www.washington.edu/imap/listarch/2000/msg00363.html`
17. D. J. Bernstein, *Using maildir format.* `http://cr.yp.to/proto/maildir.html`
18. Namesys, *ReiserFS: Journalling file system for Linux based on balanced tree algorithms.* `http://www.namesys.com/`
19. Silicon Graphics, Inc., *XFS: A high-performance journaling filesystem.* `http://oss.sgi.com/projects/xfs/`
20. *Courier Mail Server.* `http://www.courier-mta.org/`
21. Sam Varshavchik, *Benchmarking mbox versus maildir* (25 Mar 2003). `http://www.courier-mta.org/mbox-vs-maildir/`
22. University of California, Irvine, *The RAND MH Message Handling System.* `http://www.ics.uci.edu/~mh/`

23. Nicholas Elprin and Bryan Parno, “An Analysis of Database-Driven Mail Servers,” TR-13-02 (2002). `ftp://ftp.deas.harvard.edu/techreports/tr-13-02.ps.gz`

24. “Closure of Phoenix,” *University of Cambridge Computing Service Newsletter,* 183 (Oct 1995). `http://www.cam.ac.uk/cs/newsletter/1995/nl183/phoenix.html`

25. “Central Unix Service,” *University of Cambridge Computing Service Newsletter,* 151 (Jan/Feb 1990). `http://www.cam.ac.uk/cs/newsletter/1990/nl151.html`

26. *The Exim Home Page.* `http://www.exim.org/`

27. *ISO Development Environment.* `ftp://ftp.uu.net/networking/osi/isode/`

28. Greg A. Woods, *The Smail Project.* `http://www.weird.com/~woods/projects/smail.html`

29. “Mail to and from Phoenix,” *University of Cambridge Computing Service Newsletter,* 164 (Mar/Apr 1992). `http://www.cam.ac.uk/cs/newsletter/1992/nl164.html#19`

30. “The Hermes Message Store,” *University of Cambridge Computing Service Newsletter,* 172 (Oct 1993). `http://www.cam.ac.uk/cs/newsletter/1993/nl172.html#31`

31. *Sun Microsystems.* `http://www.sun.com/`

32. University of Washington, *IMAP Information Center.* `http://www.washington.edu/imap/`

33. Martien F. van Steenbergen, *File System Organization - The art of automounting* (1 Aug 1991). `ftp://blank.org/pub/misc/art_of_automounting.ps`

34. Malcolm Beattie, *WING, the Web IMAP/NNTP Gateway,* Oxford (1999). `http://www.clueful.co.uk/mbeattie/wing/`

35. Apache Software Foundation, *mod_perl.* `http://perl.apache.org/`

36. David Carter, *The Prayer Webmail System,* Cambridge. `http://www-uxsup.csx.cam.ac.uk/~dpc22/prayer/`

37. Mark Crispin, *Ten Commandments of How to Write an IMAP client.* `http://www.washington.edu/imap/documentation/commndmt.txt.html`

38. Open Source Initiative, *The Open Source Definition.* `http://www.opensource.org/docs/definition.php`

39. Mark R. Crispin, “Internet Message Access Protocol - Version 4rev1,” RFC 3501 (Mar 2003). `http://www.ietf.org/rfc/rfc3501.txt`

40. John G. Myers, “Local Mail Transfer Protocol,” RFC 2033 (Oct 1996). `http://www.ietf.org/rfc/rfc2033.txt`

41. John G. Myers and Marshall T. Rose, “Post Office Protocol - Version 3,” RFC 1939 (May 1996). `http://www.ietf.org/rfc/rfc1939.txt`

42. Tim Showalter, “Sieve: A Mail Filtering Language,” RFC 3028 (Jan 2001). `http://www.ietf.org/rfc/rfc3028.txt`

43. Philip Hazel, *The Exim Filter Specification.* `http://www.exim.org/exim-html-4.30/doc/html/filter_toc.html`

44. Barry Leiba, “IMAP4 IDLE command,” RFC 2177 (Jun 1997). `http://www.ietf.org/rfc/rfc2177.txt`

45. Aaron B. Brown and David A. Patterson, “Undo for Operators: Building an Undoable E-mail Store,” *Proceedings of the 2003 USENIX Annual Technical Conference.* `http://roc.cs.berkeley.edu/papers/brown-emailundo-usenix03.pdf`

46. Robert Siemborski, “The Mailbox Update (MUPDATE) Distributed Mailbox Database Protocol,” RFC 3656 (Dec 2003). `http://www.ietf.org/rfc/rfc3656.txt`

47. Tony Finch, *A Hermes “thermal event”.* (Jan 2004). `http://www-uxsup.csx.cam.ac.uk/~fanf2/hermes/doc/misc/orange-fire/`

48. Jim Chisholm, *UPS explosion aftermath* (Oct 2003). `http://people.pwf.cam.ac.uk/jc235/UPS/`