FreeBSD mailserver performance tuning

1. Increase kernel limits

With hundreds of exim processes, the kernel will eventually run out of file handles; after that, it will run out of sockets. Some important parameters to look for are:

```
# sysctl -a | less
...
kern.ipc.nmbclusters: 1024          # no. network memory buffers (2K each)
kern.maxproc: 532                   # max. number of processes
kern.maxfiles: 1064                 # max. number of open files
kern.maxfilesperproc: 957           # max. number of open files per process
```

Many of these can be changed on a running system:

```
# sysctl -w kern.maxfiles=16384
# sysctl -w kern.maxfilesperproc=2048
```

For this to happen automatically upon next reboot, create "/etc/sysctl.conf" with these lines:

```
kern.maxfiles=16384
kern.maxfilesperproc=2048
```

Others may say "read-only". Some of these can be set at kernel boot time, by adding the following lines to the **/boot/loader.conf** file then rebooting:

```
kern.ipc.nmbclusters="16384"    # Note: this will take up 32MB of RAM
kern.maxproc="1024"
```

An alternative method is simply to recompile the kernel with a higher value of MAXUSERS in the kernel configuration file, for example MAXUSERS 512 (the default is a value between 32 and 384 depending on how much RAM is in your system). Many of the parameters are derived from a formula based on this value, for example kern.maxproc defaults to MAXUSERS*16 + 20, so a default system has a limit of 532 processes.

For more info see http://www.freebsd.org/handbook/configtuning-kernel-limits.html

2. Enable 'softupdates'

Delivering E-mail causes spool files to be written and deleted frequently; delivering messages into a Maildir also creates and deletes files for each message. FreeBSD's old default behaviour of doing synchronous metadata writes means that each file creation/deletion causes a disk operation to occur at that moment - they could not be cached in RAM for writing later. Linux's default behaviour is do do everything asynchronously, which is fast but can cause major filesystem corruption if the power is lost during disk activity.

FreeBSD provides a solution to this: 'softupdates'. With softupdates, writes of metadata are delayed but are sequenced so that the filesystem always remains in a consistent state. With softupdates, you can increase your performance from a few hundred file creations/deletions per second to many thousands per second, without losing the safety of the UFS filesystem.

Recent version of FreeBSD enable softupdates by default on every partition you create at installation time, apart from the root partition. You can check that softupdates are enabled using 'mount':

```
# mount
/dev/ad0s1a on / (ufs, local)
/dev/ad0s1g on /u (ufs, local, soft-updates)
/dev/ad0s1f on /usr (ufs, local, soft-updates)
/dev/ad0s1e on /var (ufs, local, soft-updates)
```

To enable softupdates on an existing filesystem you need to reboot into single-user mode ("boot -s" at the boot loader command prompt) and use

```
# tunefs -n enable /mountpoint
```

3. Use SCSI disks

SCSI drives generally perform much better under heavy utilisation than IDE drives. Also, drives with a higher rotational speed (rpm) will have a shorter latency for accessing data and can therefore perform more accesses per second. Beware that higher-speed drives generate more heat, and therefore proper case cooling becomes critical. Consider buying a good-quality SCSI drive chassis, external from the server itself.

SCSI has another important advantage: you can put many devices on a single SCSI bus (7 or 14), and they can be accessed concurrently. That is, once the controller has issued a command to a drive to fetch some data, it disconnects from the bus while it works on the request, and the bus is then free to talk to other drives.

With IDE you need a separate controller for each disk. Putting two disks on one controller does not work well, because 'master' and 'slave' cannot be accessed simultaneously.

4. Spread mail directories across multiple disks

If a disk has a 5ms average latency, that limits it to 200 operations per second. To get more operations per second, you need more disks. You can achieve this simply by putting different users' mail directories on physically separate drives, e.g.

```
/mail1     -- /dev/da0s1g     (First SCSI disk)
/mail2     -- /dev/da1s1g     (Second SCSI disk)
/mail3     -- /dev/da2s1g     (Third SCSI disk)
```

This is because accesses to different disks can happen concurrently. (Note that there is no advantage in using different partitions on the same disk!). If using IDE disks, each must be on its own IDE controller.

It's possible to use 'striping' to concatenate all three disks together so that they appear as one large disk, but this is generally not a good idea for a Maildir system where we need to access many small files concurrently rather than a few big files. As well as performance issues, managing the volume is difficult (i.e. you cannot easily add a fourth disk), and if a single disk fails, the entire volume becomes unusable. It's also easier to debug problems caused by a single troublesome user if they are isolated to a single disk.

In addition to the drives where users' mail is stored, you ought to have a separate system disk for booting from (which can be IDE). This drive can also store log files and have Exim's spool directory, which is written to for every message received. This drive will therefore be doing work, although experience shows that this is unlikely to become a bottleneck (consider that receiving a million messages per hour is only 12 messages per second).

5. Use mirroring, not RAID5

If you want to protect against disk failure, for best performance you should use mirroring rather than RAID5. For example, if you have six drives, then configure them as three mirrored pairs:

```
/mail1     -- disk 1 and 2
/mail2     -- disk 3 and 4
/mail3     -- disk 5 and 6
```

Using RAID5 you would have increased data storage capacity, because you would have 5 disks containing data and one for parity. However, writes to a RAID5 array are slow: to write a single block on disk 1 requires two reads (read the old data from disk 1 and the old parity data from another disk) followed by two writes (write the new data to disk 1 and the new parity data to another disk). Given that files are being created and deleted constantly in a busy mailserver, this reduces the bandwidth available.

In contrast, with a mirrored set, writing a block just requires two writes to the two disks. Also, because there are two copies of every item of data, you get double the read bandwidth: one process can be reading from disk 1 at the same time as another process is reading different data from disk 2.

6. Put in as much RAM as possible

Any extra RAM is used as disk cache.

7. Use PCI cards, not ISA!

Especially important for network interface cards and SCSI controllers.