QuickStart: Booting VMs Faster in the Cloud

Yan Zhai, Amanda Hittson, Thomas Ristenpart and Michael Swift
Department of Computer Sciences,
University of Wisconsin Madison
{yanzhai,hittson,rist,swift}@cs.wisc.edu

Abstract

Cloud computing promises rapid adaptation to changes in workload by spinning up more virtual-machine instances. However, the ability to respond quickly depends on the time it takes the cloud provider to provision a new virtual machine and the time it takes the guest operating system to boot. We find that typical Linux instances used in Amazon's EC2 cloud can take more than 50 seconds to boot and provide application services.

We describe a new technique that greatly reduces the latency to launch a new instance without requiring OS modifications. In a measurement study, we find that I/O delay to transfer OS code and data from storage is the dominant factor in boot time. Our proposed solution leverages existing Linux ramdisk support to optimize I/O and effectively prefetch the entire OS and application data in one operation. In an evaluation in EC2, we find our approach reduces boot latency by more than 80%.

1 Introduction

Booting an operating system quickly is useful: in enterprise datacenters, faster boot means faster upgrades and failure recovery. In personal computing, faster boot reduces the time users spend waiting. In cloud environments, fast boot has an additional use: agile adaptation to workloads changes. A service deployed in the cloud that experiences a rapid load increase can quickly launch more instances to shoulder the load.

However, current cloud systems boot slower than ideal. In our experiments on Amazon's EC2, we found that it can take an average of 40 seconds for Amazon to start the virtual machine and an additional 50 seconds for applications to start.

To understand why, we studied Linux boot performance on a local testbed, where we have tight control over the system, and then in EC2. As we show in Section 2, we found that the bottleneck lies in transferring the OS image from storage servers to the virtual machine's host computer.

Based on this observation, we propose QuickStart, a technique for fast Linux boot in cloud environments. QuickStart leverages *existing* support for ramdisks during boot (*initramfs* in Linux) to stream the critical data in one large transfer, which greatly improves network efficiency. Currently, Linux places only a few files critical to OS boot, such as low-level device drivers, in the initial ramdisk. Instead, we propose populating it with *all files* needed to launch applications. Remaining files can still be referenced from network storage. QuickStart requires no kernel modification and uses existing tools to rearrange which files are packaged in the initial ramdisk. We make no assumptions about the underlying platform, so our method works with any VMM or even physical hardware.

In tests on EC2, we evaluate QuickStart on three applications: the *Nginx* and *Apache* webservers and the *MySQL* database. While speedup varies with the size and complexity of the application, on average QuickStart is 80% faster than booting Amazon's stock Linux image.

2 Understanding VM Boot Performance

We begin with a study of boot performance to identify bottlenecks in a local testbed and on Amazon's EC2.

2.1 Local Boot Performance

We emulate a cloud boot environment using an NFS server, connected by a 100mbps network, to hold VM images. The VM host has a 4-core Intel Core

Configuration	Test	Time (sec)
Local	NFS without cache	30.8
	NFS cached	7.5
	local disk	12.3
EC2	boot small	60.5
	boot medium	51.5
	reboot small	23.8
	reboot medium	15.1

Table 1: Average boot times (in seconds) on a local VM setup and on EC2.

i5-2500K with 16GB memory and a SATA disk. We use Xen hypervisor [4], version 4.1.2. Dom0 system is Gentoo Linux, customized from the official stage3 tarball, and the target DomU guest is also a Gentoo system.

For local experiments, we have three configurations:

(1) *NFS without cache*: all I/O during boot goes to the NFS.

(2) *NFS cached*: I/O goes to a local in-memory cache.

(3) *Local disk*: I/O goes to local disk using EXT4 file system. system.

The results in the top half of Table 1 report the average boot times: from `/proc/uptime` right after `sshd` starts. The averages are over 5 runs. The results are as one might expect: if the image is already in memory (NFS cached), then the boot is extremely fast. If the image is stored either locally on disk or across the network, the boot time is much slower. Using bootchart [1] to profile startup performance, we found that in the NFS without cache case, more than 20 seconds was spent within the guest waiting for I/O. In the cached case, this dropped to less than one second.

As mentioned, our local experiment used a 100mbps network, which is an order of magnitude slower than the 1gbps networks currently used in public clouds such as IaaS. The slower network therefore exacerbates the performance cost of I/O compared to real EC2 behavior. Nevertheless, we hypothesize that I/O remains the key bottleneck for boot time.

2.2 EC2 Boot Performance

To understand boot times in a commercial cloud, we ran experiments using Amazon Linux AMI 2013.03 in the us-east-1a region. Using both small and medium instances, we started a fresh VM instance and measure the boot time as we did above: we inserted into the stock AMI a script that collects `/proc/uptime` immediately after `sshd` starts. Note that medium instances have roughly twice the CPU capacity of small instances. The timing results are shown in Table 1 in the "boot small" and "boot medium" rows. As one would expect, small instances have slightly longer boots compared to medium instances due to reduced CPU capacity.

We also measure the effect of caching on boot time. Before an instance is launched, the VM images are stored in network storage and must be copied to the VM host. When an instance *reboots*, though, it uses the locally cached copy of the instance and avoids much of the network traffic [5] The last two rows in Table 1 shows boot time when *rebooting* the same EC2 instances. On average, rebooting is 50% faster than booting the first time.

All this points to I/O performance as a bottleneck for faster boot performance. In the next section we develop QuickStart which will improves performance by reducing the number of I/O operations on the critical path for boots.

3 Design and Implementation

Given that network I/O is the bottleneck for boot, two prior approaches to reducing I/O costs would seem applicable. First, one can create a stripped-down system that only starts the minimal necessary services and a kernel with only the minimal necessary drivers [3]. Second, one could put the entire root file system into a ramdisk, which will ensure high-speed sequential I/O for files needed during boot.

Stripped-down images. We gauged whether a stripped-down image might help by measuring whether Amazon VM images (AMIs) include many unnecessary files. We downloaded 1,249 public images, and found that most AMIs include hundreds of files unnecessary in a virtual environment, such as wireless and bluetooth drivers. They also start dozens of services during boot. Using `OpenRC`, we emulated the stripped-down system approach by

configuring the default Amazon Linux AMI to only start the network and *no other services*. Such an AMI, however, still required more than 30 seconds to boot on EC2. The problem still remained that I/O was not efficient, because data is fetched on demand. For example, running a script requires bringing over the shell, executables invoked by the shell, shared libraries linked to the executables, and any configuration files referenced by the script.

Ramdisks. The second common approach fetches the entire root file system into memory during boot [7], leveraging the existing Linux *initramfs* mechanism. Initramfs is an in-memory file used to provide files during boot. When the VM starts, the OS loader `PV-GRUB` fetches the kernel and ramdisk from network storage into the host's memory, and passes the address of the ramdisk to the kernel. The kernel then mounts the ramdisk, and all files on the ramdisk are accessible without additional I/O. A natural embellishment for faster boots, then, is to include the entire file system (applications, data, and all) into the ramdisk. While this promises good performance, but is a waste of memory. Even after removing many unnecessary files the ramdisk was still larger than 1GB, but small EC2 instances provide only 1.7 GB of memory leaving very little for the OS and applications.

3.1 The QuickStart Approach

We instead propose a new boot mechanism called QuickStart that combines the above two approaches. First, we build a minimal environment containing files necessary to start applications, and ensure those files are transferred efficiently with a ramdisk. To preserve generality, however, we enable the remaining files on the root volume to be accessed via normal network I/O. Thus, QuickStart can start critical applications quickly but enables full access to all the files found in a full Linux installation.

Minimum ramdisk environment Our approach tries to minimize the services required to boot a normal system, and packs most necessary files in the initramfs for boot programs to use. There are three major steps to building the ramdisk: (1) determine the necessary services—application files, network and clock services for our tests, (2) determine the files needed by the kernel and desired services, and (3) modify the `/init` script to use files from the ramdisk rather than root volume. Currently, we determine the necessary files manually by analyzing an I/O trace of the kernel during startup and using `ldd` to determine shared files. We put any executables (from the trace), libraries (from `ldd`), and configuration files into the ramdisk. Currently, we include the entire `/etc` directory for simplicity. To minimize the size of the executables, we use `BusyBox`, a package of the frequent use programs, in our environment.

Normally, the kernel frees the initial ramdisk once it has mounted the root file system. Because we rely on it for applications, QuickStart must keep it around. We therefore modify the `/init` script to remount the ramdisk under `/fast`, and direct applications to find files with their `PATH` and `LD_LIBRARY_PATH` environment variables. When files are not found on the ramdisk, the normal path search mechanism will look on the root volume, where they will be fetched from network storage.

While conceptually simple, we ran into several obstacles applying this methodology. First, determining the complete set of library dependencies is difficult. While not a correctness problem (the libraries will be found on network storage), leaving a library out of the ramdisk can slow startup time. We found two major causes of such misses: first, some applications explicitly load libraries with the `dlopen()` function, such as `libnss`. These will not be reported by `ldd`. Second, some applications have hard coded paths and will not search for files under `/fast`. For example, the C library's `popen()` function hard-codes the location of the shell as `/bin/sh`. Finally, a ramdisk may be inappropriate for files that are written to or need persistence, such as system logs.

To address these issues, we modified startup scripts to reference the `/fast` directory, and discovered explicitly loaded libraries using `strace`. We leave data files at their original location to avoid running out of space if they exceed the ramdisk capacity.

Note that one downside of the QuickStart approach is that when we modify files stored in the ramdisk, changes are not reflected to the original files. However, boot images within EC2 are already read-only, so changes to the image are not written back to image storage.

4 Evaluation

In the 1249 images from EC2, we discovered that the most popular applications were `Apache(Httpd)` and `MySQL`, which were included in 551 and 479 images, respectfully. We will therefore use these applications for evaluation. We additionally include the `Nginx` web server , as an example of a lighter-weight application. We perform experiments on both `m1.small` and `m1.medium` instance types.

4.1 Methodology

We measure the total provisioning time of applications (the time from launch request to application responsiveness) as well as the boot times (the time from the beginning of VM boot to application responsiveness). The former includes provisioning delays that are out of the control of the user, such as time spent in queues waiting for a server assignment. In more detail, we use a client to execute our experiments. The time it requests a VM instance to be launched is T_{launch}. We modified the VM's `/init` script to configure the network using `udhcpc`, and as soon as that configuration finishes, to record the value of `/proc/uptime` and send it to the client using `netcat`. The time at which this information is received by the client is T_{netcat}. From these values we can calculate the `provisioning time` $T_{pr} = (T_{netcat} - T_{launch}) - T_{uptime}$. This is the time EC2 takes to select a physical host and initiate boot.

To determine when an application becomes available, the client polls the application service (`GET /` for a web server and `show databases;` for MySQL) every second until it responds successfully. The time of the successful response is T_{probe}. The application boot time is $T_{app} = (T_{probe} - T_{launch}) - T_{pr}$, and total application launch time is $T_{total} = T_{probe} - T_{launch}$.

We compared QuickStart with three other configurations. First is a pure ramboot AMI (`RAMBoot`), but here with only the necessary applications placed inside, as opposed to the entire file system. This is the optimal case, but limiting since the full functionality of the AMI is not available to users. Second is the stripped-down AMI without adding to the ramdisk (`Minimal`), as described in the previous section. Third is the standard Amazon Linux AMI (`Standard`).

4.2 Boot Times

We report in Figure 1 the average application launch time T_{total} (entire bar) made up of provisioning time T_{pr} and application boot time T_{app} (upper bar). The averages are taken over 50 runs and we repeated the experiment both for m1.small and m1.medium instance types.

As can be seen, `QuickStart` reduces by half the launch time compared to standard Amazon AMIs. For example, `Apache` requires only 49 seconds, compared to 104 seconds for standard Amazon Linux on a small instance. When compared with the optimal `RAMboot`, QuickStart is only 10 seconds slower. The difference is due mostly to provisioning times: discounting EC2 provisioning overheads, we see `QuickStart` boot time is close to optimal for `Apache` (8.1 seconds), while `RAMBoot` takes 7 seconds. Both systems outperform Amazon Linux's 61.6 seconds by more than 80%, and the other two applications show similar improvements. `Minimal`, meanwhile, is 20 seconds faster than the standard AMI on average, but still three times slower than QuickStart. This shows that removing services helps, but not nearly as much as optimizations that use initramfs.

To help understand the I/O improvement, we recorded the I/O stats of the root file system from `/proc/diskstats` before application startup. The number of read operations were reduced by a factor of three for the two web servers when moving from the minimal AMI to `QuickStart`. For MySQL, read operations are reduced only by a factor of two. This helps explain why MySQL benefits slightly less from QuickStart.

We note that the difference between small and medium instances is approximately 2 seconds. This suggests again that the CPU is not a major bottleneck during boot for the Amazon AMIs. In QuickStart CPU improvements bring relatively more benefit, since more of the I/O is eliminated.

4.3 Initramfs Size

As we mentioned before, we use initramfs to hold necessary data. This memory currently cannot be swapped out or reclaimed, so we report our initramfs size in Table 2. Recall that in the `RAMBoot` configuration we included only application-related data, and

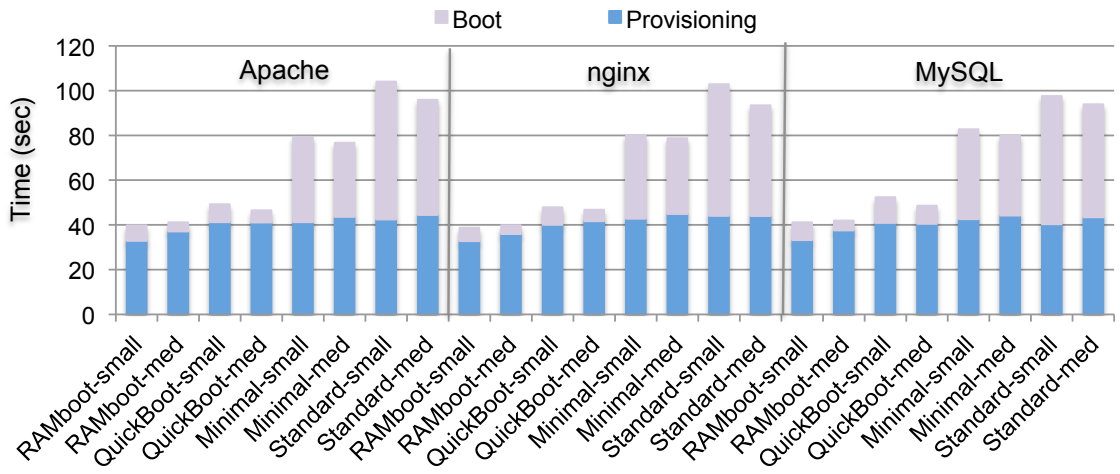

Figure 1: The average application launch time and provisioning time for three applications with four configurations, on both small and medium EC2 instances. The small/medium specify instance types;

Application Type	Initramfs Size
Mysql	35MB
Nginx	27MB
Apache	28MB

Table 2: Initramfs size of different configurations.

so it does not have a complete functional environment. To support a full environment of the Amazon Linux AMI, one requires nearly 1GB of memory, which is highly prohibitive in most settings. By comparison, the largest initramfs in our experiments with QuickStart required only 35MB.

One might be tempted to compress (most) of the initramfs in order to save space. Counterintuitively, we found that this can *slow* boot. Using `Mysql`, compressing the initramfs yielded a 10MB file, but *increased* boot time from 11.5 seconds to 14.8 seconds. On medium instances, boot time similarly increased from 8.1 seconds to 10.8 seconds. This indicates that the bandwidth savings in sending the compressed ramdisk do not outweigh the CPU time required to decompress it.

5 Related Work

Bare-metal boot. Microsoft Windows ReadyBoot [9] is perhaps most similar to our proposal, in that it optimizes boot by streamlining disk I/O, though for non-virtualized and local-storage settings. Rather than relying on a single ramdisk containing all necessary files, ReadyBoot dynamically constructs a list of files referenced during boot, and then prefetches them into memory early during boot. In a cloud environment, dynamically identifying files is an unnecessary complexity because the same disk image is used every time. Windows 8 extends this by defaulting to hibernating, where an existing snapshot of system state is loaded in a sequential transfer [11].

As discussed in Section 3, a well-known technique (c.f., [7]) for speeding up Linux boots in bare-metal settings is to place the *whole* Linux root file system in a ramdisk. We note that we are unaware of any previous suggestions to use this approach in virtualized settings, but the techniques apply there as well. The major problem with this approach, however, is that it wastes memory by storing unnecessary files in RAM. As a result, its users typically build minimally functional environments that may limit which applications can run. In contrast, our approach only places the files needed for startup in RAM, while the remainder of the root file system is accessed normally.

There have been many other techniques for faster OS boot. Specific techniques for Linux—including measuring boot time, stripping down the kernel and optimizing user applications—can be found at the Linux boottime website [2]. Many of these efforts are for embedded devices and are not yet publicly measured on a cloud environment. The ChromeOS system optimizes boot to support fast system up-

grades [6], and can reboot a laptop in under 15 seconds. The OS uses a custom BIOS and removes or delays anything not critical for user login.

VM boot. Past studies of cloud boot time across multiple cloud providers showed that boot performance is strongly dependent on the VM image size [8], which agrees with our measurements that I/O is the dominant cost. One past approach is to use cloning with migration to lower boot times [13]. However, this requires support from cloud providers, while QuickStart works with existing cloud services.

In paravirtualized systems, a modified kernel can reduce boot time, such a Xen mini OS [3]. Google proposed initializing devices in one VM and transferring them to another [10] and accelerating timer rates in a VM, since virtual devices usually do not require delays [12]. These techniques are complementary to ours, as QuickStart could use such optimized kernels to further speed boot.

6 Conclusion

In this work, we investigated how to improve provisioning time for Linux VM instances in clouds such as Amazon's EC2. Our initial experiments highlighted that network I/O for retrieving needed files from network storage is a bottleneck to boot performance. We then presented a design, called QuickStart, that optimizes the critical path of application initialization using the initramfs functionality already existent in modern clouds. This reduces the amount of network I/O and improves performance, yet does not limit functionality nor require significant memory overheads. We showed experimentally that launch time for applications such as MySQL, Apache, and Nginx are all reduced by a factor of two.

Our method currently does have some limitations. We require manual manipulation of images to create the needed `initramfs`. Likewise, updates to applications require changes to the ramdisk (which in EC2 is handled separately from a VM image). In typical usage one sets up an AMI/ramdisk and reuses it frequently, thus amortizing the setup costs. Moreover, QuickStart is flexible in that one can leave frequently modified files (e.g., application data files) on network storage while retaining performance improvements. In future work we will investigate techniques for automating more parts of the process so as to enable non-expert users to streamline their images.

References

[1] Bootchart. `http://www.gentoo.org/proj/en/base/openrc/`.

[2] Linux boot time related work collection. `http://elinux.org/Boot_Time`.

[3] Xen mini os project. `http://wiki.xen.org/wiki/Mini-OS`.

[4] Xen hypervisor project. `http://www.xen.org/products/xenhyp.html`, 2005.

[5] Amazon Web Services. Storage for the root device. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html#storage-for-the-root-device`.

[6] R. Barnette. About ChromeOS's startup process. `https://groups.google.com/a/chromium.org/forum/?fromgroups=#!topic/chromium-os-discuss/r2UIuVL8RL8`, Dec. 2011.

[7] I. Kuo. How to RAMboot for blazing speed and silence. `http://forums.justlinux.com/showthread.php?151619-How-to-RAMboot-for-blazing\-speed-and-silence`, June 2008.

[8] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Proc. CLOUD*, pages 423–430. IEEE, 2012.

[9] Microsoft Corp. Windows PC Accelerators. `http://msdn.microsoft.com/en-us/library/windows/hardware/gg463388.aspx`, Oct. 2010.

[10] T. Powell and J. Anderson. Fast booting a computing device to a specialized experience, Oct. 12 2010. US Patent 7,814,307.

[11] S. Sinofsky. `http://blogs.msdn.com/b/b8/archive/2011/09/08/delivering-fast-boot-times-in\-windows-8.aspx`, Sept. 2011.

[12] M. Tsirkin and G. Natapov. Mechanism for virtual machine boot speed-up by clock acceleration, Dec. 20 2010. US Patent App. 12/972,671.

[13] X. Wu, Z. Shen, R. Wu, and Y. Lin. Jump-start cloud: Efficient deployment framework for large-scale cloud applications. *ICDIT*, 6536(2011):112–125.