Living under a Poison Tree

Erik Quanstrom
quanstro@coraid.com

ABSTRACT

Within the last year, the transition to the *nupas* [1] mail system has been completed at Coraid. In that time, the number of messages stored and the size of inboxes has increased by an order of magnitude. Yet the amount of storage required on the WORM has been reduced by an order of magnitude and the amount of core required has been reduced by an order and a half. Most of these gains have been realized through the caching strategies made possible by the *mdir*(6) format. Much smaller but significant optimizations and bug fixes in the last year have enabled the current level of performance.

Introduction

The introduction of *nupas* has been largely successful. Most heavy users access the system through IMAP. The system is accessed daily through Apple Mail, Firefox, Opera and Outlook. Last year at this time, both mail servers ran out of memory on a daily basis. With many very large inboxes, nearly 1GB of data per day was added to the WORM. Opening large mailboxes was understandably accompanied by a delay on the order of minutes. Today, though 14 new users have been added, less than 5% of available memory is used by *upas*, and the elimination one of the two mail servers is being considered. Only 100MB of data per day is added to the WORM. Logging now accounts for more of the dump than email. The largest mailboxes can now be opened in a few seconds. This is summarized in Figure 1.

date	total messages	total MB	largest box messages	largest box MB	dump blocks	core MB
200808	12491	1143	975	249	123000	5386
200908	157075	8307	15587	790	14500	482

Figure 1

While this improvement would not have been possible without the move to caching facilitated by moving to the *mdir* format, some surprising secondary limitations were found and some bugs were found due to faulty assumptions.

Hash Handling

The initial roll out of *nupas* was somewhat disappointing. The memory savings was less than expected. Several users were using a few hundred megabytes of core each. Part of this was explained by email clients such as Apple Mail keeping several connections open at once and the increase in IMAP-capable mobile devices. Even so, it was typical for users with large inboxes to have 50MB instances of *upas/fs* immediately upon opening the inbox. *Leak*(1) was unable to complete a scan of these processes. It was determined that linearly increasing the allocation for two large memory blocks in an interleaved fashion with other small allocations was causing pathological memory fragmentation. Switching to exponential allocation fixed *leak*, but revealed that there was no memory leak.

In addition, opening large mailboxes was taking an inordinate amount of cpu. Profiling indicated that the file hash table handling accounted for the bulk of startup time. Since the scheme for handling files was to keep them in a hash table. Each file needed a hash entry. Since each message part has 30 standard files for the message body, header, subject and so on, a message with n subparts will require $30n+1$ hash entries. Each one of these entries is allocated with *malloc*(2). A 5000 message mailbox would have a minimum of 150000 hash entries, but likely at least double that number. Since there were 1227 buckets, the load factor α on the hash table, or the average number of entries per hash chain, would be at least 122 [2], [3].

Given that the load factor is so large for such a small mailbox, and since it seems that building the hash table is expensive, it may be worth analyzing how long we expect building this table to take. Interestingly this topic does not appear to be explicitly discussed in [3]. If we consider adding a single element to a hash table, it's not hard to see that computing the bucket by hashing a predetermined set of strings with a fixed value is bounded by a constant time. Since we do need to guard against duplicates, adding an element to the hash chain, will take $O(1+\alpha_t/2)$ time [4]. Since

$$\sum_{t=0\rightarrow\alpha} 1+\frac{\alpha_t}{2} = \frac{(1+\alpha)^2}{4},$$

we can expect that loading the hash table will take $O(\alpha^2)$ time.

There are two potential solutions to this problem. Either replace the current lookup with one of sub-linear time and/or reduce the number of hash entries. The latter seemed like the best first approach as this could also address excessive memory use. And, given the poor big-O performance of our algorithm, any reduction of nodes will result in quadratic speedups.

The hash entries for the 30 standard files that populate each message directory were replaced with a single dummy entry entry "xxx." The file portion of the QID was stripped out. Since all message directories are numeric and all of the standard files begin with a letter, we simply lookup the dummy entry when asked to lookup file starting with a letter. If the dummy entry is found, the given name is translated to a file id by linear search of a static table. The file id is added back to the returned QID. This change reduces the number of allocated hash entries from $30n+1$ to $n+1$. Likewise the new load factor $\alpha' = \alpha/30$ and the time to build the hash table will be $O(\alpha'^2/4) = O((\alpha/30)^2) = O(\alpha^2/900)$. While this approach fails to address the quadratic behavior, it does address the memory use and provides three orders' of magnitude headroom.

It is important to note in this analysis that IMAP clients such as Apple Mail open each mail box every minute or so to check for new messages. If there are none the mail box is immediately closed. Thus the time it takes to open a mailbox is one of the most important benchmarks in our system. Also, due to the frequent mailbox scanning without closing the IMAP connection, reducing memory fragmentation is vitally important. Based on the experience with *leak*, a few small items (e.g. mime types) that were sure to be reused were freed when mail boxes are closed. This reduced the total long term memory use for *upas/fs* driven by Apple Mail by an order of magnitude.

Last year's *upas/fs* was benchmarked against this year's for a 15200-message mailbox. One further significant change has been made: the number of hash buckets has been increased to 1999. Both the time and memory usage to start and to run *nedmail*(1) are listed. The results are summarized in Figure 2.

date	start time (s)	start core (MB)	ned time (s)	ned core (MB)
200808	72	67	233	133
200908	0.40	17	3.3	17

Figure 2

While memory use is still somewhat disappointing, start time has improved by two orders of magnitude. Since memory use is predicted to be linearly related to the number of messages, it is envisioned that this issue will not need to be revisited until 30000-message mailboxes become commonplace, when it is expected that the $O(\alpha^2)$ of hash addition will again begin to be important.

Index Scanning

The scanning of the mailbox index relied on the order of messages in a mailbox being stable. A missing or extra message near the beginning of the list of messages (assumed to be in date-of-delivery order) could result in a large number of messages being deleted from the index and the mailbox. Initially it was assumed that this case was not important, since non-stable sorting would indicate a bug in the particular mail box code. Unfortunately a few bugs were found that deleted mail. It also proved a difficult problem to tackle for *mdir* mailboxes since they are sorted by date from the UNIX from line, since the order of delivery to the *mdir* is not available. Directory order is not stable when deletions are possible, since deletions on the file server simply mark a slot free and new files fill the first free directory slot. Yet new messages that are older than some (or even all) existing mail may be added.

The solution employed was to keep an AVL tree keyed on the SHA1 checksum of each message. This allowed to matching of existing message structures to index entries to be robust in the face of ordering problems or races between the index and mailbox. The AVL tree was also employed to detect duplicate messages. Duplicate messages and other rejected messages are now silently dropped rather than deleted.

Avoiding Mail Box Scans

For very large mailboxes, scanning the mailbox can be fairly resource intensive. Since mailboxes tend to be open many times, it is desirable to avoid duplicating this effort. To accomplish this, each mail box type may save a line of mailbox-specific information

to the index. If the mail box is older than the index, then the index is read without consulting the underlying mailbox. For example *mdir* mailboxes save the QID of the *mdir* directory to the index. If the QID matches the QID saved in the index, the index is considered "newer" than the *mdir*.

When the mail box scan is avoided, the difference for our 15200-message mailbox is 0.4s for the cached case versus 3.3s for the cached case.

Mdir Scanning; *Upas/fs* Scanning

The decision to sort *mdir* mailboxes by date continues to be problematic. Since new messages must be given a higher message number than older message ids, it is possible for messages to be out of numeric order. This requires all *upas/fs* clients to be aware that relative message ids may not be stable as they are with a tradition mbox format. It would be possible to declare the order of messages in the index to be the the order of messages in a mailbox. However the same problem would arise if the index is regenerated for any reason. This would seem to be an unwise dependency.

Sorting has been a particular problem for *imap4d* due to the quixotic definition of the imap UID and sequence numbers. IMAP uses two different numbers to as handles on a message. The UID is an almost permanent identifier assigned in increasing order. Sequence numbers are only valid during a session and take on values from $1-n$, where n is the number of messages in the mail box. The difficult requirement is that if $\mathrm{UID}_n < \mathrm{UID}_m$ then one must have $\mathrm{seq}_n < \mathrm{seq}_m$. This puts the sorting of IMAP (by UID) in conflict with the sorting of *upas/fs* (by date). Rather than trying to maintain two conflicting sorted indexes, the same AVL tree solution used for index scanning was employed. Other clients such as *nedmail*(1) simply read the entire message directory and resort.

Further Work

Clearly there is a lot of room for further work. Much of the further work mentioned in [1] remains undone. There is ongoing work to multithread the the file server interface of *upas/fs*. It seems that replacing the current hashing strategy needs to be in the medium-term plans. The AVL tree strategy seems fruitful and should be reused by *mdir* and *nedmail* to avoid the need to read and sort the mail directory as a whole. Sorting of mailboxes continues to be a sticky wicket. IMAP search and listing performance needs some reevaluation.

Credits

This work wouldn't have been possible without the patience of my coworkers at Coraid. In particular Brantley Coile and Ryan Thomas deserve special mention for suffering through many buggy versions. In addition Sape Mullender has reported many important bugs and included invaluable *snapfs*(4) debugging snapshots. Finally Gabriel Diaz Lopez De la llave has provided a lot of valuable feedback through his GSOC work related to adding multithreading to *upas/fs*.

Abbreviated References

[1] E. Quanstrom "Scaling Upas", proceedings of the International Workshop on Plan 9, October, 2008.

[2] E. Quanstrom "(n)upas update", email to the 9fans list, May 21, 2009, `http://9fans.net/archive/2009/05/106`.

[3] D. E. Knuth, *The Art of Computer Programming*, vol 3., 2d ed., 1998.

[4] G. H. Gonnet, “Expected Length of the Longest Probe Sequence in Hash Code Searching”, Department of Computer Science, University of Waterloo, CS-RR-78-46, 1978.