I/O Performance HOWTO

Sharon Snider

v1.1, 05/2002

Revision History

Revision v1.1	2002-05-01	sds
	Updated technical information and links.	
Revision v1.0	2002-04-01	sds
	Wrote and converted to DocBook XML.	

Abstract

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

Table of Contents

Distribution Policy 1
Introduction 1
Avoiding Bounce Buffers 2
- Memory and Addressing in the Linux 2.4 Kernel 2
- The Problem with Bounce Buffers 2
- Locating the Patch 2
- Modifying Your Device Driver to Avoid Bounce Buffers 3

Raw I/O Variable-Size Optimization Patch 4
- Locating the Patch 4
- Modifying Your Driver for the Raw I/O Variable-Size Optimization Patch 4

I/O Request Lock Patch 5
- Locating the Patch 5
- Modifying Your Driver for the I/O Request Lock Patch 5

Additional Resources 5

Distribution Policy

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 [mailto:sniderssd@us.ibm.com].

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.

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 starting at PAGE_OFFSET. On i386 systems the default mapping scheme limits kernel-mode addressability to the first gigabyte (GB) of physical memory, also known as low memory. Conversely, high memory is normally the memory above 1 GB. High memory is not directly accessible or permanently mapped by the kernel. Support for high memory is an option that is enabled during configuration of the Linux kernel.

The Problem with Bounce Buffers

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

Systems with a large amount of high memory and intense I/O activity can create a large number of bounce buffers that can cause memory shortage problems. In addition, the excessive number of bounce buffer data copies can lead to performance degradation.

Peripheral component interface (PCI) devices normally address up to 4 GB of physical memory. When a bounce buffer is used for high 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.

Locating the Patch

The latest version of the bounce buffer patch is *block-highmem-all-18b.bz2*, and it is available from Andrea Arcangeli's -aa series kernels at http://kernel.org/pub/linux/kernel/people/andrea/kernels/v2.4/.

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 the `Code maturity level options` category and specify "y" to `Prompt for development and/or incomplete code/drivers`.

High Memory Support - To enable support for physical memory that is greater than 1 GB, select the `Processor type and features` category, and select a value from the `High Memory Support` option.

High Memory I/O Support - To enable DMA I/O to physical addresses greater than 1 GB, select the `Processor type and features` category, and enter "y" to the `HIGHMEM I/O support` option. This configuration option is a new option introduced by the bounce buffer patch.

Enabled Device Drivers

The bounce buffer patch provides the kernel infrastructure, as well as the SCSI and IDE mid-level driver modifications to support DMA I/O to high memory. Updates for several device drivers to make use of the added support are also included with the patch.

If the bounce buffer patch is applied and you configure the kernel to support high memory I/O, many IDE configurations and the 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

Modifying Your Device Driver to Avoid Bounce Buffers

If your device drivers are not listed above in the Enabled Device Drivers section, and the device is capable of high-memory DMA I/O, you can modify your device driver to make use of the bounce buffer patch as follows. More information on rebuilding a Linux device driver is available at http://www.xml.com/ldd/chapter/book/index.html.

1. A.) For SCSI Adapter Drivers: set the *`highmem_io`* bit in the *`Scsi_Host_Template`* structure.

   B.) For IDE Adapter Drivers: set the *`highmem`*bit in the *`ide_hwif_t`* structure.

2. Call *`pci_set_dma_mask(struct pci_dev *pdev, dma_addr_t mask)`* to specify the address bits that the device can successfully use on DMA operations.

   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(request_queue_t *q, u64 dma_addr)`* 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.

3. Use *`pci_map_page(dev, page, offset, size, direction)`*, instead of *`pci_map_single(dev, address, size, direction)`* to map a memory region so that it is accessible by the peripheral device. *`pci_map_page()`* supports both high and low memory.

   The *`address`* parameter for *`pci_map_single()`* correlates to the *`page`* and *`offset`* parameters for *`pci_map_page()`*. Use the *`virt_to_page()`* macro to convert an *`address`* to a *`page`* and *`offset`*. The *`virt_to_page()`* macro is defined by including pci.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 to remove the mapping established by *pci_map_page()*.

Note

pci_map_single() is implemented using *virt_to_bus()*. *virt_to_bus()* handles low memory addresses only. Drivers supporting high memory should no longer call *virt_to_bus()* or *bus_to_virt()*.

4. If your driver calls *pci_map_sg()* to map a scatter-gather DMA operation, your 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* and *offset*.

Note

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.

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 CPU utilization by reducing the number of buffer heads needed for raw I/O operations.

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 is available from SourceForge at http://sourceforge.net/projects/lse/io. The latest version is *PAGESIZE_io-2.4.17.patch*.

Modifying Your Driver for the Raw I/O Variable-Size Optimization Patch

In previous versions of this patch, changes were enabled for all drivers. However, the 2.4.17 and later versions of the patch enable the changes only for the Adaptec, Qlogic ISP1020, and IBM ServerRAID

drivers. All other drivers for version 2.4.17 and later must be modified to make use of the patch by setting the *can_do_varyio* bit in the *Scsi_Host_Template* structure.

Note

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 on a 4 K boundary.

Additional information is available on rebuilding Linux device drivers at http://www.xml.com/ldd/chapter/book/index.html.

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 performance and CPU utilization improvements possible by enabling multi-processors to concurrently drive multiple block devices.

Before the patch is applied 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 while holding the lock specific to the queue associated with the request. Requests that are made to different devices are queued concurrently, and requests that are made to the same device are queued serially.

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*. Patches that enable concurrent queuing for specific drivers are also available at SourceForge. The patch for the Emulex SCSI/FC is *lpfc_sior1-v0.249* and the patch for Adaptec SCSI is *aic_sior1-v0.249*.

Modifying Your Driver for the I/O Request Lock Patch

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 driver, the *concurrent_queue* field in the *Scsi_Host_Template* structure must be set when the driver is registered.

Note

Drivers that activate concurrent queuing must ensure that any access of the *request_queue* by the driver is protected by the *request_queue.queue_lock*.

Additional information is available on rebuilding device drivers at http://www.xml.com/ldd/chapter/book/index.html.

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/HOW-TO/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.