

MANAGED HOSTING

ARTICLES

- 2** **Verio Managed Hosting; The Hosting Service Provider for Superior IP Application Performance**

- 4** **Graphic Administration with Webmin**
New to Linux administration? Webmin can help you out.
Federico Kereki

- 8** **Building a Scalable High-Availability E-Mail System with Active Directory and More**
Cyrus-IMAP to the rescue.
Jack Chongjie Xue

- 12** **Distributed Computing with distcc**
Put your lazy machines to work.
Jes Hall

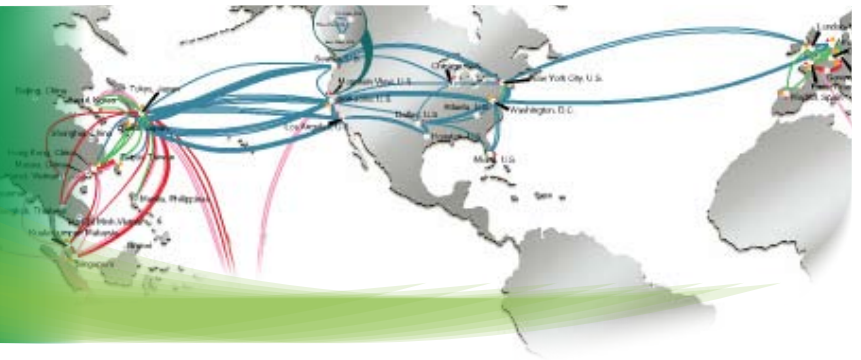
- 15** **Faster Web Applications with SCGI**
Can your Web apps go even faster?
Jeroen Vermeulen

sponsored by

VERIO

An NTT Communications Company

May 27, 2008



THE HOSTING SERVICE PROVIDER FOR SUPERIOR IP APPLICATION PERFORMANCE

SEVEN REASONS FOR CHOOSING VERIO

When you're tasked with ensuring the highest possible performance levels for IP applications, your choice for a hosting service provider is clearly Verio.

Verio's robust portfolio of Linux-based services delivers state-of-the-art performance, reliability, and security. Verio Linux VPS, Linux MPS, and dedicated servers are based on the Red Hat Enterprise Linux (RHEL) operating system to deliver the best of open-source technology and the stability of a true enterprise-class platform.

Verio and its parent company, NTT Communications, helped shape the IP industry and continuously invest in a world-class global tier network designed exclusively to move Internet traffic fast. The data-only network provides high-volume transport at commercial-strength bandwidths. There are no voice ride-alongs to impede the flow of traffic.

Following are seven reasons why a vast number of professionals in the Linux community rely on the world-class services Verio provides.

Optimum uptime – Verio's Tier-1 global IP network keeps you at the forefront of Internet technology to power your applications 24x7 and eliminate the downtime that threatens operations. Dedicated servers deliver extremely high availability to reduce the risk of downtime.

Reliable redundancy – Verio ensures server and network integrity with uninterruptible power supplies and redundant servers and connections.

Fast response time – With more data paths, routing options, and private peering points, Verio's global network ensures fewer hops and thus greater delivery speed. The network is free of outages 100% of the time. Packet loss will not exceed 0.1%. Latency will not exceed 50 milliseconds for North America, 90 milliseconds for transatlantic transmission, and 130 milliseconds for transpacific transmission. Performance is guaranteed by the industry's most aggressive SLAs.

Real-time support – Superior network support at our advanced Network Operations Center provides real-time alarming, forecasting, traffic management, event notification, and upgrades. Customers are assured fast, highly responsive, single-point-of-contact problem resolution.

Uncompromising security – Verio’s global IP network data and operations centers employ rigid physical security measures and constant monitoring.

Reliable data restoration – Verio’s world-class infrastructure ensures data restoration with data protection mechanisms, including RAID storage and virtual tape library systems to minimize or prevent data loss, should a system crash.

Seamless scalability – High-capacity bandwidth and resources deliver the seamless scalability needed to expand customer services without compromising performance.

You’re in Control

With Verio dedicated servers, you retain the management responsibility you want. Verio’s proprietary software allows you to have complete control over your hardware and applications. The services offer flexible options for storage space, bandwidth, security, and power to create an optimal environment that best satisfies your hosting requirements.

Call us at: 1-214-672-7246
Toll Free: 1-800-269-3300 or
visit www.dedicatedserver.com

VERIO
An NTT Communications Company

Graphic Administration with Webmin

First published in *Linux Journal*, issue 168

Administering a Linux server might be complicated, but Webmin can help you work quickly and safely. | **FEDERICO KEREKI**

WHEN YOU start administering a Linux system, one of the biggest challenges is learning exactly what to do, and how to do it. There simply are too many tools, settings, parameters, configuration files, daemons and what have you to consider. Obviously, if you ever want to become a full-fledged sysadmin on your own, you have to learn everything. But, until you get to that point, you still need to get things done, and you would do well by installing and using Webmin, a Web-based, comprehensive administration tool for Linux systems.

Webmin runs on your server and presents a Web-based interface, allowing you to do all sorts of system administration tasks—from the very simple to the very complex ones—without ever touching a configuration file or restarting any process or daemon on your own. As an aside, it isn't just any run-of-the-mill tool. If you mention Webmin at a Linux Users Group reunion, it's guaranteed to raise a lively argument—much akin to the “using closed graphics drivers” or “banning all non-open-source software from distributions” discussions on forums and chat channels.

For some people, the idea of using anything but the command line to manage a server is barely short of heretical, and they believe you should not even consider using Linux if you plan on employing such a tool. (A Linux user I know once said dismissively, “If you want to use graphic tools, use Windows.”) However, for other people, any tool that helps them avoid mistakes or the need to memorize a lot of parameters is a welcome addition to their toolset.

Webmin won't let you avoid actually learning about Linux though. You can't merely start using it and change configuration settings without knowing perfectly well what you are doing. If you know what needs be done and how to do it, Webmin can save you from having to memorize lists of parameters or configuration files, and it will help you get things done quickly and safely. On the other hand, don't ever use Webmin as an experimentation tool. It's quite likely you could really mess things up.

Webmin runs not only on Linux, but on UNIX and FreeBSD as well. Here's a partial list of supported systems and distributions: Asianux, Caldera, Debian, FreeBSD, Gentoo (and Sabayon), HP-UX, IBM AIX, LinuxPPC, Lycoris, Mac OS X, Mandriva (and Mandrake and Conectiva), Mepis, NetBSD, OpenBSD, PCLinuxOS, PlayStation Linux, Red Hat (and CentOS and Fedora), Scientific Linux, SCO OpenServer and UnixWare, Slackware, Sun Java Desktop System, Sun Solaris, SUSE and OpenSUSE Linux, Turbolinux, Ubuntu (and derivatives like Kubuntu or Xubuntu), Xandros, Yellow Dog Linux and Yoper Linux.

If your favorite distribution isn't included, some Webmin modules might not work, so be careful. If you are using a

distribution derived from one that is on the list, it's a fair bet you won't have any problems, but don't say I didn't warn you.

By the way, why this state of affairs? The problem is a lack of standardization. Distributions use different locations for various configuration files, and if Webmin can't find them, it won't be able to function. This may change for the better over time, when (if) all distributions fully embrace the Linux Standard Base (LSB) and comply with the standards related to file placement. But, that certainly hasn't happened yet. To mention a simple example, I'm currently using OpenSUSE, and it uses `/srv/www/htdocs` as the root for Web sites. Most other distributions use `/var/www/html`. So, you can see that a configuration module might have serious problems finding Web files if it didn't know about this difference.

What do you need to run Webmin? Just a browser, Perl, a Java Runtime Environment (JRE) for some functions and the root password. After you become familiar with Webmin, you'll be able to forget about ever editing configuration files (like all those in the `/etc` directory) or starting, stopping and reloading services. If you set up Webmin correctly, you even will be able to administer your server from a remote machine.

Installation

Webmin is available under the GPL, so you can get it without any problems. The latest version (as of the time of this writing) is 1.380, and it's being developed actively. The easiest way to install Webmin is with your favorite package manager. Even though I am an OpenSUSE user, I prefer Smart to YaST, so a simple `smart install webmin` command did the job for me. If you don't get the latest version this way, don't worry. You can fix that just by using Webmin itself; keep reading.

The other method of installation is to go to the download site, download the appropriate version for your system, and follow the instructions on the left side of the page. There are two options here. You can get the full package (with all available modules), or you can get the minimal edition and add the modules you require afterwards, using Webmin's own update features.

After installing Webmin, you need to start a service. Working as root (use `su`), do `chkconfig webmin on` (to ensure that Webmin starts every time you turn on your machine). Then do `/etc/init.d/webmin start` to start it immediately. You're all set.

Using Webmin is simple. Open your favorite browser, and navigate to `http://localhost:10000` (or the equivalent, `http://127.0.0.1:10000`), and you'll see Webmin's login page. Next, enter the user name and password for the system administrator (in many distributions, that would be root, but Ubuntu and others

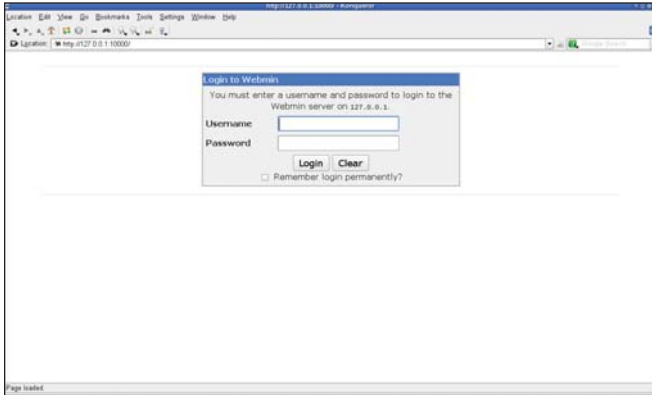


Figure 1. Initial Webmin Login Screen

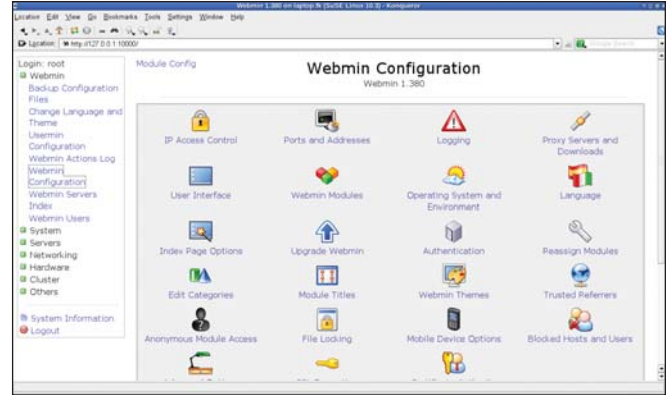


Figure 3. Webmin Configuration Screen

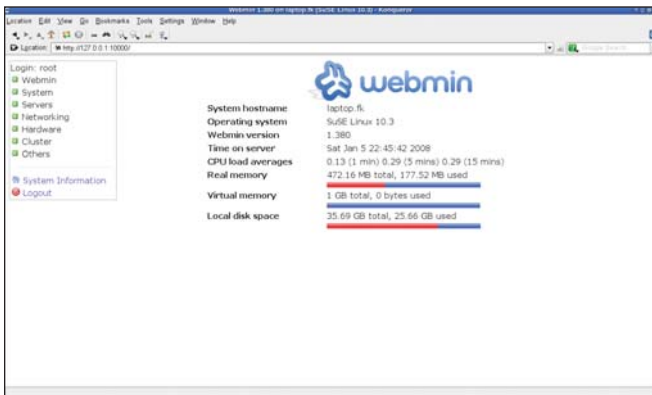


Figure 2. After logging in, you'll see a menu and system information on the screen.

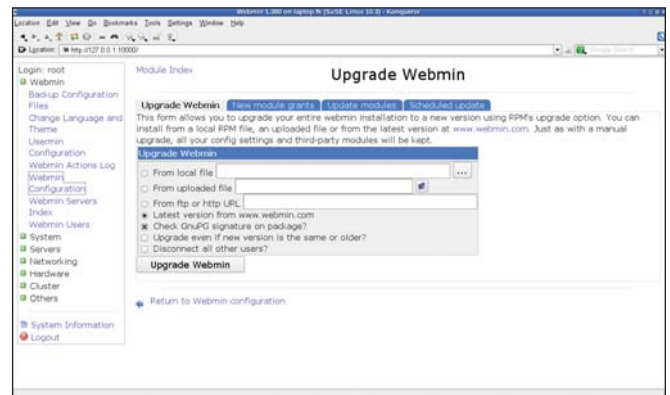


Figure 4. You can upgrade Webmin or add new modules without any other tools.

grant `sysadmin` rights to specific users instead), and click the Login button. You could check the Remember login permanently box, but that's a security risk, so I recommend not doing that.

If you want to save yourself some typing, save that address as a bookmark. For example, in Firefox, either press `Ctrl-D` or go to Bookmarks→Create new bookmark. Alternatively, for even less typing, create a desktop icon. If you use KDE, right-click on your desktop, select Create New→Link to Location (URL), enter the URL above, and click OK. (The process is similar if you use GNOME.) You can make it even snazzier by right-clicking on the newly created icon and changing its image to `/usr/libexec/webmin/images/webmin.xpm` (this path might be different for distributions other than OpenSUSE).

Upgrade

Once you have Webmin installed correctly, upgrading it or adding more modules is a breeze. On the left-side menu, select Webmin→Webmin Configuration, and you'll see a screen full of icons. If you click Upgrade Webmin (the up-pointing blue arrow), you can upgrade Webmin itself from the Internet. Note that you can click on Scheduled Update to set up a cron task that will connect to the Web and download all needed updates on its own. This is a safe option (for you'll definitely get all updates and bug fixes as soon as possible), but it's also an unsafe

one (should the Webmin Web site itself ever be hacked). So, I leave it up to you to decide whether you want to do this.

On the same Webmin Configuration page, if you click the Webmin Modules icon (the one with small boxes), you can browse all available modules on the Webmin site or even download third-party modules from other sites. Choosing the Standard Module option provides a pop-up window with dozens of modules (I haven't been able to figure out whether there's a method to the list's organization). If you click a module name, and then click Install Module, Webmin downloads it and sets it up for you.

Users and Groups

Before moving on, let's talk about security and users. Webmin has its own users, which are not the same as the operating system users. The very first time you log in, it automatically creates a root user. You shouldn't let every user work with this account. It's safer if you create specific accounts and restrict each one to needed functions. To do this, click Webmin on the left-side menu, and then Webmin Users.

When adding users, you can opt to give them a specific Webmin password or use "Unix authentication". The former option is usually safer (but only if users choose a password different from their standard passwords), and the latter option is the friendliest one. The Password Restrictions screen lets you set

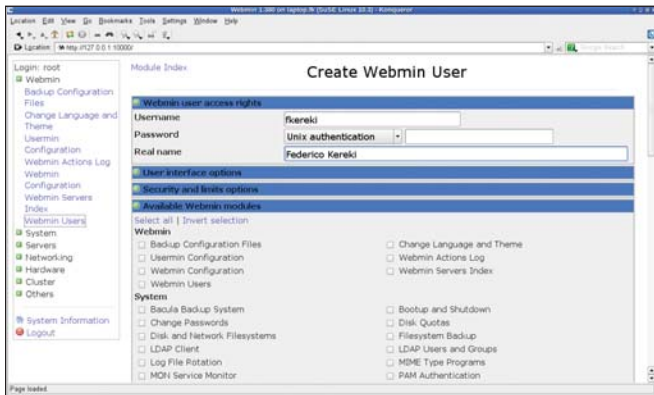


Figure 5. Webmin has its own database of users.

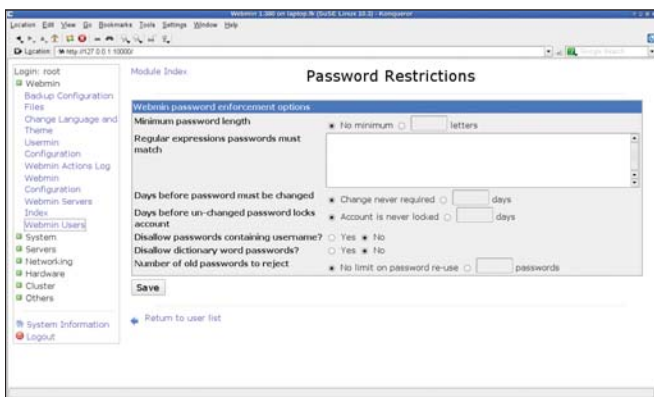


Figure 6. Using password restrictions provides higher security levels.

specific controls, so users can't use too short, simple or easy-to-guess passwords.

Instead of assigning rights to each user, you can create groups. Go to Webmin→Webmin Users, and click Create a new Webmin group. Select what functions should be allowed to members of this group, and finish by clicking Create. From now on, when you create new users, you can specify to which group they belong, and their rights will be assigned automatically.

You also should take a look at the Unix User Synchronization option, which allows the automatic synchronization of Linux users and Webmin users. You can set it up so that every time a Linux user is created/deleted, a corresponding Webmin user also is created/deleted. The Unix User Authentication option also might be of interest if you have many users who should be allowed access to Webmin. Additionally, you can use the View Login Sessions to check whatever the users might have done.

Using Webmin

Using Webmin is quite simple, as you might already have guessed from the examples above. Choose a category from the menu on the left side of the screen, and it opens up, showing a list of available modules. The main page for each module usually includes a Module Config link on its top-left corner, which lets you do some configuration, and a Help link that provides documentation on the module's functions. Here are the categories:

- **Webmin:** provides general configuration, including language and theme selection (you can use Webmin in more than 40 languages), upgrades, module installation, logging options, log browsing and more. If you want to make your installation more secure, check the Authentication option (allowing, among other things, protection against brute-force password-cracking attacks), and also check IP Access Control and Blocked Hosts and Users. If you have the Servers module installed, you can use it to scan for other Webmin servers and administrate them remotely—although it won't be as speedy.
- **System:** covers many different functions. You can control backups with the third-party option for the Bacula backup system or with a far simpler filesystem backup that uses either tar or the dump-and-restore family of commands to save directories to tape or to a file on another filesystem. Bootup and Shutdown lets you specify which services will be run at which levels, and also (obviously) to reboot or shut down the system. For user management, check Users and Groups (which allows you to create, edit or delete both users and groups) and Change Passwords, whose function is obvious. The Disk and Network Filesystems module lets you mount or unmount devices and filesystems, and Disk Quotas will be of interest if you have assigned file space quotas to users. You can schedule commands to run once (think atd) or have periodical jobs (think cron). You can get a top-like display of processes (but it won't refresh on its own) with the Running Processes option, and you can find plenty of information by clicking on a process id. Finally, to cut the list short, the Software Packages option allows you to install or remove a software package on the server remotely.
- **Servers:** this category has to do with all the possible servers you might be running, including Web-related functions, such as Apache or FTP; mail functions (Fetchmail, Postfix, QMail, Sendmail) and filters (ProcMail, SpamAssassin); file sharing (Samba); databases (MySQL, PostgreSQL); network functions (DHCP, SSH, DNS, SLP); proxying (Squid); and several similar functions. There are several options for each of these modules, so you'll want to click on each of them to see the available features.
- **Networking:** covers more-specific network-related options, including configuration (interfaces, routing, gateways, DNS client, host addresses); services; connection (ADSL client, Bandwidth Monitoring, PPP, SSL tunnels, VPN); security (Kerberos5, IPsec); firewalls (the Linux Firewall provides an iptables-based configuration, and there's an option for the Shoreline shorewall firewall too); and more, including NFS and NIS.
- **Hardware:** lets you control disks and volumes (including LVM, RAID and disk partitions; you also can use Smart to check the status of your disk units); printers; CD burning; and the system clock. If you are using GRUB, you can edit its options from here too.
- **Clusters:** includes several options you will use only if you are running two or more machines forming a cluster, with the Heartbeat monitor—a rather more specialized setup, which

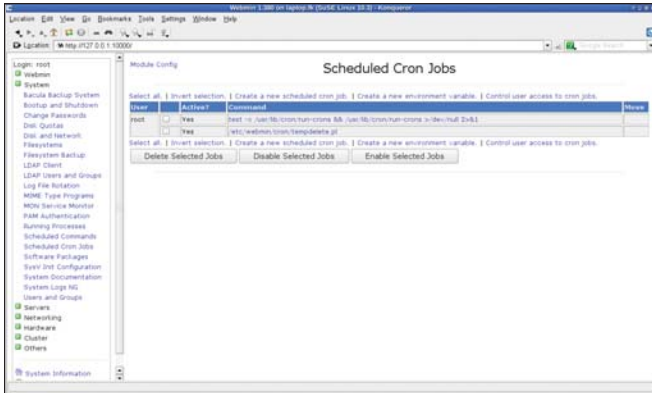


Figure 7. You can manage future (cron) jobs easily.

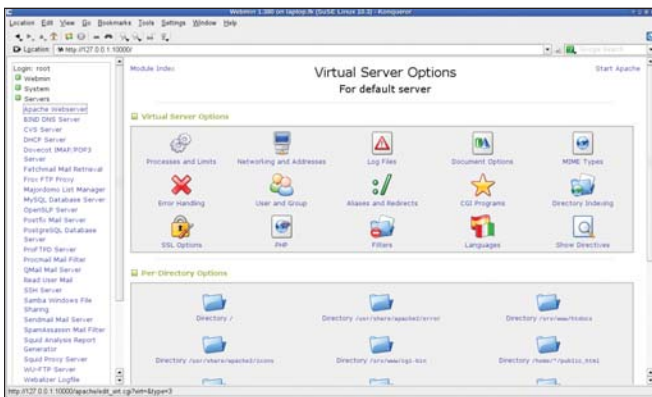


Figure 8. You can configure Apache fully with Webmin. Here, you can edit the default server attributes.

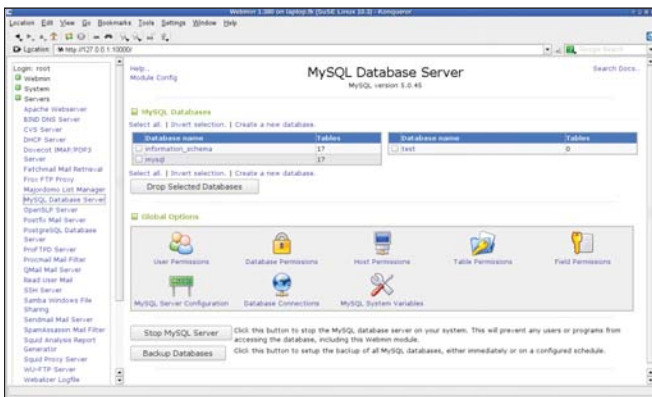


Figure 9. Webmin provides an alternative to PHPMyAdmin for configuring MySQL databases.

proves once again that you need to know what you're doing before starting to mess with Webmin.

- Others: a catch-all for several options, including a command shell (implemented via a Java applet) for full console access, or Custom Commands, which allows you to set up and execute commonly used commands, with optional parameter

Usermin: a Tool for End Users

Usermin is a close relative of Webmin, designed to allow end users to manage several administrative functions on their own, such as changing passwords and user details, managing mail (though a standard e-mail client is a better solution) and more. Usermin is available by default when you install Webmin. You can access it by navigating to `http://127.0.0.1:20000`, where you'll see an interface very much like Webmin's, but with far fewer functions. In fact, you can configure which functions will appear with Webmin. Start that program, go to Webmin→Usermin Configuration→Available Modules, and select which modules should be available via Usermin. You don't need to log in to use Usermin; it will assume the rights of the current user.

substitution—a fine tool if you need to make some commands available to inexperienced users. There also is a File Manager (another Java applet), SSH/Telnet remote login, an HTTP tunnel for accessing Web pages, data files upload and download, and more.

Conclusion

Can you benefit from Webmin? Who should use it? Jamie Cameron, Webmin's creator, said this program "may be better suited for less-experienced users who are unfamiliar with configuration file formats than for enterprise sysadmins who already have a detailed understanding of UNIX". I fully agree with that opinion, although I'd add that even if you are quite familiar with configuration files and the like, you might welcome an easier (and sometimes quicker) way of doing things.

Webmin packs a quite impressive, always growing, number of functions, but it allows you to use only what you require, through clear menus and forms, and it detects possible errors before they can do any harm. You should at least consider it for its learning value, because you can examine configuration files before and after each change, and, thus, learn how something was (or should have been) done. You can't avoid learning about each function before diving in, but Webmin provides at least an easier road to becoming a more proficient sysadmin. ■

Federico Kereki is an Uruguayan Systems Engineer, with more than 20 years' experience teaching at universities, doing development and consulting work, and writing articles and course material. He has been using Linux for many years, having installed it at several different companies. He is particularly interested in the better security and performance of Linux boxes.

Resources

Webmin and Usermin: www.webmin.com

Webmin Download Site: www.webmin.com/download.html

Linux Standard Base: www.linux-foundation.org/en/LSB

Smart: labix.org/smart

Building a Scalable High-Availability E-Mail System with Active Directory and More

First published in *Linux Journal*, issue 163

A large-scale implementation of a scalable Linux e-mail with Active Directory.
JACK CHONGJIE XUE

IN EARLY 2006, Marshall University laid out a plan to migrate HOBBIT (Figure 1), an HP OpenVMS cluster handling university-wide e-mail services. Plagued with increasing spam attacks, this cluster experienced severe performance degradation. Although our employee e-mail store was moved to Microsoft Exchange in recent years, e-mail routing, mailing list and student e-mail store (including IMAP and POP3 services) were still served by OpenVMS with about 30,000 active users. HOBBIT's e-mail software, PMDF, provided a rather limited feature set while charging a high licensing fee. A major bottleneck was discovered on its external disk storage system: the dated storage technology resulted in a limited disk I/O throughput (40MB/second at maximum) in an e-mail system doing intensive I/O operations.

To resolve the existing e-mail performance issues, we conducted brainstorming sessions, requirements analysis, product comparison and test-lab prototyping. We then came up with the design of our new e-mail system: it is named MUMAIL (Figure 2) and uses standard open-source software (Postfix, Cyrus-

IMAP and MySQL) installed on Red Hat Enterprise Linux. The core system consists of a front-end e-mail hub and back-end e-mail store. The front-end e-mail hub uses two Dell blade servers running Postfix on Linux. Network load balancing is configured to distribute load between them. The back-end e-mail store consists of two additional blade servers running a Cyrus-IMAP aggregation setup. Each back-end node is then attached to a different storage group on the EMC Storage Area Network



Figure 1. HOBBIT OpenVMS Cluster Hardware



Figure 2. Linux E-Mail Server Blades and SAN

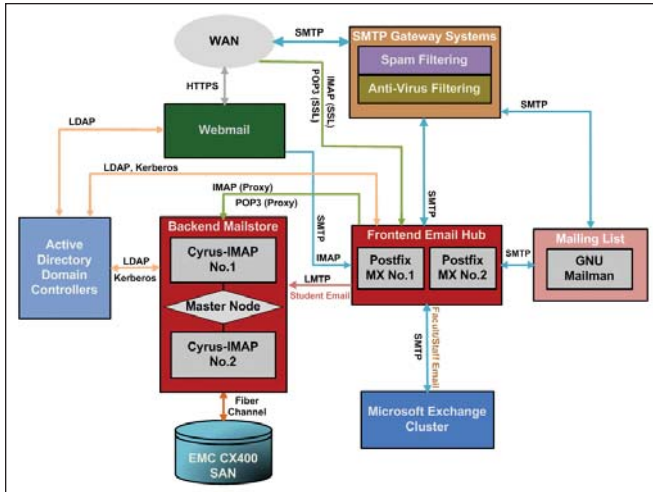


Figure 3. System Architecture

(SAN). A fifth blade server is designated as a master node to store centralized user e-mail settings. Furthermore, we use LDAP and Kerberos to integrate the e-mail user identities with Windows Active Directory (AD).

Figure 3 illustrates our new e-mail system architecture and the subsystem interactions with existing services, which include Webmail, AD and SMTP gateway. The block diagrams highlighted in red are the components to be studied in detail.

Related Solutions

Before we zoom further into our new e-mail system, I want to mention some of the existing Linux/UNIX e-mail solutions in higher education environments. First, the HEC Montreal (HEC) e-mail system discussed in a *Linux Journal* article (see Resources) influenced our design, which is based on Cyrus-IMAP and Postfix. Second, we looked into Cambridge University's solution. It uses custom IMAP proxy front-end servers and multiple pairs of Cyrus-IMAP mail store servers replicating data to each other. Furthermore, Carnegie Mellon University (CMU), which originally developed Cyrus-IMAP, uses Sendmail as the front-end mail exchanger and a Cyrus-IMAP Murder Aggregator setup on the back end. Columbia University moved its e-mail system to a Cyrus-IMAP-based solution in 2006, and the University of Indiana moved to Cyrus back in 2005. Cyrus and Postfix also are used by Stanford University.

Although the designs of these related solutions are different, most of them use a cluster-based approach that separates mail transport/delivery from the mail store. Multiple front-end MTA-MDA (Mail Transport Agent and Mail Delivery Agent) servers are set up to deliver mail to the back-end mail store, which then saves messages either in a filesystem (for example, Maildir) or a database. Most of the solutions use Cyrus-IMAP (on UNIX or Linux) as their mail store server.

Detailed Design

Some distinctive differences set our design apart from the existing solutions:

1. Instead of using a separate directory service (such as

OpenLDAP) for user authentication, our design integrates user identities with Windows Active Directory (AD).

2. Rather than using an LDAP server to store user e-mail routing settings, we designed a relational database to store these settings.
3. In the mail store setup, instead of using an active-passive high-availability cluster setup, like the HEC approach or the Cyrus replication approach developed at Cambridge, we deployed the Cyrus-Murder Aggregator. Unlike the CMU Cyrus Aggregator server allocation, which uses separate MTA server nodes, we consolidate both MTA and Cyrus Proxy functions to run on our front-end mail hub nodes.

We designed an e-mail user database (running MySQL on the Master node) to serve as a centralized data store for information including e-mail accounts, user e-mail routing, group aliases and mailing lists. Web-based user interfaces were developed using PHP to allow users to make changes to their settings in the database. Automated scripts running on the front-end nodes will query the database for user settings and build Postfix MAPs to apply these settings.

A Postfix server can be thought of as routers (not for IP packets but for e-mail). For each e-mail message, Postfix looks at the destination (envelope recipient) and the source (envelope sender) and then chooses how to route the e-mail message closer to its destination. Look-up tables called Maps (such as Transport, Virtual, Canonical and Alias Maps) are used to find the next-hop e-mail delivery location or apply e-mail address re-rewrites.

A background job is running on each of the front-end e-mail hub nodes to "pull" the e-mail settings (delivery location, e-mail alias and group alias information) stored in the e-mail user database to the Postfix maps (aliases, virtual, canonical and transport). Written in Perl, the program is configured to run periodically as a cron job.

Our design principle for the new e-mail system is to scale out from a single, monolithic architecture to multiple nodes sharing the same processing load. In a large e-mail environment, scaling out the front-end MTA system is considerably easier compared with scaling out the back-end mail store. As the front-end nodes are essentially data-less, using DNS or IP-based load balancing on multiple front-end servers is a typical practice. However, the same technique cannot be applied to design the back-end mail store where the user data resides. Without clustering, shared storage or additional software components (such as a proxy server), multiple mail store servers cannot share the same IMAP/POP3 process load under a unified service namespace. Because of this, using a single mail store server tends to be an obvious solution. However, one node usually implies elevated server hardware expenses when more powerful server hardware needs to be purchased to accommodate the ever-increasing system load. The price of a mid-range server with four CPUs is usually much higher than the total price of three or more entry-class servers. Furthermore, single-node architecture reduces system scalability and creates a single point of failure.

The Cyrus-IMAP package is proven to be robust and suitable in large settings. It differs from other Maildir or mbox IMAP

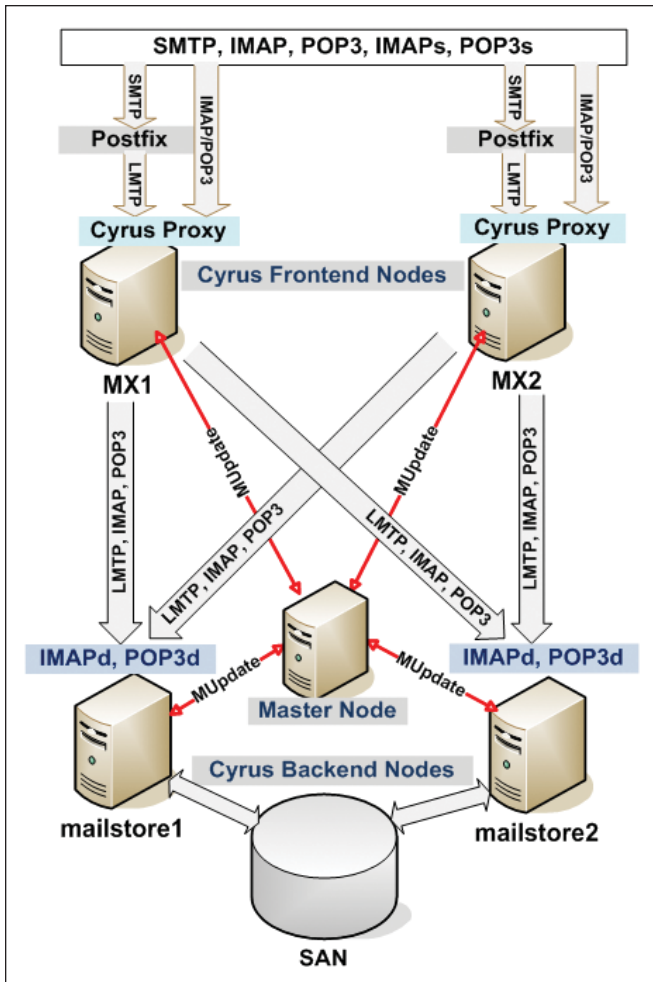


Figure 4. Cyrus-IMAP Aggregation Setup

servers in that it is intended to run as a “sealed” mailbox server—the Cyrus mailbox database is stored in parts of the filesystem that are private to the Cyrus IMAP system. More important, a multiple server setup using Cyrus Murder aggregation is supported. It scales out the system’s load by using multiple front-end IMAP proxies to direct IMAP/POP3 traffic to multiple back-end mail store nodes. Although we found other ways to scale out Cyrus-IMAP—for example, Cambridge University’s pair-wise replication approach, mentioned in the Related Solutions section of this article, or using a clustered filesystem to share IMAP storage partitions between multiple servers with products like Red Hat’s Global File System (GFS)—compared with the aggregation approach, these solutions either are too customized to support (the Cambridge approach) or involve extra cost (GFS is sold separately by Red Hat, Inc.).

So, the Cyrus-IMAP Aggregation approach was adopted. Figure 4 illustrates the setup: two Cyrus back-end servers were set up, and each handles half of the user population. Two Postfix MTA front-end nodes are designated to serve the proxy functions. When e-mail clients connect through SMTP/IMAP/POP3 to the front-end servers, the Cyrus proxy service will communicate with the Cyrus master node using the MUPDATE protocol, so that it gets the infor-

mation about which the Cyrus back-end node stores e-mail for the current client. Furthermore, the back-end Cyrus nodes will notify the master node about the mailbox changes (creating, deleting and renaming mailboxes or IMAP folders) in order to keep the master updated with the most current mailbox location information. The master node replicates these changes to the front-end proxy nodes, which direct the incoming IMAP/POP3/LMTP traffic. The MUPDATE protocol is used to transmit mailbox location changes.

Although it is not a fully redundant solution (the master node is still a single point of failure), and half of our users will suffer a usage outage if either one of the back-end nodes is down, the aggregator setup divides the IMAP processing load across multiple servers with each taking 50% of the load. As a result of this division of labor, the new mail store system is now scalable to multiple servers and is capable of handling a growing user population and increasing disk usage. More back-end Cyrus nodes can join with the aggregator to scale up the system.

Integration with Active Directory

One of the requirements of our new e-mail system is to integrate user identities with the university directory service. Because Microsoft Active Directory services have been made a standard within our centralized campus IT environment, Cyrus (IMAP/POP3) and Postfix (SMTP) are architected to obtain user authentication/authorization from AD. After the integration, all e-mail user credentials can be managed from AD. Most directory services are constructed based on LDAP. AD uses LDAP for authorization, and it has its own Kerberos implementation for authentication. The goal of an integrated AD authentication is to allow the Linux e-mail servers to use AD to verify user credentials. The technology used to support the AD integration scheme is based mainly on the Kerberos and LDAP support, which come with native Linux components, as shown in Figure 5.

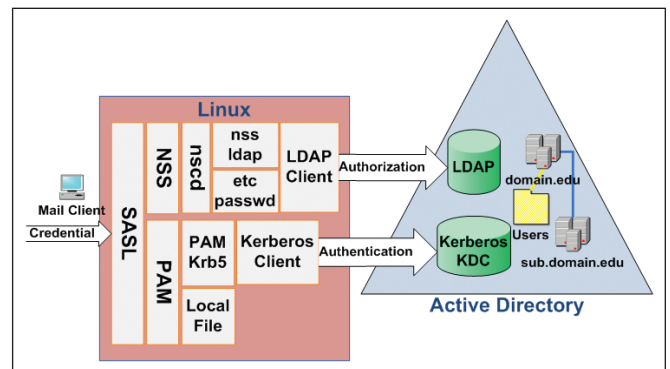


Figure 5. Linux Authentication and Authorization Against AD

Here is how it works. First, we use AD Kerberos to authenticate Linux clients. Pluggable Authentication Module (PAM) is configured to get the user credentials and pass them to the pam_krb5 library, which is then used to authenticate users using the Linux Kerberos client connection to the Key Distribution Center (KDC) on Active Directory. This practice eliminates the need for authentication administration on the Linux side. However, with only the Kerberos integration, Linux has to store

authorization data in the local `/etc/passwd` file. To avoid managing a separate user authorization list, LDAP is used to retrieve user authorization information from AD. The idea is to let authorization requests processed by Name Service Switch (NSS) first. NSS allows the replacement of many UNIX/Linux configuration files (such as `/etc/passwd`, `/etc/group` and `/etc/hosts`) with a centralized database or databases, and the mechanisms used to access those databases are configurable. NSS then uses the Name Service Caching Daemon (NSCD) to improve query performance (NSCD is a daemon that provides a cache for the most common name service requests). This can be very important when used against a large AD user container. Finally, NSS_LDAP is configured to serve as an LDAP client to connect to Active Directory to retrieve the authorization data from the AD users container. (NSS_LDAP, developed by PADL, is a set of C library extensions that allow LDAP directory servers to be used as a primary source of aliases, ethers, groups, hosts, networks, protocol, users, RPCs, services and shadow passwords.) Now, with authorization and authentication completely integrated with AD using both LDAP and Kerberos, no local user credentials need to be maintained.

In order to support LDAP authorization integration with Linux, Windows Server 2003 Release 2 (R2), which includes support for RFC 2307, is installed on each of the AD domain controllers. R2 introduces new LDAP attributes used to store UNIX or Linux user and group information. Without an extended AD LDAP schema, like the one used by R2, the Linux automatic authorization integration with AD is not possible. It is also important to mention that the SASL Authentication layer shown in Figure 3 is using Cyrus-SASL, which is distributed as a standard package by Carnegie Mellon University. The actual setup uses PAM for authenticating IMAP/POP3 users. It requires the use of a special Cyrus daemon, `saslauthd`, which the SASL mechanism uses to communicate via a Linux-named socket.

Conclusion

Our new e-mail system is mostly based on open-source software. The incorporation of Postfix, Cyrus-IMAP and MySQL helped to fulfill most of the system requirements. From the hardware perspective, the technologies used, such as Storage Area Network (SAN), blade server and the Intel x86_64 CPUs, helped to meet the requirements of fast access, system scalability and high availability. However, the use of open-source software and new hardware technologies may introduce new management overhead. Although all the open-source software packages used on the new system are mature products, compared with commercial software, they typically lack a GUI for system management. Their configuration and customization are completely based on a set of plain-text configuration files. Initially, this may present a learning curve, as the syntax of these configuration files must be studied. But, once the learning curve is passed, future management easily can be automated, as scripts can be written to manage the configuration parameters and store them in a centralized location. On the hardware side, complex settings also may imply complex network and server management settings, which also may introduce overhead during system management. However, the benefits of using the technologies discussed outweigh the complexities and learning curves involved. It is

easy to overcome the drawbacks through proper design, configuration management and system automation.

At the time of this writing, our new Linux e-mail system (MUMAIL) has been running in production for ten months. The entire system has been running in a stable state with minimal downtime throughout this period. All user e-mail messages originally on HOBBIT were moved successfully to MUMAIL in a three-day migration window with automated and non-disruptive migration processes. Users now experience significantly faster IMAP/POP3 access speed. Their e-mail storage quota is raised from 20MB to 200MB, and there is potential to increase the quota to a higher number (1GB). With the installation of gateway-level spam/virus firewalls as well as increased hardware speed, no e-mail backlog has been experienced on MUMAIL during recent spam/virus outbreaks. With an Active Directory integrated user authentication setup, user passwords or other sensitive information are no longer stored on the e-mail system. This reduces user confusion and account administration overhead and increases network security. Mail store backup speed is improved significantly with faster disk access in the SAN environment. Finally, the new system has provided a hardware and software environment that supports future growth with the adoption of a scalable design. More server nodes—both front end and back end—and storage can be added when system usage grows in the future. ■

Jack Chongjie Xue holds a Masters' degree in Information Systems from Marshall University, where he did Linux and Windows systems administration work. His job duties now include developing Business Intelligence applications and working on data mining projects at Marshall.

Resources

“Migration of Alcatel C-Mod Computer Infrastructure to Linux” by T. W. Fredian, M. Greenwald and J. A. Stillerman: www.psfc.mit.edu/~g/papers/fed04.pdf

“HEC Montréal: Deployment of a Large-Scale Mail Installation” by Ludovic Marcotte: www.linuxjournal.com/article/9323

Cyrus-IMAP Aggregation: cyrusimap.web.cmu.edu/ag.html

“Scaling up Cambridge University’s E-Mail Service” by David Carter and Tony Finch: www-uxsup.csx.cam.ac.uk/~fanf2/hermes/doc/talks/2004-02-ukuug/paper.html

CMU’s Cyrus-IMAP Configuration: cyrusimap.web.cmu.edu/configuration.html

Columbia’s Cyrus-IMAP Move: www.columbia.edu/cu/news/05/12/cyrus.html

Indiana’s Cyrus-IMAP information: uitspress.iu.edu/040505_cyrus.html

Stanford’s E-Mail System Discussion: www.stanford.edu/dept/its/vision/email.html

Windows Security and Directory Services for UNIX Guide: www.microsoft.com/downloads/details.aspx?familyid=144f7b82-65cf-4105-b60c-44515299797d&displaylang=en

“Toward an Automated Vulnerability Comparison of Open-Source IMAP Servers” by Chaos Golubitsky: www.usenix.org/events/lisa05/tech/golubitsky/golubitsky.pdf

Distributed Compiling with distcc

First published in *Linux Journal*, issue 163

You don't need a cluster to get cluster-like performance out of your compiler.

JES HALL

ONE OF THE most frustrating aspects of open-source development is all the time spent waiting for code to compile. Right now, compiling KDE's basic modules and libraries on a single machine takes me around three hours, and that's just to get a desktop. Even with a Core 2 Duo, it's a lot of time to sit around and wait.

With another pair of Core Duo machines at my disposal, I'd love to be able to use all of their processing power combined. Enter distcc.

distcc is a program that allows one to distribute the load of compiling across multiple machines over the network. It's essentially a front end to GCC that works for C, C++, Objective C and Objective C++ code. It doesn't require a large cluster of compile hosts to be useful—significant compile time decreases can be seen by merely adding one other similarly powered machine. It's a very powerful tool in a workplace or university environment where you have a lot of similar workstations at your disposal, but one of my favourite uses of distcc is to be able to do development work on my laptop from the comfort of the café downstairs

It doesn't require a large cluster of compile hosts to be useful—significant compile time decreases can be seen by merely adding one other similarly powered machine.

and push all the compiles up over wireless to my more powerful desktop PC upstairs. Not only does it get done more quickly, but also the laptop stays cooler.

It's not necessary to use the same distribution on each system, but it's strongly recommended that you use the same version of GCC. Unless you have set up cross-compilers, it's also required that you use the same CPU architecture and the same operating system. For example, Linux (using ELF binaries) and some BSDs (using a.out) are not, by default, able to compile for each other. Code can miscompile in many creative and frustrating ways if the compilers are mismatched.

Installation

The latest version of distcc, at the time of this writing, is 2.18.3. There are packages for most major distributions, or you can

download the tarball and compile it. It follows the usual automake procedure of `./configure; make; make install`; see the README and INSTALL files for details.

distcc needs to be called in place of the compiler. You simply can export `CC=distcc` for the compilers you want to replace with it, but on a development workstation, I prefer something a little more permanent. I like to create symlinks in `~/bin`, and set it to be at the front of my PATH variable. Then, distcc always is called. This approach used to work around some bugs in the version of ld that was used in building KDE, and it is considered to have the widest compatibility (see the distcc man page for more information):

```
mkdir ~/bin
for i in cc c++ gcc g++; do ln -s `which distcc` ~/bin/$i; done
```

If `~/bin` is not already at the beginning of your path, add it to your shellrc file:

```
export PATH=~/bin:$PATH
setenv PATH ~/bin:$PATH
```

for bourne- and C-compatible shells, respectively.

Client Configuration

Each client needs to run the distcc daemon and needs to allow connections from the master host on the distcc port (3632). The daemon can be started manually at boot time by adding it to `rc.local` or `bootmisc.sh` (depending on the distribution) or even from an `inetd`. If distccd is started as an unprivileged user account, it will retain ownership by that UID. If it is started as root, it will attempt to change to the distcc or nobody user. If you want to start the daemon as root (perhaps from an init script) and change to a user that is not distcc or nobody, the option `-user` allows you to select which user the daemon should run as:

```
distccd -user jes -allow 192.168.80.0/24
```

In this example, I also use the `-allow` option. This accepts a hostmask in common CIDR notation and restricts distcc access to the hosts specified. Here, I restrict access only to servers on the particular subnet I'm using on my home network—machines with addresses in the 192.168.80.1–192.168.80.254 range. If you are particularly security conscious, you could

Code can miscompile in many creative and frustrating ways if the compilers are mismatched.

restrict it to a single address (192.168.80.5) or any range of addresses supported by this notation. I like to leave it pretty loose, because I often change which host is the master depending on what I'm compiling and when.

Compiling

Back on the master system on which you plan to run your compiles, you need to let distcc know where the rest of your cluster is. There are two ways of achieving this. You can add the hostnames or IP addresses of your cluster to the file `~/distcc/hosts`, or you can export the variable `DISTCC_HOSTS` delimited by whitespace. These names need to resolve—either add the names you want to use to `/etc/hosts`, or use the IP addresses of the hosts if you don't have internal DNS:

```
192.168.80.128 192.168.80.129 localhost
```

The order of the hosts is extremely important. distcc is unable to determine which hosts are more powerful or under less load and simply distributes the compile jobs in order. For jobs that can't be run in parallel, such as configure tests, this means the first host in the list will bear the brunt of the compiling. If you have machines of varying power, it can make a large difference in compile time to put the most powerful machines first and the least powerful machine last on the list of hosts.

Depending on the power of the computer running distcc, you may not want to include localhost in the list of hosts at all. Localhost has to do all of the preprocessing—a deliberate design choice that means you don't need to ensure you have the same set of libraries and header files on each machine—and also all of the linking, which is often hugely processor-intensive on a large compile. There is also a certain small amount of processing overhead in managing shipping the files around the network to the other compilers. As a rule of thumb, the distcc documentation recommends that for three to four hosts, localhost probably should be placed last on the list, and for greater than five hosts, it should be excluded altogether.

Now that you have your cluster configured, compiling is very similar to how you would have done it without distcc. The only real difference is that when issuing the make command, you need to specify multiple jobs, so that the other machines in the cluster have some work to do.

As a general guide, the number of jobs should be approximately twice the number of CPUs available. So, for a setup with three single-core machines, you would use `make -j 6`. For three dual-core machines, you would use `make -j 12`. If you have removed localhost from your list of hosts, don't include its CPU or CPUs in this reckoning.

distcc includes two monitoring tools that can be used to watch the progress of compile jobs. The console-based `distccmon-text` is particularly excellent if your master host is being accessed via SSH. As the user the compile job is running as, execute the command `distccmon-text $s`, where `$s` is the number of seconds at which you would like it to refresh. For example, the following:

```
distccmon-text 5
```

updates your monitor every five seconds with compile job information.

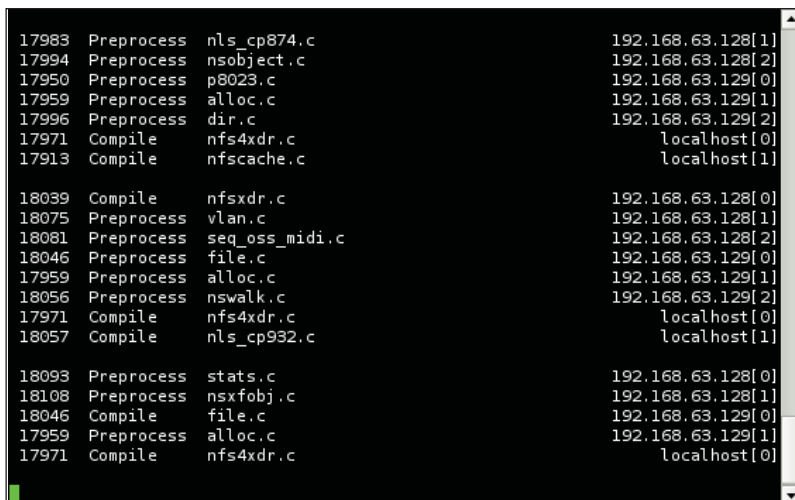


Figure 1. distccmon-text Monitoring a Compile

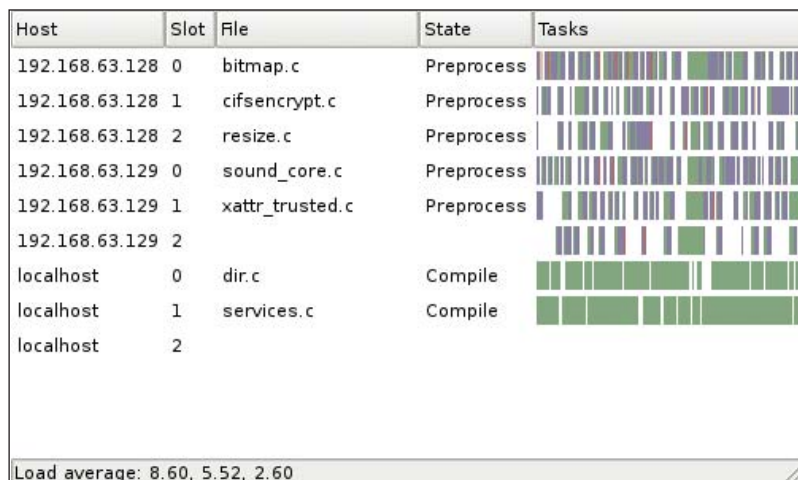


Figure 2. Graphical distcc Monitoring

The graphical `distccmon-gnome` is distributed as part of `distcc` if you compile from source, but it may be a separate package depending on your distribution. It provides similar information in a graphical display that allows you to see at a glance which hosts are being heavily utilised and whether jobs are being distributed properly. It often takes a few tries to get the order of hosts at the most optimal—tools like `distccmon-gnome` make it easier to see whether machines are being under- or over-utilised and require moving in the build order.

Security

`distcc` relies on the network being trusted. Anyone who is able to connect to the machine on the `distcc` port can run arbitrary commands on that host as the `distcc` user. It is vitally important that `distccd` processes are not run as `root` but are run as the `distcc` or `nobody` user. It's also extremely important to think carefully about your `-allow` statements and ensure that they are locked down appropriately for your network.

In an environment, such as a home or small workplace network, where you're securely firewalled off from the outside world and people can be made accountable for not being good neighbours on the network, `distcc` is *secure enough*. It's extremely unlikely that anyone could or would exploit your `distccd` hosts if they're not running the `dæmon` as `root` and your `allow` statements limit the connecting

The console-based `distccmon-text` is particularly excellent if your master host is being accessed via SSH.

machines appropriately.

There is also the issue that those on the network can see your `distcc` traffic—source code and object files on the wire for anyone to reach out and examine. Again, on a trusted network, this is unlikely to be a problem, but there are situations in which you would not want this to happen, or could not allow this to happen, depending on what code you are compiling and under what terms.

On a more hostile network, such as a large university campus or a workplace where you know there is a problem with security, these could become serious issues.

For these situations, `distcc` can be run over SSH. This ensures both authentication and signing on both ends, and it also ensures that the code is encrypted in transit. SSH is typically around 25% slower due to SSH encryption overhead. The configuration is very similar, but it requires the use of `ssh-keys`. Either passphraseless keys or an `ssh-agent` must be used, as you will be unable to supply a password for `distcc` to use. For SSH connections, `distccd` must be installed on the clients, but it must not be listening for connections—the `dæmons` will be started over SSH when needed.

First, create an SSH key using `ssh-keygen -t dsa`, then add it to the target user's `~/.ssh/authorized_keys` on your `distcc` hosts. It's recommended always to set a passphrase on an SSH

key for security.

In this example, I'm using my own user account on all of the hosts and a simple bash loop to distribute the key quickly:

```
for i in 192.168.80.120 192.168.80.100; do cat ~/.ssh/id_dsa.pub
  >| ssh jes@$i 'cat - >> ~/.ssh/authorized_keys'; done
```

To let `distcc` know that it needs to connect to the hosts under SSH, modify either the `~/.distcc/hosts` file or `$DISTCC_HOSTS` variable. To instruct `distcc` to use SSH, simply add an `@` to the beginning of the hostname. If you need to use a different user name on any of the hosts, you can specify it as `user@host`:

```
localhost @192.168.80.100 @192.168.80.120
```

Because I'm using a key with a passphrase, I also need to start my SSH agent with `ssh-add` and enter my passphrase. For those unfamiliar with `ssh-agent`, it's a tool that ships with OpenSSH that facilitates needing to enter the passphrase for your key only once a session, retaining it in memory.

Now that we've set up SSH keys and told `distcc` to use a secure connection, the procedure is the same as before—simply make `-jn`.

Other Options

This method of modifying the hostname with the options you want `distcc` to honour can be used for more than specifying the connection type. For example, the option `/limit` can be used to override the default number of jobs that will be sent to the `distccd` servers. The original limit is four jobs per host except `localhost`, which is sent only two. This could be increased for servers with more than two CPUs.

Another option is to use `lzo` compression for either TCP or SSH connections. This increases CPU overhead, but it may be worthwhile on slow networks. Combining these two options would be done with:

```
localhost 192.168.80.100/6,lzo
```

This option increases the jobs sent to `192.168.80.100` to six, and it enables use of `lzo` compression. These options are parsed in a specific order, so some study of the man page is recommended if you intend to use them. A full list of options with examples can be found on the `distcc` man page.

The flexibility of `distcc` covers far more than explained here. One popular configuration is to use it with `ccache`, a compiler cache. `distcc` also can be used with `crossdev` to cross-compile for different architectures in a distributed fashion. Now, your old SPARC workstation can get in on the act, or your G5 Mac-turned-Linux box can join the party. These are topics for future articles though; for now, I'm going to go play with my freshly compiled desktop environment. ■

Jes Hall is a UNIX systems consultant and KDE developer from New Zealand. She's passionate about helping open-source software bring life-changing information and tools to those who would otherwise not have them.

Faster Web Applications with SCGI

First published in *Linux Journal*, issue 158

Speed up your Web applications with SCGI. | JEROEN VERMEULEN

IF YOU'RE OPERATING a Web server, chances are, you're not merely serving up static text and images. You're likely to be running some Web applications as well, where pages are generated on the fly by some program or script using CGI (Common Gateway Interface). Think of blogging software, bug trackers, news sites and content management systems—anything that turns the browser from a document viewer into a user interface. And, you probably write or at least tweak some of these yourself.

This article shows how to build faster Web applications using an alternative to CGI called SCGI (Simple Common Gateway Interface). SCGI is a protocol, not just a program, but its authors also provide a reference implementation, which is what we use here. It includes modules to use SCGI from Apache or `lighttpd` and Python classes to help you create SCGI applications. Implementations in other languages are available, but we examine the combination of Apache 2.x and Python here.

Where Does the Time Go?

Normally, a Web application runs briefly, but very frequently, in child processes of the Web server. When a client requests a page, the Web server consults its configuration and finds that the request should go to the application. It delegates the request to a child process, which in turn loads and runs the application program. The program may be a binary or a script in Perl, Python or PHP, shell commands, or just about anything else. The CGI standard defines how the program receives details about the request, including requested URL, requested body, authenticated user identity and originating IP address. The program reads these, produces a page in answer to the client's request, and exits. All this happens again at the next request.

Loading, running and exiting programs can be costly. It does make sense for sloppy programs: they may use memory without ever freeing it up again, for instance. In that case, you want the program to run briefly and then let the operating system clean up after it. But, with today's popular languages—Perl, Python, PHP, Java and shell scripts—there really aren't many problems with this. A well-written application really should be able to handle multiple requests in a single run.

Faster Service with SCGI

SCGI lets your program start once and continue servicing requests for as long as it likes. It works like this: a separate server process, called an SCGI server, runs separately from the Web

server and manages one Web application. The Web server forwards all requests for that application to the application's SCGI server. It passes on details about the request in much the same form as in regular CGI.

The SCGI server delegates the request to a child process, just like the Web server did with a regular CGI application. The child process also runs the application, but that's where the similarity ends. Instead of exiting after it's done with that one request, the application can sit and wait for a new one. Each of the SCGI server's child processes runs one instance of the application, each sleeping until there is work for it to do.

The SCGI server spawns a new child process when none are available to take on the latest request—up to a configurable maximum, of course. It also cleans up crashing or exiting child processes, so your Web application can still bail out if things go wrong. But, most of the time, when a request arrives, the application is ready and waiting for it. That's why Ruby on Rails, the Web application framework, comes with the option to run on SCGI; it would be too slow otherwise.

Other Advantages

If the speedup isn't enough for you, there's more. The SCGI server process can be running on the same system as the Web server, but it doesn't have to be. You can offload the server by delegating some Web applications to separate systems, preferably behind a firewall where only the Web server can access them.

Even with just a single server, you can use SCGI to contain vulnerabilities. A normal CGI application starts out running under the same user identity as the Web server process. If an attacker manages to subvert a normal CGI application, your entire Web site may be at risk. An SCGI server, on the other hand, can run under its own user identity, so it can't easily affect the Web server or other applications even if it does run amok. Conversely, you don't need to give the Web server access to the application's code or data anymore; only the application as run by the SCGI server needs access. Everyone else must go through the Web server, which in turn talks to the SCGI server.

You also can run an application in a chroot environment or a virtualized server. With CGI, that quickly becomes expensive and hard to manage. When using SCGI, you start only one server process in your isolated environment—whether it's a chroot jail, a virtualized server, a different user identity or another machine—and the entire application will stay there.

Installing SCGI

You need two components: the Python classes for building SCGI applications and a module for your Web server to make it “speak SCGI” to the applications. If you use Red Hat package management (RPM), you can install these using `yum install python-scgi apache2-mod_scgi`; users of Debian’s `apt` can use `apt-get install python-scgi libapache2-mod-scgi`.

You also can install either component by hand. The Apache module requires a C compiler and Apache’s `apxs` script. Some distributions keep `apxs` in a separate development package rather than installing it as part of the regular Apache package.

Assuming you now have those components, next download the source tarball `scgi-1.12.tar.gz`, and run the commands shown in Listing 1.

Listing 1. Installing SCGI by Hand

```
# Unpack source directory scgi-1.12 from tarball
tar xzf scgi-1.12.tar.gz
cd scgi-1.12
# Build the Python part
python setup.py build
# Install Python module; we'll need root privileges
sudo python setup.py install
# Now build and install the Apache module
cd apache2
sudo make install
# Enable the SCGI module in Apache. This may fail,
# depending on your Apache version, but no matter.
sudo a2enmod scgi
# Make Apache's new configuration take effect
sudo /etc/init.d/apache2 force-reload
```

Test Run

Now, let’s make sure it all works. The Python package is a module with some classes, and normally, you’d write your application as a program that imports that module. For debugging, however, you also can run it as a standalone application. When it receives a request from the Web server, it simply prints the request’s details as a text page. Perfect for a first test—no coding required!

Find the `scgi_server.py` module on your system. It should be installed in `/usr/lib/python2.4/site-packages/scgi` (the 2.4 may be 2.3 or 2.5 on your system). Then, run the module:

```
cd /usr/lib/python2.4/site-packages/scgi
python scgi_server.py
```

This listens for requests from the Web server on a TCP port on your system, using port 4000 by default. You can make it listen on a different port by passing the desired port number as a command-line argument, such as:

```
python /usr/lib/python2.4/site-packages/scgi/scgi_server.py 63000
```

The module keeps running until you kill it, so start it in a separate shell. Remember, you don’t need to run an SCGI server as root or even under the Web server’s identity.

Now that the SCGI application is waiting for requests, pick a location on your Web site to delegate to the application. Let’s say you want it to answer all requests for `“/scgitest”` on this server. Write an Apache configuration snippet, as shown in Listing 2, to a new file in `/etc/apache2/conf.d`.

The SCGI server doesn’t really need to run on the same

Listing 2. Apache Configuration Snippet

```
# Load the SCGI module. This is really only needed
# if you installed manually and the "a2enmod scgi"
# command failed.
LoadModule scgi_module /usr/lib/apache2/modules/mod_scgi.so

<Location "/scgitest">
    # Enable SCGI
    SCGIHandler On
    # Other properties for /scgitest, such as access
    # control
    # ...
</Location>

# Hostname and port number where SCGI server for
# /scgitest is running.
# Port 4000 on localhost (127.0.0.1) is the default.
SCGIMount /scgitest 127.0.0.1:4000
```

Listing 3. `scgi_server.py` returns request details.

```
SERVER_SOFTWARE: 'Apache'
SCRIPT_NAME: '/scgitest'
REQUEST_METHOD: 'GET'
SERVER_PROTOCOL: 'HTTP/1.1'
QUERY_STRING: ''
CONTENT_LENGTH: '0'
HTTP_ACCEPT_CHARSET: 'UTF-8,*'
HTTP_USER_AGENT: 'Mozilla/5.0'
SERVER_NAME: 'testserver.example.org'
REMOTE_ADDR: '10.99.11.99'
SERVER_PORT: '80'
SERVER_ADDR: '192.0.34.166'
DOCUMENT_ROOT: '/srv/www/'
SERVER_ADMIN: 'webmaster@example.org'
HTTP_HOST: 'testserver.example.org'
REQUEST_URI: '/scgitest'
HTTP_ACCEPT: 'text/html,text/plain,*/*;q=0.5'
REMOTE_PORT: '47088'
HTTP_ACCEPT_LANGUAGE: 'en'
SCGI: '1'
HTTP_ACCEPT_ENCODING: 'gzip,deflate'
```

machine as the Web server, as you can see here. Simply make sure that the SCGI server's port is properly firewalled, so that only your Web server can reach it! That way, your application can be sure that all CGI parameters have been validated by the Web server first. If an attacker could connect directly to your SCGI application, you wouldn't be able to trust that information. The CGI parameter `AUTHENTICATED_USER`, for instance, tells your application that the request comes from a particular logged-in user. You can believe that only if you hear it from a properly configured Web server.

Make Apache reload its configuration with `sudo /etc/init.d/apache2 reload`. Your server should now serve a new location, `/scgitest`, that simply prints your request's CGI parameters when you access it. Verify this by looking it up in a browser. If your server's address is `example.org`, point your browser at `http://example.org/scgitest`. You should see a page that looks like Listing 3.

If that's not what you see, take a look at the shell where you ran the module. It may have printed some helpful error message there. Or, if there is no reaction from the SCGI server whatsoever, the request may not have reached it in the first place; check the Apache error log.

Once you have this running, congratulations—the worst is behind you. Stop your SCGI server process so it doesn't interfere with what we're going to do next.

Writing an Application

Now, let's write a simple SCGI application in Python—one that prints the time.

Even with just a single server, you can use SCGI to contain vulnerabilities.

We import the SCGI Python modules, then write our application as a handler for SCGI requests coming in through the Web server. The handler takes the form of a class that we derive from `SCGIHandler`. Call me unimaginative, but I've called the example handler class `TimeHandler`. We'll fill in the actual code in a moment, but begin with this skeleton:

```
#!/usr/bin/python
import scgi
import scgi.scgi_server

class TimeHandler(scgi.scgi_server.SCGIHandler):
    pass # (no code here yet)

# Main program: create an SCGIServer object to
# listen on port 4000. We tell the SCGIServer the
# handler class that implements our application.
server = scgi.scgi_server.SCGIServer(
    handler_class=TimeHandler,
    port=4000
)
# Tell our SCGIServer to start servicing requests.
# This loops forever.
server.serve()
```

You may think it strange that we must pass the `SCGIServer` our handler class, rather than a handler object. The reason is that server object will create handler objects of our given class as needed.

This first incarnation of `TimeHandler` is still essentially the same as the original `SCGIHandler`, so all it does is print out request parameters. To see this in action, try running this program and opening the `scgitest` page in your browser as before. You should see something like Listing 3 again.

Now, we want to print the time in a form a browser will understand. We can't simply start sending text or HTML; we first must emit an HTTP header that tells the browser what kind of output to expect. In this case, let's stick with simple text. Add the following near the top of your program, right above the `TimeHandler` class definition:

```
import time
def print_time(outfile):
    # HTTP header describing the page we're about
    # to produce. Must end with double MS-DOS-style
    # "CR/LF" end-of-line sequence. In Python, that
    # translates to "\r\n".
    outfile.write("Content-Type: text/plain\r\n\r\n")

    # Now write our page: the time, in plain text
    outfile.write(time.ctime() + "\n")
```

By now, you're probably wondering how we will make our handler class call this function. With SCGI 1.12 or newer, it's

easy. We can write a method `TimeHandler.produce()` to override `SCGIHandler`'s default action:

```
class TimeHandler(scgi.scgi_server.SCGIHandler):
    # (remove the "pass" statement--we've got real
    # code here now)

    # This is where we receive requests:
    def produce(self, env, bodysize, input, output):
        # Do our work: write page with the time to output
        print_time(output)
```

We ignore them here, but `produce()` takes several arguments: `env` is a dict mapping CGI parameter names to their values. Next, `bodysize` is the size in bytes of the request body or payload. If you're interested in the request body, read up to `bodysize` bytes from the following argument, `input`. Finally, `output` is the file that we write our output page to.

If you have SCGI 1.11 or older, you need some wrapper code to make this work. In these older versions, you override a different method, `SCGIHandler.handle_connection()`, and do more of the work yourself. Simply copy the boilerplate code from Listing 4 into the `TimeHandler` class. It will set things up right and call `produce()`, so nothing else changes, and we can write `produce()` exactly as if we had a newer version of SCGI.

Listing 4. Boilerplate Code for SCGI 1.11 or Older

```
# Insert this definition into your handler class:
class TimeHandler(scgi.scgi_server.SCGIHandler):

    # ...

    def handle_connection(self, conn):
        input = conn.makefile("r")
        output = conn.makefile("w")
        env = self.read_env(input)
        bodysize = int(env.get('CONTENT_LENGTH',0))
        try:
            self.produce(env, bodysize, input, output)
        finally:
            output.close()
            input.close()
            conn.close()
```

Once again, run the application and check that it shows the time in your browser.

Next, to make things more interesting, let's pass some arguments to the request and have the program process them. The convention for arguments to Web applications is to tack a question mark onto the URL, followed by a series of arguments separated by ampersands. Each argument is of the form name=value. If we wanted to pass the program a parameter called pizza with the value hawaii, and another one called drink with the value beer, our URL would look something like `http://example.org/scgittest?pizza=hawaii&drink=beer`.

Any arguments that the visitor passes to the program end up in the single CGI parameter `QUERY_STRING`. In this case, the parameter would read "pizza=hawaii&drink=beer". Here's something our `TimeHandler` might do with that:

```
class TimeHandler(scgi.scgi_server.SCGIHandler):
    def produce(self, env, bodysize, input, output):
        # Read arguments
        argstring = env['QUERY_STRING']
        # Break argument string into list of
        # pairs like "name=value"
        arglist = argstring.split('&')

        # Set up dictionary mapping argument names
        # to values
        args = {}
        for arg in arglist:
            (key, value) = arg.split('=')
            args[key] = value

        # Print time, as before, but with a bit of
        # extra advice
        print_time(output)
        output.write(
            "Time for a pizza. I'll have the %s and a swig of
```

```
%s!\n" %
        (args['pizza'], args['drink'])
    )
```

Now the application we wrote will not only print the time, but suggest a pizza and drink as passed in the URL. Try it! You also can experiment with the other CGI parameters in Listing 3 to find more things your SCGI applications can do.

Porting Applications

Once you're comfortable writing programs using SCGI, you may want to try adapting existing applications to use it. Some well-known Web applications, such as MoinMoin (a wiki) and Trac (a wiki-based collaborative development environment), are implemented as Python modules. Both of these examples come with CGI scripts in Python that can be called from Apache. The CGI scripts are very short; they really don't do anything except import the application's modules and invoke a function on them.

If you find an application like that, all you really need to do to make it work with SCGI is take that little bit of Python code and move it into a `produce()` method, as in the examples you've seen here. If you have SCGI 1.12 or newer, you also might want to take a look at an alternative SCGIHandler method, `produce_cglike()`.

Conclusion

That's about all we have room for. If you wonder about how the CGI parameters work, try looking at the CGI standard, which calls them "request meta-variables" (see Resources).

Finally, a word of warning. You'll notice that that last example program dies horribly if you fail to pass the expected arguments. The SCGI server replaces the failing processes, so in this case, there's no real problem. But, this should remind you how careful you need to be when writing Web applications. Never trust the input you receive from outside! If a program can be crashed, someone can probably subvert it or take it out of action. People all over the world do that sort of thing for fun or profit, so take the risk seriously. ■

Jeroen Vermeulen works for the Open Source department of the Thai Software Industry Promotion Agency. He's currently working on Suriyan, a server system for those who don't have time for server systems.

Resources

SCGI Downloads: quixote.python.ca/releases

SCGI Home Page: www.mems-exchange.org/software/scgi

CGI Standard: [ftp.rfc-editor.org/in-notes/rfc3875.txt](ftp://rfc-editor.org/in-notes/rfc3875.txt)

More on SCGI with Python and Apache2: thaiopensource.org/development/suriyan/wiki/UsingScgi

Perl Interface: search.cpan.org/~vipercod/SCGI/lib/SCGI.pm

Lisp Interface: randallsquared.com/download/scgi

Trac: trac.edgewall.com

MoinMoin: moinmoin.wikiwikiweb.de

Hosting solutions in all shapes and sizes

Verio dedicated hosting solutions and a wide variety of managed services make it easy and cost-effective to procure the exact level and type of Internet services your business needs.

Managed hosting

Dedicated hosting services are offered in all shapes and sizes. You can select a prepackaged plan for ultrafast deployment or task Verio's expert engineering staff with designing a custom configuration in line with the applications you plan to run. In either case, you can rest assured your dedicated hosting plan will be hassle-free and will deliver guaranteed uptime, security, data protection, privacy, and stability.

Managed hosting solutions are built on Sun™ and HP ProLiant® servers, and they are offered with a large range of options for processing, memory, and storage resources.

Colocation solutions

Our colocation solutions are ideally suited for businesses that must closely manage their online assets, but demand enterprise-class security.

Server and network protection measures include uninterruptible power supply and redundant network connections. Proprietary software gives you complete on-site and/or remote control over your hardware and applications. Verio colocation solutions are highly flexible and can accommodate your precise space, bandwidth, security, and power requirements. Reliability and performance are guaranteed by the industry's most aggressive SLAs.

Smart Content Delivery

Verio Smart Content Delivery (SCD) was developed to resolve the many application performance problems that prevent businesses from providing reliably fast connectivity. SCD marries the cost benefits of a monitored, redundant, shared architecture with the performance and stability of high-end carrier-class equipment.

Three technology components — load balancing, global server load balancing, and reverse proxy caching — work together to deliver fast, reliable Web content, rich media, and e-commerce transaction processing.

Hosted Exchange

Verio Hosted Exchange service transforms email into a high-performance, real-time messaging and collaboration environment. As a user of this service, you reap the many productivity benefits of Microsoft® Exchange without incurring ownership costs or consuming valuable management resources. Verio Hosted Exchange is offered on a simple, flat per-person, per-month fee.

The solution is SAS 70 certified and is equipped to help your company meet strict new governance requirements. Robust security features include three-tiered antivirus protection, antispyware, and encryption solutions.

Why choose Verio dedicated hosting?

Our HP ProLiant Servers are certified to operate with Microsoft Windows® Server 2003 (R2) and Red Hat® Enterprise Linux® 4.0 operating systems. We offer standard pre-installed software for all Windows Server 2003 packages: IIS 6, POP3, Remote Desktop, Windows 2003 Support Pack, Windows 2003 Resource Kit, Windows 2003 Critical updates (from the date of your server order), and HP Support Pack.

We also offer standard pre-installed software for all Red Hat Enterprise Linux 4.0 packages, including pre-bundled applications. HP ProLiant Servers can be configured and/or modified to meet the requirements of your particular business.

Sun servers from Verio are certified to operate with Solaris™ 10, one of the computing industry's most tested and feature-rich operating systems. Sun servers offer standard pre-installed software and come with mirrored boot drives for the ultimate in resiliency and performance. Sun servers can be configured and/or modified to meet your business requirements.

Supreme service and support

World-class support plays a large role in Verio's delivery of its premium hosting solutions and managed services. Our customer operations and support functions guarantee 100% satisfaction.

- A consultative approach helps Verio's expert support teams understand your needs and quickly provision and deploy the solution.
- Thorough project management coordinates solutions, installations, weekly project meetings, communications planning, testing, and QA.
- Constant monitoring guarantees integrity and performance.
- PowerPortal allows you to monitor the performance of your solution infrastructure.
- SLAs guarantee performance.
- Direct access to certified technical support engineers 24x7x365 — with no queue and no triage to level 1 ticket-takers — ensures your services remain secure, protected, private, and stable.



Build on us.

VERIO

HP ProLiant Servers

HP ProLiant DL140-G3 Base Bundle

Starting At:
\$242/mo

- Dual-Core Intel Xeon CPU
- 1 GB RAM
- 2 x 80GB SATA Hot-Plug Drives
- Certified to operate with Microsoft Windows Server 2003 (R2), and Red Hat Enterprise Linux 4.0 Operating Systems

HP ProLiant DL380-G5 Base Bundle

Starting At:
\$399/mo

- Dual-Core Intel Xeon CPU
- 2 GB Memory
- 2 x 36GB (3G) SAS Hot-Plug Drives
- RAID Controller with Battery-backed Cache (BBWC)
- Redundant Power Supplies
- Certified to operate with Microsoft Windows Server 2003 (R2), and Red Hat Enterprise Linux 4.0 Operating Systems

HP ProLiant DL385-G2 Base Bundle

Starting At:
\$420/mo

- Dual-Core AMD CPU
- 2 GB Memory
- 2 x 36GB (3G) SAS Hot-Plug Drives
- RAID Controller with Battery-backed Cache (BBWC)
- Redundant Power Supplies
- Certified to operate with Microsoft Windows Server 2003 (R2), and Red Hat Enterprise Linux 4.0 Operating Systems

**LEARN MORE ABOUT VERIO DEDICATED SERVERS
& HOSTING AT 1-800-269-3300 OR VISIT
WWW.DEDICATEDSERVER.COM**

Sun Servers

Sun V245 Base Bundle

- 1.5 GHz UltraSPARC IIIi CPU
- 1 GB Memory
- 2 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Redundant Power Supplies
- Solaris 10

Sun T1000 Base Bundle - NEW !!!

- 1 x 6-core 1.0 GHz/3-MB UltraSPARC T1 CPU
- 2 GB Memory
- 2 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Redundant Power Supplies
- Solaris 10

Sun V245 Cluster Base Bundle

- 2 x Sun Fire V245 servers
- 1 x 1.5 GHz UltraSPARC IIIi CPUs
- 1 GB Memory
- 2 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Solaris 10
- VERITAS Foundation Enterprise HA for Oracle
- Cluster Storage: 2 x Sun StorEdge 3320 SCSI Arrays: each with 5 x 146 GB SCSI Drives & dual Ultra320 SCSI RAID controllers

Sun V445 Base Bundle

- 2 x 1.6 GHz UltraSPARC IIIi CPU
- 8 GB Memory
- 2 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Redundant Power Supplies
- Solaris 10

Sun T2000 Base Bundle - NEW !!!

- 1 x 8-core 1.0 GHz/3-MB UltraSPARC T1 CPU
- 4 GB Memory
- 4 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Redundant Power Supplies
- Solaris 10

Sun T2000 Base Bundle - NEW !!!

- 2 x Sun Fire V445 servers
- 2 x 1.6 GHz UltraSPARC IIIi CPUs
- 4 GB Memory
- 4 x 73 GB SAS Hot-Plug Drives
- Hardware RAID Controller
- Solaris 10
- VERITAS Foundation Enterprise HA for Oracle
- Cluster Storage: 2 x Sun StorEdge 3320 SCSI Arrays: each with 5 x 146 GB SCSI Drives & dual Ultra320 SCSI RAID controllers

Sun and Solaris are trademarks of Sun Microsystems, Inc. in the United States and other countries. HP ProLiant is a registered trademark of Hewlett-Packard Company. Microsoft and Windows are registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. Red Hat is a registered trademark of Red Hat, Inc., in the United States and other countries. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries. Verio and the Verio logo are trademarks and/or service marks of Verio Inc. in the United States and other countries. All other names are trademarks or registered marks of their respective owners.
©2007 Verio Inc. All rights reserved

Build on us.
VERIO