KNBD - A Remote Kernel Block Server for Linux

Jeff Becker
MRJ Technology Solutions
NASA Ames Research Center

I am developing a prototype of a Linux remote disk block server whose purpose is to serve as a lower level component of a parallel file system. Parallel file systems are an important component of high performance supercomputers and clusters. Although supercomputer vendors such as SGI and IBM have their own custom solutions, there has been a void and hence a demand for such a system on Beowulf-type PC Clusters. Recently, the Parallel Virtual File System (PVFS) project at Clemson University has begun to address this need (1). Although their system provides much of the functionality of (and indeed was inspired by) the equivalent file systems in the commercial supercomputer market, their system is all in user-space. Migrating their IO services to the kernel could provide a performance boost, by obviating the need for expensive system calls.

Thanks to Pavel Machek, the Linux kernel has provided the network block device (2) with kernels 2.1.101 and later. You can configure this block device to redirect reads and writes to a remote machine's disk. This can be used as a building block for constructing a striped file system across several nodes. Figure 1 shows its structure and operation.

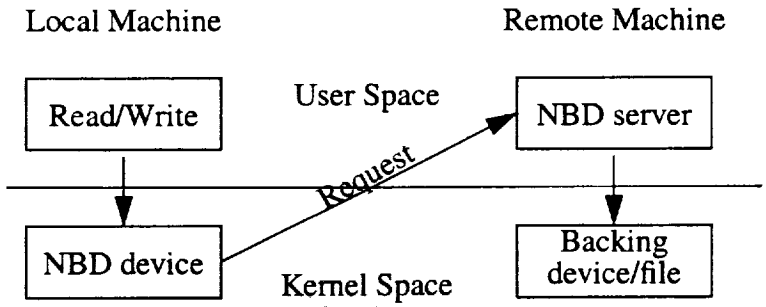

FIGURE 1. nbd structure

Reads and writes to the device are redirected through a TCP socket to a user-land server. The server then issues the request (through a system call) on a backing device or file and returns any results to the nbd driver through the socket. The driver passes these back to the caller. The connection is set up as follows. First, the server is started on the remote machine with the port to listen for connections given as a command line parameter. The backing device or file is also specified on the command line. A user-land client is then started on the local machine, with the remote machine name and port given as command line parameters. The client creates the socket to the server, and then hands it to the nbd driver through an ioctl(). Once the connection is initialized, you can use nbd like any other local block device, e.g., make a file system on it, use it as part of a RAID configuration, etc. A primary performance- related shortfall is that the server is in user-

space. Thus every request from nbd must cross from the local kernel to user-space on the remote machine. The server then makes an expensive system call to access the backing device, and the path (and kernel/user crossings) are reversed on the return to the local machine. This shortfall is eliminated by moving the server to the kernel. This is the motivation for knbd whose structure is shown in figure 2.

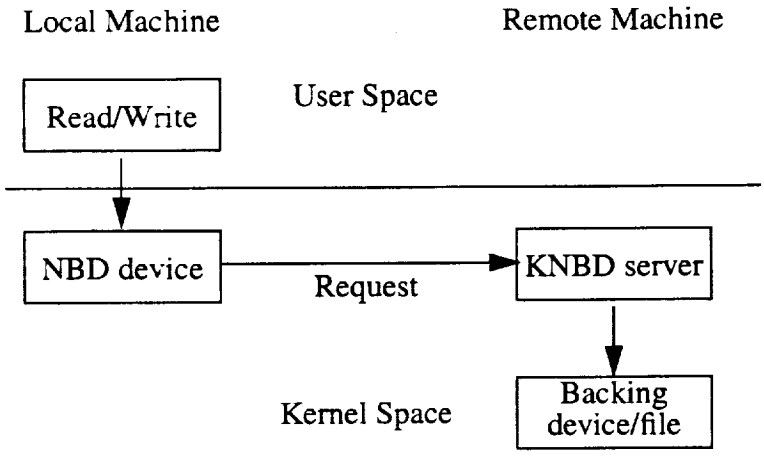

FIGURE 2. knbd structure

In order to simplify development and testing, the kernel nbd server is designed as a kernel module. It is also designed to be used with the standard nbd device and client. Module initialization takes a port and a backing device as parameters using the Linux MODULE_PARM macros. The module_init routine sets up a TCP socket and starts a kernel thread to listen for a connection from the nbd client. The kernel thread basically follows the structure of the user-space nbd server. It does an initial handshake, e.g., to tell the client the size of the backing device/file, allocates a local knbd buffer, then enters a loop to wait for requests. Upon receiving a request, the server thread executes the following pseudo-code:

```
decode and check request (address, length, type (read or write))
if write, get data from client nbd and put it in the local knbd buffer
while (length > 0)
      get a buffer head (using getblk())
      if (type == read)
            fill in the buffer from the buffer cache (ll_rw_block();
            wait_on_buffer (buffer head);)
            copy data from buffer to the local knbd buffer
      else /* type == write */
            copy data from local knbd buffer to buffer cache buffer
            mark buffer cache buffer dirty (so bdflush moves it to the
            backing device/file)
      subtract amount read or written from length
      release the buffer head (with brelse())
send status of request back to client nbd
```

I've tested the server on i386 platforms running Red Hat Linux 6.0/Linux kernel 2.3.18. I first configured the module to be backed by the linux loop device (backed by a large file). I then started

the client on another machine. After verifying the initial socket setup and handshake worked correctly, I ran mkfs on the client nbd device to make an ext2 file system. I then verified that I could successfully mount the nbd device and perform miscellaneous file operations on it (such as cp, ls, cat, vi etc.).

I also ran the bonnie I/O benchmark to compare the userland nbd server to knbd on the above configuration. The knbd server outperformed the userland server on both block reads (1054 K/sec vs 1044 K/sec) and block writes (3888 K/sec vs 2561 K/sec). The latter is about a 50% increase in performance!

As a more strenuous test, I loaded the block server module onto a third machine, and started a second client (on /dev/nbd1) on the machine running the first client (on /dev/nbd0). I then set up a RAID0 (striped) device (/dev/md0) across nbd0 and nbd1. I was then able to mkfs on /dev/md0 and mount it. This was followed by successfully untar'ing and building a large software source base (for the fte editor).

At this time, I have submitted the remote block server code to several people for testing and review. I have already received one favorable reply from Marcelo Tosatti. In addition to performing successful tests of my code, he has also kindly provided a patch to build and run the module on Linux 2.2. While I await other responses, I am pursuing a port of nbd to NetBSD. This operating system does not provide a network block device, so this will have to be ported along with the kernel block server. The user-land client should port over without much modification. In a related activity, I plan to evaluate the Slice Block IO server for FreeBSD, part of the Trapeze project at Duke University (3). It seems to provide similar functionality, and may be useful in simplifying the porting effort. However, Slice is not currently available in either source or binary form.

An additional future direction is to extend the Linux knbd server to provide multithreading, i.e., to fork off a new kernel thread for each unique client. I would also like to pursue reordering of events in order to boost performance (suggested by Pavel Machek).

References.

1. http://ece.clemson.edu/parl/pvfs
2. http://atrey.karlin.mff.cuni.cz/~pavel/nbd/nbd.html
3. http://www.cs.duke.edu/ari/trapeze/