Speedup Kernel Bisect by 10X with Kexec and Remote Images

Abstract

Kernel DRM (Direct Rendering Module) testing is very important for determining graphics driver quality, but because of long compiling costs and mandatory rebooting, it uses much more time compared with user space drivers. This is especially a big burden when we do git-bisect which requires multiple circles of kernel building and booting. We have designed and implemented a quick kernel bisect system based on continuous integration and kexec, making the kernel validation and bisecting faster and easier.

Problem Statement

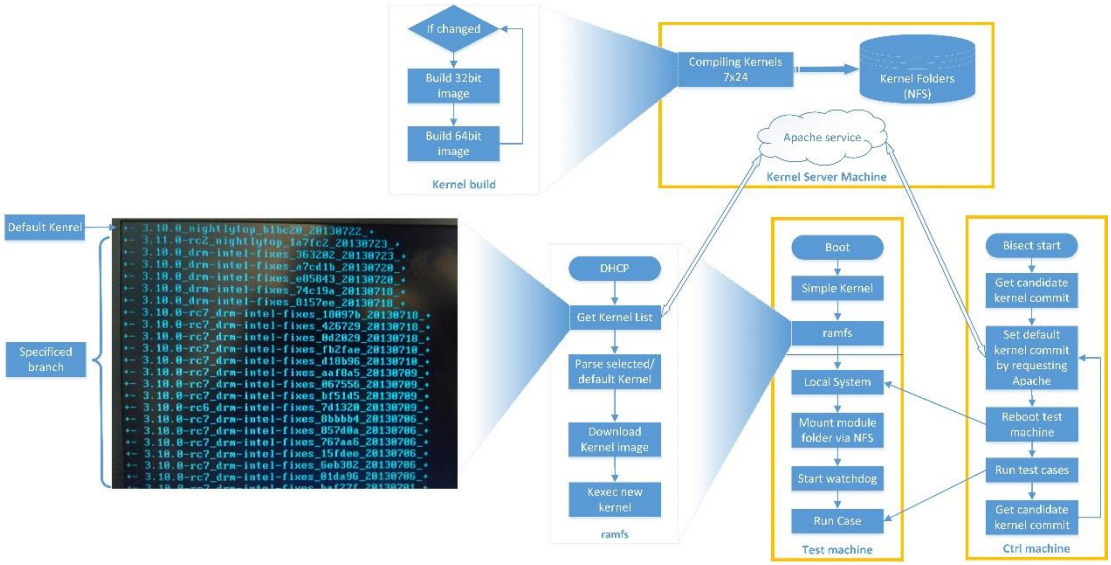
There are two main problems during kernel testing:

First, we can install a candidate kernel image on an alive system, but if system hangs happens after running a test case, we have to reboot the system with a good kernel before installing another candidate kernel image.

Second, hang issue bisecting requires too much time.

Solution Description

We designed and implemented a quick kernel bisect system with the following work flow chart.



Three kinds of PCs are needed in this system: Kernel server, test machines and a bisect control machine.

[Kernel Server]

A continuous kernel images building program is running on the Kernel Server. The server can watch the kernel source reposition, compile source code to available kernel images for both 32 bit and 64 bit if any new kernel patches commit found, and save all the kernel images in a NFS(network file system) exported folder. If any error happens during kernel compiling, a mail notification would be sent out.

An Apache service is also set up on the kernel server which is used to create an available kernel images list while the test machine is requesting it. Specifically, the available kernel images list is made up by two parts, a user customized version of the kernel on top and all other branches of kernel images following.

[Test machines]

We separated the system booting into two stages: a fixed kernel with a ramfs and the real candidate kernel image with the local OS.

During the first stage, we booted the machine with a fixed kernel with an initrd built-in and a customized root file system. This stage should guarantee a good system booting up and load rootfs.

- The rootfs gets the kernel server address passed by GRUB (GRand Unified Bootloader).
- The rootfs initialization program requests the available kernel list from kernel server.
- After parsing the selected or default kernel version (2 seconds timeout), it would download the kernel image from the NFS on the kernel server.
- The last step in stage one is booting the downloaded kernel with kexec. (Kernel execution, is a mechanism of the Linux kernel that allows "live" booting of a new kernel "over" the currently running kernel) Then the machine boots the local system with the specified kernel.

[Bisect control machine]

When kernel regression occurs, a control machine which includes the kernel source is needed to do the git-bisect. The control machine can modify the customized kernel list through the Apache service. Once the customized kernel list is modified, the test machine can boot with the new specified kernel image during the next boot up automatically without image copying or manual install.

[Auto bisect process]

The auto bisect process is very simple with the new kernel install process. After bisect starts:

- It gets the candidate kernel commit then sets the default kernel version of the specified test machine via the Apache service on the Kernel server.
- It reboots the test machine remotely.
- Wait for test machine booting up and run a test cases.
- Loop above steps, finally get the first bad commit.

Impact

This solution reduces time cost a lot by following two improvements. First, 2-steps system booting makes it very convenient for testing on specified versions of kernel because it allows only one reboot for installing a new kernel even when the system is hang. Second, compiling and storing all kernel commits beforehand saves a lot of time during regression bisect.

The following table shows that it only takes 1/10th the amount of time for a candidate kernel testing circle: compiling, installing and reboot test machine with new kernel.

	Install new kernel	Regression bisect (10 circle)	System hang
Old process	10 minutes	10x10 minutes	Twice reboot
New process	1 minutes	10x1 minutes	Once reboot