What HPC Networking Requires from the Linux Kernel

Brice Goglin

What HPC Networking Requires from the Linux Kernel

By Brice Goglin, Ph.D., Myricom Inc.

Archive of the article that was initially published at http://www.hpcwire.com/hpc/811570.html as part of HPCwire's Daily Coverage From LinuxWorld Conference & Expo San Francisco - 2006.

High performance computing is a realm in which Linux dominates over the other operating systems. In the Top500 list released in June, almost three supercomputer sites out of four run Linux. Drivers and middleware for high-end HPC cluster hardware, typically for specialized high performance interconnects such as Myrinet, InfiniBand or QsNet, are generally more widely available, stable, fully featured and efficient on Linux than on any other OS.

It is noteworthy that in the Top500 list, particularly toward the top of the list, a large portion of the Linux clusters are using specialty (non-Ethernet) networks with communication software stacks specialized for HPC applications. The other Linux clusters, which today tend to be toward the bottom of the list, rely instead on a traditional Sockets-API/IP/Ethernet combination. These two different kinds of Linux clusters are representative of a history almost as old as the first Linux clusters.

The first association between HPC and Linux -- the first class of "Beowulf" clusters -- has been based on a "COTS" (Commercial Off The Shelf) approach on both the hardware side (standard Ethernet cards) and the software side (the traditional Sockets-API/IP-stack/Ethernet-driver combination). This is the approach on which the core Linux kernel community has been focusing on almost exclusively.

During this same period, a number of people outside the core Linux kernel community have been (ab)using and modifying Linux in non-standard ways to provide more optimized HPC networking solutions with specialty networks and network stacks. The goal of high bandwidth and low latency in communications between computing nodes, as well as low host-CPU overhead, required optimizations all the way from the application program's buffers to the network port. In this class of cluster, the application typically talks directly to the networking hardware (kernel bypass, a.k.a. OS bypass), and the hardware is able to DMA data directly to and from any parts of a user-space process (zero-copy techniques).

There would be significant benefits for all parties to work on adding a better framework for the specialized HPC networks to the base Linux code. These changes to Linux would make the corresponding network stacks more efficient and much easier to maintain. Now might be a good time for the community to take on this task, inasmuch as a convergence between HPC networking and traditional networking is starting to take place.

The most important changes needed in the Linux kernel are not necessarily major, but are at a central part of the kernel: memory management. In addition to those features, there are some other problem areas, such the x86/x86_64 support for Write-Combining, which is critical for efficient Programmed Input/Output (PIO) and support for Message Signaled Interrupts (MSI). These capabilities still lack good support in Linux even though the hardware features have been widespread for a number of years. HPC network users and vendors are often the early adopters of such features, and the first ones to feel the need for some improvements. As 10 Gigabit Ethernet is gaining popularity, it also benefits from some of those kernel improvements, and hopefully there is enough critical mass with 10 Gigabit Ethernet to improve the situation faster.

The kernel community advocates that all drivers/modules code should go into the standard kernel code base. However, the changes envisioned in this article would probably be mostly used by externally maintained modules. In some cases, having a specialized HPC networking stack inside the standard Linux code base, as opposed to just some framework support, may be counter-productive. Most people running production clusters generally do not want to upgrade to the latest "development" kernel on their thousands of nodes. They want to use a proven, stable kernel. This fact of life is why HPC vendors still have to support the old kernels (Linux 2.4.x) that some machines have been running for one or two years, while the vendors also need to support more recent kernels (2.6.17) and their various distribution flavors (Red Hat and SuSE, for example, both ship heavily patched kernels).

It's not that supporting external modules is easy. Doing so while the internals of the Linux kernel differ significantly across all these releases already makes the work much harder than one might think. The code requires some tweaks and architecture-specific hacks that make the code less readable. In fact, it makes the code downright ugly by the high standards of Linux developers. But the bottom line is that users require compatibility and these optimizations for legitimate reasons. Having an extra "clean" variant of the code inside the kernel just adds an additional burden; it is mostly useless since the code is no longer recommended by the time it reaches users (at least a few months for master kernel releases, up to more than one year for distribution kernels). A common situation for such complex, fast changing, in-kernel subsystems (such as InfiniBand) is that most users replace them with an external up-to-date version.

Let us go into more detail on the Linux kernel additions that would most significantly help high performance interconnect modules: Linux support to allow the HPC stack to easily manage asynchronous zero-copy (DMA from user space) transfers. Indeed, for years now the better network interfaces that are used in clusters have been smart enough to allow direct PIO or DMA transfer between the hardware device and user space in a secure way, involving the kernel only for setup purposes. Their software stacks have been designed to avoid any memory copies between the application buffers and the network. This approach enables low-overhead communications because the host processor has very little work to do compared with what's involved in a standard TCP/IP stack. However, zero-copy operation actually hides a very high software overhead in the initialization phases, called memory registration, because it requires pinning-down in physical memory the portion of the user space that might need to be accessed via DMA.

Some well-known caching strategies are now widely used in HPC to reduce the visible overhead of memory registration; an alternative solution is to duplicate the page table of the process in the NIC. The caching strategies are very efficient for most applications, but have some strong technical constraints in real implementations. In particular, they require one to trace the changes in the application address space to keep the cache up-to-date. Most of the time, this scheme has been implemented in the user-space library or middleware by intercepting those system calls from the application that may change the address-space layout. However, the way in which the application actually invokes these calls may vary so much, due to user-space developer freedom, that the interception ends up being difficult to implement properly in the general case.

Having kernel support for this interception would make things much easier. A few hundred lines of kernel code to configure notification hooks would definitely solve all the problems and make the memory registration cheap. Quadrics has been using such a strategy for years, but it requires their customers to patch their kernel. It would be much better to get support in standard un-patched ("vanilla") kernels, but it would first require that HPC interconnect vendors agree on what support they actually need and design the new in-kernel interface for these notification hooks.

In fact, the emergence of 10 Gigabit Ethernet hardware might help HPC vendors get this additional support into Linux. In order to handle the jump in bandwidth coming with 10 Gigabit Ethernet, several new designs for the Linux networking stack have been proposed. The latest one, named "asynchronous and zero-copy communications," might appear in the near future to handle TCP/IP traffic while overcoming the performance limitations of the current Sockets-based API. It might require generic memory management additions quite similar to what would benefit HPC networking stacks; thus, it might help bring people who are working on specialized HPC stacks and people who are working on traditional TCP/IP networking closer together.

However, for the foreseeable future this possible Linux "asynchronous and zero-copy communications" extension for TCP/IP won't alleviate the need for HPC communications stacks targeting applications written on top of the standard MPI (Message Passing Interface) API. Indeed, the TCP/IP zero-copy features currently being discussed are limited to applications using communication buffers in a restricted way, in which the application can receive data into arbitrary parts of its address space; this does not fit the MPI model very well. It is quite unlikely that MPI will be replaced anytime soon in the HPC world as the dominant communication API. Thus, as long as MPI continues to be used, HPC-specific communication stacks providing MPI will have to deal transparently with zero-copy and with other features possible only with specialized HPC hardware. In any case, a base framework to ease these tasks in the Linux kernel is still needed.

To conclude, it should be emphasized that HPC networking stacks development is still active and evolving. However, managing zero-copy communication schemes is now sufficiently mature to add a common base infrastructure to the Linux base kernel to handle those schemes. Another motivation for this addition is the likely convergence (maybe even blurring) between the solutions used for traditional TCP/IP networking and the solutions used for HPC MPI applications. On the hardware side, we expect 10 Gigabit Ethernet to fit both TCP/IP and HPC MPI requirements with ease. On the software side, we expect the TCP/IP stack to use buffer and memory-management schemes similar to what has been used in HPC for a long time.

The author would like to thank Dr. Loic Prylli, Myricom, for helping to write this article.

Brice Goglin earned his Ph.D. at the Ecole Normale Superieure de Lyon in 2005. His research interests include high-speed networking and operating systems. Brice started working for Myricom as a summer intern during his graduate studies, and became full-time upon completion of his Ph.D., working on the development of the Myri-10G software stacks. Brice will return to France this fall to do research at INRIA.