Tuning Sendmail for Large Mailing Lists

Rob Kolstad – Berkeley Software Design, Inc.

ABSTRACT

One of BSDI's mail servers hosts what might be the Internet's busiest mailing lists: inet-access@earth.com. This list now has about 2,000 subscribers and occasionally processes traffic as high as 200 separate messages per day. This 400,000 message per day aggregate is a taxing load for a non-optimized mailing system.

When the project started, the mail queues sometimes lagged as much as five days (hundreds of thousands of messages) behind. A single message could take more than five hours to attempt delivery to members of the list. Furthermore, the disk load was high as sendmail processed the queues repeatedly (trying to deliver previously missed mail). All in all, the system's efficiency was remarkably low.

This paper describes the procedures undertaken to reduce delivery times to under five minutes (for all 2,000 subscribers) and mitigate problems associated with unavailable hosts.

Outline

First, the paper discusses the problem of large mailing lists and processing them. After outlining the goals for the new processing system, the state of the old system is described. The methodologies employed to reduce overhead and increase throughput are then discussed along with tools for measuring and aiding performance. Finally, the final state of the system is presented.

The Problem

Mailing Lists

Mailing lists are surely the first (and arguably only) 'push technology.' New information is moved toward the consumer shortly after it is produced. These with online mail systems (vs. POP-style mail systems) who additionally employ *biff*(1), can receive notification of mail delivery in real-time. This convenience and timeliness have motivated mailing lists' popularity since the first days of ARPANET mailers. As the 'net proliferates, lists are increasingly for a variety of discussions and for a variety of communities.

Mailing list sizes run from the tiny (a handful of recipients) to the huge. InfoBeat (formerly Mercury Mail) creates customized content for various mailing lists and (in August, 1997) sends out more than 200,000 messages every afternoon.

Surprisingly, mailing lists continue to be popular in spite of other technologies like USENET News and the World Wide Web. This might be because of the pure 'push' properties of the mailing lists. I hypothesize that people have integrated mail reading into their daily (or hour or minutely) task schedule and that the mingling of mailing lists into such a paradigm is just too convenient to change.

Mailing List Servers

The convenience and efficacy of receiving mailing lists depends on the mailing list servers (hereafter 'list servers' or just 'servers'). If a list server fails to deliver a message in a timely manner (or at all!), much of the effectiveness of mailing lists is lost. As lists grow, delivery becomes an ever more resource-intensive procedure that is complicated by all sorts of factors, including: host outages, network link outages, DNS service problems, slow clients, slow networks, and packet loss. Any of these factors can increase mail delivery time from the 50-1000 millisecond timeframe to the multi-day timeframe (as much as seven orders of magnitude).

Furthermore, queueing a large number of messages to a list server can seriously impact its performance on other tasks. Sendmail's standard paradigm for processing disk queues can cause severe resource saturation when mail queues have more than one or two thousand messages ready to be delivered. This saturation can impact server efficiency so that only a few messages per minute are delivered. Tuning such systems is painful since disk operation (including paging in editors, for example) is impacted. Mixing high priority mail delivery (e.g., corporate mail) with a high volume mailing list can produce disastrous and unacceptable results like 12 hour delivery times for mail from one desktop to another desktop in the same office.

The properties suggest a set of goals for a good mail server environment.

Goals for Mailing List Servers

The combined needs of the message recipients and list administrators yield a set of common-sense goals:

- Fast delivery of messages (low latency)
- Reasonable consumption of server resources

- Easy (or least low time commitment) administration
- Use of existing tools for list processing
- Ability to monitor results to ensure goals are being met

Delivery speed is surely a primary consideration of mailing lists. Arguments can be made that reasonable maximum delivery times run anywhere from a few minutes to an hour. Analyses of "as fast as possible" suggest that a maximum delivery time of 5-10 minutes for a few thousand users is reasonable and acceptable to the vast majority of users.

Reasonable server resource usage is an important goal for list servers. Otherwise, they do not scale well with increased list size and become too expensive for organizations to support (both in terms of hardware resources and both direct and indirect administration costs).

Reasonable administration costs is another important goal. List servers that require more than a few minutes of daily maintenance start to become problematical when:

- ○ the list administrator is unavailable (e.g., for vacation or a conference),
- ○ management is looking for ways to reduce overhead, or
- ○ the personal patience of the administrator is tried (for any reason).

Use of existing tools is a prerequisite for continuing goals of low-impact administration, low-impact costs (both direct and indirect) for running a mailing list, and ability to share innovations with other list administrators. Our site would probably opt out of running mailing lists if costs for extra software were required.

Monitoring performance and resource usage becomes important once the hardware costs of server mailing lists are exposed. Neither network bandwidth, server hardware, nor administration time is free. If high quality service is not in the offing, these costs are best directed towards other, higher impact and more effective services.

Initial State

BSDI hosts the inet-access@earth.com mailing list. It currently support approximately 2,000 subscribers with anywhere from 100 to 200 messages per day. At the time this analysis began, the message had just over half that many subscribers and slightly less traffic.

Probably the biggest problem was that delivery times were starting to exceed five hours and the system load seemed to be increasing very quickly. Paging was high; disk I/O was high; machine performance was sagging.

A quick check revealed 50,000 messages in the mail queue. This was surely one of the causes of the high disk I/O as sendmail explored the queue with each new process instantiated to mop up previously undelivered mail.

Checking *ps*(1), showed over 100 sendmail processes running. Normally, this wouldn't seem very painful for a mail server, but in this case each process consumed over 2.5 MB of memory! This caused the paging and exacerbated the disk load.

The load average ran from 5 to 20 even though the CPU was not saturated. The (mostly) busy disks and paging were keeping jobs from running.

Interestingly enough, most queued deliveries would 'catch up' overnight and the mail queues would be quite tolerably small by the time the first morning batch of list mail began to be distributed. This put a certain kind of cap (18 hours) on mail delivery time, but the overall big picture was quite alarming since other mail going through the mail delivery machine would be delivered in hours instead of milliseconds.

The 'Fast Fix'

As a quick-and-dirty fix to increase throughput, Tony Sanders (inet-access's owner and list maintainer) gathered statistics about mail delivery times. Here is a typical line (displayed here broken down into fields) that he used to analyze the delivery time:

```
Sep  6 10:54:25 – date message was delivered
ace – hostname upon which sendmail is running
sendmail[24338]: – process name and PID
KAA24336: – date message was delivered
to=peter.j.scott@jpl.nasa.gov, –
            recipient
ctladdr=kolstad (101/0), – sender
delay=00:00:07, – delay from time message
            was queued until delivery
xdelay=00:00:06, – time for this particular
            delivery attempt
mailer=smtp, – mailer used
relay=mailhub.jpl.nasa.gov.
            [137.78.18.34], –
            destination machine
stat=Sent (Message received and
            queued) – final status
```

On a daily basis, Tony Sanders gleaned the xdelay information for each recipient (across dozens of mail messages in a given day) and then sorted outgoing mail list so that those with lower xdelay averages were delivered before those with higher averages. This had the property of rewarding 'good citizens' with outstanding delivery time (i.e., the first 100 or so good citizens received their mail within a minute of its arrival at the list server). Furthermore, 'better' citizens were not punished by having their mail delayed by multi-minute timeouts behind unavailable hosts. Those people at the end of the queue still waited over five hours for their mail to be delivered.

The First Suggestion

I entered the scene because one of my mailing lists (the USA Computing Olympiad list) was being delivered ever more slowly. What used to take 20 minutes was taking hours.

I checked out the system and observed the symptoms reported above. I suggested that the 1,500 person mailing list be split into 75 lists of 20 recipients. My reasoning was that 75 lists of recipients would be processed in parallel and all the 'waiting' (for hosts to answer, etc.) would be parallelized and there would 'always' be some host ready to communicate. And, of course, how long could 20 deliveries take? I was hoping to deliver all the messages in a minute or two by this increase in parallelism.

In a bizarrely political process, I was completely overruled by our system administration staff who complained quickly and bitterly that we could not possibly support the RAM and process slot requirements of 75 parallel sendmail processes. The fears were based on the notion that three or four messages would arrive in a small interval and thus stress the system with 4 x 75 = 300 processes and, worse, 4 x 75 x 2.5 = 750 MB of virtual memory requests. I was surprised that sendmail used up so much data space (since the code space is shared), but I am a strong believer in delegating authority and this particular authority had in fact been deleted.

Second Try, First Suggestion

I bargained with the administrators. I suggested that we split list into four lists of 375 recipients each. They were not pleased. However, being the company president, the subtle force of that office won the day. We fixed the outgoing mailing list processor to mail to four different aliases of 375 recipients instead of one big alias of 1,400 recipients.

Primitive analysis tools showed a an improvement in throughput of roughly 4x. 'Maximum' mail delays were reduced from over five hours to less than two hours. The delivery rate was increased proportionally. The number of processes in use did, in fact, increase. The RAM usage was high but not nearly as high as the 2.5 MB per process that had been feared.

It was difficult to measure throughput since the mail queue status indicators are only updated every ten message deliveries. It was at this point that we committed a huge error and changed the update rate in the /etc/sendmail.cf file from 10 to 1:

```
# checkpoint queue runs after every
# N successful deliveries
O CheckpointInterval=1
```

Do not do this at home.

An Aside on the Environment

For better or for worse, the inet-access mailing list environment was a 'live' environment. Everyone involved in this project had to take care not to endanger the ultimate throughput of the list. We knew that we lagged as many as 50,000 messages by the time prime-time for mailing ended each day. A bad move would result in getting more than one day behind and, thus, potentially failing to catch up overnight when the traffic was reduced. We feared that falling too far behind would leave us unable to catch up ever.

Second Experiment

Heartily encouraged by the results of four lists instead of one, I wanted to increase the number of lists while decreasing the number of users per list.

It was clear at this point that monitoring tools were necessary that could:

- monitor the instantaneous rate of delivery and
- summarize the day's performance.

Without such tools, evaluating the status of a new experiment would be difficult if not impossible.

It was decided to watch processes, RAM use, disk I/O rates, and network I/O rates. A bottleneck in any of these areas would cause a 'hard limit' on performance.

As the number of recipients in each 'list chunk' was decreased, delivery rates and throughput increased.

This was puzzling, to an extent, because of the initial fears of process table crowding and virtual memory consumption. Observation of the RAM use showed that a sendmail process delivering a single message never used much RAM. Later observations showed that only 'older' sendmail processes used lots of RAM, suggesting speculation about a memory leak or a table that grows with status information about destination hosts. At any rate, having solved the puzzle, it was easy to envision ever more list chunks with ever smaller sets of recipients.

Disk I/O continued to increase as the number of entries in /var/spool/mqueue increased. Network I/O never got very high. Our T-1 line was never seeing much usage, never more than 10-15%.

Delivery times had declined to less than one hour. This caused queue sizes to be considerably reduced so that list delivery was keeping up with the lowest possible expectation of throughput. This was a good milestone.

Head Scratching

We observed RAM usage was now relatively low. We were trying to find the bottlenecks that were reducing throughput. We observed:

- Disks are getting busier
- CPU isn't that busy
- Load average isn't getting worse

- Network isn't the problem

The actual cause of less-than-maximum throughput was unknown.

Maximum Throughput

Of course, when one is concerned about achieving maximum possible throughput, one should try to figure out precisely what the maximum throughput could possibly be.

I wrote a small program, 'mailtest.c.' This program implemented what is arguably the smallest set of steps necessary to deliver a message, it:

- Opened port 25 on the destination machine (note that DNS resolution is in some other process),
- ran the SMTP protocol,
- sent approximately the shortest message that the mailing list saw,
- closed the remote port, and
- exited.

On a PentiumPro/200 processor, this task (in repeated observations) ran in about 30 ms. This is implies that the maximum throughput of a mailer (under the most ideal of all possible conditions) running on a PPro200 processor with a nearby recipient machine whose name was already resolved was:

3600 sec/hour / 30 ms/message = 120,000 messages/hour.

The main bottleneck for this particular experiment was the CPU though the network was running close behind.

Statistics

It was time to build tools. The first tool ('batchstat') was designed to show a day's summary of mail throughput. See Figure 1 for a sample of its output.

Initial observations were performed by watching the output of the mailq command. Tony Sanders wrote a slightly modified version called mailqq that showed the number of messages to be delivered instead of the names of all recipients for a message. It also found the sum for all messages waiting to be delivered.

The process of running mailqq while the /var/spool/mqueue queueing file update interval was set to update after each message was delivered gave a relatively depiction of mail throughput, though its effect on disk performance was quite debilitating (since it read so many files to find its statistics). More on this later.

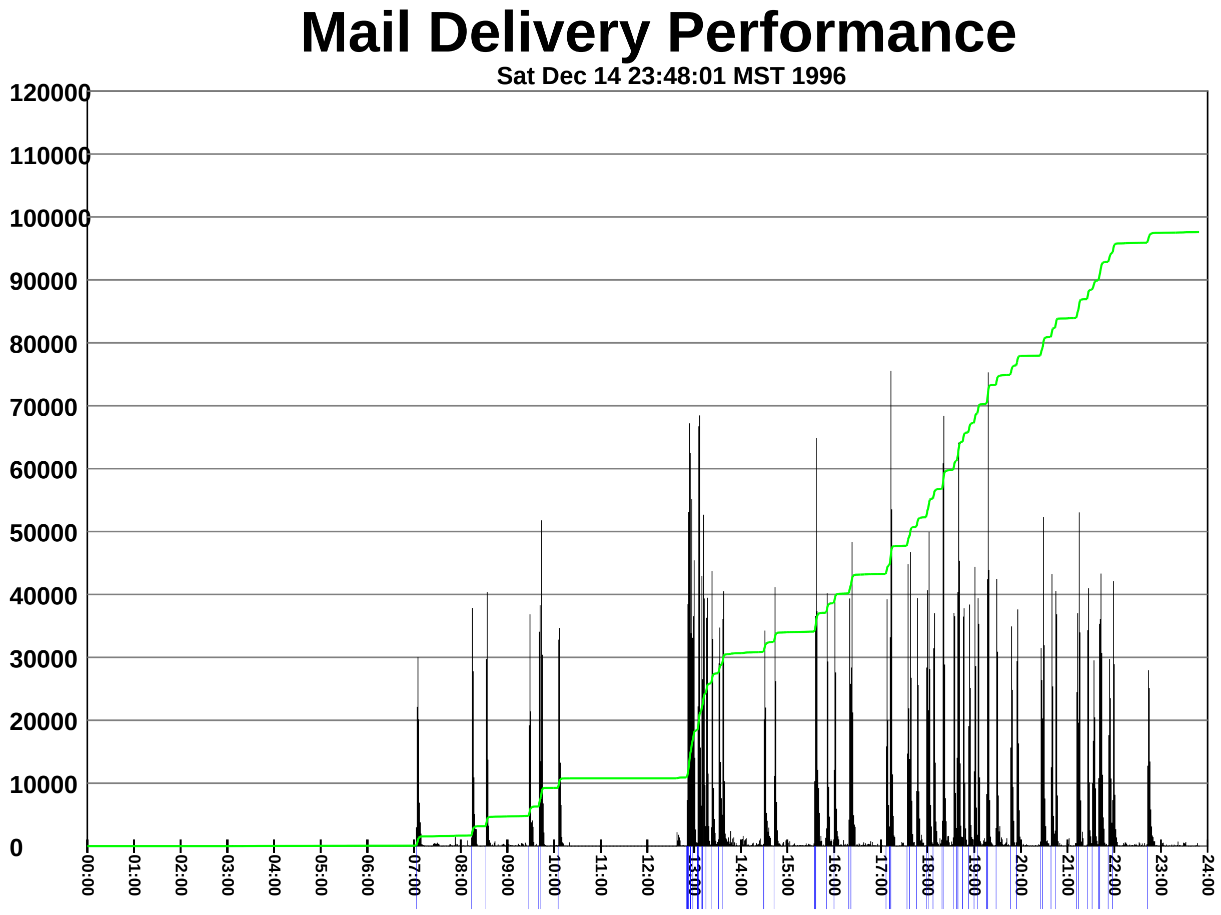

Figure 1: Initial batchstat output.

batchstat

The batchstat program labels the output page, the left axis in a processing rate measured in messages/hour, and the bottom axis in time throughout a single day. Note that statistics for a day often start at 2:00 am instead of midnight. This doesn't change our observations, though.

The dark black vertical spikes running up from the X axis depict 'instantaneous message delivery rates' through the day. These are actually averages over a minute or two. The light vertical lines below the X axis show when a new message arrived for delivery to the mailing list. Later versions of this program went out of their way to ensure at least one pixel between these bars so that high arrival rates could be observed accurately.

The lighter gray bar that slowly increases in value across time is the integral of the spikes: it shows the total messages delivered so far since midnight. In this example, about 98,000 messages were delivered across the day.

The obvious goal of a mailing list delivery program is to deliver all the messages the instant a message arrives. On this graph, such behavior would display as a spike whose height is infinitely high for a very short period of time. Higher thinner/narrower spikes for each incoming message show better performance of a mailing list handler.

realstat

Waiting an entire day to see if the spikes showed up for mail deliveries was a nailbiting experience. The realstat program read the realtime output shown in /var/log/maillog so it could display performance in a small window. Figure 2 shows realstat's output during a slow period.

```
  9:00:00   480/  2
  9:00:15   960/  4
  9:00:30   960/  4 --
  9:00:45  2160/  9 *--
  9:01:00  1920/  8 *--
  9:01:15   720/  3 --
  9:01:30  2400/ 10 **--
  9:01:45   720/  3 --
  9:02:00  1440/  6 *-
  9:02:15   960/  4 -
  9:02:30   960/  4 --
  9:02:45  1440/  6 *--
```

Figure 2: Realstat Output During Slow Period

The first column shows the time of the observation. The observation time is actually the 15 seconds leading up to the time of observation. The second column shows the hourly rate of message delivery. The next column shows the actual number of messages delivered in the previous interval. This number is scaled and presented graphically as '*'s. The set of dashes shows, in the same scaling, the number of messages that were attempted to be delivered but, for some reason, failed. For this graph, far more messages failed to be delivered than succeeded.

mailstat

The mailstat program calculates a numerical summary of messages delivered since the /var/log/maillog file began. See Figure 3 for sample output.

```
mailstat: Sat Feb 22 09:05:40 MST 1997
              failed      deliveries
MMM DD HH mhosts/recipt mhosts/recipt
=== == == ============= =============
Feb 22 02    433/   612   1111/  1111
Feb 22 03    495/   696   1298/  1298
Feb 22 04    431/   615   1137/  1137
Feb 22 05    421/   610    810/   810
Feb 22 06    422/   606    717/   717
Feb 22 07    411/   587    931/   931
Feb 22 08    427/   616   1039/  1039
Feb 22 09     22/    22    105/   105
========= ============= =============
Totals      3062/  4364   7148/  7148
```

Figure 3: Mailstat output, slow day.

The first three columns show the date and hour for the summary shown to the right. Subsequent columns show the number of hosts that a message to which a message actually failed to be delivered and the number of messages that failed (which, surely, is always at least as high as the number of hosts). Similar statics follow for successful deliveries. Figure 4 shows mailstat output when the queue is full of messages that can't be delivered for some reason.

```
mailstat: Sat Feb 22 09:03:49 MST 1997
              failed      deliveries
MMM DD HH mhosts/recipt mhosts/recipt
=== == == ============= =============
Feb 22 03  18811/ 18811    522/   522
Feb 22 04  27065/ 27065    574/   574
Feb 22 05  29342/ 29346   1738/  1887
Feb 22 06  29973/ 29978      8/     8
Feb 22 07  26668/ 26675   1556/  1690
Feb 22 08  11768/ 11787   1347/  1464
Feb 22 09   1893/  1896    566/   606
========= ============= =============
Totals    145520/145558   6311/  6751
```

Figure 4: Mailstat (few deliveries, many failures).

Further Experiments

We continued to increase the delivery parallelism. Eventually, there 100 lists of 15-20 people each. This caused ever-decreased delivery time, a very busy machine, and incredibly busy disks.

Examination of the disk I/O rates showed that each mail delivery was causing lots of disk I/O. This brought to mind the update of the queueing files for each mail delivery. Because sendmail is super-safe in its algorithms for changing on-disk data structures, many of the operations for updating the queueing files ended up requiring synchronous disk operations. While this enabled observation of disk queue sizes, it required an unacceptable disk I/O load burden. Besides, one can observe /var/log/maillog to get an even better understanding of statistics for mail delivery. There was no double that synchronous disk operations were destroying performance.

The configuration file was changed to 'update the queueing files every 10 deliveries.' This change immediately removed the disk I/O bottleneck from the system's performance. Figure 5 shows 275,000 messages transmitted in one day with very high throughput. Even though the throughput is averaged across 60 seconds, some of the spikes are trending toward the theoretical maximum throughput (though the conditions are far less than ideal).

Next Analysis Step

Trying to characterize both the high and low performance periods led to a certain set of discoveries. First, the number of hosts unavailable for delivery has a profound impact on throughput. Why? Because a sendmail process waits a long time in the vain hope that a response might be received to the open on port 25. Second, the number of messages available to deliver impacts performance. How? Because when lots of messages can be delivered, more processes can run in parallel and deliver more messages per unit time.

It was observed that the mail queue still had substantial size, even though messages were being delivered quickly through the day. Figure 6 shows some (edited) mailqq output when the queue size was nonzero.

Each message has a few 'stragglers' that are not being delivered. A quick perl script ('latedudes') shows the message counts and names of recipients with messages in the mailqueue:

```
 82 todd@acc.com
127 glennh@netstation.net
127 ispmail@zhi.dialup.access.net
127 jnussbaum@americandata.net
127 kevinc@rrt.com
127 mp3@cyber-gate.com
127 nevin@shadowave.com
127 tcosta@biznm.com
164 rdavis@masschaos.de.convex.com
200 whenpigsfly@worldsrv.net
```

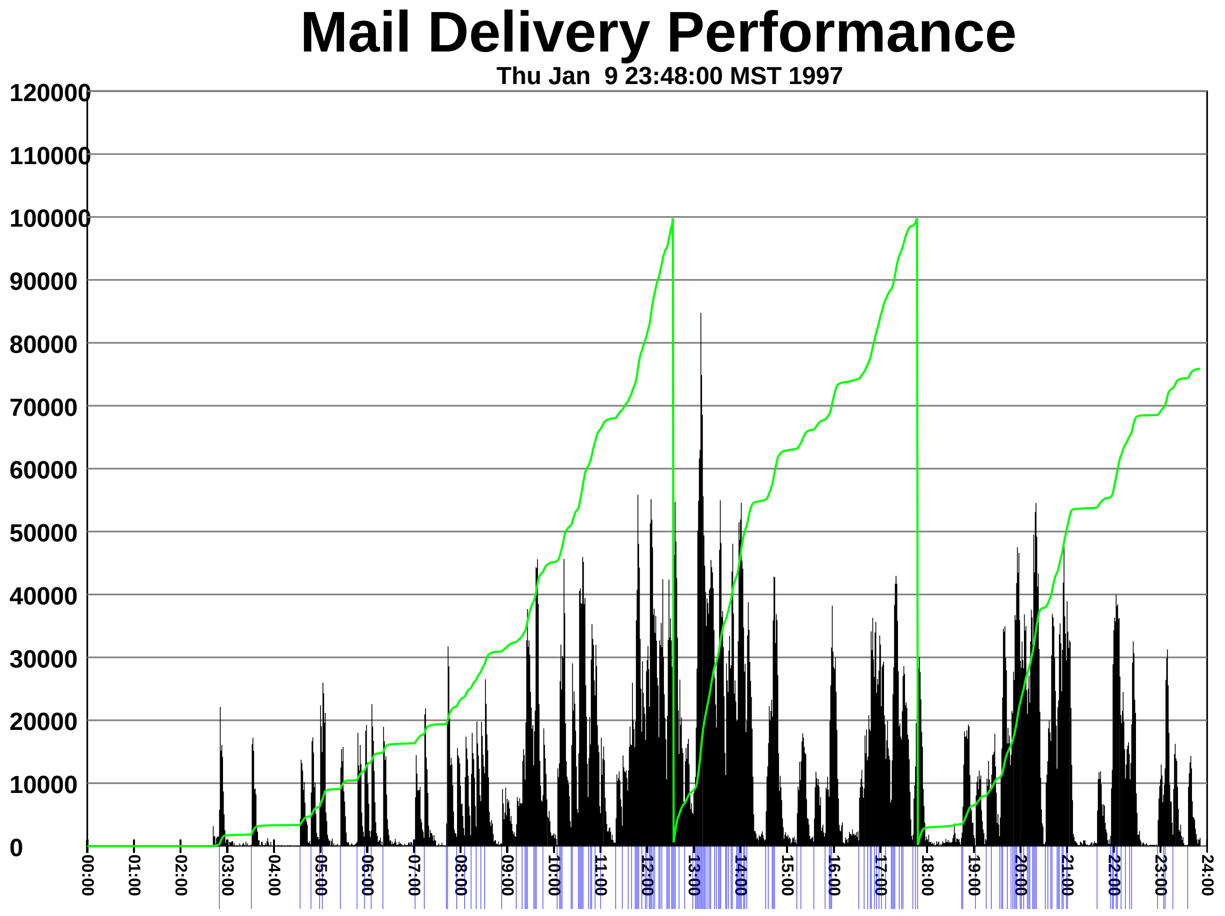

Figure 5: Improved throughput (small message queues, disk bottleneck removed).

```
  388 cbrown@matnet.com
  559 robert_thompsen_at_usr-css...
  593 berney.ortiz@mailserver.oi...
  595 list.inet-access@optimum.net
```

What a surprise to find some recipients had not accepted mail for many days. In the worst case, each of these messages would have its own queue file in the mail spool.

The impact of each of these recipients can range from the trivial to the dramatic. Here's what happens when sendmail tries to send to a host that is 'busy, hung, or dead.' First, sendmail tries to connect to the host. Maybe the connection succeeds (but host turns out to be slow or net is losing lots-o-packets). Maybe the connection fails.

Each step in the SMTP protocol from connection through completion has long time-out (like as much as 300 seconds). This means that a particular sendmail process idles for five minutes waiting for a reply. This reduces throughput – especially when 100 sendmail processes are conducting this exercise in parallel.

At any point in time, 1-3% of recipients – and these recipients are ISPs – are unavailable. InfoBeat reports a number closer to 10% for average users. This is not to say that users and ISPs have control on all possible outages – cable cuts do make a difference. Nevertheless, in a mailing list with 1,400 - 2,000 participants, 2% is anywhere from 28 to 40 recipients. This means a total of 6,000 to 8,000 messages per day can not be delivered!

Next Step

Obviously, it would be advantageous to reduce timeouts for initial contact/mail transmission. This would enable 'good citizen' sites to continue their good throughput. Some sort of 'reaper' process could come along later to deliver to slower (or dead) hosts.

Happily, all these times are configurable in sendmail. We reduced them 5x. This sped up initial mail delivery, though some messages were, of course, not delivered. A second sendmail.cf file with slower timeouts was created and run three times/hour.

Note that this is all in the context of sendmail already remembering when a host is unavailable and not trying that host again for a one hour period.

Reducing Queue Search Time

The most painful part of running mailing lists is the *manual* removing and adding people to the list. For the inet-access list, Tony had a policy of not removing people from the list for a bounce or even for two days of list bounces. This made the outgoing queue grow significantly.

So we created 10 more queues to run separately. We moved jobs moved from queue to queue when older than a certain amount of time. We automatically scheduled ever more 'reaper' processes to run those (presumably smaller) queues.

Regrettably, it never seemed to help. Performance differences were barely measurable, if at all. This idea was abandoned.

Summary of Modifications So Far

All in all, it doesn't take many changes to speed sendmail's throughput dramatically.

First of all, use lots of parallelism (hundreds of processes in parallel). Of course, one should reduce impact of unavailable recipients by keeping track of hosts that won't answer and by reducing the timeout for hosts that can't keep up with standard Internet speeds.

Stragglers continue to have effects by slowing mailq commands (which touch each file) and slowing sendmail itself for queue runs (which also touch each file).

The mail delivery time was reduced and throughput increased with ever better spikes; see Figure 7. In fact, the high load performance is also outstanding. Figure 8 shows deliveries after one main spooling machine was down until noon one day.

```
[...]
nrecipients  length  date                sender
3 BAA08598  1554 Sat Feb 22 01:02 <inet-access@earth.com>
3 BAA08677  1017 Sat Feb 22 01:29 <inet-access@earth.com>
3 FAA10201  1438 Sat Feb 22 05:24 <inet-access@earth.com>
3 FAA10208  1438 Sat Feb 22 05:24 <inet-access@earth.com>
3 HAA10369  1527 Sat Feb 22 07:46 <inet-access@earth.com>
3 IAA10524*  423 Sat Feb 22 08:52 <inet-access@earth.com>
4 HAA10371  1527 Sat Feb 22 07:46 <inet-access@earth.com>
4 HAA10383  1527 Sat Feb 22 07:46 <inet-access@earth.com>
4 IAA10544   423 Sat Feb 22 08:53 <inet-access@earth.com>
4 IAA10558   423 Sat Feb 22 08:53 <inet-access@earth.com>
5 IAA10541*  423 Sat Feb 22 08:53 <inet-access@earth.com>
```

Figure 6: Mail queue analysis after several nondeliveries.

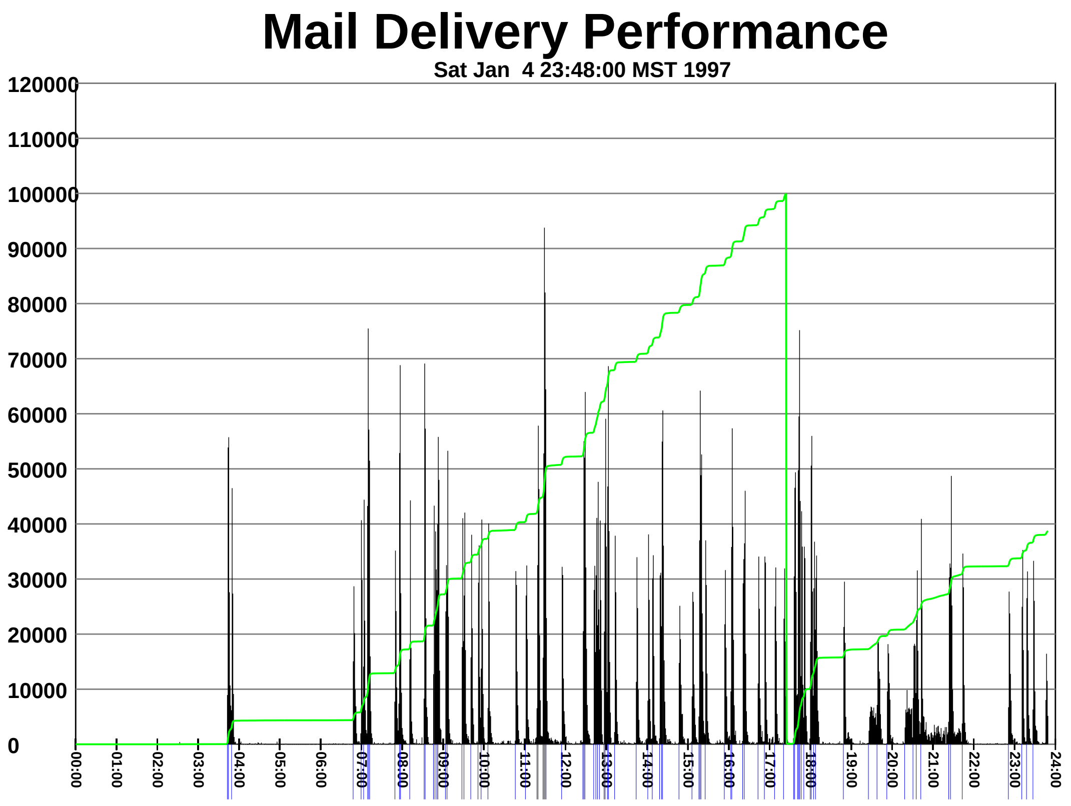

Figure 7: Good performance after modifications.

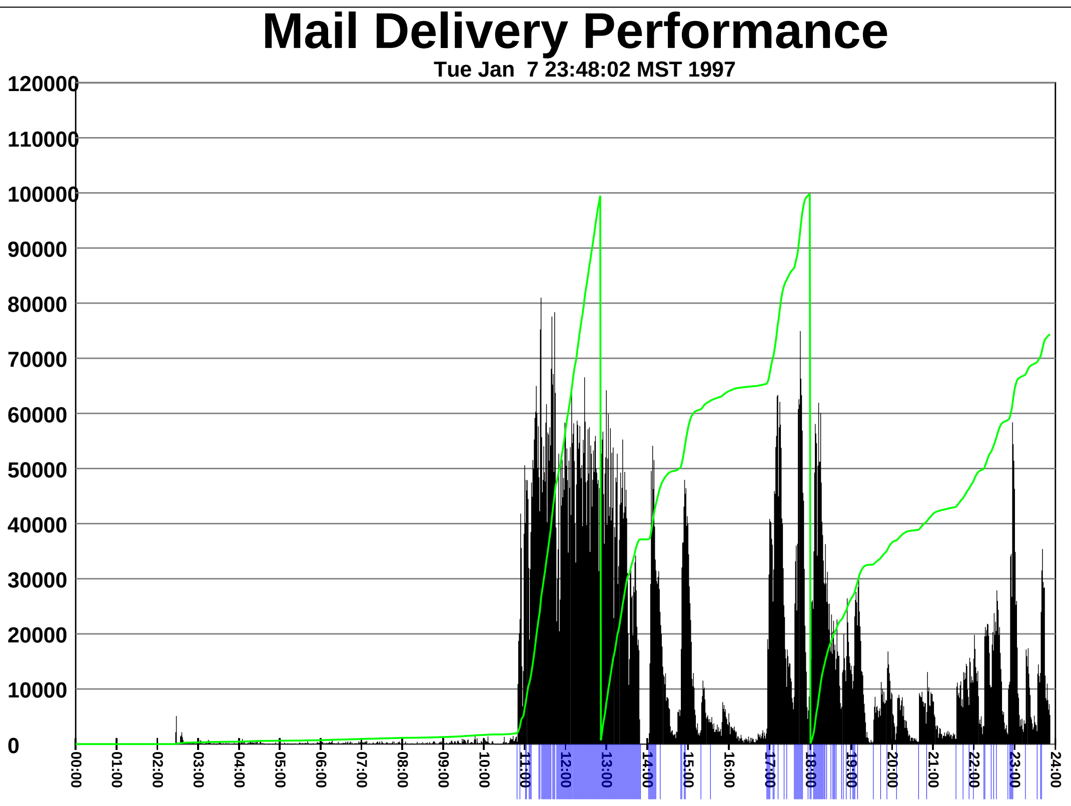

Figure 8: Good performance under load.

Futures

Now that the system is up, running, and manageable again, it's easy to see some of the future improvements that might be tried.

First of all, stragglers should be coalesced into a single message/recipient-list pair. This would reduce the queue sizes trivially. One can even envision creating a large digest for stragglers and having one queued file per straggler instead of one queued file per message. Obviously, it depends on the ratio of stragglers to messages per day.

Secondly, one might consider a policy of just deleting messages older than, say, 24 hours. These people can always look at an archive.

A grandiose plan (and I do mean unmanageable) would be to rewrite sendmail for a high – but constant – number of concurrent transmission processes coupled with the use of extended SMTP to send multiple messages once a machine is up. This would make sendmail treats its queues in a sort of contrapositive way to the way it treats them now (sending all messages to one site rather than one message to all sites).

To be fair, though, sendmail already close to achieving highest possible bandwidth for a given speed of network connections. It's not clear any of these measures would increase actual throughput.

Also note that even with these improvements, messages smaller than 10KB or so don't push T-1 speeds to their limit yet. It takes pictures for that.

Availability

All scripts mentioned here are available by sending a short yet descriptive e-mail request to the author <kolstad@bsdi.com> .

Conclusion

It is not that difficult to reduce mailing list latency dramatically. We now have a script to insert in /etc/aliases to break message into parts. The time to deliver 95% of a mail queue was reduced from 5 hours to 3.5 minutes. The unavailability of recipients is still the biggest performance problem.

And, most amazingly, sendmail does about as good as can be done in delivering large amounts of mail! Figure 9 shows the latest performance figures as traffic has increased and a new file system design has reduced file I/O dramatically.

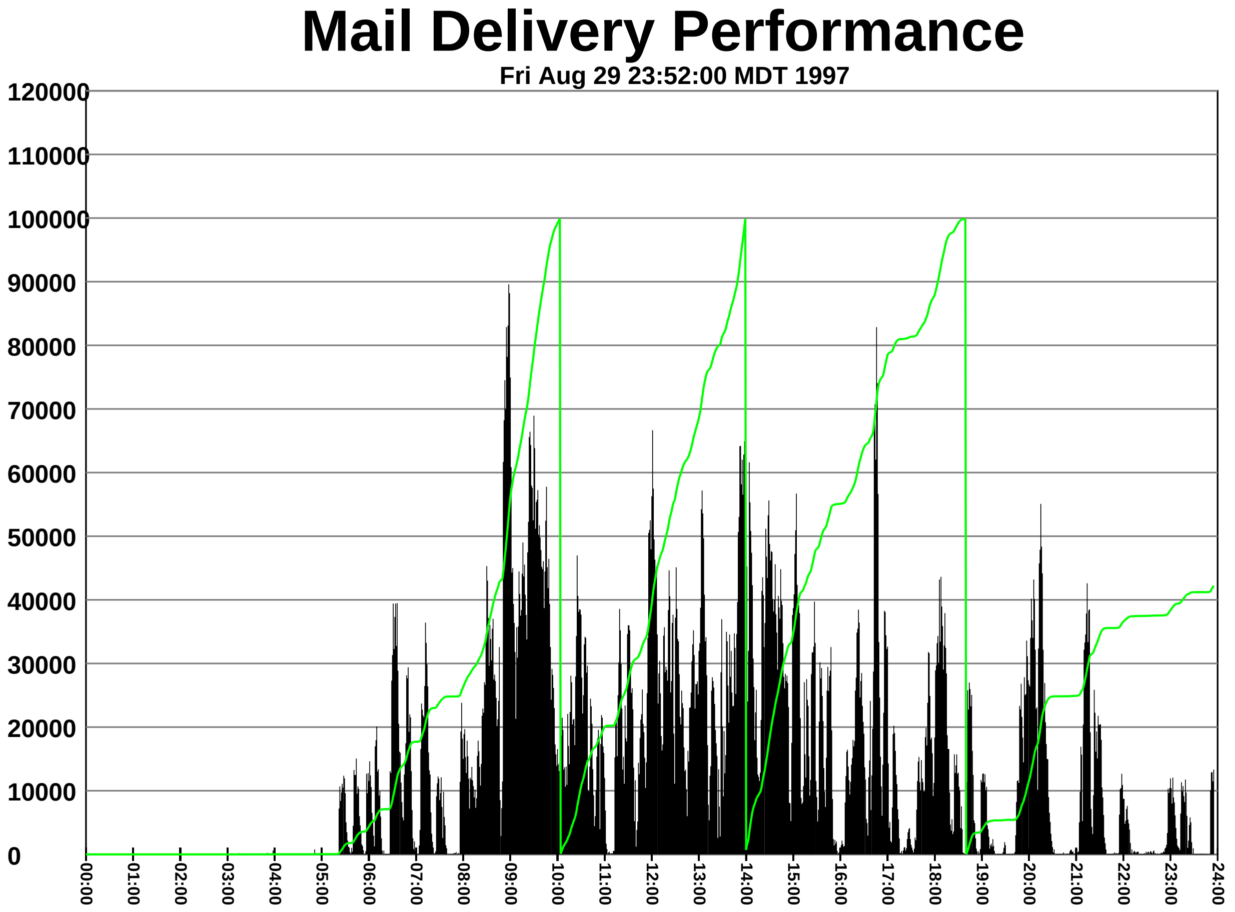

Figure 9: High traffic with new file system.