Inside the Ubuntu 10.04 boot process

BOOT POLISHING

Lorelyn Medina, 123RF.com

Ubuntu developer Scott James Remnant shows why you can expect faster boot times when firing up Ubuntu 10.04 Lucid Lynx.

BY SCOTT JAMES REMNANT

Ubuntu 10.04 sees the cumulation of a year of work by the Ubuntu developers to decrease the time it takes an Ubuntu machine to boot. Boot performance is an area the development team has visited a number of times since the original launch of Ubuntu, most particularly for Ubuntu 5.04, our second release.

This most recent effort was in reaction to Intel's demonstration of a five-second boot at the inaugural Linux Plumbers Conference in 2008. Although that demonstration was with a very stripped-down software image – and directly tailored to the hardware it was running on – it showed that Linux had improved to the point where faster times were possible even on a distribution like Ubuntu that supports a very wide range of hardware and software configurations.

Reducing boot speed is a little like losing weight: you can exercise, go on a diet for a while, and see what weight you get to; or you can pick a target weight and figure out what level of exercise and diet you need to achieve that weight in the available time.

For the development of Ubuntu 9.10, we already knew that we had a large number of easy problems to solve that would give a decent benefit in boot time – so-called "low-hanging fruit." Thus we settled on planning to solve the problems we knew about and accept whatever resulting boot time that gave us.

For Ubuntu 10.04, we had a lot less low-hanging fruit and more hard work, so we picked a very ambitious target boot time that we didn't entirely expect to hit, then divided that time into budgets for the different sections of the boot sequence (see Table 1). The times in Table 1 were set for an SSD (solid-state drive)-based Dell Mini 10v netbook with an Intel Diamondville hardware platform and an Atom processor.

Picking a single hardware platform for benchmarking is incredibly important because times can vary wildly with different hardware. We thought the hardware we chose was a good compromise. Most users' desktops and laptops will have a much faster CPU than the Atom, so they will get a faster boot speed; but most will have a rotational hard disk drive rather than an SSD, so they will get a comparatively slower boot as a result.

The division between budget points is not at all arbitrary. The clock is started once the Kernel is loaded into memory and begins initialization; we don't include the time it takes for the BIOS to start because the BIOS is out of our control and we can't accurately time it. The kernel provides an accurate clock, so the kernel startup makes sense as the first stage of the optimization process.

Kernel and initramfs

The first budget section comprises the time to initialize the Linux system, set up core parts of the hardware such as the PCI and USB buses, and locate and mount the root filesystem.

A big area of improvement in the kernel initialization was contributed by the Moblin project as part of their boot performance work. Previously, the kernel set up each subsystem one at a time, waiting for each to complete before moving onto the next; the process has now changed so that subsystems may be set up asynchronously, with the kernel waiting for all parallel work to complete before passing control to user space.

Locating and mounting the root filesystem generally is not actually performed by the kernel, but by an *initial RAM filesystem* (*initramfs*) that is unpacked alongside it by the bootloader. The easiest way to think of the *initramfs* is as a very stripped-down Linux system whose sole purpose is to mount the root filesystem and load the full Ubuntu system.

Table 1: Component Startup Targets

Section	Budget
Kernel and initramfs	2s
Plumbing	2s
X Server	2s
Desktop	4s
Total	**10s**

You might wonder why we need this at all, and why the kernel cannot directly mount the root filesystem. This approach is necessary for supporting such things as resuming from hibernate (suspend to disk), software RAID, Logical Volume Management (LVM), and, most notably, the wide range of different SCSI, IDE, and Serial ATA controllers.

Other fast-booting Linux systems, such as Moblin, reduce boot time by removing this *initramfs* section, however, we did not want to lose the flexibility it gives us. Ubuntu engineers, instead, worked to build on Moblin's other kernel work and changed the kernel such that the set up of the *initramfs* happens asynchronously.

You can see both the asynchronous initialization work by Moblin engineers, and the asynchronous *initramfs* initialization (the *populate_rootfs* thread) work by Ubuntu engineers in the output of a kernel boot graph shown in Figure 1. The graph is generated by extracting log messages from the kernel log after boot, using special messages added by the *initcall_debug* command-line option that delineate the different kernel threads.

Additional work tightened up the code in the *initramfs* itself. The majority of time was getting lost in the loop that waited for the root filesystem device, with the difficulty of supporting mounting by filesystem UUID or label. This loop worked by waiting for the *udev* daemon (which receives messages from the kernel about hardware and creates device nodes in response) to become idle, checking whether the expected device node or symbolic link had been created, then probing the contents of the device node for a valid filesystem. If any of these steps failed, the loop slept for a short time then repeated.

The obvious problem with this approach was that the *udev* daemon did more than simply create the device node for the root filesystem; it also loaded such other drivers as were necessary for the system and waited for other pieces of hardware to be ready. As such, it generally did not become idle for a second or two after the root filesystem was actually ready. The additional probe of the filesystem added time too; the udev daemon had already probed the filesystem as part of the creation of the device node, so this was needless duplication of effort.

This loop has been replaced by a small utility that, instead, listens for messages from the udev daemon and waits for a message indicating that the root filesystem is ready.

Plumbing

The root filesystem is not the only filesystem we need to wait for. Many systems mount additional disk drives at various points on the filesystem, and at the very least, a number of different virtual filesystems also need to be mounted. Also, the root filesystem must be checked and made writable.

This work previously required a small number of large, complicated, shell scripts that performed a check and mounted each filesystem in turn. All of this was done in sequence; it could only occur after we were reasonably sure all devices had been detected – and again – while the udev daemon was idle.

In Ubuntu 9.10 and 10.04, a small daemon that is connected to both the *udev* daemon and the Upstart *init* daemon listens to *udev* events for those matching filesystems described in */etc/fstab* and only acts on them when the device is ready. Filesystem checks on different devices can occur in parallel, and each filesystem is mounted when ready. The connection to Upstart allows the *init* daemon to start services once virtual, local, or remote filesystems are all mounted and allows the system to finish booting once all filesystems are mounted.

This daemon eliminates a large part of the critical path for booting the system and puts an end to needless delays waiting for things like sound card drivers simply to determine that the disk drives are ready. The daemon also removes the time limit, so slower disk drives now work.

These changes allow us to begin to take full advantage of the new Upstart *init* daemon, which has also been connected to the *udev* daemon, so that hardware-related events are also available. This makes much of the previous critical path sequence of the boot fully asynchronous.

The difference is apparent when comparing boot charts of Ubuntu 9.04 with Ubuntu 10.04 Beta 2: Looking at the main process part of the chart, you can see that, in 9.04 (Figure 2), most processes waited for the previous process to complete before running; whereas in 10.04 (Figure 3), much more happens in parallel.

Removing HAL

As you can see, a large part of our boot performance work was rearranging things so that the system is not needlessly waiting for something irrelevant to complete before moving on to doing something else. Another area of work, however, was to strip out areas where different pieces of software are doing the same work; the largest benefit was in the removal of the Hardware Abstraction Layer (HAL).

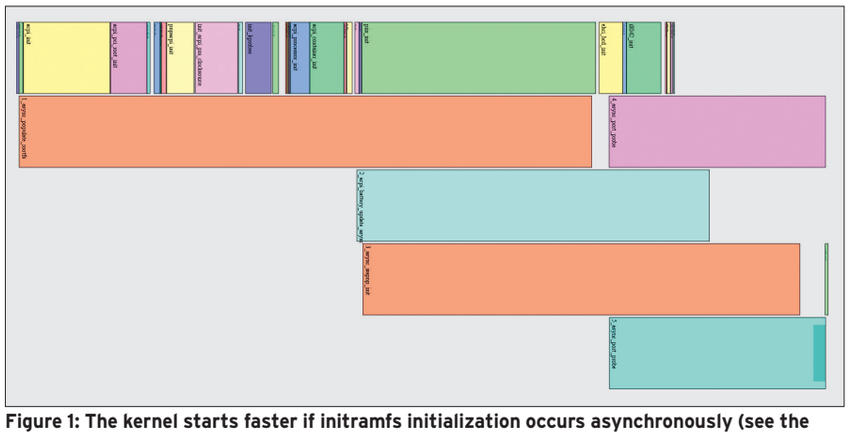

Figure 1: The kernel starts faster if initramfs initialization occurs asynchronously (see the populate_rootfs thread).

HAL comes from a time when the Linux kernel was not as free with information about the system as it is today; it was originally designed to probe the kernel and hardware to find out which devices were connected, collect information about those devices, and notify other parts of the system about any changes.

In the past handful of years, the Linux kernel has gotten increasingly better at performing that role itself. Nearly all known information about hardware is published in a tree under the new */sys* filesystem, and notifications of changes are sent by the kernel to the udev daemon, which performs its own probes where necessary and notifies other parts of the system.

Patches to move software from using HAL to using *udev* directly, or to the *udisks* (née DeviceKit-disks) and *upower* (née DeviceKit-power) daemons that wrap *udev* to add method calls, had been slow to come, and one of the last was the X Window System server itself,

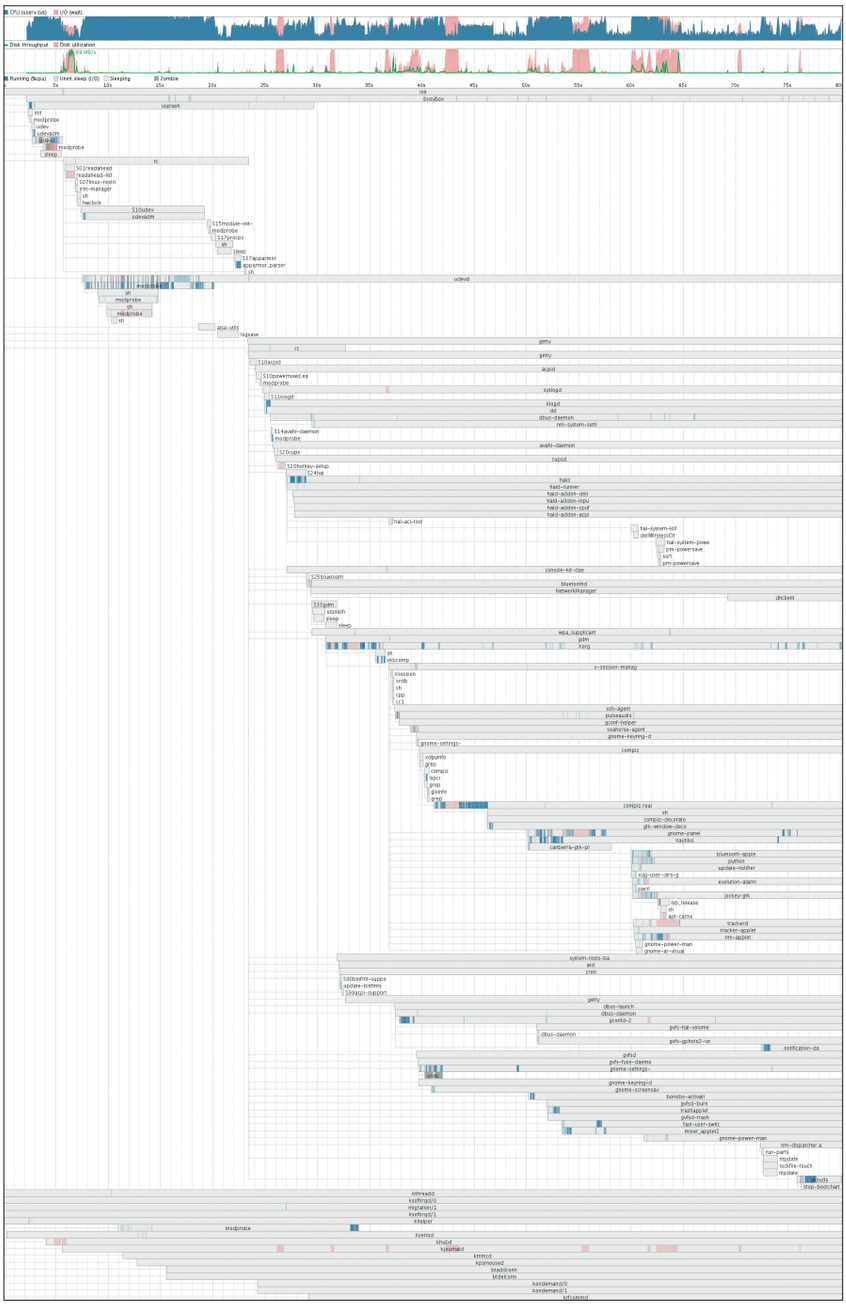

Figure 2: In Ubuntu 9.04, many boot processes were left waiting for previous tasks to complete. See how, in the chart, each bar is staggered throughout the boot time.

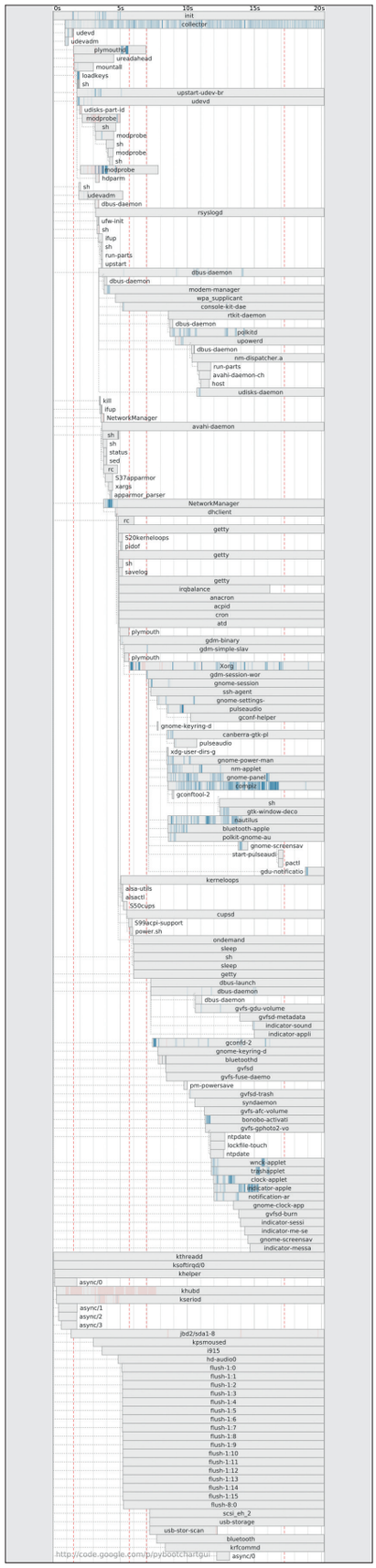

Figure 3: In 10.04, a much higher percentage of the boot processes happen in parallel.

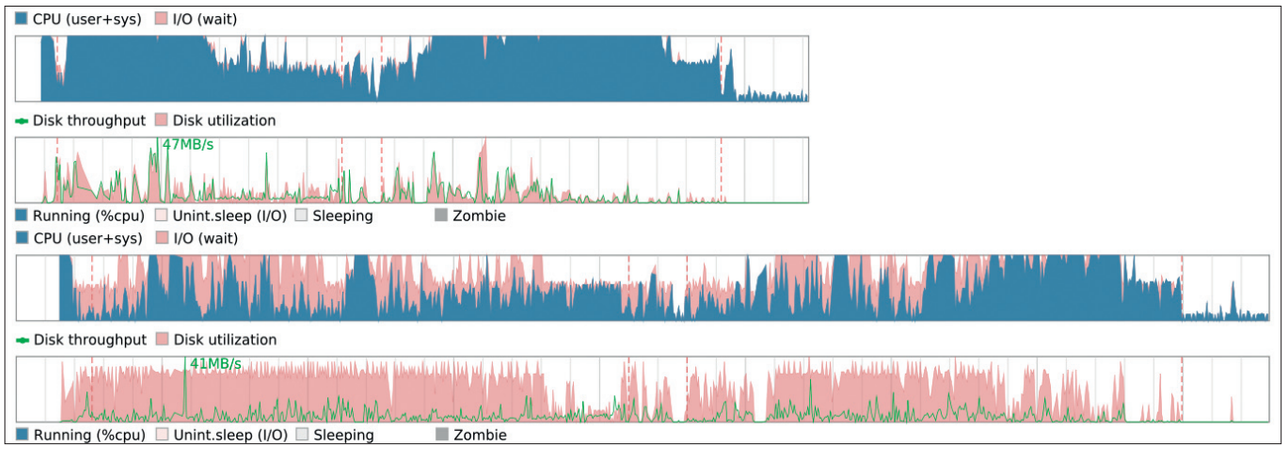

Figure 4: I/O wait time has a significant influence on overall boot performance. With both an SSD (top) and HDD (bottom), the lower red graph shows a large amount of disk activity with low throughput (the green line).

which used HAL to support hot-pluggable input devices.

With these patches all in place (some of them written by Ubuntu engineers and others by upstream contributors), the HAL daemon could be removed. Eliminating HAL removed a chunk of boot time, since the system no longer has to probe all hardware twice.

ureadahead

As I mentioned at the beginning of this article, the reference platform included an SSD disk. This choice was deliberate because solid-state disks perform better than traditional rotational hard disk drives. Engineers measure disk performance in two ways: the time it takes to read a contiguous chunk of data from the disk, and the time it takes to move the disk head to that data in the first place (the seek time).

Seek time is where rotational hard disk drives perform extremely poorly; a significant amount of time is lost simply waiting for the disk head to move to the next piece of data. This loss only gets worse once files begin to fragment and are split across different parts of the disk.

This lost time is called *I/O Wait* – literally, the time waiting for input or output that could be better spent doing other things.

It shows up nicely on the boot chart software as red space in the otherwise blue CPU graph (see Figure 4). The lower (predominantly red) of the two graphs illustrates the utilization of the disk, and the green line indicates throughput.

In a comparison of an solid-state and a hard disk drive on otherwise identical hardware, you can see that much more time is spent waiting for the rotational disk than the solid-state disk (indeed, solid state is almost fast enough to not show any wait time), and throughput of the rotational disk is substantially lower as well.

The *ureadahead* tool was developed to combat this problem, especially for rotational hard disk drives. *ureadahead* works by recording the chunks of files that are read during boot, it then reads those files into memory in a single pass the next time according to the on-disk location of the individual chunk. By doing this single location-ordered pass, a lot of the seek time is eliminated, and the use of the disk becomes much more efficient.

The rest of the boot proceeds significantly faster since the necessary files are already in memory. A comparison of SSD and HDD on the exact same hardware and software image as before shows the difference (Figure 5).

As you can see, this makes an immense difference for the rotational hard drive and brings it much closer to the boot time of the solid-state disk-based system, although this gap obviously widens again as the disk becomes more fragmented. More efficient disk usage improves SSD as well.

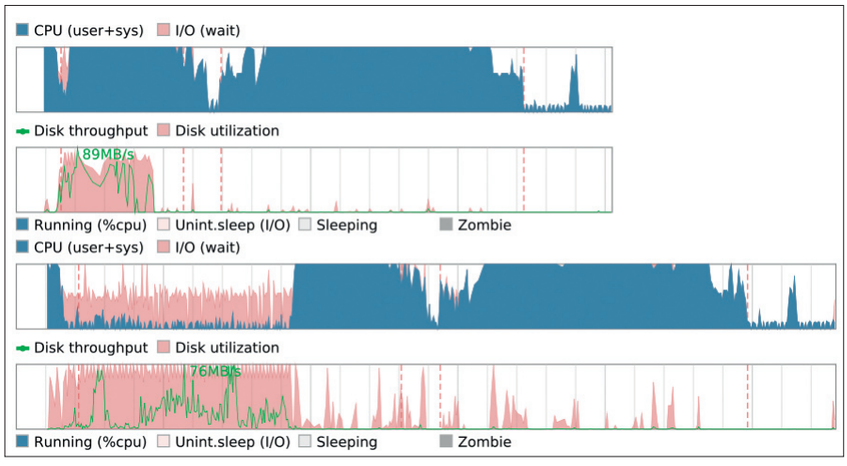

Figure 5: The ureadahead program reads boot-related files into memory in a single pass, thus speeding up the boot process by reducing I/O wait time. Compared with Figure 4, there is much less disk activity with much higher throughput.

Conclusion

More work is needed to decrease Ubuntu boot time; future plans involve feeding the *ureadahead* analysis into a defragmentation tool, so that the chunks of files needed to boot are all contiguous on disk, thus further minimizing seek time. ■