# Impacts on the I/O Performance of the Virtual Disk in Xen

by

Quan Zhang

B. Computer Science and Technology, Nankai University, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

January 2012

# Abstract

Storage systems are complicated, especially after the virtualization technology has been introduced. As is a key metric, the I/O throughput of the storage in the virtual machine environment is remarkably lower than that in the non-virtualized environment. This disparity should be alleviated because of the growing popularity of virtual machines.

In this paper, four factors, which affect the I/O throughput of the generic Linux storage subsystem and further degrade virtualization on this storage subsystem, are testified by quantitative approaches, leading to the effective solutions of improving the I/O throughput. The improvements are verified by experiments; however, the overhead of virtualization is inevitable. Therefore, a further way to offset the overhead is also given in this thesis.

# Table of Contents

# List of Tables

# List of Figures

# Glossary

**ISCSI** Internet Small Computer System Interface

**RPC** Remote Procedure Call

**TCP** Transmission Control Protocol

# Acknowledgments

Foremost, I would like to show my sincere gratitude to my supervisor, Andrew Warfield, and my co-supervisor, Norm Hutchinson. Their guidance helped me in the whole process of research and writing this thesis.

Besides my supervisors, I would like to thank the people in my lab, especially Dutch T. Meyer, for their help during my research work in the lab.

My sincere thanks also goes to Lidong Zhou, for offering me the internship opportunity in Microsoft Research Asia, where I regained my nerve after the failure of the previous research work.

I am also grateful to my parents, who gave my birth and support me throughout my life.

Last but not the least, I would like to thank my lover, Kate Tang, who offered me lots of help and colored my life.

# Chapter 1

# INTRODUCTION

The computational power of processors is still showing a growth rate in orders of magnitude every 5 years. Although the storage capacity keeps a similar trend; unfortunately, hard disk performance has not been able to rise at nearly the same rate [7]. As a result, storage subsystem has been the dominant bottleneck in the overall system until the hard disk is abandoned.

The performance of storage systems is not just decided by the specifications of the hard disk, but is also affected by the software. In other words, the design of the storage subsystem can have a great influence on the I/O performance. However, not much attention is focused on this aspect of the design.

Xen platform leads the resurgence of interest for diverse uses, including server consolidation, cloud computing and secure computing platforms [1]. After its appearance, additional control flow needed by the storage subsystem in that virtual machine worsened the I/O performance. This retrogress aggravates the bottleneck as mentioned above. As Xen is becoming an industry standard for virtualization, this performance issue is pressing to be solved.

Although the mechanism of virtualization on the storage subsystem is commonly known, it is still hard to analyze what factors can impact the I/O performance. Without this knowledge, it is impossible to improve the I/O performance of the virtualized storage system in the virtual machine. Hence, the first part of this paper (Chapter 2 & Chapter 3) will analyze the origins of these impacts and testify them using quantitative approaches. Afterwards, verifications and improvements

made specifically to overcome these impacts will be presented in Chapter 4. Because the impacts are inevitable, a system to offset the overhead on the performance will be given in Chapter 5. This part will constitute detailed design and implementation of the system. Finally, evaluation and further works will be discussed in Chapter 6 and Chapter 7 of this paper.

# Chapter 2

# ANALYSIS

## 2.1 Simplified Control Flow of the Non-virtualized Storage System

Figure 2.1 shows the simplified control flow during the execution of *dd* with a direct I/O operation. The procedure can be divided into five stages. Stage one is from the entry point of *dd* to the insertion of a new request into the request queue of the generic block layer. Stage two is from dequeuing of the request until the submission of the request to the device controller. Stage three is the process by which the disk controller accesses the data on the disk. This process involves positioning and transferring the data. Stage four starts when the interrupt handler is activated and ends at the awakening of the dd process. Stage five ranges from the resumption until the end of *dd*. Meanwhile, multiple layers in Figure 2.1 illustrates the fact that the polymorphic request is processed by different agents, including

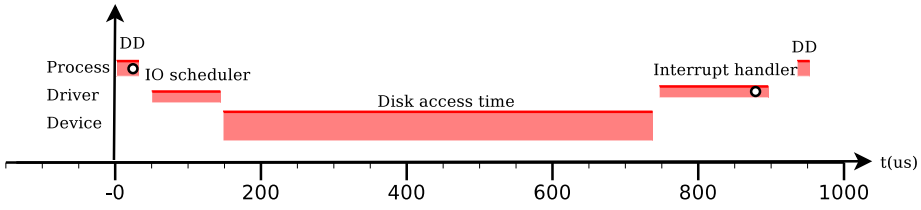


**Figure 2.1:** Simplified control flow of the non-virtualized storage system

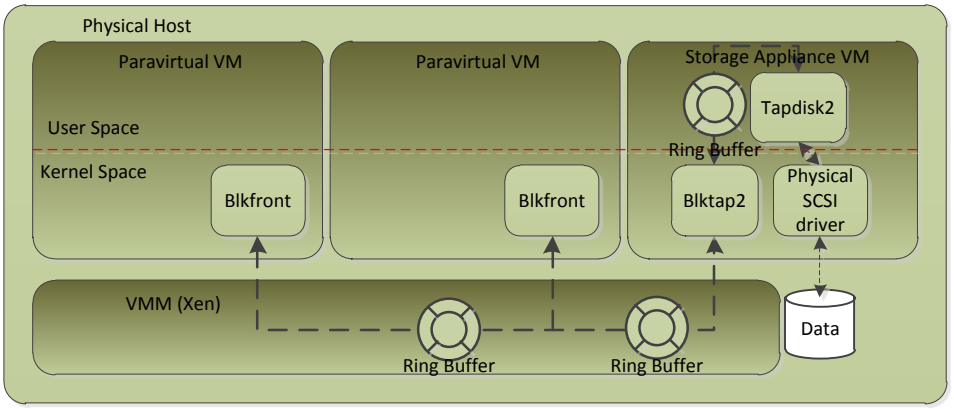


**Figure 2.2:** Mechanism of storage para-virtualization in Xen

the *dd* process, the I/O scheduler routine, the device controller and the interrupt handler.

Asynchronous I/O or buffer I/O is different from direct I/O. Multiple I/O requests (*bio*s) can aggregate into one request in the request queue during stage one. The merged request is then submitted to the disk, thereby reducing the time spent on positioning the request and stages two and four.

## 2.2 Storage Virtualization in Xen

Xen provides a para-virtualized Linux machine, which means that the Linux OS running in the virtual machine is different than that of the generic Linux. In the storage subsystem, the block device driver (*blkfront*) in the virtual machine redirects the I/O requests in the request queue to *blkback* which is a component in the storage appliance virtual machine by a ring buffer. The *blkback* routine which is activated by a event in Xen submits the requests to the request queue. Then, the virtual block device driver (*blktap2*) operates on the queue and uses another ring buffer to redirect the I/O requests to *tapdisk2* in the user space of the storage appliance virtual machine. Finally, the ring buffer is a form of character device polled by *tapdisk2*, so that *tapdisk2* is aware of the arrival of the new I/O request and sequentially submits it to the physical disk. It should be noted that the I/O responses are proceeded in the reverse flow as reflected in Figure 2.2.

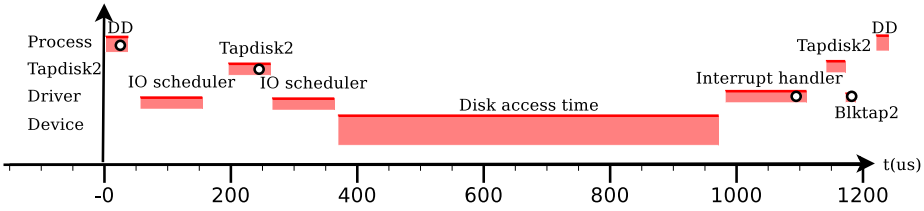


**Figure 2.3:** Simplified control flow of the virtualized storage system

A ring buffer is logically a lockless structure that stores the I/O descriptors. Each descriptor can represent a request that consists of several segments while the maximum segments in one request arbitrarily is set to eleven. The number of I/O descriptors is determined by the size of the ring buffer.

In this paper, logical control flow that describes the I/O request operated by *blktap2* and *tapdisk2* is extracted to allow for simple investigation. This control flow is also mapped to a I/O operation on a virtual disk (*tapdev*) visible to the storage appliance virtual machine. Further experiments illustrate that the I/O path inside *blkfront* and *blkback* has same impacts. Therefore, by taking the same enhancement methods as described below, the I/O performance in the virtual machine would get the same effective improvements.

## 2.3 Simplified Control Flow of the Virtualized Storage System

After the above logic is supplemented to constitute the virtual disk in the storage appliance virtual machine, the control flow on *dd* with a direct I/O operation on the virtual disk is sorted into nine stages as depicted in Figure 2.3. The additional time cost is due to the execution of *blktap2* and *tapdisk2* and the extra context switch to *tapdisk2*.

# Chapter 3

# QUANTITATIVE APPROACHES

## 3.1 Experimental Setup

Two machines connected via a LAN hub to each other are used for experiments presented in this paper. One machine is mainly used to analyze the impacts on the I/O throughput from Xen. This machine has an Intel Core 2 Quad CPU, clock rate 2.66GHz with 4GB of RAM and a Seagate 320GB 7200.11 hard drive, running Xen 4.1. The other machine used as the server in Chapter 5 has an Intel Core 2 Dual CPU, clock rate 3.00GHz with 2GB of RAM and a Seagate 320GB 7200.11 hard drive, running Ubuntu 10.10.

The additional time cost as shown in Figure 2.3 accounts for the fact that the I/O throughput on the virtual disk (virtualized storage system) is worse than that on the physical disk (non-virtualized storage system). This fact is further illustrated in Figure 3.1. On the other hand, Figure 2.3 indicates that the time cost on *blktap2* and *tapdisk2* is relatively constant, which implies that the gap between the I/O throughput on the virtual disk and that on the physical disk should be reduced as the block size of *dd* with a direct I/O write operation increases. Contrarily, their ratio actually decreases when the block size is bigger than 32KB. This contradiction suggests that performance implications of *blktap2* and *tapdisk2* can be further advanced.

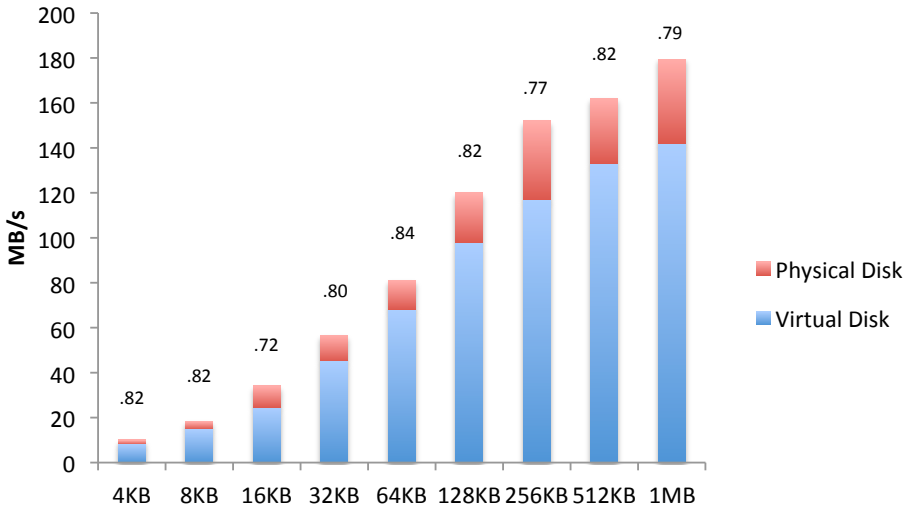


**Figure 3.1:** Difference between the I/O throughput of the virtual disk and that of the physical disk and their ratios (block size of *dd* with a direct I/O write operation is varible)

## 3.2 I/O Requests Aggregation (I/O Aggregation)

It is observed from Figure 3.1 that the performance is enhanced when the block size of *dd* with a direct I/O write operation is increased. Theoretically, there should be a gain in the time cost on the disk when the block size is increased because the data transfer time on the disk raises. However, the time cost on *dd* and the device driver (shown as stages two and four in Figure 2.1) should stay relatively constant. This theory infers that the time cost per unit size of direct I/O is decreasing when the block size is increasing, which explains the benefits of I/O aggregation.

In order to use quantitative approach to prove the above hypothesis, two trace points tagged in Figure 2.1 are inserted into the kernel. One of these trace points is in the *dio_bio_submit* function, and the other is in the *dio_bio_end_io* function. Both trace points can provide higher resolution time cost on the driver and the disk access. They can also reflect the I/O request aggregation and detect the upper bound of the request aggregation.

Table 3.1 records the time elapse between the two trace points on a direct I/O

**Table 3.1:** Time cost on the driver and the disk access of a direct I/O write operation with different block sizes

| Request size (KB) | Time cost (us) | Number of *bio*s |
|---|---|---|
| 4 | 345 | 1 |
| 8 | 373 | 1 |
| 16 | 399 | 1 |
| 32 | 476 | 1 |
| 64 | 605 | 1 |
| 128 | 887 | 1 |
| 256 | 1412 | 1 |
| 512 | 2383 | 1 |
| 1024 | 4830 | 2 |

write operation with different block sizes and the number of internal *bio*s. Take the 1MB block size as an example, because the number of *bio*s submitting to the request queue is two, there are four trace points with the time elapse between the first and last point. From the variations on the number of *bio*s, it is inferred that the maximum size of one request in the queue is less than 1MB. The above is true given that the maximum size of one *bio* is equal to that of one request under the condition of I/O with data in single buffer. Hence, the 1MB size of a direct write should split to two *bio*s before submitting to the queue, thereby increasing the constant time cost on stage one and five, the driver ( stages two and four as depicted in Figure 2.1) and the disk's data positioning if there is no further aggregation inside the driver. All of these time costs impact the I/O throughput.

Suppose that X is the constant time cost and Y is the variant time cost per unit I/O size; then,

$$\boxed{\text{Time cost} = X + Y \times \text{Size}}$$

After replacing the time cost with values of 128KB, 256KB and 512KB from Table 3.1, X is approximated to be 335 us, while Y is around 4 us per byte.

It should be emphasized that when the block size is 1MB, data transfer is the major time cost while the constant time accounts for less than 10%. Therefore, increasing the maximum request size in the request queue does not have a significant effect. Moreover, the gap between the end of stage four and the beginning of

stage five in Figure 2.3 that represents the time cost for the context switch to the *dd* process is another illustration of how the constant time plays a role in the I/O throughput of *dd*.

Two more trace points are added as shown in Figure 3 to investigate whether *blktap2* and *tapdisk2* have any further impacts on the benefits of I/O aggregation. They are in functions *dio_bio_submit* and *dio_bio_end_aio*, because *tapdisk2* uses direct asynchronous I/O. By adding these two trace points, the additional time cost on *blktap2* and *tapdisk2* can be calculated. Suppose that $X_1$, $X_2$, $X_3$ and $X_4$ are the sequential time-stamps of those four trace points; then,

$$\boxed{\text{Additional time cost} = (X_2 - X_1) + (X_4 - X_3)}$$

Table 3.2 lists the concrete values for the additional time cost on *blktap2* and *tapdisk2* using the above equation. From the number of *bio*s, it is inferred that the maximum size of the request in the request queue is limited by *blktap2* and *tapdisk2*. This limitation restricts the benefits gained from I/O aggregation by increasing the constant time cost in stages one, two, seven and eight as shown in Figure 2.3. In fact, the maximum size of the request in the request queue is found to be determined by that in the ring buffer, after investigating at the source code level and testifying through the number of *bio*s. Furthermore, it is observed from Table 3.2 that the larger the block size of a direct I/O write operation is, the higher the time cost will be on *blktap2* and *tapdisk2*. These two factors constitute the first element of how Xen hurts the I/O throughput.

## 3.3 Length of Request Queue

When using buffer I/O or asynchronous I/O, the length of the request queue takes effect. It determines the maximum number of segments held in the request queue, so as to decide the number of context switches to the *dd* process. Because theoretically:

$$\boxed{\text{Number of context switch} = \text{Number of requests} \div \text{Length of queue}}$$

In addition, the size of the merged *virtual request* increases with the number of segments held in the request queue. Therefore, more benefits can be obtained from I/O aggregation by decreasing the number of system calls in *tapdisk2*.

**Table 3.2:** Additional time cost on *blktap2* and *tapdisk2* during a direct I/O write operation with different block sizes

| Request size (KB) | Time cost (us) | Number of *bios* |
|---|---|---|
| 4 | 111 | 1 |
| 8 | 117 | 1 |
| 16 | 136 | 1 |
| 32 | 171 | 1 |
| 64 | 299 | 2 |
| 128 | 416 | 3 |
| 256 | 678 | 6 |
| 512 | 1315 | 12 |

**Table 3.3:** Additional time cost on *blktap2* and *tapdisk2* during a direct I/O write operation with different block sizes

| Request size (KB) | Number of *bios* |
|---|---|
| 4 | 1 |
| 8 | 1 |
| 16 | 1 |
| 32 | 1 |
| 64 | 2 |
| 128 | 3 |
| 256 | 6 |
| 512 | 12 |

Two trace points are inserted into the kernel to investigate whether *blktap2* and *tapdisk2* would have a similar impact on the length of the request queue. One is in the function *submit_bh* and the other is in *end_bio_bh_io_sync*. Both trace points are nearby the points tagged in Figure 2.1; however, they are used to trace buffer I/O instead of direct I/O. Table 3.5, which records the results traced from buffer I/O consisting of 4096 write operations each with a 4KB block size, shows the number of request submissions and the number of segments inside each of the request for both physical and virtual disks.

Table 3.5 demonstrates that *blktap2* and *tapdisk2* limit the length of the request queue, thereby increasing the time cost on the driver (as shown in stages two and eight of Figure 2.3), the time cost on the execution of *tapdisk2* and the time cost

**Table 3.4:** Additional time cost on *blktap2* and *tapdisk2* during a direct I/O write operation with different block sizes

| Request size (KB) | Number of *bios* |
|---|---|
| 4 | 1 |
| 8 | 1 |
| 16 | 1 |
| 32 | 1 |
| 64 | 1 |
| 128 | 2 |
| 256 | 3 |
| 512 | 6 |

**Table 3.5:** Different numbers of request submissions and segments inside each request on both physical and virtual disks

| | Physical disk | Virtual disk |
|---|---|---|
| Number of submissions | 4 | 8 |
| Number of segments | 2396 | 1636 |
| | | 352 |
| | 647 | 352 |
| | | 352 |
| | 555 | 352 |
| | | 352 |
| | 498 | 352 |
| | | 348 |

on the context switch to *tapdisk2*. By investigating at the source code level and verifying through the number of segments, it is discovered that the length of the request queue is actually limited by the size of the ring buffer.

## 3.4 Virtual Request Fragmentation

In *tapdisk2*, *virtual request* is partitioned into 4KB and then merged together. The possibility of this merge depends on whether the addresses of both the physical disk and the virtual memory of continuous *virtual request*s are contiguous. As stated above, this contiguity affects the number of system calls in *tapdisk2* (the context

switch between *tapdisk2* and the kernel). The number of merged *virtual request*s is traced to observe this effect, and it is concluded that three circumstances can affect the contiguity:

- Improper address in the virtual memory allocated for *virtual request* in *tapdisk2*
- Variation on the address of the physical disk of *virtual request*
- Dysfunctional I/O scheduler that reorders the requests

## 3.5 Miss Rate of *Meta-data* Cache

Miss rate of *meta-data* cache is another source of performance implication. It is used when there is an additional address translation layer in *tapdisk2*. For instance, *Parallax* [6], a virtual storage system that uses deduplication to provide fast clone and snapshot on the virtual disk has two address translation layers: *V2P* and *P2M*. *V2P* is used for deduplication and *P2M* is used to solve defragmentation problems. However, additional access to *meta-data* causes extra I/O, leading to the decrease of the I/O throughput. As a result, *meta-data* cache is added to counteract this effect. Therefore, miss rate of *meta-data* cache is a significant metric when investigating whether *blktap2* and *tapdisk2* has any impact on the I/O throughput.

Additional requests to access *meta-data* on the disk are tagged and traced in *Parallax* to measure the miss rate of *meta-data* cache. Table 3.7 records the miss rates for *V2P* and *P2M* during the execution of *dd* with 4KB buffer I/O for 32768 times.

**Table 3.6:** Difference between the I/O throughput on the virtual disks with and without *Parallax*

| Block size (KB) | Virtual disk 1 (MB/s) | Virtual disk 2 (MB/s) |
|---|---|---|
| 4 | 18 | 20 |
| 8 | 29 | 31 |
| 16 | 40 | 46 |
| 32 | 59 | 64 |

**Table 3.7:** *V2P & P2M* miss cache rates in *Parallax*

| *V2P* miss rate | *P2M* miss rate |
|---|---|
| 0 | 0.98% |

# Chapter 4

# IMPROVEMENTS

After using quantitative approaches to detect the impacts of *blktap2* and *tapdisk2* on the I/O throughput, the opportunity to eliminate these impacts appears. More specifically, because the size of the maximum request and that of the ring buffer are the determinant factors, they need to be enlarged in order to offset the limitations on I/O aggregation and the length of the request queue. Currently, the maximum request size is 44KB and the size of the ring buffer is 4KB. After changing the maximum request size to 88KB and the size of the ring buffer to 8KB, Figure 4.1 shows a decrease in the additional time cost (refer to Table 3.2) during the execution of *dd* with 64KB direct I/O write operation on the left portion; and the time cost between the first trace point in the function *submit_bh* and the last trace point in *end_bio_bh_io_sync* during the execution of *dd* with 4KB buffer I/O write operations for 4096 times on the right. This way of measurement reflects the improvements more precisely than the result of *dd* because of the variant time cost on the context switch to *dd*. Even though the variant time cost on the context switch to *tapdisk2* is included in the time cost on buffer I/O, the observation that the number of *bio*s has been reduced to 1 represents the progress of I/O aggregation and the reduction in the number of request submissions as shown in Table 4.1 indicates that the time cost on *tapdisk2* decreases with the number of context switches to *tapdisk2*. Both best prove the improvements on the I/O throughput.

However, after the size of the ring buffer is increased and the number of request submissions is reduced, the decrease in the time cost appears to be insignificant.

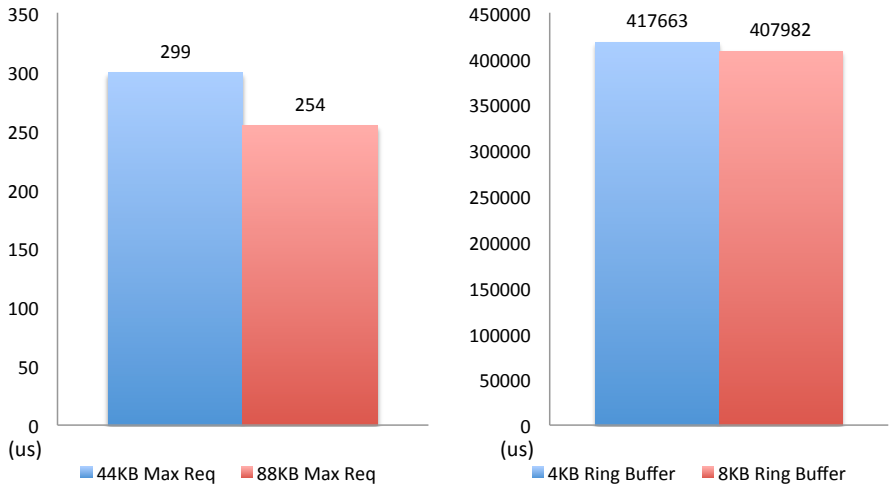


**Figure 4.1:** Improvements after increasing the maximum request size and enlarging the ring buffer size

When tracing the size of the merged *virtual request* in *tapdisk2*, it is found that this size remains constant after the ring buffer grows. This is because *blktap2* does not map the continuous requests into contiguous addresses of the ring buffer. After changing the mapping algorithm in *blktap2*, new improvements are shown in Figure 4.2 where the left portion is the time cost between the two trace points and the right is the I/O throughput. This is the first circumstance that affects the *virtual request* fragmentation in *tapdisk2*.

The size of *P2M meta-data* cache in *Parallax* should be increased to improve the miss rate of the cache. After changing the cache from the current 10 slots to 128 slots which is enough to hold all *P2M meta-data* during 4MB buffer I/O for 128 times, the I/O throughput between *Parallax* based virtual disk and the original virtual disk shows no difference.

To defragement *virtual request*, a sort algorithm is implemented in *tapdisk2* to reorder the request according to the address of the physical disk. The address of each request in the virtual memory also needs to be remapped so that the requests can get merged. The improvements made during buffer I/O with 112KB interleaving write operations for 100 times, which is demonstrated in Table 4.2, accounts

**Table 4.1:** Different numbers of request submission and segments with both the 4KB ring buffer and the 8KB ring buffer

| | Virtual disk 1 | Virtual disk 2 |
|---|---|---|
| Size of ring buffer (KB) | 4 | 8 |
| Number of submissions | 8 | 5 |
| Number of segments | 1678 | 1635 |
| | 363 | 715 |
| | 352 | |
| | 352 | 704 |
| | 352 | |
| | 352 | 704 |
| | 352 | |
| | 295 | 338 |

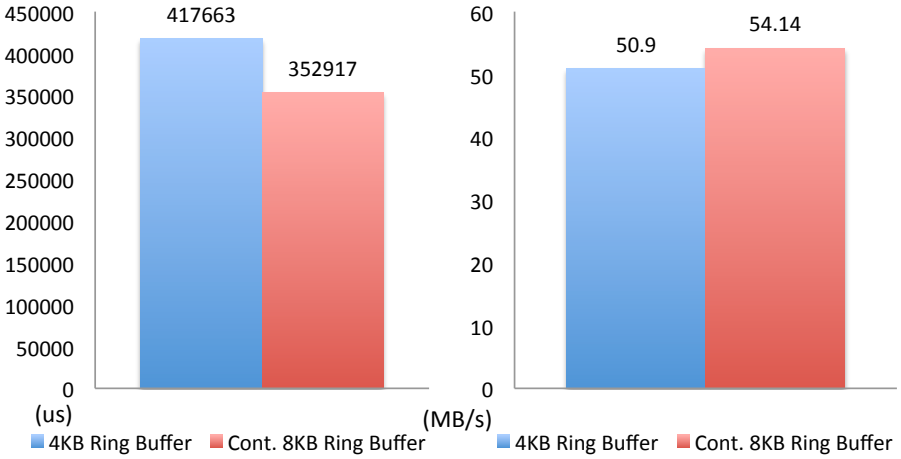


**Figure 4.2:** Improvements after revising the mapping algorithm in *blktap2*

for the third circumstance of the *virtual request* fragmentation.

**Table 4.2:** Different time costs before and after implementing the sort algorithm, during buffer I/O with 112KB interleaving write operations on the virtual disk for 100 times

| | Time cost |
|---|---|
| Before | 1.419s |
| After | 0.359s |

## 4.1 Optimum Values for Maximum Request Size and Ring Buffer Size

Theoretically, to completely avoid these impacts, the maximum request size in the ring buffer should be no less than that in the request queue of the beneath driver (1MB) and the size of ring buffer should not be the bottleneck in the pipelined model working for buffered I/O.

Therefore, after changing the maximum request size in the ring buffer to 1408KB in order to avoid the impacts from the ring buffer on the I/O throughput [1], Figure 4.3 shows the improvements similar to that in Figure 4.2 during direct I/O with 64KB, 128KB, 256KB and 512KB write operations.

It is observed that the number of segments inside each request is hardly larger than 700 as determined by the maximum disk throughput from Table 3.5. Therefore, it is sufficient to set the ring buffer to 8KB which can hold 704 segments. However, the size of the ring buffer should be larger when using parallel I/O in *tapdisk2* as discussed in Chapter 5.

[1] The maximum request size in the ring buffer connecting *blkfront* and *blkback* cannot be set over 352KB in the current verison of Xen.

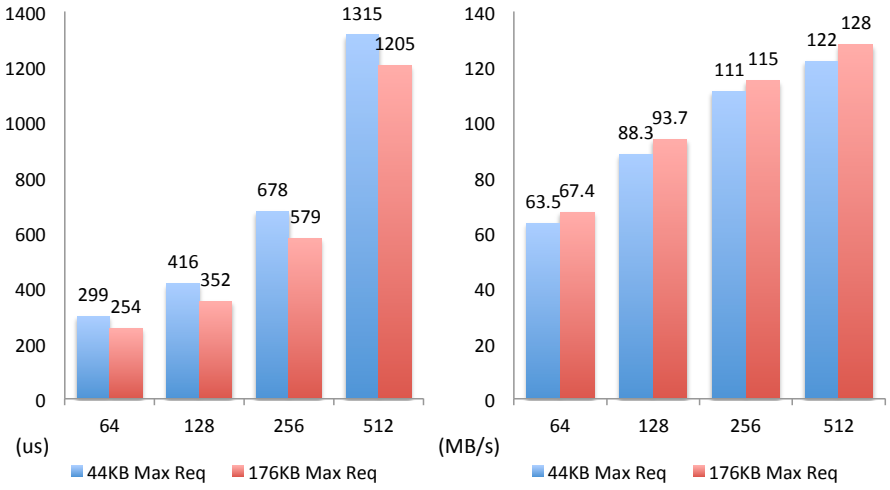


**Figure 4.3:** Improvements made by changing the maximum size of request in the ring buffer to 704KB

# Chapter 5

# VIRTUALIZED STORAGE SYSTEM WITH PARALLEL I/O

Although the above factors responsible for the difference in the time costs between the physical disk and the virtual disk can be eliminated, the basic time cost on the execution of *blktap2* and *tapdisk2* is unavoidable (approximately 100us during 4KB direct I/O but can increase with the size of the request). This time cost still causes a descent in the I/O throughput; however, it can be counteracted by implementing a distributed storage system that supports parallel I/O for Xen. Theoretically, the disk access time in Figure 2.3 can be halved when there are two disks in the distributed environment working in parallel.

## 5.1 Design

In order to distribute the I/O requests from the virtual disk to several machines, there should be an address mapping layer that translates the address of the requests to a tuple containing the identification of the machine, the identification of the disk and the address of the disk. Therefore, when designing such a storage system, the initial step is to decide who controls the address mapping.

A centralized control is much more intuitive and manageable when compared to peer-to-peer architecture where the mapping information is distributed in the environment. This is because only the machine that does the translation requires the

mapping information. Therefore, the best place to store the mapping information is the client-side (machine hosting Xen). By doing so, further communications between machines when translating the address is avoided. However, three questions need to be considered,

1. How to establish the mapping?
2. How to control the concurrent access?
3. How to prevent single point of failure?

Each server that constitutes the disk array sends a notification along with the IP address of the server and the logical unit number of the disk to *master* that acts as a central node. After *master* receives this message, it stores an array of tuples $< IP, LUN, ADDR$ *(starting address of the disk)*$>$ identifying *extent*s which are segments of the servers' storage capacity. An *extent* is also a basic unit for concurrency control with an attached *allocation lock*. Therefore, when a client needs to set up a virtual disk, it first sends a message that contains the number of required *extent*s to *master*. *Master* then locks the *extent*s and returns the tuples to the client who uses each tuple together with the starting address of the virtual disk of each *extent* to establish the mapping.

*Master* rarely becomes the bottleneck of the storage system because it is only involved in the mapping establishment stage. In other words, its failure does not affect the availability of the storage system unless new *extent*s are required by the client. The array of tuples stored in *master* and the mapping in the client-side can be replicated to prevent the single point of failure. In order to replicate on the virtual disk, each *extent* keeps several different identifications of the servers in the mapping information.

## 5.2 Implementation

Figure 5.1 is a sketch of the distributed storage system implementation based on the above design. The mapping information and the configuration of the virtual disk such as the disk size, the number of replicas, the number of stripes and the size of each stripe are stored in the file. This file can be used by only one virtual disk,

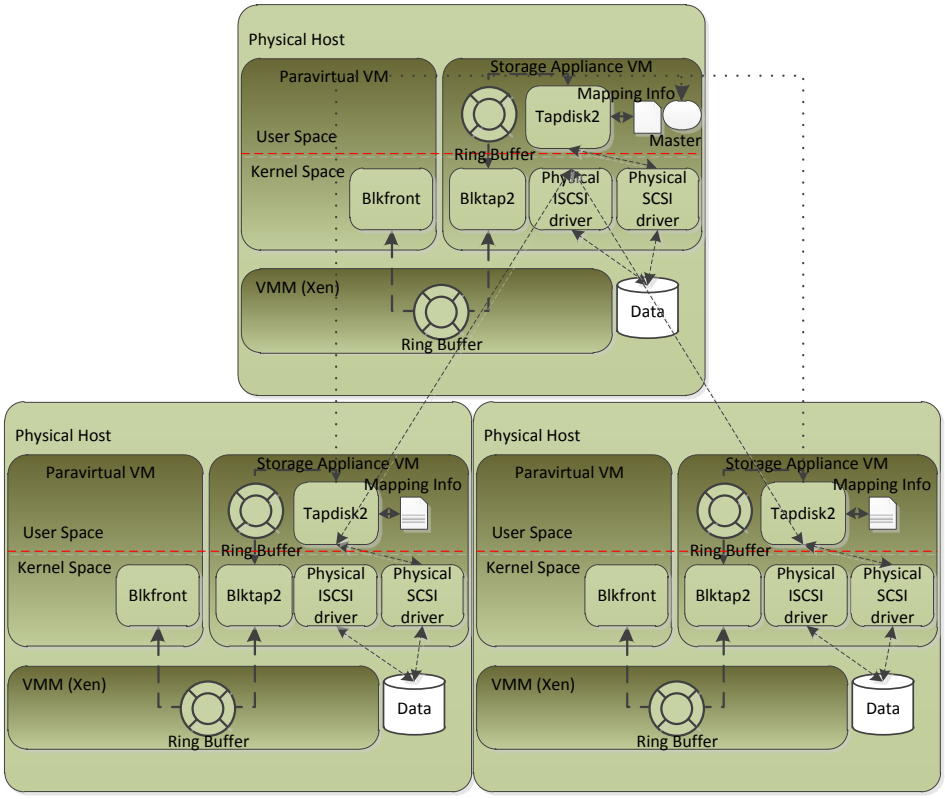


**Figure 5.1:** Architecture of the distributed virtual disk in Xen

thus it should have an exclusive lock. The client uses a filesystem that supports replication to keep the file in order to prevent the single point of failure.

The communications between the client and the *master* as well as that between the *master* and the server is implemented by Sun Remote Procedure Call (RPC) based on Transmission Control Protocol (TCP) protocol. Because the call semantics of Sun RPC with TCP protocol is *at least one*, the virtual disk identification and the address of the virtual disk of each *extent* should also be stored in the *master* to ensure the correct logic even if communications fail.

The name of the file that stores the mapping information is used as the only argument for *tapdisk2* when creating a new virtual disk. Every time an I/O request is redirected to *tapdisk2*, the address of the virtual disk of the request is divided into two parts, *extent ID* and the offset in the *extent*. *Tapdisk2* uses the *extent ID* as

the key to obtain the tuple from the file before caching it. After the address of the server is calculated, the request is submitted to the corresponding server by using Internet Small Computer System Interface (ISCSI) protocol. The reason in using the ISCSI protocol is that it has a small overhead of less than 10%. Each request queue in the client-side is assigned for each of the server to store the asynchronous I/O request. Another internal queue in the client-side is required to hold the request on the virtual disk in order to support replication. When all replicated requests are successfully replied, the response can be returned to *blktap2*.

# Chapter 6

# RESULTS AND ENHANCEMENT

One of the second circumstances that can affect the *virtual request* fragmentation is the use of striped I/O to change the address of the virtual disk of the requests in *tapdisk2*. One solution is to rearrange the addresses of the memory that store the buffer pages of the requests, making them contiguous when *tapdisk2* aggregates the requests in each of the queues.

## 6.1 Final Improvements

Figure 6.1 shows the I/O throughput during *dd* with 512KB direct I/O on the left and that during *dd* with 4KB buffer I/O on the right under three circumstances: the physical disk, the virtual disk and the virtual disk with parallel I/O (two disks array). It clearly demonstrates the effect of parallel I/O on offsetting the overhead of *blktap2* and *tapdisk2* in that the I/O throughput on the virtual disk with parallel I/O is even larger than that on the physical disk.

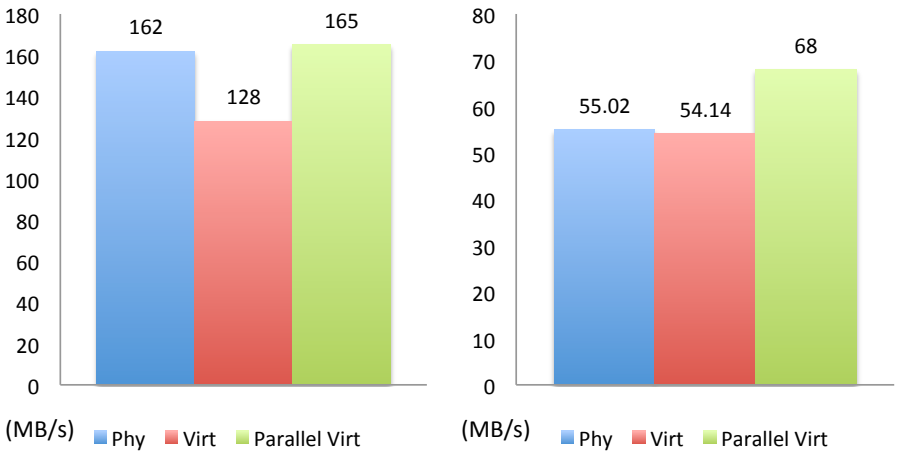


**Figure 6.1:** Improvements made by implementing the parallel I/O

# Chapter 7

# RELATED AND FUTURE WORKS

Blktrace [9] is a tool to trace the I/O request on the block device. The trace points are set at each action on the request queue. The analysis method in this paper, which uses a metric at the microscopic level rather than a common benchmark, is inspired by blktrace. This method is also effective when analyzing the impacts of the intra-request dependencies in *Parallax*[6] on the I/O throughput.

Xenoprof [5] is a profiling toolkit that can obtain statistical information such as clock cycles, cache and distribution of TLB misses on each routine in the virtual machine. It provides another perspective when analyzing the performance overhead in Xen, but not as directly as using special points tracing.

Sheepdog [8] is a distributed storage system for Qemu. It differs from the distributed virtual disk described in this paper in that Sheepdog uses the distributed system to control the mapping. Therefore, its mapping information is distributed among machines. Moreover, the basic unit for each mapping, which is a consistent hash table, is 4MB. The advantage of Sheepdog is that it is much more flexible when adapting the joining and leaving of a machine. However, additional communications needed for translation can cause overhead. Besides, whether the distributed disk needs such flexibility is also questionable.

Vineet *et.al* [3] present their study on the I/O overhead by using a simulation method. However, this technique can affect the underlying disk throughput and

change the length of the request queue that impacts the I/O throughput.

Further studies include testifying the scalability of the distributed virtual disk system as well as providing additional mechanisms to recover from the failure of the distributed virtual disk. Furthermore, other methods should be considered when investigating the other sources of the impacts on the I/O throughput [4], such as I/O scheduling [2] [9] [10].

# Chapter 8

# CONCLUSION

This paper provides a thorough investigation of the performance implications of virtualization. Instead of a systematic method, this is done by using quantitative approaches at the microscopic level based on modeling the virtual disk in Xen. There are four different variables that determine the four aspects affecting the I/O performance: maximum request size, length of request queue, *virtual request* fragmentation and the miss rate of *meta-data* cache. Therefore, improvements on the I/O throughput is achieved by changing these four variables. In order to offset these inevitable impacts on the I/O throughput of the virtual disk, a distributed virtualized storage system is given, and ways of building such a system is discussed. Finally, the results made from the improvements on the I/O throughput are presented.

# Bibliography


[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi:http://doi.acm.org/10.1145/945445.945462. URL http://doi.acm.org/10.1145/945445.945462. → pages

[2] D. Boutcher and A. Chandra. Does virtualization make disk scheduling passe? *SIGOPS Oper. Syst. Rev.*, 44:20–24, March 2010. ISSN 0163-5980. doi:http://doi.acm.org/10.1145/1740390.1740396. URL http://doi.acm.org/10.1145/1740390.1740396. → pages

[3] V. Chadha, R. Illiikkal, R. Iyer, J. Moses, D. Newell, and R. J. Figueiredo. I/o processing in a virtualized platform: a simulation-driven approach. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 116–125, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-630-1. doi:http://doi.acm.org/10.1145/1254810.1254827. URL http://doi.acm.org/10.1145/1254810.1254827. → pages

[4] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. Io performance prediction in consolidated virtualized environments. In *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering*, ICPE '11, pages 295–306, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0519-8. doi:http://doi.acm.org/10.1145/1958746.1958789. URL http://doi.acm.org/10.1145/1958746.1958789. → pages

[5] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual*

*execution environments*, VEE '05, pages 13–23, New York, NY, USA, 2005. ACM. ISBN 1-59593-047-7. doi:http://doi.acm.org/10.1145/1064979.1064984. URL http://doi.acm.org/10.1145/1064979.1064984. → pages

[6] D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. J. Feeley, N. C. Hutchinson, and A. Warfield. Parallax: virtual disks for virtual machines. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 41–54, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-013-5. doi:http://doi.acm.org/10.1145/1352592.1352598. URL http://doi.acm.org/10.1145/1352592.1352598. → pages

[7] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965. → pages

[8] M. S. MORITA Kazutaka, FUJITA Tomonori. A distributed storage system for qemu/kvm, 2010. → pages

[9] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 1–10, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-796-4. doi:http://doi.acm.org/10.1145/1346256.1346258. URL http://doi.acm.org/10.1145/1346256.1346258. → pages

[10] Y. Xu and S. Jiang. A scheduling framework that makes any disk schedulers non-work-conserving solely based on request characteristics. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association. ISBN 978-1-931971-82-9. URL http://dl.acm.org/citation.cfm?id=1960475.1960484. → pages