# Startup Time in the 21st Century: Filesystem Hacks and Assorted Tweaks

Benjamin C.R. LaHaise
*Intel Corporation*
bcrl@linux.intel.com

## Abstract

While processors have relentlessly increased in performance over the past few years, the amount of time it takes a modern Linux distribution to go from the bootloader to a working shell remains relatively large and painful. Several key points in the boot process offer the chance to make more efficient use of otherwise idle time in the system to perform tasks that are required by later stages of initialization. The missed opportunities range from the precious seconds lost while Grub idly awaits user input to the seek-bound thrashing of init scripts and filesystem checks.

To improve this situation, a block device cache called BootCache is filled via sequential reads earlier in the boot process. This helps remove the IO bottleneck from the boot process, enabling further performance tuning through traditional profiling techniques. This paper examines the impact of BootCache on startup time and regular workloads, as well as the new bottlenecks that are revealed by the modified system.

## 1 Background

The inspiration for this work was a talk presented at OLS in 2005 during which Bert Hubert presented actual measurements of the latencies associated with disk IO during application startup. These measurements showed a substantial amount of time being wasted while the system waited on IOs that caused the disk to seek. These delays are of particular interest to many of us who spend time waiting for laptops to boot. Laptops tend to have horrendously slow drives, often spinning at 4200 rpm compared to the more typical 7200 rpm of current desktop drives. This raises the question: how much benefit does removing the seek bottleneck provide when IO is started early enough? What are the issues of concern in implementing a cache to make sequential streaming reads possible? Can such a cache be useful for workloads outside of booting?

## 2 Is it worthwhile?

The first step in looking at any potential optimization to solve a problem is to see if the effort spent will actually accomplish anything. Thankfully, the Linux kernel has a standard measurement of system idle time which is useful in estimating how much time is spent waiting on IO. Barring a few moments when the startup scripts wait several seconds for user input, the startup scripts should not be spending much time sitting idle.

| | Uptime | Idle time |
|---|---|---|
| System 1 | | |
| to init | 13.2s | 6.4s |
| to rc.local | 38.0s | 24.9s |
| System 2 | | |
| to init | 8.3s | 4.0s |
| to rc.local | 46.3s | 36.3s |

Table 1: Idle time during boot

Simply getting to the login prompt involves the system sitting idle for approximately 25s on each boot for a fairly minimal set of daemons being started on a pruned FC4 install. A more complex system (FC5 default install) spends over 36s in idle time. This is ripe for improvement.

## 3 A first cut

There has been some experimentation with using the `readahead()` syscall to prefetch data into the cache, but this suffers from a number of problems. The most notable drawback is that it does not eliminate the time wasted by disk seeks.

This leads into the main requirement of BootCache, which is that all IO should be sequential. Sequential streaming is a task that disks are much better tuned for, with many disks able to read at rates of more than 60 MB/s. With that in mind, a rough prototype of BootCache was written.

For the purposes of the prototype, the BootCache modules take the approach of dumping the contents of the kernel's page cache and buffer cache into a simple log file which can be replayed on boot. The order in which data is recorded is determined via a log of cache references collected by the system during boot. The prototype is rather grotesque in that it hooks directly into the page cache and buffer cache directly. All of this functionality is included in the `mkbootcache` module, which performs these tasks as part of its initialization function.

The `mkbootcache` module operates by performing multiple passes over the access log. Each pass attempts to write out the data of either a buffer cache page or a page cache page. If the page is dropped from the cache or not valid, the entry is dropped. This is necessary because the log of what pages are contained in the BootCache must be present at the beginning of the cache.

One important element of `mkbootcache` is that it must ensure that the cached copy of any blocks stored on disk remains up to date with the original. This is accomplished by snooping all writes to the root filesystem's block device. When a write overlaps a block in the cache, `mkbootcache` steps in and writes out a copy to the cache before allowing the request to proceed. This step is extremely tricky to get right, as the order of block writes is especially important to journaling filesystems. With `mkbootcache` in place and keeping the data coherent, the cache's log file is now ready to be used on boot.

On boot, a module called `trystuffcache` is loaded immediately after the root filesystem is mounted. This module attempts to replay the log file and stuff data back into the page cache and buffer cache. For the paranoid during testing, it would only compare the log against the actual data on disk, which made debugging substantially easier.

| | Without BootCache | With BootCache |
|---|---|---|
| to BootCache | n/a | 8.0s |
| to rc.sysinit | 12.3s | 15.7s |
| to login | 44.9s | 30.8s |

Table 2: Fedora Core 5 boot times

## 4 How does BootCache improve things?

For a laptop installed with Fedora Core 5, boot time to the login prompt takes 44.9s with an unmodified kernel. With a BootCache in place, boot time is reduced to 30.8s. This 14s improvement (a 32% reduction in boot time) includes the time it takes to load the BootCache log from disk. Even though the log comes in at a whopping 205MB (mostly due to FC5's readahead-preloading many desktop applications).

There is an even more impressive improvement in the case of preloading the cache for a `git diff` operation. Without the cache being stuffed, `git diff` takes 1m 06s after a fresh boot, yet with BootCache stuffing the cache, it only takes 0.2s. Even including the run time of `trystuffcache`, BootCache comes out ahead.

## 5 Improvements

In writing the prototype BootCache and making it work using the cache-stuffing technique, there were quite a number of small hurdles to overcome. Cache coherency was most tricky and results in increased overhead for requests passing through to the underlying block device. Those requests affecting the BootCache area (especially inodes and superblocks) must be written out twice. Depending on the journaling mode of the filesystem, the cache and the original blocks can end up out of sync.

To simplify and make the system more robust, it is probably better to eliminate the duplication of blocks and instead focus on block-based readahead. This would have to go hand-in-hand with reordering the layout of files on disk to place those accessed during boot in a compact sequential area on the disk. Then, by performing readahead on this area of the disk, the benefits from cache-stuffing can be achieved while the complexity and coherency issues of the cache-stuffing process are eliminated.

## 6 Further Information

Before starting this work, it was unclear how much of an improvement to boot time the BootCache functionality would actually provide. Thankfully, a 32% reduction in boot time is of definite utility. As BootCache is a work in progress, there will be updates. These updates will be made available at `http://www.kvack.org/~bcrl/bootcache/`.

# Proceedings of the Linux Symposium

## Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

## Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

## Review Committee

Jeff Garzik, *Red Hat Software*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat Software*
Ben LaHaise, *Intel Corporation*
Matt Mackall, *Selenic Consulting*
Patrick Mochel, *Intel Corporation*
C. Craig Ross, *Linux Symposium*
Andrew Hutton, *Steamballoon, Inc.*

## Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
David M. Fellows, *Fellows and Carr, Inc.*
Kyle McMartin