

Linux® Quality Assurance Utilizing An Automated Nightly Test System

Jason Baietto

Concurrent Computer Corporation

2881 Gateway Drive, Pompano Beach, FL 33069

Abstract

Linux kernel development happens at an extremely fast pace. Companies that develop products that make use of and enhance the Linux kernel must continuously merge their local source trees with the latest kernel releases or risk missing important bug fixes and new features. In this environment it is critical to have a system in place for continuously monitoring and tracking kernel health. This paper describes the test system developed and used by Concurrent Computer Corporation.

1. Introduction

The open source Linux kernel development process presents a very challenging Quality Assurance problem. Every week dozens of developers around the World provide hundreds of kernel source code changes that are continuously analyzed and often incorporated by Linus Torvalds into “test” versions of the official Linux kernel. These test kernel versions are periodically “blessed” and released as new official versions of the stable Linux kernel series. The release frequency is surprisingly high; new releases generally occur every two to four weeks, and occasionally even more rapidly.

Companies that are doing their own kernel development work need to continually import these frequent releases to ensure compatibility between locally developed features and the ever-changing body of kernel code known as Linux.

Concurrent generally merges all official kernel.org releases within one or two days of their public availability. Failure to keep up with the changes can result in missing key bug fixes. Even worse, locally developed features can get stranded in time, usable only with outdated kernel versions, requiring significant effort to bring the features up-to-date and working again with the most recent kernels.

To better survive in this chaotic environment, Concurrent has developed a comprehensive testing framework called the “Automated Nightly Test System” or ANTS for short. Every night, ANTS builds the complete RedHawk product from source, performs a complete product installation and then runs component tests, system tests and benchmarks on the newly installed systems, providing concise test reports upon completion. This testing is fully automated and is done in parallel on each iHawk model that is sold and supported by Concurrent, including 64-bit AMD Opteron™ based systems.

ANTS has been very useful at Concurrent. With Linux kernel development happening in so many places (both internally and externally), it is critical that problems associated with kernel changes are identified and corrected as quickly as possible after their introduction. As more time elapses between problem introduction and problem detection, the number of possible code changes at fault increases, making the job of determining the root cause of the problem that

much more difficult. ANTS usually discovers kernel bugs within 24 hours of their introduction.

2. Building

The RedHawk product is a collection of several additional RPMs that are installed on top of a previously installed Red Hat system. Of primary importance are the `ccur-kernel` packages which provide alternative real-time kernels for the system. However, the packages containing additional user-level libraries and tools are also very important and also require extensive testing.

The ANTS process begins each night with building all of the RedHawk product RPMs from source. A full set of RPMs is built for each architecture that is supported by RedHawk, currently the IA32 and AMD64 architectures. In addition, full sets of packages are built for each major version of RedHawk that is still supported, currently the 1.4 version and the 2.0 version. At the time of this writing, there are three nightly builds on three different build machines.

Different build machines are used because the RedHawk product is self-hosted, meaning that, for example, the 2.0 version of RedHawk is required to be built on a system that is installed with and is already running RedHawk 2.0 (or at least a very recent preliminary version). This dramatically simplifies the build system because it eliminates the need for cross compilers, and it also ensures that the kernels that are built and shipped are built using the exact same tool-chain that a custom kernel built out of `/usr/src` at a customer's site will use.

A simple crontab entry is used to start each

build. Concurrent uses the locally developed SMS source management system, which provides a “development environment” (a.k.a. copy of the source tree) for each build to occur in. The build first “imports” the code changes that have been committed during the day's engineering efforts and ensures that the build is done with the latest code available. After the import, a simple `make release` in the top-level directory does all the work.

The end result of a nightly build is a directory filled with fresh new RPMs, ready to be installed onto eagerly awaiting test systems.

3. Installing

The nightly built RPMs are installed into “virtual root directories” that reside on the same system that was used to build the RPMs. Installing packages in places other than the root directory of the currently running system is accomplished by using the `rpm` command's `--root` option. The `--root` option issues a `chroot()` system call so that package installation happens within the virtual root directory; package installation scripts affect only the virtual root and do not alter the RPM database on the build system.

A virtual root directory is an exact copy of the root file system that is created and populated during the installation of the Red Hat product. Recall that RedHawk installs on top of a Red Hat system and provides several additional packages, including alternative kernels. After the nightly built RPMs are installed, the virtual root directory becomes a fully installed RedHawk root file system and is ready for action.

At this point various directories full of tests, diagnostics and benchmarks are copied into

the `/var` directories of the virtual root directories. The scripts and tests in these directories will be used in subsequent stages of the ANTS run when test execution begins.

In addition to the nightly built RedHawk product RPMs, packages from other product teams at Concurrent are also installed into each of the virtual root directories. For example, all components of the optional NightStar™ debugging and performance analysis tool suite are installed. This allows automated testing of the NightStar Tools, in addition to the kernel testing that ANTS was primarily designed to perform.

A virtual root directory exists for each test machine that will be performing automated testing. In theory it is possible for multiple test machines to share a single virtual root directory, but as the cost of disk storage is relatively inexpensive it seems that the complexity is not warranted. In addition, separate directories provide better isolation between test systems.

4. Booting

A virtual root directory is used in two ways by a test machine. First, the test machine fetches the kernel to boot out of the virtual root's `/boot`. Second, the test machine mounts the virtual root directory as its root file system. Note that this second task requires that the Linux kernel has been built with the `CONFIG_ROOT_NFS=y` option. All of the RedHawk kernels are built with this option enabled to ensure that the same kernel binaries that are shipped can be tested within the ANTS framework.

A grub boot CD is used to accomplish the first step of fetching the kernel. Grub sets up

a serial console, performs a DHCP broadcast, and then goes into its command prompt mode waiting for commands. Serial console access is essential because ANTS relies heavily on the ability to control the target machine remotely via a serial console. Using DHCP is not strictly necessary, but it allows a single grub CD image to be used on all test machines, and ensures that networking configuration problems will be discovered early on in the boot process.

The ANTS master “boot” script is responsible for rebooting the test systems that will be used for the nightly testing runs (note that these test systems are available for general use by Concurrent kernel developers during the day). The boot script runs on an ANTS host machine, which by convention is the same as the build machine, but that is not necessarily required. The boot script attempts to connect to the serial console of the target test machine and bring the machine down as nicely as possible. It uses a Perl::Expect state machine and can recognize almost every state that the serial console might be in (e.g. `getty login`, `bash shell`, `grub prompt`). However, as a fall back if the boot script is unable to bring the system down cleanly it accesses a network of Western Telematic remote power switches to power cycle the system.

Regardless of how the system is rebooted, the end result is a test machine sitting in a grub command prompt awaiting further instructions. At this point the boot script instructs grub to load the kernel via TFTP from the virtual root directory. It then instructs grub to boot the fetched kernel with the kernel boot parameters required to specify the virtual root directory as the kernel's NFS root directory. Note that the boot option to

have Linux redirect its console to the serial port is also supplied, as this is critical for continued control of the system by ANTS.

5. Testing

After a target system has booted the newly built kernel and mounted the updated virtual root directory it continues the init process until it eventually produces a getty login prompt on the serial console. The boot script, which has been patiently waiting for the login prompt, now goes into action and immediately logs into the console as the root user. The boot script invokes the “ants” testing script (previously placed in `/var`) and then takes another break; the boot script will not run again until all testing is complete. Note that the boot script is running on the ANTS host machine, while the ants script is running on the ANTS target.

The first thing the ants script does is copy all of the test, diagnostic and benchmark directories from `/var` into `/tmp`. The reason for this is that `/tmp` is configured to be a local file system on the test machine's hard drive. This improves test performance, minimizes testing hiccups due to NFS traffic, and avoids NFS file-locking issues that prevent some test suites from passing; it also better matches how customers actually run RedHawk systems. Note that the swap partition is also local to the test machine's hard drive, for obvious reasons.

Once the test directories are copied, the ants script runs the tests. It first executes a set of generic tests that all ANTS test machines run. Then, it executes a machine-specific list of tests, if a machine-specific list exists for the current machine. After that, the night's testing is complete for the system.

Currently, there are 51 directories of locally developed kernel tests that ANTS executes. These directories include tests for all of RedHawk's key features including shielding, frequency-based scheduling, priority inheritance, rescheduling variables, high-resolution accounting, POSIX® timers, processor affinity, CPU control, post-wait services and even the Real-Time Clock Interface Module (RCIM) made possible because every iHawk system is equipped with an RCIM card. There are also tests that collect performance information, including lock hold times and process dispatch latency. Note that the tests are compiled on the target system before they are executed, thus testing the installed development environment of the target system.

After the locally developed kernel tests run, the locally developed “Hawker” diagnostic suite is run on every machine. Hawker is a system-level test suite that uses the NightStar tools in combination with many RedHawk kernel features, and it is a key part of Concurrent's quality assurance and acceptance testing. Every system shipped from Concurrent's manufacturing floor must pass a full Hawker diagnostics run; if it does not pass, it does not ship.

Finally, after Hawker runs, a collection of benchmarks and test suites that are freely available on the Internet are run. The subset of benchmarks and test suites that are run varies from time to time, depending upon how well they support the version of the Linux kernel that RedHawk is currently using (as well as how well they support the version of the user-land tools currently provided by Red Hat). The list of benchmarks and test suites used includes LMBench, the Linux Test Project, the Open POSIX® test suite, and the

Linux Standard Base test suite.

When testing is fully complete the ants script prints out “ANTS TESTING COMPLETED” and exits. The boot script sees the magic string and goes into action again, with two final duties to perform. First, it invokes the “cook” script to process the logged output from all of the tests, diagnostics, benchmarks and test suites that ran during the ANTS run. The cook script produces a nice summary and also a per-directory list of test results; the summary is mailed to the ANTS maintainers and also to an internal newsgroup where it is archived. Finally, the boot script brings the machine back down to the grub prompt, where it will be ready for kernel developers to boot with their test kernels when they arrive in the morning.

6. Complications

Unfortunately, things do not always go according to plan, and there are several obstacles that can prevent an ANTS run from completing smoothly. For example, occasionally a test will start but never complete. Early versions of ANTS simply waited indefinitely, preventing the run from completing. A per-test timeout of 15 minutes was subsequently imposed, after which the test is marked as “failed with timeout” and testing resumes.

A more serious predicament occurs when the kernel panics or hangs during an ANTS run. To resolve this, an overall timeout of 8 hours for the entire ANTS run was imposed. If the boot script does not see the magic string before the timeout, it assumes the worst and reports an ANTS timeout. At this point the ANTS maintainers examine both the host and target logs to discover what happened. One

nice feature of ANTS is that the target log is always available on the host system, because the log is written into the virtual root directory which physically resides on the ANTS host. It is generally always obvious which test caused the crash by looking at the last test to start executing in the target log.

Another snag with early versions of ANTS was the fact that it assumed that it exclusively owned all of the test machines after midnight. This was not popular with kernel developers who were counting on being able to run overnight tests on their test kernels. To resolve this, a test machine reservation system was created. Any test system that is marked as reserved will not participate in the evening's ANTS runs.

An ongoing issue is the lack of support in grub for many of the gigabit Ethernet controllers that are found on Concurrent's high-end iHawk systems. To resolve this, an inexpensive (\$10-\$20) PCI network interface card is installed in each test machine with a separate network cable attached. The RedHawk kernel is intentionally *not* compiled with the support for this PCI card. The result is that grub can be used to TFTP boot the new kernel via the PCI card, but the RedHawk kernel will perform all networking (including mounting of the NFS root directory) via the embedded gigabit controller, thus ensuring that the embedded network controller is exercised and tested during the ANTS run.

Finally, early ANTS runs had intermittent test failures which were eventually tracked down to the fact that the target's system clock was significantly in error. It turned out that the ants script was running tests on the target before the NTP daemon had a chance to set and synchronize the time. The boot script

was changed to wait until the target system had been up for two minutes before invoking the ants script, thus giving the system adequate time to stabilize.

7. Future Directions

The RedHawk product currently ships with three different pre-built kernels. Internally, we refer to these kernels as the “debug”, “trace” and “high-performance” kernel flavors. Currently, individual ANTS test machines are statically assigned one of the three kernel flavors to use during its test runs. The flavor assignment is periodically shuffled manually to ensure that all flavors are tested on all supported product models. In the future, support will be added to automatically round-robin the kernel flavors on a given test machine, to achieve better test coverage.

Tests that fail during the ANTS run are always reported clearly, as there is no ambiguity regarding what constitutes a test failure. However, currently there is no automated benchmark analysis logic in ANTS. This means that human intervention is required periodically to ensure that no regressions in benchmark performance have occurred. In the future, it is hoped that effort can be spent addressing this limitation. One hindrance to this effort is the fact that the result summaries produced by various benchmarks vary widely in output format and most do not seem to have been written with machine parsing in mind.

Hopefully, ANTS will continue to develop and future versions will integrate even more test suites and benchmarks into the test runs.

8. Conclusion

As this paper reveals, a complex integration and test process is needed to ensure that the latest Linux kernel updates are capably merged into a continually engineered and enhanced Linux kernel environment. If such a process is utilized, the enhanced Linux kernel will be a quality, up-to-date operating system. Without it, there is the risk of missing new features and bug fixes, and the possibility that additions that are incorporated will introduce new problems that raise quality issues late in the product development cycle or post-release. Concurrent has addressed these issues with its ANTS system, which regularly monitors and tests the latest the Linux world has to offer merged with Concurrent's real-time deterministic enhancements.

9. References

The following resources were found to be valuable during the development of ANTS:

- Perl::Expect Home Page
<http://sourceforge.net/projects/expectperl/>
- The Linux BootPrompt-HOWTO
<http://www.linux.org/docs/ldp/howto/BootPrompt-HOWTO.html>
- NFS-Root-Client Mini-HOWTO
<http://www.linux.org/docs/ldp/howto/NFS-Root-Client-mini-HOWTO/>
- LMBench - Tools for Performance Analysis
<http://www.bitmover.com/lmbench/>
- Linux Standard Base Testing
<http://www.linuxbase.org/test/>