Editors: Paul F. Dubois, paul@pfdubois.com
George K. Thiruvathukal, gkt@cs.luc.edu

GENTOO LINUX: THE NEXT GENERATION OF LINUX

By George K. Thiruvathukal

A PARTICULARLY INSPIRED EPISODE OF THE AMERICAN TV SHOW *FRASIER*, "THEY'RE PLAYING OUR SONG," FEATURES THE TWO BROTHERS, FRASIER AND NILES CRANE, talking about a new jingle Frasier is composing for his radio talk show, a project whose scope has evolved into a minor symphonic work:

Niles: Whatever happened to the concept of "less is more"?
Frasier: Ah, but if less is more then just think how much more "more" will be.

One of the reasons scientific programmers love Linux is its less-is-more philosophy. We can configure it to be anything from a desktop replacement with USB port support to a blade in a large SMP compute engine to a powerful Web server. Although Linux's market penetration in these various sectors remains to be seen, plenty of people are pumping resources into the Linux world.

The first item of business when installing a Linux system is to pick a distribution, or *distro*. The plethora of options includes Red Hat, Lindows, Debian, SuSe, Mandrake, Gentoo, and even the emerging Chinese standard, Red Flag. I've used just about every version of Linux and found that almost all suffer at some point from the same management complexity found in Microsoft Windows. This is important for those of us who don't want to spend a lot of time becoming Linux experts.

In this article, I'll explain why Gentoo Linux is a good choice for scientists, and how its structure gives us the flexibility and ease of management we need.

Gentoo Linux

I should point out early on that I'm the pickiest of users. I want my systems to run optimally for scientific and computer-science research; toward this goal, I want the OS and its applications to do nothing that affects my setup's performance and reliability. In my experience, most Linux distros are weak in certain areas:

- *Binary packages*. Virtually all OSs—and most Linux distros—are shipped with precompiled binaries. This means that upgrading a particular tool usually requires a full OS upgrade because most packages have numerous dependencies.
- *Kernel compilation*. Linux distros typically have a kernel precompiled to run on all of a particular architecture family's machines. On the Intel x86 platform, for example, the kernel is typically configured for the 386 or 586 variants. However, many processors run less than optimally when running these instruction sets. In particular, Intel's latest Pentium 4 and Itanium are hobbled when the kernel doesn't use newer instructions and optimization opportunities.
- *Differing platform use*. Most Linux distros allow optional desktop setups, but different approaches to managing the OS exist, depending on intended use. A desktop system, for instance, requires USB devices that have the ability to plug or unplug a device at any time, as well as other peripherals to provide a more user-friendly experience.

Gentoo Linux is a significant distro for several reasons. First, the entire OS is maintained from source code. Even when binary packages are directly installed, metadata shows precisely how the package was configured and built. Second, we need install the OS only once. We can get upgrades of the latest packages from one of the myriad Gentoo mirrors. Finally, it's completely free, regardless of intended use and configuration. Unlike Red Hat Linux, Gentoo Linux doesn't have a commercial edition, meaning users don't have to worry that something might be missing in the free version.

Let's look at some of Gentoo's salient features.

Metadata and Portage

Perhaps the Gentoo distro's most significant feature is its use of metadata—particularly to let us answer questions about what version of a package is installed, how that package was built, and whether a newer version is available. Metadata itself is not an altogether new concept in Linux distros, but Gentoo takes it to a new, network-centric level. Let's look at an example:

```
$ emerge —search gcc

* sys-devel/gcc
  Latest version available: 3.3.3-r6
  Latest version installed: 3.3.3-r6
  Size of downloaded files: 23,203 kB
  Homepage: http://www.gnu.org/software/
     gcc/gcc.html
  Description: The GNU Compiler Collection.
     Includes C/C++, java compilers, pie and
     ssp extentions
  License: GPL-2 LGPL-2.1
```

This example shows that I have the latest version of the gcc compiler installed (the latest version available versus installed is 3.3.3-r6). The command used to obtain the information, `emerge`, is part of the Gentoo system, which maintains package information and keeps the packages up to date. This command's most brilliant aspect is that we can use it at any time to discover the latest available packages. We do this with `emerge sync`, which synchronizes our local Portage database with one hosted at a randomly selected Gentoo mirror site. Gentoo has invested significant effort in ensuring that its mirroring scheme is highly reliable, regardless of the selected mirror.

For the most part, we can install any package of interest simply by using `emerge -search` to find it. (Obviously, we need to have an idea of what we're looking for—at least part of its common name.) Once we know the name, we can pretend to install it (by using `emerge -pretend`) and then install it for real, using `emerge` with the package name as the lone argument. (The core idea behind pretending is twofold: one, to see what packages will be installed as a result of `emerge`, and two, to see how many packages will be installed. Unlike most installer programs out there today, `emerge` lets you see the potential effect of something before you actually do it.)

Package Build Complexity

Typical system administration in Linux requires ongoing maintenance of multiple packages. Most seasoned system administrator/hackers, such as yours truly, tend to download packages from the Web and build them by hand:

```
./configure -{}-prefix=/usr/local
   (myriad of options)
make
make install
```

Only a handful of packages can be built from source code without downloading and building several dependent packages beforehand. Take, for example, the world-famous (well, at least among us geeks) Emacs editor. We can build Emacs in two ways: as a console application or as a graphical application using several different widget sets.

In the past, the system administrator would download Emacs source code from ftp.gnu.org and then invoke the `configure` command with several options to include programming libraries for the desired functionality. By default, Emacs is designed to be compiled as a console application. Today, however, most users want the ability to run a graphical version on their desktops. Linux users typically use Emacs with their favorite desktop (Gnome or KDE), so they want this support compiled into the Emacs binary.

With Gentoo Linux, the administrator can query the Portage metadata to determine what flags can be set to direct the build. You can find the entire list of flags (known affectionately as `USE` flags) on the Gentoo site in the Documentation section (www.gentoo.org/doc/). Let's look at the flags we can use to build the Emacs package, which is already installed on my system:

```
gkt@develop gkt $ etcat -u emacs
[ Colour Code : set unset ]
[ Legend : (U) Col 1 - Current USE flags ]
[ : (I) Col 2 - Installed With USE flags ]
U I [ Found these USE variables in : app-
   editors/emacs-21.3-r1] + - X : Adds
   support for X11
+ + nls : unknown
+ - motif : Adds motif support
   (x11-libs/openmotif x11-libs/lesstif)
- - leim : Adds input methods support to
   Emacs
+ - gnome : Adds GNOME support
- - Xaw3d : Adds support of the 3d athena
   widget set
- - debug : Tells configure and the makefiles
   to build for debugging.
```

The `etcat` command is part of the Gentoo Linux toolkit, which lets us examine the metadata. The `U` column indicates how the package would be built if we used the default `USE` flags (specified in the configuration file `/etc/make.conf`); the `I` column indicates how the package was actually built. On my system, which is running the `etcat` command, I dis-

```
# Copyright 1999-2004 Gentoo Technolo-
gies, Inc.
# Distributed under the terms of the GNU
General Public License v2
# $Header: /var/cvsroot/gentoo-x86/app-
editors/emacs/emacs-21.3-r3.ebuild,v 1.5
2004/06/10 19:44:32 vapier Exp $

inherit flag-o-matic eutils

DESCRIPTION="An incredibly powerful, ex-
tensible text editor"
HOMEPAGE="http://www.gnu.org/software/ema
cs"
SRC_URI="mirror://gnu/emacs/${P}.tar.gz
  leim? ( mirror://gnu/emacs/leim-
  ${PV}.tar.gz )"

LICENSE="GPL-2"
SLOT="0"
KEYWORDS="~x86 ~ppc ~sparc -alpha arm -
hppa ~amd64 -ia64 ~s390"
IUSE="X nls motif leim gnome Xaw3d less-
tif"

RDEPEND="sys-libs/ncurses
  sys-libs/gdbm
  X? ( virtual/x11
    >=media-libs/libungif-4.1.0.1b
    >=media-libs/jpeg-6b-r2
    >=media-libs/tiff-3.5.5-r3
    >=media-libs/libpng-1.2.1
    !arm? (
    Xaw3d? ( x11-libs/Xaw3d )
    motif? (
      lesstif? ( x11-libs/lesstif )
      !lesstif? ( >=x11-libs/openmotif-
      2.1.30 ) )
    gnome? ( gnome-base/gnome-desktop )
    )
  )
  nls? ( sys-devel/gettext )"
DEPEND="${RDEPEND}
  >=sys-devel/autoconf-2.58"

PROVIDE="virtual/emacs virtual/editor"
SANDBOX_DISABLED="1"

DFILE=emacs.desktop

src_compile() {

  # -fstack-protector gets internal com-
piler error at xterm.c (bug 33265)
  filter-flags -fstack-protector

  epatch ${FILESDIR}/${P}-amd64.patch
  epatch ${FILESDIR}/${P}-hppa.patch

  export WANT_AUTOCONF=2.1
```

Figure 1. Gentoo Ebuild file for Emacs. This file is used to describe the build process for the Emacs package.

abled X11 support and opted for a pure console application.

Here's the kicker: Suppose I decide later that I want an installation of Emacs that supports both the console and the Gnome desktop using the Gnome Toolkit (GTK) library. I can pretend to build the Emacs package from the command line using the `emerge` command discussed earlier:

```
gkt@develop gkt $ USE="gnome" emerge -up
   emacs
```

Pretending is what differentiates Gentoo from most other Linux distros. As you can see from the following output, adding Gnome support on my server would require me to install many packages:

```
root@develop gkt # USE="gnome" emerge -up
   emacs
These are the packages that I would merge,
   in order:
Calculating dependencies...done!
[ebuild U ] dev-libs/glib-2.4.1 [2.2.3]
[ebuild U ] dev-libs/atk-1.6.1 [1.4.1]
[ebuild U ] dev-libs/libxml2-2.6.7 [2.6.6]
[ebuild N ] x11-themes/hicolor-icon-theme-
   0.4
[ebuild U ] x11-themes/gnome-icon-theme-
   1.2.1 [1.0.9]
... Many lines omitted for brevity's sake!
[ebuild N ] gnome-base/gnome-keyring-0.2.0
[ebuild U ] gnome-base/libgnomeui-2.6.0
   [2.4.0.1]
[ebuild U ] x11-themes/gnome-themes-2.6.0
   [2.4.1]
[ebuild U ] gnome-base/gnome-desktop-
   2.6.0.1 [2.4.1.1]
[ebuild U ] app-editors/emacs-21.3-r2
   [21.3-r1]
```

This command might look odd to casual Unix users. Essentially, what's happening is that a local shell variable definition is temporarily affecting the environment for the `emerge` command's execution. The definition will take precedence over the

```
    autoconf

    local myconf
    use nls || myconf="${myconf} —disable-
  nls"
    if use X ; then
      if use motif && use lesstif; then
        export LIBS="-L/usr/X11R6/lib/less-
  tif/"
      fi
      myconf="${myconf}
        —with-x
        —with-xpm
        —with-jpeg
        —with-tiff
        —with-gif
        —with-png"
      if use motif ; then
        myconf="${myconf} —with-x-
  toolkit=motif"
      elif use Xaw3d ; then
        myconf="${myconf} —with-x-
  toolkit=athena"
      else
        # do not build emacs with any
  toolkit, bug 35300
        myconf="${myconf} —with-x-
  toolkit=no"
      fi
    else
      myconf="${myconf} —without-x"
    fi
    econf ${myconf} || die
    emake || die
  }

  src_install() {
    einstall || die
    einfo "Fixing info documentation..."
    rm -f ${D}/usr/share/info/dir
    for i in ${D}/usr/share/info/*
    do
      mv ${i%.info} $i.info
    done

    einfo "Fixing permissions..."
    find ${D} -perm 664 |xargs chmod 644
    find ${D} -type d |xargs chmod 755

    dodoc BUGS ChangeLog README

    keepdir /usr/share/emacs/${PV}/leim

    if use gnome ; then
      insinto /usr/share/gnome/apps/Applica-
  tion
      doins ${FILESDIR}/${DFILE}
    fi
  }
```

Figure 1., continued.

USE variable defined in the global environment, which is set in /etc/make.conf. In normal day-to-day Gentoo Linux use, we don't have to set the USE variable on the command line; however, it's a handy trick when we want to ensure that a particular library (or set of libraries) is compiled into a binary.

Sneaking Around the Emerge System

First-time Gentoo Linux users will find themselves frustrated by something known as a *masked package*—so named because they can't be built, either due to an architectural limitation or because they haven't been tested extensively. For example, the application we want to emerge might have been tested and known to work on the Intel x86 family, but not on a PowerPC or SPARC.

Every package available via the Portage system (and emerge) provides an ebuild file, which you might have noticed in the previous emerge command output. Figure 1 shows the text for the Emacs ebuild file; you can find ebuild on your new Gentoo system in the /usr/portage/app-editors/emacs directory. This file contains many variable definitions that control how the package is built:

- *DESCRIPTION* is a summary of why we should install the package.
- *HOMEPAGE* is where we can download the file to install it.
- *SRC_URI* is where the emerge system downloads the source during installation.
- The *LICENSE* for every package in Gentoo is clearly specified in the ebuild file, which helps us sleep at night, knowing that any package installed on the system is legal, from an open-source perspective.
- *KEYWORDS* are where we see firsthand the interaction with the USE flags. Emacs, for example, can be built on just about any system, including x86, PowerPC (ppc), and SPARC (sparc). However, it can't be built on Gentoo Linux running on an Alpha, which more than likely means that someone at Gentoo or in its user community tried it without success.
- *DEPEND* describes the dependency tree for compiling the package. The syntax is a somewhat complex beast based on regular expressions.

For brevity's sake, I can only provide an introduction here, but knowing just a bit about how the Ebuild files work can

```
#!/sbin/runscript
# Copyright 1999-2003 Gentoo Technolo-
gies, Inc.
# Distributed under the terms of the GNU
General Public License v2
# $Header: /home/cvsroot/gentoo-x86/net
   -www/apache/files/2.0.40/apache2.initd,
   v 1.13 2003/10/31 07:17:45 rajiv Exp $

opts="${opts} reload"

depend() {
  need net
  use mysql dns logger netmount
  after sshd
}

start() {
  ebegin "Starting apache2"
  [ -f /var/log/apache2/ssl_scache ] && rm
     /var/log/apache2/ssl_scache
  env -i PATH=$PATH /sbin/start-stop-
     daemon —quiet \
    —start —startas /usr/sbin/apache2 \
    —pidfile /var/run/apache2.pid — -k
start ${APACHE2_OPTS}
  eend $?
}

stop() {
  ebegin "Stopping apache2"
  /usr/sbin/apache2ctl stop >/dev/null
  start-stop-daemon -o —quiet —stop
     —pidfile /var/run/apache2.pid
  eend $?
}

reload() {
  ebegin "Gracefully restarting apache2"
  /usr/sbin/apache2 -t ${APACHE2_OPTS}
     &>/dev/null
  if [ "$?" = "0" ]
  then
    if [ -f /var/run/apache2.pid ]
    then
      kill -USR1 $(</var/run/apache2.pid)
      eend $?
    else
      svc_start
      eend $?
    fi
  else
    if [ -f /var/run/apache2.pid ]
    then
      svc_stop
    fi
    #show the error(s)
    /usr/sbin/apache2 -t ${APACHE2_OPTS}
    eend 1
  fi
}
```

Figure 2. The script for /etc/init.d/apache2. This script starts, stops, or restarts the server at boot time.

prove helpful when troubleshooting the packages that fail to build properly—a rare event in Gentoo—or adding your own packages. Gentoo's greatest strength as a Linux distribution is that it is community-focused, allowing software developers anywhere to contribute their own packages and make them available through the intuitive `emerge` command.

Install the Kernel Yourself and Be Happy!

Gentoo Linux is one of the few Linux distros that insists we build a proper kernel on our own, which means we don't get a precompiled kernel under any circumstances. At first, you won't be thrilled with this idea, but if you use any other Linux distro and find yourself wanting to rebuild the kernel, you might end up at a loss. Worse, most Linux distributions make it difficult to upgrade the kernel without basically waiting for an upgrade CD (the Microsoft approach), reinstalling new packages, and rebooting your computer.

Because the kernel is the code at the nucleus of any operating system, building it sounds scarier than it really is. Gentoo Linux provides first-time users with a script that configures the kernel exactly as the Gentoo development team has it configured on the distribution media. The program, `genkernel`, will automatically set up the kernel with the most commonly needed options and perform an optimized build for your particular processor.

Initialization Scripts

Initialization scripts present one of the biggest headaches when it comes to managing Linux systems, but they're essential for bringing up the needed system services in the proper order. Such scripts are typically found in the `/etc/rc.d/rcN.d` directory, where `N` corresponds to a numbered run level. (In my experience with Solaris and Red Hat Linux systems, a server typically runs at run level 2 or 3.) The entries in the `rcN.d` directory are actually symbolic links to shell scripts in the `/etc/init.d` directory.

Suppose we wanted to run the Apache Web server (httpd). We'd want this service started, say, after the networking support starts. In a Red Hat setup, we could ensure that this occurs by having one script named S05net and another named S25apache.

(The actual names and numbers don't matter, provided the network script name lexicographically precedes the Apache script name.) The `init` process is a bit cumbersome for a system administrator (especially at the command line) because it places a great premium on clever script names and rebooting the system to ensure it really comes up correctly after any change.

The Gentoo Linux folks have developed a superior approach for managing the complex (and often unknown) interactions between startup scripts. They defined a pattern for organizing the script, a sample of which is shown in Figure 2. This figure shows four functions: `depend()`, `start()`, `stop()`, and `reload()`. The `depend()` function provides details of how the service is to be run. Without going into excessive details, the service needs networking (`need net`) before starting. This makes sense: we need networking to start a Web server. The service also uses other services (`dependent services`)—for example, the default Apache setup script needs MySQL, DNS, Logger, and network mounting. Finally, Gentoo also gives us the ability to specify that a service should start after another service, even if the service doesn't depend on that service's availability. The big innovation is that Gentoo lets every script provide partial ordering information that can be sorted topologically to establish an execution order automatically and at runtime. This saves the system administrator from having to devise a creative naming scheme for ensuring that services start as expected.

Most packages, when emerged, provide a startup script (known historically in Unix as "rc" scripts) named after the package itself—for example, the apache2 package provides a script bearing the same name. Gentoo does not require the system administrator to actually create the script, but after emerging the package, he or she must install the script separately using the `rc-update` command and indicate the run levels at which the script is to be started. Gentoo Linux provides multiple run levels (boot, nonetwork, and default), so you'll probably boot the OS at the default when setting up your own Gentoo Linux system. Here's how to install the apache2 service:

```
/usr/sbin/rc-update add apache2 default
```

Gentoo Linux automatically recomputes the correct order in which the scripts are to be started and starts apache2 according to the constraints established in the `depend()` function. It's also possible to uninstall a service without uninstalling the package: instead of using the `rc-update add` command just listed, you simply use the `del` command.

The ability to manage startup scripts so easily and intuitively is a major advance over other Linux distributions and a huge win for managing several Linux systems consistently and coherently.

Install What You Need, *Then* What You Want

Unfortunately, we're in an age of more is more. With the potential for terabytes of disk space and gigabytes of RAM, does it matter how much disk space and memory are consumed when there's seemingly plenty to burn? Why not just install the veritable cornucopia of free software packages and be done with it? In practice, this mentality leads to installing a bunch of stuff on your system that you simply don't need. True, you might have resources to burn, but unwanted services running in the background still affect system performance, often dramatically.

With Gentoo Linux, a minimal but fully usable system is installed when performing the base stage3 installation (see the "Getting Started" sidebar). The stage3 installation is a minimal Gentoo setup that gets the system going without having to compile absolutely everything from scratch (which can take a long time even on the fastest systems). Notably, the base system contains only the most essential materials for building other packages: basic Unix commands, the gcc compiler (with support for C and C++), the standard C and C++ libraries, and the Portage system. Most users will want more, even those of us who truly believe that less is more. I tend to do a lot of Python hacking (part of the base system), Web programming (`emerge apache2`), and work with scientific programming tools. Here are the commands for installing some of my favorite scientific programming packages:

MPICH emerge sys-cluster/mpich
LAM MPI emerge sys-cluster/lam-mpi
HDF emerge dev-libs/hdf or emerge dev-libs/hdf5
LAPACK emerge app-sci/lapack

When in doubt, use `emerge –search package-name-substring`, where you type a few characters of the package name in place of `package-name-substring`. This gives you a list of matches from which you can then perform an `emerge` similar to the above examples.

You might argue that you could spend a substantial amount of time selecting which packages are or aren't necessary. There's really no solid counterargument to that, but I want as few inessential packages installed on my computers as possible. I'm willing to pay the price of having to install something later that I truly needed but originally forgot to install. Using the Portage tools, I can find all the installed packages I consider useful and use this list to clone the setup on another system in my cluster. The Gentoo community knows

Chez Thiruvathukal

In this issue's installment, I'm taking the opportunity to discuss some interesting developments in the open-source community that I think will be of interest to all the scientific programmers out there. We've already seen several major announcements (or accomplishments, depending on how you view these things).

Mono

The Mono project aims to build a complete implementation of the .Net platform and C# (pronounced "C sharp") programming language developed by Microsoft. The .Net platform is Microsoft's answer to the Java platform, which by now everyone knows—along with the browser—played a major role in defining the dot-com era. I'm particularly excited about this development, which I think will be instrumental in ensuring a long-term competitive environment for software. Using Mono, you can develop .Net applications on any platform, including Linux. In fact, on the Gentoo Linux platform discussed in the main text, it's a simple matter of typing `emerge mono` to install it.

A perfectly natural question to ask about any 1.0 release is, "How good is it, really?" I've been working on a commercial parallel-processing project this summer using (grrr...) the Microsoft platform. Most of the code I've written thus far can be compiled and run cleanly and efficiently using the Mono C# compiler and runtime environment. For a 1.0 release, it's notable that Mono can compile the codes we tend to write in high-performance computing. My early experiments suggest about 20 to 30 percent improvement over the Visal Studio .Net compiler and runtime environment.

Of course, it's not all perfect. The Windows and Windows Forms frameworks are not fully working yet, which makes it difficult to write portable graphical applications. However, a short-term solution allows programmers to leverage native GUI libraries such as the GTK toolkit (so-called GTK#). The next release will offer the ability to develop Windows applications that run natively on Windows and use the WINE library on Linux. As an aside, the GTK# toolkit has already been put to use by the mono.develop project (www.monodevelop.com), which aims to be a complete integrated development environment (IDE) for developing C# applications. You can also get this in the Gentoo project by doing `emerge monodevelop`.

Anyway, if you're interested in seeing what C# code looks

that not all users want to exert such fine-grained control, so it's arranging packages into groups to allow a collection of related packages to be installed in one fell swoop. The Portage system is so flexible that there's no reason such convenience can't be extended to users. Two predefined groups of packages—system and world—refer to essential system packages and all packages that have been installed to date, respectively. With these names, the user can periodically update packages (after synchronizing the Portage metadata) by using `emerge -update system` or `emerge -update world`. This makes the Gentoo setup's ongoing maintenance almost (but not completely) trivial. You still need to know a thing or two about Linux system administration, but Gentoo takes the Linux world one step closer to user-friendliness when it comes to administration. Plus, its ability to reliably perform live, real-time updating from source code is unique.

Clustering: An Advanced Need

Let's touch briefly on the topic of cluster computing, which is one of my present areas of research. Gentoo facilitates cluster maintenance in a variety of ways. I can't go into exhaustive detail, but I can share my general approach to using Gentoo Linux in a clustering environment, especially a homogenous one (that is, where every system in the cluster is nearly the same in terms of hardware).

Easy-to-Clone Reliably

Installing Gentoo Linux in a cluster is more or less a matter of getting one of the nodes up and running. Once the node is configured as desired, we can reliably clone the setup by using a tool like `rsync` (yes, you can emerge it, if it's not installed already) to replicate the file systems from the first node to a new node. This works a bit like magic, but Gentoo's distro CD is a key enabler. The Gentoo folks call it a live CD because it lets you boot the OS, mount read-only file systems on the source, and then perform synchronization.

distcc

Cluster computing has the potential to be a killer app for Gentoo Linux. Packages can be compiled in parallel on the like, check out my Web site dedicated to Java and HPC (www.jhpc.info); there, I provide a snapshot of a multithreaded class library (written for my first book), which has just been ported to C# (www.jhpc.info).

Eclipse

Meanwhile, back on the other side of the world (Java), the Eclipse Project has released version 3.0 of what is more than likely the most impressive Java project that few people are aware of. Eclipse is an open-source project that emerged from IBM's Visual Age software development. Originally an internal project, IBM realized that its investments in Eclipse had greater potential for payoff by involving the broader Java software development community in extending the Eclipse platform, mostly through the creation of plug-ins and experimental projects.

Eclipse goes beyond the traditional IDE by supporting the notion of configurable *perspective*—an example is a language editing/browsing perspective. In Eclipse, perspectives exist for Java, C/C++, and XML language editing. Another example is a source-code repository browsing perspective, which lets you examine a concurrent versioning system (CVS) tree similar to Windows Explorer. The Eclipse framework goes one step further by allowing for near seamless integration between perspectives—the source-code browsing plays nicely with both Java and C code. You can easily switch between perspectives, and Eclipse doesn't get confused. The Eclipse Project describes its approach as an environment for "everything and anything or nothing in particular."

This makes for an environment that is configurable for just about any type of development.

Subversion

Subversion is an emerging alternative to the aforementioned CVS, which for all practical purposes, is the incumbent version control software when it comes to open-source and academic research projects. It's not likely to go away anytime soon, especially given the number of projects that use it. If you judge by SourceForge use alone, there are tens of thousands of projects actively using CVS.

CVS is not without its share of problems, especially from a systems viewpoint. Setting up CVS to support even a small development team requires extreme care and, therefore, expertise. Familiarity with creating users and groups on Unix is a must, and the "`setgid()`" bit must be enabled to ensure group ownership of files contained in project directories so all members of the group can continue to check out, update, and commit files. The Subversion Project (subversion.tigris.org) aims to create a tool that's mostly compatible with CVS from the user perspective while simultaneously employing modern systems design techniques. Instead of keeping version files in a hierarchical directory structure (that mirrors the project's source hierarchy), all information is kept in a single embedded database—specifically, the Berkeley DB storage engine. The advantages of databases over file systems include transactional safety, consistency, and portability. In just a short time, the Subversion team has ported its C code to all modern OS platforms, so Windows users can also run the Subversion server or client.

Subversion is a project worth keeping any eye on. I'm going to hold out just a bit longer—at least for my Java projects—until the integration with Eclipse (my favorite software tool) is completed.

computing cluster by using the distcc service, which offloads lengthy compiles to nodes in the cluster as they are brought online. In my experience, this can be a significant timesaver because numerous packages are built in parallel, especially as I'm trying to get my system going.

Familiar Packages Available Painlessly

Perhaps one of the arguments for using Gentoo over other Linux distros is the availability of high-performance computing (HPC)-friendly packages via the Portage system. Parallel virtual machine (PVM), message-passing interface (MPI), and others are all available via the trusty `emerge` command. For scientific programmers and scientists, this means that a parallel programming environment can be established with minimal pain and suffering. I've never run into any problems with these so-called staples of HPC compiling and running cleanly on Gentoo Linux; all you need to do is edit a few configuration files (such as MPI's famous `machines` file to list the host names), and you're good to go!

OpenMosix

One of my students, Sean McGuire, actually set up our laboratory at Loyola University, Chicago, with Gentoo Linux. Being clever, he decided to experiment with a special kernel known as OpenMosix, which allows for processes to be created from any node and launched on any node under suitable migration conditions. OpenMosix is useful for environments that feature batch-style parallel processing, wherein the goal is to maximize the throughput of jobs that have minimal data I/O requirements or transaction environments in which the I/O is offloaded to a transaction server. Gentoo Linux again makes it possible to take advantage of experimental kernels and software tools that other distributions tend to avoid like the plague.

At my private company and at my university, I have built my own low-cost computing clusters using ShuttleX nodes (www.shuttle.com), which at the time of this writing run approximately US$400 apiece. The Nimkathana cluster (www.nimkathana.com/products/low

Getting Started

Getting started with Gentoo is easy. It's a bit more work than setting up something like Red Hat, but that's a price well worth paying, especially if you relish the thought of having a clue about what's being installed and how to configure things. The Gentoo Linux documentation is excellent, with many helpful tips on how to do useful types of setups such as running a mail-hosting environment.

Here are the steps:

- Visit www.gentoolinux.org. It has an installation document that you'll want to download and print. A printed version of the document on which you can write notes is extremely helpful.
- Download the ISO image to create an installation CD. This more than likely means that you should have a CD-burning utility on your computer. On OS X, you can run the Disk Utility; on Windows, a great program is Ahead Nero (www.ahead.de/us/index.html).
- The site provides many different images. For most users with modern PCs, the images having Pentium, Pentium4, or Athlon in the name should be adequate. (One tidbit: regardless of whether you use Pentium or Athlon processors, both are compatible with the 686 instruction sets.) Gentoo provides a universal CD, but you should pursue this option only if you don't know what architecture your computer is using.
- Once you've burned the CD-ROM, try booting your computer with it. This might not work if your CD-ROM is disabled as a boot device, which forces you to go into the BIOS setup to enable CD-ROM booting. Usually, you need to specify the order in which the various devices should be tried. These days, you can select just about anything as a boot device. If you've installed Windows on your system, you might already be set to boot from CD-ROM, but check before proceeding.

Once you're booted, you can complete the setup as described in the installation guide. The process is relatively straightforward from this point: you create partitions and file systems on the partitions, and then proceed to install the base OS. Next, you'll build the kernel, install a bootloader (to let you boot Linux or Windows), and you're done.

In addition to the previous steps, some questions should be on your mind:

- Do you plan to keep Windows on the same computer? This is known as the dual-boot option. If you plan to install both Windows and Linux, you should seriously think about making a backup...now.
- Do you know what hardware is in your computer? Certain devices, usually found in ultra-cheap PCs, are designed to work only with Windows. Some modems and integrated video logic don't work with Linux and never will, but such problems can usually be remedied by replacing or adding the device in question. If you have integrated video, for example, you'll want to go out and get a relatively standard video card that is known to work with Linux.
- Are you going to use Linux mostly in terminal mode or graphical mode? If you don't plan on running desktop applications and just want to tinker around with Linux in console mode, you can get a setup running with minimal pain and suffering. Most of my servers are configured to run exclusively in terminal mode, which I then access remotely from my laptop running OS X (and sometimes Windows). If running in terminal mode, I strongly recommend that you install OpenSSH (which can be done with `emerge` and `rc-update` as described in the article).

My ultimate advice? Learn the answers to all these questions before attempting to install Gentoo or any version of Linux.

_cost_clustering), which varies between 8 and 12 nodes, is an example of such a cluster built for under $5,000, with access to a storage server built for under $2,500. We use this cluster for data mining and simulations, and it runs Gentoo Linux exclusively. Who would have thought that we could build a terascale cluster for under $5,000, let alone have access to another terabyte off-cluster for $2,500? However, the best aspect of the cluster isn't its cost: it's the ability to install a great Linux distro, Gentoo Linux, and keep it up to date at all times without ever having to reinstall the OS again (barring an unanticipated hardware failure).

This article only scratches the surface of what's possible with Gentoo Linux. With a robust metadata framework and network-centric architecture, the Gentoo approach provides a great example of the Internet's potential to make OSs easier to maintain (live and in real time) and keep up to date in an age in which security has risen to the forefront of serious problems, necessitating the ability to incorporate patches before vulnerabilities are exploited. Although I didn't cover it here, Gentoo releases its own security advisories. In short, the OS sports a modern design and is well-suited to everything from desktop to advanced computing.

George K. Thiruvathukal is an associate professor of computer science at Loyola University, Chicago. He is also president and CEO of Nimkathana Corporation, which does research and development in high-performance cluster computing, data mining, handheld/embedded software, and distributed systems. He wrote two books covering concurrent, parallel, distributed programming patterns and techniques in Java and Web programming in Python. Contact him at www.cs.luc.edu/gkt.