I/O Performance HOWTO

Sharon Snider

v1.0, 2002−04−05

Revision History

Revision v1.0	2002−04−05	Revised by: sds
Wrote and converted to DocBook XML.		

This HOWTO covers information on available patches for the 2.4 kernel that will improve the I/O performance of your Linux operating system.

Table of Contents

1. Distribution Policy

2. Introduction

This HOWTO provides information on improving the input/output (I/O) performance of the Linux operating system for the 2.4 kernel. Additional patches will be added as they become available.

Please send any comments, or contributions via e−mail to Sharon Snider.

3. Avoiding Bounce Buffers

This section provides information on applying and using the bounce buffer patch on the Linux 2.4 kernel. The bounce buffer patch, written by Jens Axboe, enables device drivers that support Direct Memory Access (DMA) I/O to high−address physical memory to avoid bounce buffers.

This document provides a brief overview on memory and addressing in the Linux kernel, followed by information on why and how to make use of the bounce buffer patch.

3.1. Memory and Addressing in the Linux 2.4 Kernel

The Linux 2.4 kernel includes configuration options for specifying the amount of physical memory in the target computer. By default, the configuration is limited to the amount of memory that can be directly mapped into the kernel's virtual address space. The mapping starts at PAGE_OFFSET (normally 0xC0000000). On i386 systems the default mapping scheme limits the kernel−mode addressability to the first gigabyte (GB) of physical memory, also known as low memory. High−address physical memory is normally the memory above 1 GB. This memory is not directly accessible or permanently mapped by the kernel. Support for high−address physical memory is an option that is enabled during configuration of the Linux kernel.

3.2. The Problem with Bounce Buffers

When DMA I/O is performed to or from high−address physical memory, an area is allocated in memory known as a bounce buffer. When data travels between a device and high−address physical memory, it is first copied through the bounce buffer.

Systems with a large amount of high−address physical memory and intense I/O activity can create a large number of bounce buffer data copies. The excessive number of data copies can lead to a shortage of memory and performance degradation.

Peripheral component interface (PCI) devices normally address up to 4 GB of physical memory. When a bounce buffer is used for high−address physical memory that is below 4 GB, time and memory are wasted because the peripheral has the ability to address that memory directly. Using the bounce buffer patch can decrease, and possibly eliminate, the use of bounce buffers.

3.3. Locating the Patch

The latest version of the bounce buffer patch is *block−highmem−all−<version>.gz* , and it is available from Andrea Arcangeli's −aa series kernels at http://kernel.org/pub/linux/kernel/people/andrea/kernels/v2.4/.

3.3.1. Configuring the Linux Kernel to Avoid Bounce Buffers

This section includes information on configuring the Linux kernel to avoid bounce buffers. The Linux

Kernel−HOWTO at http://www.linuxdoc.org/HOWTO/Kernel−HOWTO.html explains the process of re−compiling the Linux kernel.

The following kernel configuration options are required to enable the bounce buffer patch:

- Development Code − To enable the configurator to display the High I/O Support option, select Code Maturity Level Options category and specify "y" to prompt for development and/or incomplete code/drivers.
- High−Address Physical Memory Support − To enable high memory support for physical memory that is greater than 1 GB, select Processor type and feature category, and enter the actual amount of physical memory under the High Memory Support option.
- High−Address Physical Memory I/O Support − To enable high DMA I/O to physical addresses greater than 1 GB, select Processor type and feature category, and enter "y" to HIGHMEM I/O support option. This configuration option is a new option introduced by the bounce buffer patch.

3.3.2. Enabled Device Drivers

The bounce buffer patch provides the kernel infrastructure, small computer system interface (SCSI), and IDE mid−level driver modifications to support DMA I/O to high−address physical memory. Updates for several device drivers to make use of the added support are included with the patch.

You will need to apply the bounce buffer patch and configure the kernel to support high−address physical memory I/O. Many IDE configurations and the peripheral device drivers listed below perform DMA I/O without the use of bounce buffers:

aic7xxx_drv.o
aic7xxx_old.o
cciss.o
cpqarray.o
megaraid.o
qlogicfc.o
sym53c8xx.o

3.4. Modifying Your Device Driver to Avoid Bounce Buffers

The entire process of rebuilding a Linux device driver is beyond the scope of this document. However, additional information is available at http://www.xml.com/ldd/chapter/book/index.html.

Modifications are required for all device drivers that are not listed above in the Enabled Device Drivers section.

If your device driver is capable of high−address physical memory DMA I/O, you can modify your device driver to make use of the bounce buffer patch by making the following modifications:

For SCSI Device Drivers: set the `highmem_io` bit in the Scsi_Host_Template structure, then call `scsi_register ( )`.

For IDE Drivers: set the `highmem` in the ide_hwif_t structure, then call `ide_dmaproc ( )`.

1. Call `pci_set_dma_mask ( )` to specify the address bits that the device can successfully use on DMA operations. Modify the code as follows:

   ```
   int pci_set_dma_mask (struct pci_dev *pdev, dma_addr_t mask);
   ```

   If DMA I/O can be supported with the specified mask, `pci_set_dma_mask ( )` will set `pdev->dma_mask` and return 0. For SCSI or IDE, the mask value will also be passed by the mid−level drivers to `blk_queue_bounce_limit ( )` so that bounce buffers are not created for memory directly addressable by the device. Drivers other than SCSI or IDE must call `blk_queue_bounce_limit ( )` directly. Modify the code as follows:

   ```
   void blk_queue_bounce_limit (request_queue_t *q, u64 dma_addr);
   ```

2. Use `pci_map_page (dev, page, offset, size, direction)` to map a memory region so that it is accessible by the peripheral device, instead of `pci_map_single (dev, address, size, direction)`.

   The address parameter for `pci_map_single ( )` correlates to the page and offset parameters of `pci_map_page ( )`. `pci_map_page ( )` supports both the high and low physical memory.

   Use the `virt_to_page ( )` macro to convert an address to a page/offset pair. The macro is defined by including pc.h. For example:

   ```
   void *address;
   struct page *page;
   unsigned long offset;

   page = virt_to_page (address);
   offset = (unsigned long) address & ~PAGE_MASK;
   ```

   Call `pci_unmap_page ( )` after the DMA I/O transfer is complete, the mapping established by `pci_map_page ( )` should be removed by calling `pci_unmap_page ( )`.

   Important:

   `pci_map_single ( )` is implemented using `virt_to_bus ( )` . This function call handles low memory addresses only. Drivers supporting high−address physical memory should no longer call `virt_to_bus ( )` or `bus_to_virt ( )`.
3. Set your driver to map a scatter−gather DMA operation using `pci_map_sg ( )`. The driver should set the page and offset fields instead of the address field of the scatterlist structure. Refer to step 3 for converting an address to a page/offset pair.

If your driver is already using the PCI DMA API, continue to use `pci_map_page ( )` or `pci_map_sg ( )` as appropriate. However, do not use the address field of the scatterlist structure.

4. Raw I/O Variable−Size Optimization Patch

This section provides information on the raw I/O variable−size optimization patch for the Linux 2.4 kernel written by Badari Pulavarty. This patch is also known as the RAW VARY or PAGESIZE_io patch.

The raw I/O variable−size patch changes the block size used for raw I/O from hardsect_size (normally 512 bytes) to 4 kilobytes (K). The patch improves I/O throughput and central processing unit (CPU) utilization by reducing the number of buffer heads needed for raw I/O operations.

4.1. Locating the Patch

You can download the patch from one of the following locations:

- Andrea Arcangeli has made the patch available at http://www.kernel.org/pub/linux/kernel/people/andrea/kernels/v2.4/2.4.18pre7aa2/. The name of the file is *10_rawio−vary−io−1*.
- Alan Cox has included the patch in the *2.4.18pre9−ac2* kernel patch. The patch is available at http://www.kernel.org/pub/linux/kernel/people/alan/linux−2.4/2.4.18/.
- The patch can be found as part of the IO Scalability Package at http://sourceforge.net/projects/lse/io. The name of the patch is *PAGESIZE_io−<version>* listed under the *Raw I/O Enhancements* release.

4.2. Modifying Your Driver for the Raw I/O Variable−Size Optimization Patch

Modifications are required for all device drivers using version 2.4.17 patch. However, rebuilding device drivers is beyond the scope of this document. Additional information is available at http://www.xml.com/ldd/chapter/book/index.html.

In previous versions of this patch, changes were enabled for all drivers. However, the 2.4.17 and later versions of the patch enable only the changes for the Adaptec aic7xxx and the Qlogic ISP1020 SCSI drivers. All other drivers for version 2.4.17 must be modified to make use of the patch.

You will need to modify the code as follows:

Set the `can_do_varyio` bit in the Scsi_Host_Template structure before calling `scsi_register ( ).`

Drivers that have the raw I/O patch enabled must support buffer heads of variable sizes (b_size) in a single I/O request because `hardsect_size` is used until the data buffer is aligned to the 4 K boundary.

5. I/O Request Lock Patch

This section provides information on the I/O request lock patch, also known as the scsi concurrent queuing patch (sior1), written by Johnathan Lahr.

The I/O request lock patch improves scsi I/O performance on Linux 2.4 multi−processor systems by providing concurrent I/O request queuing. There are significant I/O preformance and CPU utilization improvements possible by enabling multi−processors to concurrently drive multiple block devices.

Initially block I/O requests are queued one at a time holding the global spin lock, `io_request_lock`. Once the patch is applied, SCSI requests are queued which holds the specific queue lock targeted by the request. Requests that are made to different devices are queued concurrently, and requests that are made to the same device are queued serially.

5.1. Locating the Patch

You can download the I/O request patch from Sourceforge at http://sourceforge.net/projects/lse/io. The latest version is *sior1−v1.2416*.

Additional patches that enable concurrent queuing can be downloaded from Sourceforge. The patch for the Emulex SCSI/FC is *lpfc_sior1−v0.249* and the patch for Adaptec SCSI is *aic_sior1−v0.249* .

5.2. Modifying Your Driver for the I/O Request Lock Patch

Modifications are required for all device drivers. However, rebuilding device drivers is beyond the scope of this document. Additional information is available at http://www.xml.com/ldd/chapter/book/index.html.

The I/O request lock patch installs concurrent queuing capability into the SCSI midlayer. Concurrent queuing is activated for each SCSI adapter device driver. To activate the device, the `concurrent_queue` field in the `Scsi_Host_Template` must be set when the system registers the driver.

You activate concurrent queuing when you apply the patch. Concurrent queuing ensures access to the drivers `request_queue`. by This access is protected by the `request_queue.queue_lock` acquisition.

6. Additional Resources

The following list of Web sites provides additional information on modifying device drivers and configuring the Linux kernel.

- Information on Dynamic DMA mapping is available at http://lwn.net/2001/0712/a/dma−interface.php3.
- Kernel−HOWTO is available from the Linux Documentation Project at http://www.linuxdoc.org/HOWTO/Kernel−HOWTO.html.
- Linux Device Drivers, 2nd Edition published by O'Reilly is available online at http://www.xml.com/ldd/chapter/book/index.html.