Linux-VServer

Resource efficient context isolation

Herbert Pötzl Micah Anderson Björn Steinbrink

Everyone is eager to virtualize their working environment to take advantage of the abstraction layer it provides. Some may require resource isolation for enhanced security, others may need development environments for testing and debugging. Whatever your needs are, virtualization will save you resources through utilizing them more efficiently. This is done by exploiting synergies built on proven technologies, improving availability and reducing downtime, adding scalability through duplication and gaining a certain degree of hardware independence.

Virtualization can be done on different levels, each one with its own advantages and disadvantages

Gains from virtualization

The gains from virtualization are rapidly being uncovered, however the most obvious savings are in maintenance. Maintaining ten virtual instances of a service, application, or system, that are all very similar to each other, is much easier than maintaining ten separate machines, with ten different operating system installations, patch levels, security updates, etc. Keeping all of your virtual instances on one machine is much more resource efficient, and easier to manage.

Different virtualization levels

Virtualization can be done on different levels, each one with its own advantages and disadvantages and each one requiring different implementation techniques. Basically you can virtualize:

- Services (web, mail, ICQ, shell...)
- Applications (desktop, word processing...)
- Userspace (jails, vservers, sandboxes...)
- Hardware (virtual machines, hardware partitions...)

Linux-VServer excels at handling the level of system and application virtualization, by virtualizing exactly those pieces that are required and no more, with as little overhead as possible.

Linux-VServer excels at handling the level of system and application virtualization, by virtualizing exactly those pieces that are required and no more

What "native performance" really means

If we look at virtual machines, whose design includes binary translation or hardware partitioning, to run many instances of different operating systems, or the more recent paravirtualization techniques, like Xen or UML which strive to

Linux-vserver home page

Linux-VServer

Linux VServer Project

[illegible]

News

[illegible]

reach “native performance” inside the virtual machine, you might ask, “why is another approach needed?”

Para-virtualization performance measurements are based on a single unit running in a virtual guest environment. As you add more units, more overhead is incurred. The Linux-VServer project is designed to scale virtual units without incurring this additional overhead.

Let’s see what this actually means by hypothetically putting each service into its own isolated environment. We’d have a virtual unit for a web server, one for the database server, an FTP server, probably a mail server, a shell server, an IMAP server, maybe even some IRC services, etc. Let’s assume we need a dozen different virtual units for our overall “Server” to run.

Reducing the overhead by eliminating the kernel

With Xen or UML you have to provide each unit with a kernel, some memory, disk space, a network, and, of course, some CPU share. This in turn means that you would have about a dozen kernels running, each doing their own file caching, disk buffering, network processing and a bunch of other things that kernels usually do. For example, a syscall to read a file is first processed by the guest kernel, to be then handed upwards and result in an actual I/O by the host kernel, which in turn has to hand back the data to the guest kernel before it reaches the process. Now you might rightfully ask: why would I do that?

- Why add latency and overhead of a dozen running kernels?
- Why buffer and handle the same data many times?
- Why have several network stacks if one is enough?

And this is where Linux-VServer (and, of course, other free and commercial implementations of the same idea) come into play. By virtualizing the interface between processes and the kernel, so that every process (or group of processes) gets a limited view of reality, we can build units very similar to real machines, which can work side by side on the same hardware. Those units can run anything, from a single process to a whole distribution, without the need for a separate kernel, and therefore without the need to process any data twice.

Faster than the real thing?

In a Linux-VServer virtualized environment you don’t have a kernel for each instance, but instead the implementation uses contexts and the mostly unknown Linux Capability System to ensure secure interfacing with the kernel. This means that Linux-VServer does not add invisible overhead for each new guest. Instead, you can expect the same performance in a Guest server as compared with the Host server because processes running in the Guest are talking directly to the kernel itself.

Extending the “chroot” concept

The way this is achieved is through context separation and by applying the well-known concept of a “chroot” to a much larger set of resources than is typically done in traditional “jails”. Although the Linux-VServer implementation uses the tried and true chroot concept, it is important to note that it also resolves some fundamental flaws in chroot itself, therefore resolving any traditional chroot() escapes. These concepts are then applied to context separation so that process namespace and network addressing can be isolated appropriately. Context separation makes processes have scope that prohibit them from interacting in unwanted ways between processes inside the context and processes belonging to other contexts. This means that in a Guest the groups of processes that run there are isolated from the other Guests on the system, as well as from the host system itself.

To complete the virtual environment several kernel interfaces are modified to return "virtualized" information. Virtualized information allows you to have separate servers whose uptime, the host and domain name, machine type and kernel version are all different in respect to its virtual environment. Similar changes are made for context memory availability and disk space, even on a shared partition.

In addition to that, the administrator of the Host can get a lot of useful information regarding the guest, and in turn control the resources available to each guest, by specifying limits and tuning the scheduler to adjust the process priorities or even stop scheduling processes when the context has used up its CPU share.

Sharing resources by "unification"

Resource sharing is further improved by a concept called "unification" which is based on "protected" hard links, which cannot be altered, but unlinked (to allow updates). Files that are common between different Guests are shared in a manner that does not reduce the level of security of the isolation. Files that are not likely to change, such as libraries or binaries are "unified" so that the amount of disk space, inode caches, and memory mappings for shared libraries is reduced. The Linux-VServer unification process performs the necessary steps to find common files and then hard link them between contexts protecting them against unwanted modification while still allowing them to be removed in the process of updating software inside the Guest.

Resource sharing is further improved by a concept called "unification" which is based on "protected" hard links, which can not be altered, but unlinked (to allow updates)

Hardware independence allows for many platforms

Linux-VServer is fairly hardware independent, which makes it available on basically all known Linux platforms, may it be x86 or x86_64, sparc/64, powerpc/64, mips, alpha or more exotic architectures like sh64, ia64, s390, uml and xen (as soon as it gets into mainline). It is available for 2.4 kernels (with the focus more on stability) as well as recent 2.6 kernels (where new enhancements and features are added).

The current development version contains the following features:

- virtual namespace support (like chroot, but more secure)
- configurable context procfs permissions/visibility
- tagged filesystem support (for shared disk limits)
- modification of utsname information
- resource limits (AS, RSS, NPROC, Files, Locks, IPC, etc.)
- socket, process and memory accounting
- token bucket priority scheduler, hard scheduler

Finally, it should be mentioned that Linux-VServer is a non commercial community project and so you are welcome to join the development or participate in any other way you would like to. For more details have a look here (http://linux-vserver.org) or just visit us via IRC on #vserver at irc.oftc.net.

Copyright information

About the author

Herbert Pötzl has studied Computer Sciences and has taught Object Oriented Software Engineering at the Technical University of Vienna. He is currently working as a Consultant for Unix and Linux System Integration and Server Consolidation, and since November 2003 has been the Project Leader for the Linux-VServer Community Project.