Linux Software RAID Level 0 Technique for High Performance Computing by using PCI-Express based SSD

Jae Gi Son, Taegyeong Kim, Kuk Jin Jang, *Hyedong Jung
Department of Industrial Convergence, Korea Electronics Technology Institute, Korea
jgson@keti.re.kr, taegyeong@keti.re.kr, kjang@keti.re.kr, *hudson@keti.re.kr

Abstract—The Linux-based legacy server systems are configured and used with software RAID to improve the performance of the disk I/O. Server systems requiring high performance prefer a special SSD that connects directly with the PCI-express bus to the SATA interface. However, the problem is that the current Linux kernel and software RAID are difficult to optimize the high-performance SSD based on PCI-Express because it is designed to be optimized for the conventional hard disk. Therefore, we propose the efficient method using re-combination and re-mapping techniques to improve the performance of software RAID level-0 provided on the Linux kernel level. This proposed method is designed to have more bandwidth at a time by reducing the number of system calls considering the block I/O characteristics of Linux kernel and RAID level 0. As a low-level I/O benchmarking tool, XDD is used to evaluate the performance of the proposed method. According to the experimental results, our performance gains are 28.4% on write bandwidth and 13.77% on read bandwidth compared with legacy software RAID. Moreover, CPU occupancy rates are decreased 81.2% and 77.8%, respectively.

Keywords—Software RAID, PCI-E Express SSD, High Performance Computing, Disk I/O, Memory Block Device

1. INTRODUCTION

As big data, parallel and distributed processing become widespread, the demand for high-speed data storage technologies continues to rise. Moreover, more recent applications require fast response time with high throughput. However, as hard-disk input-output speed depends on mechanical movement, the speed lacks in comparison to processor and network transfer speed. Therefore, flash-memory based SSD have been increasingly used in personal computers and laptops and research is actively pursued related to non-volatile memory [1]. Currently companies such as FusionIO, Intel, etc. have developed and commercialized PCI-Express based high performance SSD products. Unlike SSD based on the common SATA interface, PCI-Express based SSD directly access the system bus architecture. Therefore, in this paper we designate all such devices as memory block devices. Despite the improved performance of PCI-Express based SSD when compared to hard disk drives of SATA SSD, system using such devices are configured using software RAID (Redundant Array of Inexpensive Disks). This is due to the non-existence of hardware-level RAID for SSD. Therefore in most Linux systems, software RAID (MD: Multiple Disk) is used to improve performance. However, the software RAID provided with standard Linux distributions was developed without consideration for high-speed block devices resulting in suboptimal performance when used with block devices. This is due to the fact that software RAID does simple mapping of input-output (IO) requests to the actual devices which are collected and optimized by the IO scheduler. In the case of high-speed block memory devices, as the devices are connected directly to the system bus in order to improve performance, the IO Scheduler does not exist. Therefore, a need exists to design the architecture for software RAID taking memory block devices into consideration. Therefore, in this paper we propose a method for improving the performance when applying software RAID Level 0 provided by standard Linux distributions. The proposed method has the following characteristics: First, the number
of system calls within the Linux system is reduced to lower processor overhead. Second, the fast response times and high throughput of memory block devices are considered to improve the performance of software RAID.

The paper is organized as follows. In section 2, the technological background is described followed by an explanation of the proposed method for software RAID Level 0 for PCI-Express based high-performance SSD. Section 3 describes proposed method and error handling. In section 4, the performance is evaluated. Section 5 is the conclusion and direction for future research.

2. BACKGROUND

In Linux, two general approaches for software RAID are provided. These are the multiple disk (MD) and device mapper (DM). Though these two methods are similar, MD was developed with RAID considerations. DM was developed for virtual block devices resulting in a framework which allows for direct access to real block devices. In this section we concentrate on MD which has shown better performance comparatively.

2.1 RAID LEVEL 0 IN MD

An MD device is a Linux virtual block device consisting of several disk drives combined together to provide larger capacity and improved reliability [2]. Figure 1 depicts the general hierarchical structure of an MD device and general disk hierarchy. From Figure 1, MD functions as a virtual device between the file system layer and the block device layer. The virtual MD layer receives IO requests and distributes them accordingly to the underlying disk layer.

RAID Level 0 comprised of data striping to provide improved performance. In general, RAID systems interleave sections of consecutive data on different devices in order to improve parallelism. These sections of data which are interleaving and allow parallel access are called stripes [3]. For example, for writing 1MB file, the files are divided into sections of 64KB and recorded by interleaving the section across each of the disks. Even though the data itself is not accessed in parallel, performance improves when compared to using a single device.

![Figure 1. General disk hierarchy (left) and general hierarchical structure of an MD device. (right)](image)

The MD provided by the Linux kernel handles 'bio request' in the sequence depicted in Figure 2 in order to execute IO on the actual device.

![Figure 2. RAID level 0 processing of an MD](image)

Additionally, the procedure for handling of 'bio request' is shown in Code 1. As shown in Code 1, the key procedure of software RAID can be analyzed as follows:

1. From the software RAID device of bio (bio bi_bdev) passed to the `generic make request()` function, a function call is made to the callback `md make request`
2. The `md make request()` function makes a subsequent call to the proper `make request` function depending on the RAID Level. For RAID Level 0, the `raid0 make request()` function is called.

3. From the RAID Level 0 handler `raid0 make request()` analyzes the sector and size of the bio passed to the function and maps it to the actual device such as a hard disk drive. For bio which exist across two block devices are divided and mapped accordingly.

4. The mapped block device make a call once again to the `generic make request` and passes the `bio request` to the actual device.

1. `__generic_make_request()` Call `make_request` followed by `bio->bi_bdev`.
2. `md_make_request()` Call `raid0_make_request` for raid 0 module.
3. `raid0_make_request()` Check real block device to process the request, map `bio->bi_bdev` to the block device.
4. `__generic_make_request()` Send switched `bio->bi_bdev`.
5. `__make_request()` Insert request to request queue of real block device.
6. Process `request` in real block device.

**Code 1. Procedure of request in MD**

As can be seen in the key highlight of the MD handling procedure, sequential IO events occur and in the case of large data recursive calls can result in drawbacks of high system overhead. In particular, performance improvement is limited due to the IO characteristics of disk devices and the stripe-level transfer of data for conciseness of software RAID. In the case of memory block devices with fast response times and high throughput, random memory access is possible therefore it is advantageous to send as much data as possible. However, the current software RAID is unable to utilize such characteristics.

**2.2 Device Mapper**

The device mapper was not originally developed for RAID but as a framework for storage devices. Thus the architecture allows for direct mapping of a virtual block device to an arbitrary physical device. DMRAID was developed to add support for RAID in DM [4].

3. **Linux Software RAID Level 0 Technique for PCI-Express based High-Performance SSD**

**3.1 Proposed RAID Level 0 Technique**

PCI-Express based high performance SSD have greatly improved IO performance when compared to general disk drives. However, as the current Linux IO layer is optimized for low-speed devices, it is unable to utilize the full performance potential of high speed block devices. Unlike slower hard disk drives, high-speed memory block devices are directly connected to the PCI-Express bus and therefore is unaffected by the Linux IO scheduler. Moreover, as the architecture of direct memory access (DMA) is organized in Scatter-Gather structure, which allows for transfer of non-consecutive segments of memory, large data transfers are possible. The proposed methods was designed to operate under the following assumptions.

The memory block device is a device with fast response time and high throughput supports random access in a Scatter-Gather DMA structure. Analyzing the MD procedure from the Linux kernel under this assumption, an increase in performance can be expected from sending the first bio (block IO) and the (number of devices)*n+1 bio. Figure 3 depicts how the Linux kernel IO process when the strip size is 4 KB. Each 4KB bio is added on the underlying physical device in sequence.

**Figure 3. Block IO process in MD of Linux kernel**

As can be seen in Figure 3, bio1 and bio5 occur on the same physical device. A significant gain in performance can occur if bio1 and bio5 can be
processed in one IO event. In Figure 4, a total of 6 system calls occurs to transfer 24KB. In other words 
(Number of system calls) = (IO Data size) / (stripe size).

Taking into consideration the characteristics of the 'bio request' and the MD process of the Linux kernel, and the assumption of low delay and high throughput of the high-speed memory block device the following methods are proposed:

1. Variable-length IO: The fixed stripe size of the bio request of MD is changed to allow variable length
2. Remapping & Reassemble: Remapping and reassemble occurs taking into consideration the characteristics of RAID Level 0 of MD

To support high-speed memory block devices such as PCI-Express based SSD according to the proposed RAID Level 0 method, the included software RAID Level 0 of the standard Linux kernel was modified and improved according to the proposed method. The improved software RAID in this paper is designated as Enhanced MD.

3.2 ERROR HANDLING

In IO systems, error handling is an important factor that has a significant impact on performance. bio of the Enhanced MD applied with the proposed method generates a new 'bio request' through remapping & reassembly. The error handling of the newly generated 'bio request' is forwarded to the Linux 'bio error handler'.

4. PERFORMANCE EVALUATION

4.1 EVALUATION ENVIRONMENT

The performance of the proposed method for software RAID Level 0 was evaluated using the PCI-Express based 910 SSD from Intel. The Intel 910 SSD is configured with 2 physical SSD on a single interface card. For the performance evaluation, 2 interface cards were connected to PCI-Express 4x for a total of 4 physical SSD. The specifications of the evaluation environment are shown in detail in Table 1. We used CentOS with 2.6.32_x86_64 kernel and set the stripe size to 4096 bytes. Intel Xeon X5550 and 8GB memory were used for the system.
Table 1. Evaluation environment

<table>
<thead>
<tr>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>RAM</td>
</tr>
<tr>
<td>OS</td>
</tr>
<tr>
<td>Kernel</td>
</tr>
<tr>
<td>SSD Device</td>
</tr>
<tr>
<td>Stripe Size</td>
</tr>
</tbody>
</table>

For evaluation, the stripe size was set to 4KB using the mdadm utility of MD. For comparison, an unmodified MD installed by default on CentOS 6.0 was used with the same environment to compare measurements of CPU usage and IO bandwidth. The performance evaluation tool XDD was used [5]. The XDD benchmark is a tool to measure disk performance and can be used to analyze low-speed IO in software RAID. The tool is commonly used to effectively evaluate block device performance. The XDD performance test was set to run with IO on 16 threads to allow operation under a regular system load.

4.2 Experimental Results

The developed software RAID Level 0 showed optimal performance with IO of size greater than (number of disk times stripe size) due to the characteristics of the key operating algorithm. From Figure 5 comparing the write bandwidth, it can be seen that the bandwidth greatly improves with Enhanced MD. When the IO size is 4-16 KB, the results are similar to the characteristics of RAID Level 0. In this case, the request are handled as sequential writes to a single memory block device. There is a slight performance degradation when the IO size in 8KB. When the IO size is 32 ~ 1024 KB, significant gains in performance is achieved by the proposed method. When the size is over 2048 KB, it was analyzed that performance improvements are no longer achieved due to the fact that the size is over the maximum page cache allocation of Linux. Overall, the write bandwidth of Enhanced MD achieved an improvement in performance on average of 28.24%.

Figure 5. Comparison of writing throughput

Figure 6 compares the CPU usage from the evaluation. From the results, the overall CPU usage across all sizes is less with Enhanced MD. This was analyzed to be due to the proposed methods remapping and reassembly, which results in a significant reduction in the number of system calls. As shown in Figure, the CPU usage is reduced by an average of 77.8%. The results for read performance were also improved in general as shown in Figure 7. As before with the result of write bandwidth, similar performance when compared with MD is shown for sizes of 4 - 16KB due to the characteristics of RAID.
Level 0. However, for IO sizes of 32 - 128 KB, the memory allocation of the proposed method was analyzed to be the cause of the performance degradation in page caching. In general, the read bandwidth showed an improvement of about 13.77% on average.

Figure 7. Comparison of reading throughput

The CPU usage for reads is similar to that of write operations as shown in Figure 8. From the figure, the overall CPU usage has been reduced significantly. These results confirm the positive improvements of the proposed system. For read operations there was an average of 81.2% reduction in CPU usage.

Figure 8. Comparison of CPU usage in reading

From the overall analysis of the entire experimental results, it can be seen that the performance of the proposed method shows significant improvement compared to the software RAID of standard Linux. Moreover, reduction in CPU overhead and improvements in IO performance can lead to improvements in overall system performance. In particular, as shown from the results from XDD experiments, by improving low-level IO performance, the proposed method will lead to significant improvements to systems which do not use page caching such as database systems.

5. CONCLUSION

The standard Linux kernel and software RAID is optimized for traditional block devices and is not suitable for current high-speed memory block devices. In the case of server systems which require high performance, limitations are shown in performance despite the use of high-speed memory block devices configure with SW RAID. Therefore, this paper proposed a method which provides a solution to the performance degradation when using high-speed memory block devices with the standard Linux kernel and software RAID Level 0. The proposed method has the following characteristics. First, the stripe unit of data used in RAID block IO was improved for variable length resulting in a reduction of system call within the Linux kernel. This reduces the overhead of the processor. Second, taking into consideration the fast response time and large bandwidth of the memory block device, the performance of software RAID is greatly improved. In addition, the performance was confirmed through evaluation using the low-level IO performance evaluation tool XDD. Based on the results in this paper, further research is needed to apply the method to RAID Level 4 and Level 5 while improving overall system reliability and safety. Furthermore, research regarding block migration on write operations which occurs due to the characteristics of SSD is needed.

ACKNOWLEDGEMENT

This work was partly supported by the R&D program of MSIP/LEIT [10044910, Development of Multimodality Imaging and 3D Simulation-Based Integrative Diagnosis-Treatment Support Software System for Cardiovascular Diseases] and [10045299, Development of Integrated Resource Management Solution for VDI Service System].
REFERENCES


