High Performance Web Caching With Squid

By Joe Cooper

Table of Contents

Introduction

Squid is an excellent open source web caching proxy package, but it requires quite a lot of tuning to achieve the kind of performance seen in commercial proxies. This article presents several independently useful ideas for tuning a web caching system. If all of them are implemented, extremely high performance can be achieved from modest hardware. Further, this article will describe, for illustrative purposes, a web cache that I recently built for prototype purposes. This box was tested before and after tuning using the Polygraph web cache benchmarking suite and performance climbed from about 50 reqs/second to over 100 reqs/second. A very large difference for no extra hardware expense. This article assumes that you already know how to install and configure Squid, but would like to achieve faster response times and be able to handle much heavier loads with your web cache.

I feel obligated to warn you that this process is not for the faint of heart, there will be patching, compiling, configuring, and dragons on this journey. But this is one area where putting in a day or two of hard work can pay off very big. But, if after reading through this, you find yourself saying, "Huh?", you might want to consider purchasing one already built (Swell Technology sells them) or pay a Linux nerd to build one for you using these instructions.

Getting the Most From Your Hardware

Squid can easily be crippled by disks that are not performing up to spec. Web caching is an "I/O bound" application, meaning that it can only go as fast as it gets the data onto and off of the storage media. So the first thing necessary is to ensure you have the right hardware for caching.

Hard Disks
You will need to use either 7200 RPM UDMA 66 drives or fast (7200 or 10k RPM) SCSI drives. A good UDMA 66 drive with the appropriate controller can perform nearly as well as a comparable SCSI drive for significantly less money. However, if you choose UDMA, keep in mind that the CPU will need to be slightly faster to make up for additional cycles used for I/O.

I've chosen for my system 2 Seagate Barracuda 13.6 GB 7200 RPM UDMA 66 hard disks. IBM and Maxtor also make excellent performing 7200 RPM drives. In my tests the Seagate edged out the Maxtor which edged out the IBM, each by a very small margin.

Processor
Squid in it's default configuration, as found in a default RPM or in a standard compile, does not need a very fast CPU to support 1 or 2 disks. However, a large performance boost can be achieved by using a threaded version of Squid. The threads will use a lot of CPU horsepower. So, if you are going to use threads as documented below, consider getting a rather fast processor. A 550MHz K6−2 is what I've chosen for my system. Something near that speed should be about right for a single or dual disk system.

RAM
It's pretty easy to calculate the amount of RAM needed for any given web cache. A safe number is 10MB RAM for every 1 GB of cache space on disk. I usually allot a little more than this when using a threaded Squid. Of course, you also need to figure out what your system uses for other things and subtract that from the total of available RAM. For my prototype unit I've chosen 256 MB.

Other Hardware
The other hardware is of very little importance. Any combination of high quality components should perform about the same in a web caching box.

Software Modifications

The Kernel
If you choose to use UDMA 66 drives, you will likely need to recompile your kernel with Andre Hedrick's Unified IDE patch, which provides drivers for the most common UDMA 66 controllers. If you're completely new to kernel patching and compiling it has been documented elsewhere (see *The Kernel HOWTO*). Hedrick's patches can be found at http://www.kernel.org/pub/linux/kernel/people/hedrick/. You will need to download the one for your kernel or download the latest kernel and the latest patch. I recommend using the newest of both, since UDMA support seems to improve every few days.

This patch can be added to your kernel simply by typing the following from your /usr/src/linux directory, assuming the ide patch is in the /usr/src directory.

```
patch -p1 < ../ide.2.2.16.all.20000825.patch
```

ReiserFS
Simply switching your cache directories to ReiserFS will increase squid performance by about 20%. ReiserFS is a journalling filesystem that patches easily into the recent 2.2 series kernels. ReiserFS is significantly faster than ext2 when dealing with thousands of small files. Since small files are pretty much all a web cache deals with, we're going to use ReiserFS on our cache partitions. You can find ReiserFS at http://devlinux.com/projects/reiserfs/. An added benefit of a journalled FS is that in the event of an abnormal shutdown your system will recover faster and without as many lost files as ext2.

Again, the patch is easy:

```
patch -p1 < ../linux-2.2.16-reiserfs-3.5.24-patch
```

Configuration
There are a few configuration details in the kernel you will need to change to make use of the new patches we've applied. Further, there are a few changes one can make to improve the performance of the Kernel for use in a server.

Here is a list of the important items to alter, you'll also need to turn on whichever UDMA chipset or SCSI controller you have:

```
CONFIG_EXPERIMENTAL=y CONFIG_MODULES=y CONFIG_MODVERSIONS=y
CONFIG_KMOD=y CONFIG_NET=y CONFIG_PCI=y CONFIG_BLK_DEV_IDE=y
CONFIG_BLK_DEV_IDEDISK=y CONFIG_IDEDMA_AUTO=y
CONFIG_IDEDMA_PCI_EXPERIMENTAL=y CONFIG_PACKET=y
CONFIG_NETLINK=y CONFIG_RTNETLINK=y CONFIG_FIREWALL=y
CONFIG_UNIX=y CONFIG_INET=y CONFIG_IP_FIREWALL=y
CONFIG_IP_ROUTER=y CONFIG_IP_ALIAS=y CONFIG_REISERFS_FS=y
```

This isn't a complete kernel config, obviously. You will also need network hardware drivers and will need to make a few other decisions when deciding what the box will need to do. I've already included in this list all

that is needed for a transparent proxy if that is the route you plan to take.

Note for users of older versions of ReiserFS: CONFIG_YRH_HASH − This chooses the Yuri Rupasov Hash option in ReiserFS. It is a slightly more effective hash type for squid. New versions of ReiserFS use the −h option of mkreiserfs for this.

Now simply compile and install your new kernel, run lilo, and reboot.

After rebooting, you can build the ReiserFS tools found in the /usr/src/linux/fs/reiserfs/utils directory. Type the following:

```
mkdir bin; make; make install
```

Now you can convert your squid partitions to ReiserFS like so (X is your planned cache partition), note that the −h option allows you to specify the hash type in new ReiserFS versions:

```
/sbin/mkreiserfs −h r5 /dev/hdaX
```

Add the following line to your /etc/fstab:

```
/dev/hdaX /cache reiserfs notail,noatime 0 0
```

You can then mount your new cache directory:

```
mount /cache
```

Note: The notail option above tells ReiserFS not to pack tails, which is a way for the FS to save file space. Since we need speed much more than a few megabytes of saved space, we'll disable it. And the noatime option should be used regardless of what filesystem you use, for a web cache. It disables access time updates on files, saving one write for every read.

Recompiling Squid

I've chosen to use a somewhat older version of Squid, version 2.2.STABLE5, plus the latest patch snapshot from Henrik Nordstrom. There are two reasons for this. First, squid−2.3.STABLE4 is not as stable as 2.2.STABLE5+Henrik's patches. Second, squid−2.3.STABLE4 is not as fast as squid−2.2.STABLE5+Henrik's patches. I know it seems counter−intuitive, but it's true. You can spend a few weeks running Polygraph and webbench loads on both and prove it to yourself, or you can take my word for it. Unecessary and probably fruitless rant: There will probably evetually come a time when there is a faster and more stable Squid, but even the latest DEVEL snapshot of 2.4 as of yesterday afternoon (08/29/2000) is nowhere near as fast as the version used here. I've gotten a lot of emails from folks who insist on using 2.3 or 2.4 and they just aren't seeing the same kind of performance I'm talking about here and they want me to tell them how to make their boxes stop crashing under load. I'd love for 2.4 to be faster too, but it just isn't yet. We've all got to live with it for now.

So you'll need Henrik's patch snapshot. This can be found at this address:

http://squid.sourceforge.net/hno/

Applying this patch is easy:

```
patch -p1 < ../squid-2.2.STABLE5-hno.20000202.snapshot
```

One more thing you need to do before compiling squid is raise the number of file descriptors on your system. This requires two steps in recent 2.2.x series kernels. First you need to edit this line in the /usr/include/bits/types.h file:

```
#define __FD_SETSIZE 8192
```

It is 1024 by default. 1024 is not quite enough for even a single disk cache after all of these optimizations.

Next you need to raise the limit in the shell so the configure script will see the added descriptors:

```
ulimit -HSn 8192
```

This raises the hard and soft limits to 8192. You will need to be root to do this.

In a high load environment, squid wants to be compiled with the async-io configure option enabled. Further, you will want to increase the number of threads Squid is given, the default is 16. This number depends heavily on the speed of your processor and the number of disks. For a 550 MHz CPU and two disks, I use 30 threads. Here is the configure script I use (which is actually part of my RPM spec file):

```
CFLAGS="-DNUMTHREADS=30" ./configure --prefix=/usr \
--exec_prefix=/usr --bindir=/usr/sbin
--libexecdir=/usr/lib/squid \ --localstatedir=/var
--sysconfdir=/etc/squid --enable-async-io \ --enable-snmp
--enable-poll
```

This will compile a squid that is very close to what the Red Hat SRPM creates, however, if you use Red Hat, I would suggest you edit the official Red Hat SRPM or download one of my SRPMs from this address:

http://www.swelltech.com/pengies/joe

There are a couple of patches to the makefiles and config files that are needed if you want a really Red Hat compliant squid. Also, you may want to compile with optimizations for your compiler, as it can make about a 5% difference in performance.

Finally, it's time to make and install. This step is just like the Squid directions say. Or you can rpm -Uvh it if you built from the SRPM.

Configuring Squid
Now after all of these things, you will have a Squid that is capable of handling a much higher load than the default compile. But, there are a few more things we need to do before we have a web cache running full tilt. Thankfully there will be no more patching or compiling.

First thing to do is configure squids squid.conf file.

If you used the RPM or the configure line above, you're squid.conf will be located in /etc/squid.

The important configuration options for performance are below:

cache_mem
This is one of the top two options as far as performance impact goes. The default value here is 8 MB, which is a safe option for just about any system size. But since your box is not lacking in memory it can safely be raised to 32 MB, and even 48 if you have stripped all other services off of your server. If your box is lacking in memory, go back to the hardware section and reread the part about memory...you need a bunch of memory for a fast web cache. You should be aware that cache_mem does not limit the process size of squid. This sets how much memory squid is allowed to set aside for "hot objects" which are the most recently used objects. Having said all that, keep in mind that the buffer cache in Linux is also quite good, so the gains you'll see by raising this are very small, but still measurable.

cache_dir
This is where you set the directories you will be using. You should have already mkreiserfs'd your cache directory partitions, so you'll have an easy time deciding the values here. First, you will want to use about 80% or less of each cache directory for the web cache. If you use any more than that you will begin to see a slight degradation in performance. Remember that cache size is not as important as cache speed, since for maximum effectiveness your cache needs only store about a weeks worth of traffic. You'll also need to define the number of directories and subdirectories. The formula for deciding that is this:

```
x=Size of cache dir in KB (i.e. 6GB=~6,000,000KB) y=Average
object size (just use 13KB z=Number of directories per first
level directory (((x / y) / 256) / 256) * 2 = # of directories
As an example, I use 6GB of each of my 13GB drives, so:
6,000,000 / 13 = 461538.5 / 256 = 1802.9 / 256 = 7 * 2 = 14 So
my cache_dir line would look like this: cache_dir 6000 14 256
```

Those are really the two important ones. Everything else is just picking nits. You may wish to turn off the store log since there isn't much one can do with it anyway:

cache_store_log none
If, after all of this you still find your squid being overloaded at times, try setting the max_open_disk_fds to some low number. For a single disk cache, 90 is a good choice. For a dual disk cache, try 140. To get the perfect number for your cache you'll need to keep an eye on the info page from the cache manager during a particularly heavy load period. Watch the file descriptor usage, and then set this number about 10% below it. This can help your cache ride out the heaviest loads without getting too bogged down. That being said, I've also found that this can cause other problems...so use at your own risk.

Also, the following may be set to improve performance marginally:

half_closed_clients off

maximum_object_size 1024 KB

cache_swap_high 100%

cache_swap_low 80%

Ok, that's all for squid configuration. Now there's one last thing to do to complete the picture. You need to edit your squid startup script to include the following three lines:

```
ulimit −HSn 8192 echo 1024 32768 >
/proc/sys/net/ipv4/ip_local_port_range echo 1024 >
/proc/sys/net/ipv4/tcp_max_syn_backlog
```

This will raise the file descriptor limit for the squid process, and will open up a ton of IP ports for use by squid. It won't need quite this many, but better safe than sorry. The final line raises the maximum syn backlog to a nice high number so our Squid will never be starved for requests, due to the kernel limiting network connections.

That's it! You now have the fastest squid on your block.

Links of interest

Home of Squid

Henrik Nordstrom, Squid hacker extraordinaire, lives here.

All of my experimental Squid SRPMS and binaries are here along with scripts and tools.

The ReiserFS homepage.

Andre Hedrick's home, source of IDE patches.