SEE TEXT ONLY

FreeBSD IN SCIENTIFIC COMPUTING

By Jason **Bacon**

I've been running both FreeBSD and Linux uninterrupted since the mid-1990s. I've run many different Linux distributions over the years, most recently focusing on CentOS. I also have significant experience with Mac OS X and NetBSD and have experimented with many other BSD platforms. As a staunch agnostic with a firm belief in the value of open standards, I like to remain familiar with all the options in the POSIX world so I'm always prepared to choose the best tool for the job.

Over the years, I've watched barriers to FreeBSD use fall one by one. Open-source software continues to spread like the Blob, filling in niches once held exclusively by commercial and other closed-source software. OpenOffice/LibreOffice, OpenJDK, Clang, Flang, and many other high-quality open-source products have made it possible for most people to do everything they need on FreeBSD with ease. The few remaining limitations of what can be run on FreeBSD have become increasingly esoteric and they, too, seem destined to disappear in time.

My need to run MS Windows for personal use came to an end more than a decade ago, although I still support it to some extent for work. Mac OS X has filled the few remaining needs that had been served by Windows, but even my Mac has now been reduced to a once-per-year platform for running tax software. I could, of course, use FreeBSD for this as well with a web-based tax program, if not for the reality of computer security in the cloud. The bottom line is that I now find FreeBSD easier and more pleasant to use than commercial operating systems for most of my work and personal computing.

Most of my professional work since late 1999 has been in support of scientific computing. From 1999 until 2008, I supported fMRI brain-mapping research for a multidisciplinary group including neurologists, neuropsychologists, cell biologists, psychiatrists, and biophysicists. During most of this time, I was the sole IT support person for several labs, peaking at over 60 researchers. I was responsible for maintaining many Unix workstations as well as managing all the research software needed for fMRI analysis. The need for a full-featured and extremely reliable operating system became very clear, very quickly. FreeBSD answered that call and made it possible for me to single-handedly keep this important research moving forward for many years.

Since 2009, I have been supporting a wide range of research, including engineering, bioinformatics, physics, math, chemistry, public health, business, and psychology. A major part of this environment has been our HPC clusters running CentOS Linux. FreeBSD has also played an important role as a development and testing platform, and the primary OS on our educational HPC cluster and HTCondor grid. It has also provided the model for how we manage most open-source software on our CentOS systems, using pkgsrc, a cross-platform package manager from the NetBSD project, originally derived from FreeBSD ports.

Already an Important Part of HPC

FreeBSD already plays crucial roles in research, including many high-performance computing (HPC) environments, although some may not rec-

ognize it by name. FreeBSD is the foundation of some of the most popular high-performance storage appliances such as FreeNAS, Isilon, NetApp, and Panasas. FreeBSD is also an important component of Juniper network switches, pfSense firewalls, Apple's OS X, and runs servers for many web-hosting and cloud services. FreeBSD enjoys strong support from other vendors as well, including Mellanox, which is currently developing FreeBSD drivers for their network adapters.

Outside scientific research, FreeBSD is used by some of the biggest players in the business, who need to maximize performance and reliability. Ever watch a movie on Netflix? If so, you're a FreeBSD user. The Netflix servers streaming movies to your TV or computer run FreeBSD. FreeBSD has also been used for Yahoo! servers for decades. For a current list of notable companies using FreeBSD, see the FreeBSD Handbook (https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/nutshell.html#introduction-nutshell-users).

FreeBSD as a Computing Platform

Scientific computing has reached a sort of utopia in recent years due to fast, cheap hardware and full-featured, free software. For a typical workstation or laptop, it's hard for a scientist to go wrong. There are multiple free operating systems to choose from, mostly based on BSD, Linux, and Solaris. Any one of them will serve the needs of a typical scientist quite well. There are POSIX compatibility layers for MS Windows such as Cygwin and Windows Subsystem for Linux (WSL), which facilitate running Unix software directly on Windows, and free virtual machines, such as VirtualBox and qemu, that allow us to run multiple operating systems at the same time on the same machine.

Unsurpassed Reliability

But what about those of us who manage many multi-user servers for research? There has always been, and always will be, a severe shortage of skilled systems managers. Nowhere is this shortage felt more painfully than in scientific research. The law of supply and demand dictates that experienced systems managers are expensive. Scientific researchers make their living by repeatedly begging for grant money to keep their labs running. With few exceptions, most researchers have too little funding and cannot hope to compete with wealthy corporations for the limited IT talent pool. In fact, most principle investigators in academic research earn far less than an experienced Unix sysadmin.

In this scenario, choosing a system that minimizes IT man-hours is crucial. We must enable a small IT staff to be as productive as possible, by avoiding system outages and performing common tasks such as software deployment as quickly and cleanly as possible. This is where FreeBSD stands out among its peers. FreeBSD's unsurpassed reliability and ease of management save precious IT man-hours that could be used for more creative endeavors. It can literally reduce the need for man-hours by an order of magnitude over the methods often used by inexperienced sysadmins in the research community.

Experienced systems managers in this situation know to stay away from bleeding-edge operating systems, which are likely to bring repeated surprises that will distract them from more creative work. For this reason, the vast majority of large HPC clusters run Enterprise Linux rather than other distributions with the latest kernel, compilers, and other system software. This is not a criticism of bleeding-edge platforms. In fact, it's important to all of us that many typical users use them and work the bugs out of the latest new kernel features, compilers, etc. Today's bleeding-edge is tomorrow's Enterprise.

Enterprise Linux systems achieve a high level of reliability and long-term binary compatibility by taking a snapshot of a recent bleeding-edge system and declaring a moratorium on major upgrades. For example, Redhat Enterprise and CentOS are based on a snapshot of Fedora. The down side of this approach is that the tools and libraries that ship with it are outdated and unable to support the latest open-source scientific software. For example, Redhat Enterprise 7, the latest release as of this writing, ships with GCC 4.8.5 (June 2015) and GNU libc 2.17. The latest releases are GCC 8.1 and libc 2.27. The only way to build the latest open source on such a system is by installing newer core tools and libraries. Replacing them outright would sacrifice the stability and binary compatibility with commercial software for which Enterprise Linux is designed. The solution we employ on our CentOS systems is to leave all these core tools in-place and use pkgsrc, a cross-platform package manager, to install newer tools alongside them. Many others have resorted to using containers or virtual machines to provide a more modern environment, isolated from the outdated Enterprise base.

FreeBSD allows us to avoid these issues by offering stability matching or exceeding that of Enterprise Linux while providing more modern tools in the base. As of this writing, FreeBSD 11.2 and the upcoming FreeBSD 12 release both provide clang 6 (March, 2018) as a base compiler and easily support most current open-source scientific software.

Minimize Management Time

A FreeBSD system can be installed from scratch in about 5 minutes and fully configured for many

purposes in less than an hour, thanks to well-designed tools in the base system and the FreeBSD ports system for managing add-on packages (more on this later). The FreeBSD handbook is a well-written and excellent tutorial for most common tasks and is kept fairly up-to-date. No need to search the web and risk following outdated or erroneous instructions.

Ongoing maintenance is quick and easy using freebsd-update, a binary update system for installing bug fixes and security patches in the base system. Unlike many other operating systems, FreeBSD security notices include clear instructions on how to apply patches, including when a reboot or service restart is required.

Software Management with FreeBSD Ports

To go from idea to a published paper in scientific computing involves four steps:

1. Develop the software
2. Deploy the software
3. Learn the software
4. Run the software

Steps 1 and 3 are mostly platform-independent. Unfortunately, step 2, software deployment, is often one of the major bottlenecks in scientific computing. Step 4, running the software, is a lesser, but real bottleneck requiring the ability to deploy an optimized build of the software. FreeBSD ports has the potential to easily eliminate these bottlenecks, as it allows the user to easily install from a huge collection of binary packages and just as easily install any package from source with additional compiler options. FreeBSD also has strong support for OpenMP, pthreads, and MPI, for cases where parallel computing is needed to reduce run times.

The vast majority of scientific software is open source, developed on a variety of (usually bleeding-edge) platforms, and often deployed via very primitive methods. Many developers don't target package managers at all, but instead provide precompiled binaries for their own development platform and maybe a few others, alongside cryptic instructions for performing a "caveman" install, manually building from source after installing dependencies, or worse, using their bundled dependency software. The chances of their build-from-source instructions working for the average user are almost nil.

Part of the problem is that the developers are mostly scientists with little or no computer science training, very often self-taught graduate students developing software for their dissertation. They don't know much about sustainable development and systems management practices. If their software lives beyond their graduation date, it will likely get cleaned up so that it's more portable and easier to deploy. However, given the constant, rapid progress of science and the rotating door of student-developers, the research computing community is basically doomed forever to a world full of nascent, disorganized code.

The FreeBSD ports system can alleviate this problem in two ways:

FreeBSD ports has one of the largest collections of existing packages of any package manager. At the time of this writing, FreeBSD users can install any of more than 32,000 packages with one simple command. If the package you need is not already in the collection, chances are that most or all the dependencies are there, so the effort needed to create a new FreeBSD port is often a small fraction of that required to do a caveman install. Note also that FreeBSD's port options substantially increase the number of possible software installations. This has to be considered when comparing the ~32,000 FreeBSD ports against the number of binary packages in systems that do not support convenient builds from source. Many of the binary packages in such systems are merely different builds of the same software.

Imagine a scenario where instead of thousands of scientists each wasting forty hours struggling with the same caveman installation, one of them creates a FreeBSD port and everyone else in the world from that day on can install the software in seconds. That's many thousands of man-hours redirected from senseless, duplicated IT effort to productive scientific exploration. This is the potential of FreeBSD ports and other package managers.

There is another important difference between binary package managers, which quickly install precompiled packages along with dependencies, and package managers that make it convenient to build from source, such as FreeBSD ports, Gentoo Portage, MacPorts, and pkgsrc. Binary packages install much faster, of course, but they may suffer from compatibility, security, and performance problems. To be portable, they must be compiled with static libraries and limited to common CPU features, which precludes utilizing new instructions and other CPU features that may significantly improve performance. In extreme cases, you may see a 30% improvement in speed from a non-portable binary utilizing all available CPU features. This can save thousands of core-hours on an HPC cluster running a large analysis.

With FreeBSD ports, building an optimized binary from the source code, utilizing the best features of your CPU, is as easy as installing the binary package. It will take longer for the computer to build and install, of course, but the effort for you is about the same. To install a portable (possibly slow) binary package, we might use the following:

```
pkg install canu
```

To build an optimized version from source, we would adjust our build settings in `/etc/make.conf` (e.g. by adding CXXFLAGS+=-march=native), and run the following:

```
cd /usr/ports/biology/canu
make install
```

If a FreeBSD port does not already exist for the software you need, consider creating one. It's not as difficult as one might assume. There is a rather steep learning curve to becoming a FreeBSD ports committer, who can add ports to the system only after extensive quality control measures.

However, ports need not be committed before you can use the ports system to deploy them. In fact, all ports are deployed and tested before being committed. The learning curve for creating a basically functional port or upgrading an existing port is fairly small. Anyone who knows how to write a Makefile and use the build system employed by the upstream developers can learn to do this fairly quickly. The Porter's Handbook (https://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/) covers most of what you would need to know, starting from the beginner level all the way to becoming a FreeBSD committer.

If you install binary packages from the FreeBSD ports system, they can quickly and easily be updated using the following command:

```
pkg upgrade
```

(Note that this may replace your optimized from-source install with a newer binary package, so watch for this and rebuild the port from source after the pkg upgrade if necessary.)

The port frameworks used to build from source can also be easily updated using portsnap or svn. This is a great feature for those who want to keep their systems running the latest of everything. But what if you need to keep the same version of a program running through a long-term study spanning several months or even years? Running **pkg upgrade** could effectively break your study.

It is possible, though not well-tested at this stage, to deploy multiple ports trees under different prefixes. Ports can also be installed this way without root privileges. The FreeBSD ports project branches snapshots every three months under different prefixes. The ports in these quarterly snapshots are never upgraded, although they may receive bug and security patches.

I have been experimenting with deploying quarterly snapshots under prefixes such as `/sharedapps/ports-2018Q1`, with corresponding installation to `/sharedapps/local-2018Q1`. Software can be installed statically here and never upgraded, while other software installed to the standard prefix is upgraded regularly with **pkg upgrade**.

This system follows a better-supported feature of the pkgsrc package manager, which is designed to be bootstrapped on any POSIX platform under any prefix the user desires. We have been using pkgsrc this way on our CentOS systems for years.

We could also use pkgsrc this way on FreeBSD, but there is a strong motivation to use the FreeBSD ports in a similar fashion, mainly because it has a larger collection than pkgsrc at this time. Some work remains to be done to utilize FreeBSD ports this way, but the basic support for this sort of deployment already exists. We just need to put it into use and iron out the wrinkles.

Base Features Beneficial to Science

There is a strong interest in the advanced ZFS filesystem in the scientific community. Its performance, flexibility, and data protection features are very attractive to users who invest immense amounts of time and money generating files containing their research results.

FreeBSD's RootOnZFS features allow us to deploy a FreeBSD installation booting from a ZFS filesystem using a simple menu interface in the installer. Any modern PC with multiple disks can be up and running with FreeBSD on a RAIDZ array in a matter of minutes.

ZFS is not suited for every purpose, however. ZFS is rather memory-hungry. On an HPC compute node, where local disks are used only to house the operating system and provide temporary storage, we may not want ZFS competing for memory resources with the computational processes. The same reasoning would apply to any other machine devoted mainly to CPU- or memory-bound tasks. Fortunately, FreeBSD's UFS2 filesystem also provides solid performance and reliability, with a very low memory footprint.

FreeBSD provides enterprise reliability and easy management, but what about that other big advantage of Enterprise Linux, support for commercial software products? Fortunately, FreeBSD's Linux compatibility module allows us to run most Linux binaries, *with no performance penalty,* trivial memory overhead, and a very modest amount of disk space and effort. FreeBSD's Linux compatibility is often erroneously referred to as *emulation* or a compatibility *layer*.

In reality, it is neither. The basis of FreeBSD's Linux compatibility is a kernel module that *directly* supports Linux system calls. The module activates an alternative function pointer table to directly invoke Linux-compatible kernel functions when a Linux binary is being run.

To complete the Linux-compatible environment, we simply need to install the Linux versions of any shared libraries and tools required by Linux programs, the same as we would on a real Linux system. The FreeBSD ports system provides tools for easily installing RPMs used by the RHEL/CentOS Yum package manager. Creating a FreeBSD port that installs software from the CentOS Yum repositories is trivial in most cases, and many such ports you might need already exist.

FreeBSD's Linux compatibility is fairly robust and capable of running the vast majority of commercial Linux binaries. I have personally run many versions of Linux Matlab on FreeBSD machines, with full functionality, including the Java desktop and MEX compilation system (using Linux GCC compilers).

At this point, though, I would advise most FreeBSD users to run Octave, a full-featured, open-source Matlab-compatible suite. It's virtually identical to Matlab for most typical users, it's free, and can be installed in seconds.

There is, of course, some additional work required to run Linux binaries on FreeBSD. Running something as complex as Matlab or ANSYS may require a fair amount of effort. Simpler applications such as shared-memory LS-DYNA are trivial to install. Linux binaries depending on MPI (Message Passing Interface) for distributed parallel computing could also be a challenge. If you rely heavily on complex closed-source Linux applications, you may be better off running an Enterprise Linux system.

In order for FreeBSD to become a competitive platform in HPC, it would also need to more easily support a few other subsystems that currently only work well on Linux, such as nVidia's CUDA GPU platform (although the open standard OpenCL is beginning to gain traction) and Infiniband interconnects that are commonly used for distributed parallel computing.

There are a few remaining features that are likely to ensure the dominance of Enterprise Linux in HPC for a while.

As it stands, though, FreeBSD is already an excellent platform for the needs of most typical scientific computing. If you mainly run open source software and prefer to spend your time doing science rather than IT maintenance, FreeBSD will serve you well. ●

The lead sysadmin for research computing at the University of Wisconsin-Milwaukee, Jason Bacon has been working with computers since 1983, and with FreeBSD and Linux since 1995. He is the author of *The C/Unix Programmer's Guide*, a comprehensive guide for beginning to intermediate C/C++ programming under UNIX, Linux, Macintosh OS X, and similar systems. He is proficient in Spanish and German, with a working knowledge of French and Mandarin. When he isn't inside working on various systems, he can often be found outside, cycling, sea kayaking, cross-country skiing, hiking, and scuba diving.