Real-Time and Performance Improvements in the 2.6 Linux Kernel

William von Hagen

Abstract

Work on improving the responsiveness and real-time performance of the Linux kernel holds even more promise for the future.

The Linux kernel, the core of any Linux distribution, constantly is evolving to incorporate new technologies and to improve performance, scalability and usability. Every new kernel release adds support for new hardware, but major version upgrades of the kernel, such as the 2.6 Linux kernel, go beyond incremental improvements by introducing fundamental changes in kernel internals. Many of the changes to the internals of the 2.6 Linux kernel have a significant impact on the overall performance of Linux systems across the board, independent of hardware improvements. The 2.6 kernel provides substantial improvements in system responsiveness, a significant reduction in process- and thread-related kernel overhead and a commensurate reduction in the time between when a task is scheduled and when it begins execution.

Released in late 2003, the 2.6 kernel now is the core of Linux distributions from almost every major Linux vendor in the enterprise, desktop and embedded arenas. Kernel and system performance are critical to focused markets such as embedded computing, where high-priority tasks often must execute and complete in real time, without being interrupted by the system. However, system performance and throughput in general equally are important to the increasing adoption of Linux on the desktop and the continuing success of Linux in the enterprise server market.

This article discusses the nature of real-time and system parameters that affect performance and highlights the core improvements in performance and responsiveness provided by the 2.6 kernel. Performance and responsiveness remain active development areas, and this article discusses several current approaches to improving Linux system performance and responsiveness as well as to achieving real-time behavior. Kernel and task execution performance for various Linux kernels and projects is illustrated by graphed benchmark results that show the behavior of different kernel versions under equivalent loads.

Latency, Preemptibility and Performance

Higher performance often can be realized by using more and better hardware resources, such as faster processors, larger amounts of memory and so on. Although this may be an adequate solution in the data center, it certainly is not the right approach for many environments. Embedded Linux projects, in particular, are sensitive to the cost of the underlying hardware. Similarly, throwing faster hardware and additional memory at performance and execution problems only masks the problems until software requirements grow to exceed the current resources, at which time the problems resurface.

It therefore is important to achieve high performance in Linux systems through improvements to the core operating system, in a hardware-agnostic fashion. This article focuses on such intrinsic Linux performance measurements.

A real-time system is one in which the correctness of the system depends not only on performing a desired

function but also on meeting a set of associated timing constraints. There are two basic classes of real-time systems, soft and hard. Hard real-time systems are those in which critical tasks must execute within a specific time frame or the entire system fails. A classic example of this is a computer-controlled automotive ignition system—if your cylinders don't fire at exactly the right times, your car isn't going to work. Soft real-time systems are those in which timing deadlines can be missed without necessarily causing system failure; the system can recover from a temporary lack of responsiveness.

In both of these cases, a real-time operating system executes high-priority tasks first, within known, predictable time frames. This means that the operating system cannot impose undue overhead in task scheduling, execution and management. If the overhead of tasks increases substantially as the number of tasks grows, overall system performance degrades as additional time is required for task scheduling, switching and rescheduling. Predictability therefore is a key concept in a real-time operating system. If you cannot predict the overall performance of a system at any given time, you cannot guarantee that tasks will start or resume with predictable latencies when you need them or that they will finish within a mandatory time frame.

The 2.6 Linux kernel introduced a new task scheduler whose execution time is not affected by the number of tasks being scheduled. This is known as an O(1) scheduler in big-O algorithmic notation, where O stands for order and the number in parentheses gives the upper bound of worst-case performance based on the number of elements involved in the algorithm. O(N) would mean that the efficiency of the algorithm is dependent on the number of items involved, and O(1) means that the behavior of the algorithm and therefore the scheduler, in this case, is the same in every case and is independent of the number of items scheduled.

The time between the point at which the system is asked to execute a task and the time when that task actually begins execution is known as latency. Task execution obviously is dependent on the priority of a given task, but assuming equal priorities, the amount of time that an operating system requires in order to schedule and begin executing a task is determined both by the overhead of the system's task scheduler and by what else the system is doing. When you schedule a task to be executed by putting it on the system's run queue, the system checks to see if the priority of that task is higher than that of the task currently running. If so, the kernel interrupts the current task and switches context to the new task. Interrupting a current task within the kernel and switching to a new task is known as kernel preemption.

Unfortunately, the kernel cannot always be preempted. An operating system kernel often requires exclusive access to resources and internal data structures in order to maintain their consistency. In older versions of the Linux kernel, guaranteeing exclusive access to resources often was done through spin-locks. This meant the kernel would enter a tight loop until a specific resource was available or while it was being accessed, increasing the latency of any other task while the kernel did its work.

The granularity of kernel preemption has been improving steadily in the last few major kernel versions. For example, the GPL 2.4 Linux kernel from TimeSys, an embedded Linux and tools vendor, provided both an earlier low-latency scheduler and a fully preemptible kernel. During the 2.4 Linux kernel series, Robert Love of Novell/Ximian fame released a well-known kernel patch that enabled higher preemption and that could be applied to the standard Linux kernel source. Other patches, such as a low-latency patch from Ingo Molnar, a core Linux kernel contributor since 1995, further extended the capabilities of this patch by reducing latency throughout the kernel. A key concept for the TimeSys products and these patches was to replace spin-locks with mutexes (mutual exclusion mechanisms) whenever possible. These provide the resource security and integrity required by the kernel without causing the kernel to block and wait. The core concepts pioneered by these patches now are integral parts of the 2.6 Linux kernel.

Approaches to Real-Time under Linux

Three projects for real-time support under Linux currently are active: the dual-kernel approach used by the RTAI Project and by products from embedded Linux vendors such as FSMLabs; a real-time Linux project hosted by MontaVista, an embedded Linux vendor; and freely available preemptibility and real-time work being done by Ingo Molnar and others, which is discussed openly on the Linux Kernel mailing list and which the MontaVista Project depends upon. In addition to these core kernel projects, other supporting projects, such as robust mutexes and high-resolution timers, add specific enhancements that contribute to a complete solution for real-time applications under Linux.

The dual-kernel approach to real time is an interesting approach to real-time applications under Linux. In this approach, the system actually runs a small real-time kernel that is not Linux, but which runs Linux as its lowest-priority process. Real-time applications specifically written for the non-Linux kernel using an associated real-time application interface execute within that kernel at a higher priority than Linux or any Linux application, but they can exchange data with Linux applications. Although this is a technically interesting approach to running real-time applications while using a Linux system, it avoids the question of general Linux kernel preemption and performance improvements. Therefore, it is not all that interesting from a core Linux development perspective.

MontaVista's Project to further real-time Linux leverages much of the existing work being done by Ingo Molar and other Linux kernel contributors, but it includes some additional prototype patches available only on the MontaVista Web site. The current patches available there are for a release candidate for the 2.6.9 Linux kernel (rc4). Therefore, they did not apply cleanly against official drops of the Linux kernel, which is moving toward 2.6.11 at the time of this writing. As such, the results from this project could not be included in this article.

The real-time, scheduling and preemptibility work being done by Ingo Molnar, the author of the O(1) Linux scheduler, and others has a significant amount of momentum, enhances the core Linux kernel and provides up-to-date patches designed to improve system scheduling, minimize latency and further increase preemptibility.

These patches have an enthusiastic following in the Linux community and include contributions from developers at many different groups and organizations, including Raytheon; embedded Linux vendors such as TimeSys; and from the Linux audio community. These patches provide capabilities such as heightening system responsiveness and minimizing the impact of interrupts by dividing interrupt handling into two parts, an immediate hardware response and a schedulable interrupt processing component. As the name suggests, interrupts are requests that require immediate system attention. Schedulable interrupt handling, more commonly known as soft IRQs, minimizes the impact of interrupts on general system responsiveness and performance.

The illustrations in the next section focus on comparing benchmark results from various vanilla Linux kernels against those obtained by applying the real-time, scheduling and preemptibility patches done by Ingo Molnar and others. These patches are up to date and provide complete, core Linux kernel enhancements that can provide direct benefits to Linux users who want to incorporate them into their projects and products.

The Sample Benchmark

In 2002, the *Linux Journal* Web site published an article titled “Realfeel Test of the Preemptible Kernel Patch”, written by Andrew Webber. This article used an open benchmark called Realfeel, written by Mark Hahn, to compare preemption and responsiveness between the standard Linux 2.4 kernel and a kernel against which Robert Love's preemption patch had been applied. Realfeel issues periodic interrupts and compares the time needed for the computer to respond to these interrupts and the projected optimal response time of the system. The difference between the time when an interrupt request is issued and when it is handled is an

example of latency, and the difference between predicted and actual latency is known as jitter. Jitter commonly is used as a way of measuring and comparing system responsiveness.

This article uses the same benchmark application as Webber's article but imposes substantially more load on the system when measuring results. This is a technique commonly applied when benchmarking real-time operating systems, because even non-real-time operating systems may exhibit low latencies in unloaded or lightly loaded situations. The graphics in the next sections also present the results differently to make it easier to visualize and compare the differences between latency on various Linux kernels.

Benchmark Results

The results in this section were compiled using a medium-strength Pentium-class system with a single 1.7GHz AMD Athlon processor and 512MB of system memory. The system was running the GNOME desktop environment and the system processes associated with the Fedora Core 3 Linux distribution, with up-to-date patches as of Feb 10, 2004. The system kernels tested were a vanilla 2.6.10 Linux kernel, the 2.6.10-1.760_FC3 kernel available as a Fedora Core 3 update, a vanilla 2.6.11-rc3 kernel and a 2.6.11-rc3 kernel with Ingo Molnar's current real-time and preemption patch. All of these kernels were compiled against the same kernel configuration file, modulo new configuration options introduced in the newer kernel sources.

In multiprocessing operating systems such as Linux, the system never is dormant. System processes such as the scheduler always are running. If you are using a graphical user interface (GUI), interfaces such as KDE, GNOME or standard X Window system window managers always are waiting for input events and so on. In order to examine true preemptibility and real-time performance, additional load was imposed on the system by starting various processes while each set of benchmark results were being collected. As mentioned previously, the system was running GNOME with four xterms open—one to run the realfeel benchmark, another to run a script that constantly ran recursive find and ls processes on the system's root partition and two in which 2.6.x Linux kernels, with separate source directories, were being compiled from a clean state.

Figure 1 shows a plot of the results of the Realfeel benchmark run on a stock Fedora Core system for a period of one minute. The system was running kernel version 2.6.10-1.760_FC3, which is a 2.6.10 kernel with various patches and enhancements applied by Red Hat. Each dot in the figure represents the jitter between an interrupt request and its handling. The X axis is the sample time in 1/60 of a second. Negative jitter numbers are displayed when the system responded to the interrupt faster than the projected standard time. As you can see from the figure, a fair number of these interrupt requests were handled exactly as expected, resulting in a visibly dark line along the 0 value of the Y axis.

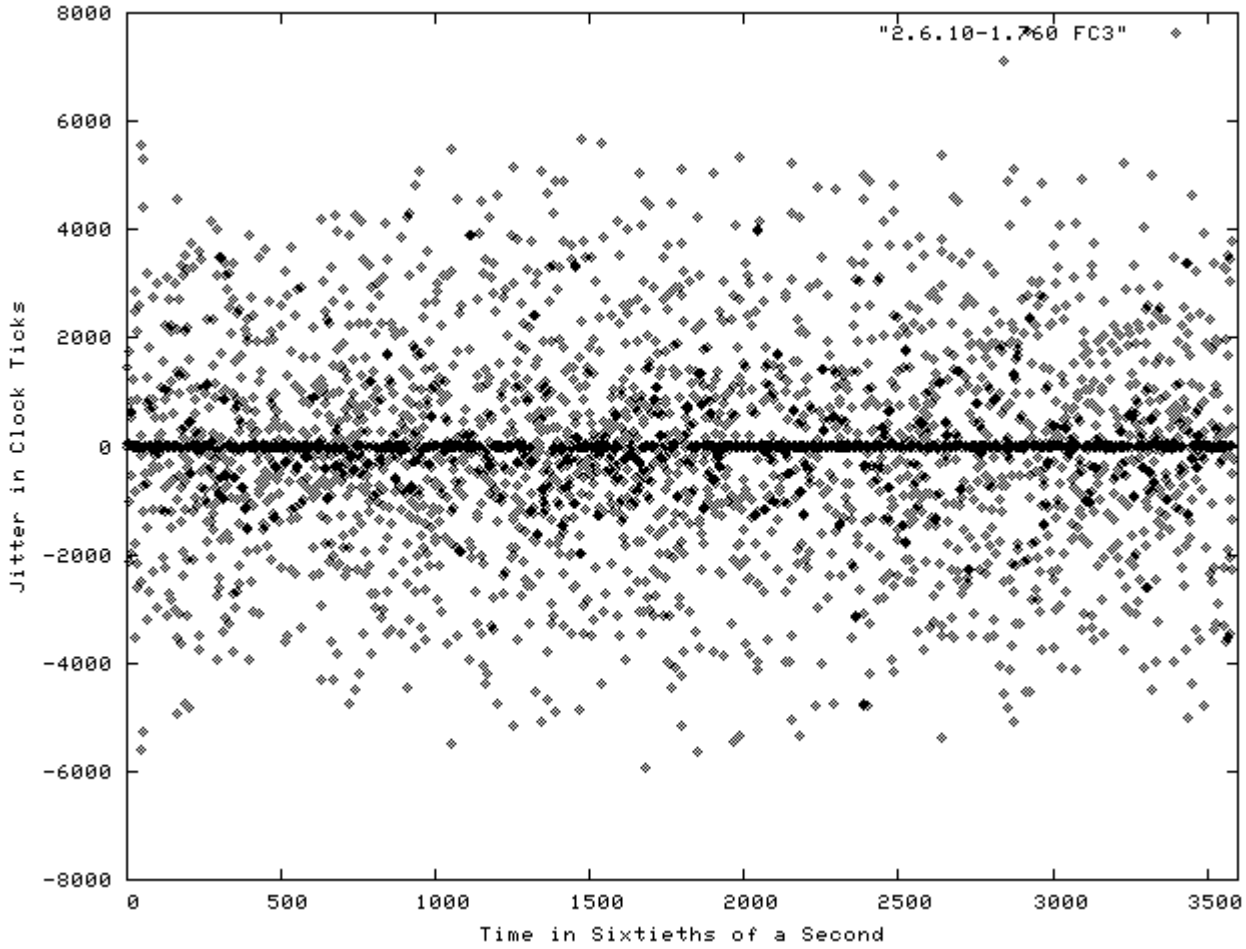

Figure 1. Jitter Results on a Stock Fedora Core Kernel

Figure 2 shows a plot of the results of the Realfeel benchmark run on the same system with a vanilla 2.6.11rc3 kernel, which is release candidate 3 of the upcoming 2.6.11 kernel. These results also were collected over a period of one minute. As you can see from these results, the 2.6.11-rc3 kernel provides improved results from the FC3 kernel, with many more instances where the jitter between an interrupt request and its handling was zero.

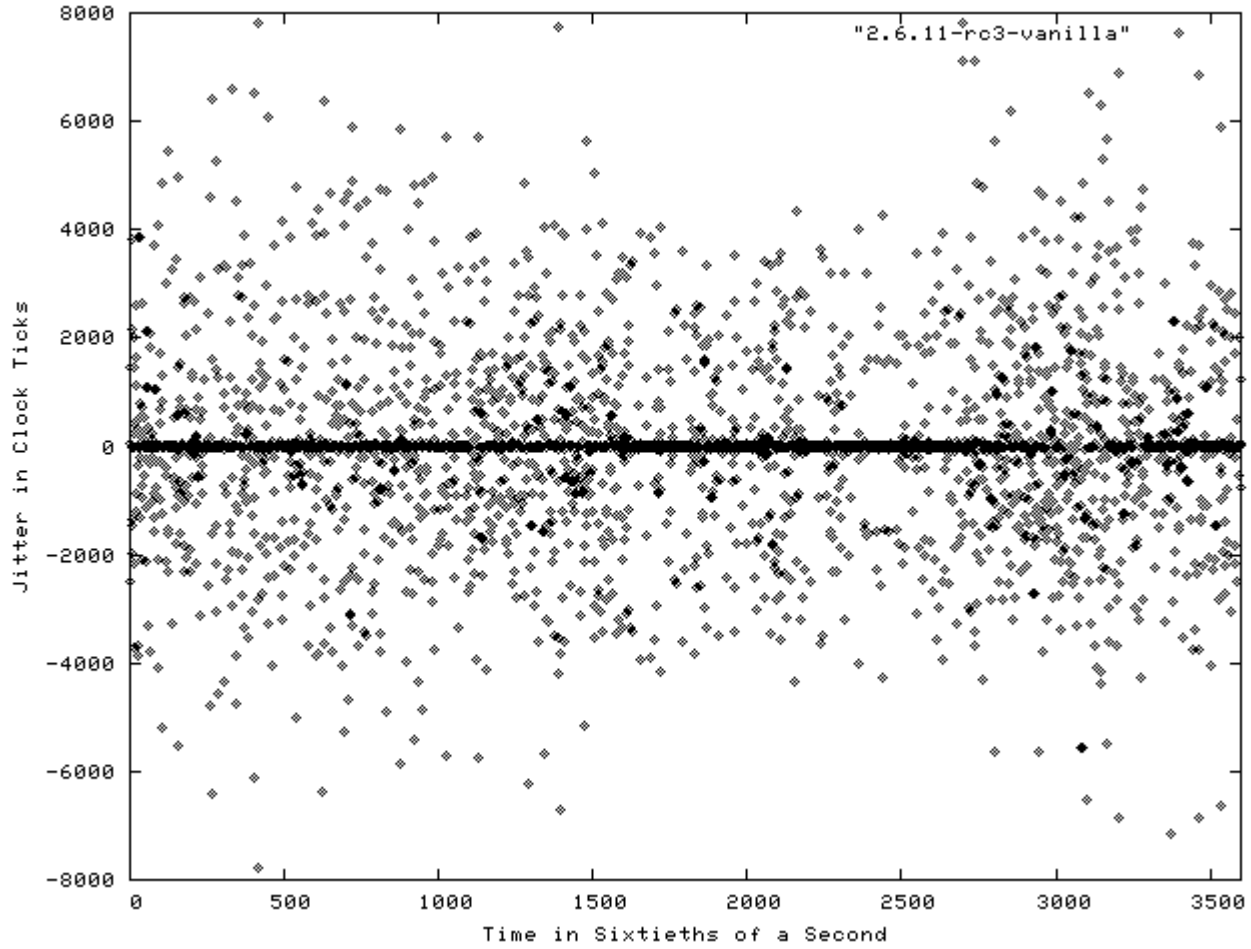

Figure 2. Jitter Results on a Vanilla 2.6.11-rc3 Kernel

Figure 3 shows a plot of the results of the Realfeel benchmark run on the same system with a 2.6.11rc3 kernel to which Info Molnar's real-time/preemption patches have been applied. These results also were collected over a period of one minute, with the same load generators as before. As you can see from these results, the real-time/preemption patch provides impressively better jitter results, with relatively few departures from handling interrupts within the expected period of time. On the target system, these improvements translate into a much more responsive system, on which expectations about program execution are much more predictable than they are when running the vanilla FC3 or stock 2.6.11-rc3 kernels.

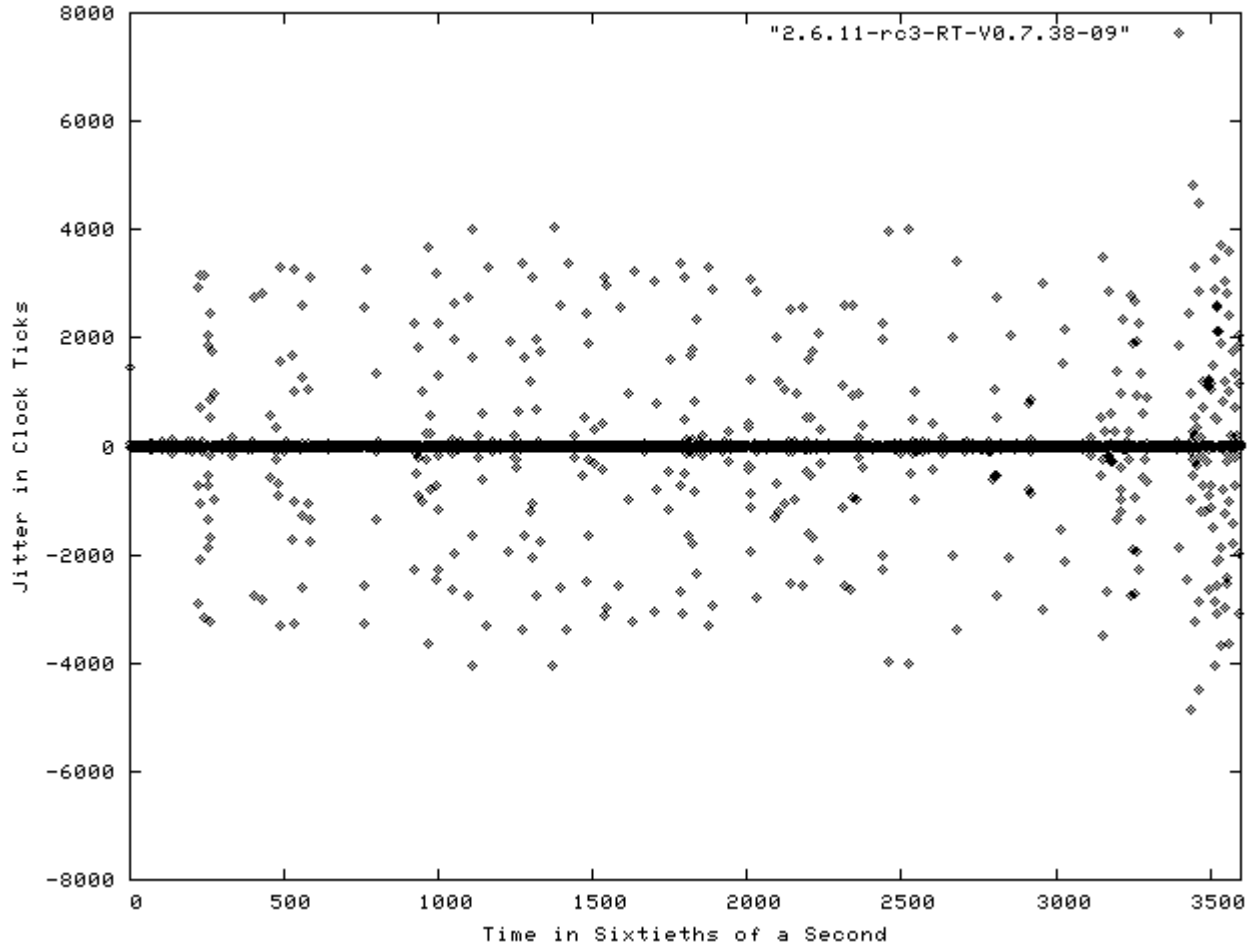

Figure 3. Jitter Results on a 2.6.11-rc3 Kernel with Real-Time/Preemption Patches

Summary

The improved scheduling, SMP and scalability improvements in the 2.6 Linux kernel provide higher performance Linux systems than ever before, enabling them to make better use of system resources and more predictably execute kernel and user tasks as requested by the system. Further improvements are available but currently are available only by patching your system manually or by obtaining a Linux distribution from a vendor such as TimeSys, which already incorporates and tests these high-performance patches.

The very existence of GNU/Linux as a free, open-source kernel and robust execution environment is something of a marvel. The contributions of individuals and, more recently, corporations to improving its performance will lead to an even brighter future. These and other improvements to Linux argue for and help guarantee the adoption of Linux as the preferred operating system for embedded, server and desktop applications.

Resources for this article: http://www.linuxjournal.com/article/8199.