Does Virtualization Make Disk Scheduling Passé?

David Boutcher and Abhishek Chandra
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455
{boutcher, chandra}@cs.umn.edu

ABSTRACT

We examine whether traditional disk I/O scheduling still provides benefits in a layered system consisting of virtualized operating systems and underlying virtual machine monitor. We demonstrate that choosing the appropriate scheduling algorithm in guest operating systems provides performance benefits, while scheduling in the virtual machine monitor has no measurable advantage. We propose future areas for investigation, including schedulers optimized for running in a virtual machine, for running in a virtual machine monitor, and layered schedulers optimizing both application level access and the underlying storage technology.

1. INTRODUCTION

Traditional disk I/O scheduling has been designed to address the characteristics of electromechanical disks which have been the prevalent disk technology for several decades. Disk scheduling algorithms such as the elevator (SCAN) algorithm re-order disk I/O operations to minimize the time spent waiting for disk heads to physically traverse the disk. A variety of application-dependent file access patterns coupled with the disk schedulers' attempt to reduce the impact of slow seek times leads to trade-offs between fairness, latency, and efficiency, so the current Linux kernel contains four different disk I/O schedulers with different characteristics [13]. The need for such scheduling may become obsolete as storage systems move to new technologies and new environments. The specific question addressed in this paper is whether such scheduling is still relevant in the face of new technologies such as virtualization which form the basis of computing environments such as clouds and virtualized hosting platforms.

The adoption of virtual machines brings some interesting challenges to disk I/O scheduling. In a virtualized environment, multiple "guest" operating systems run on top of a virtual machine monitor. In most cases each of the guest operating systems have virtual disks which often share a single shared physical disk. In many cases the guest operating systems will schedule individual disk I/O operations using algorithms such as an elevator algorithm before passing them on to the virtual machine monitor, which itself will optimize the order of I/O operations. Given the lack of coordination between the layers, it is possible for different scheduling algorithms to conflict with each other. For instance, if both the guest and VMM are using an elevator scheduling algorithm and the schedulers spend equal amounts of time scheduling I/O operations in each direction, approximately 50% of the time the guest and the VMM will be scheduling I/O operations in opposite order. Further, the scheduling algorithm providing the best performance is often dependent on the specific workload. In a virtualized environment, different system images may be running different workloads. We show in this paper that choosing the right schedulers and at the appropriate levels of the virtualization stack, can indeed have a significant impact on the performance of an application. Specifically we demonstrate that carefully choosing the scheduler *closest* to the application has the greatest impact on performance.

There are other technologies, such as SAN storage, FLASH storage, and distributed storage used in cloud computing infrastructure which also bring into question the value of traditional I/O scheduling with its focus on minimizing disk arm movement. The results shown here suggest that even in these environments there may still be value in the additional processing required to optimize the way I/O operations are delivered to underlying storage devices.

We begin by presenting an experimental study of Linux I/O schedulers in the presence of virtualization. We then present some of the insights gained from this study and the potential impact of these findings on the design and use of schedulers in virtualized environments.

2. EXPERIMENTAL STUDY: EVALUATING I/O SCHEDULERS

The Linux operating system provides an excellent environment for comparing disk I/O scheduling algorithms, since the different I/O schedulers can be easily enabled and disabled on a per-device basis. Further, there are a variety of virtual machine monitors available. We elected to evaluate all combinations of the I/O schedulers in VMM and guest operating systems. The four Linux I/O schedulers are the Noop scheduler, which provides no I/O reordering, the Completely Fair Queuing scheduler (CFQ), the Anticipatory scheduler, and the Deadline scheduler. These schedulers are described in section 2.3 in more detail.

We consider two significant metrics for I/O performance

in a virtual environment. They are *throughput* and *fairness*. In this paper we demonstrate that the choice of schedulers at different system levels can significantly affect these metrics. We define these metrics as follows:

Throughput is the aggregate amount of work, performed by the system. In our experiments this is the number of I/O oriented transactions performed in aggregate across all virtual machines being evaluated on the system.

Fairness is the equality of work divided among different concurrent environments. We use Jain's fairness measure [6] to quantify the fairness between virtual machines. Jain's fairness measure ranges between 0 (completely unfair) and 1 (completely fair) and is defined as $fairness = \frac{(\sum x_i)^2}{(n \cdot \sum x_i^2)}$ where x_i is the transactions per second recorded in virtual machine i.

2.1 Experimental Setup

Two benchmarks were used to conduct the analysis. The Flexible File System Benchmark (FFSB) is a multi-threaded benchmark that provides a mix of read and write operations. FFSB was configured with 128 threads, and a mix of read and write operations. The second benchmark is Network Appliance's PostMark[8], which has become an industry-standard benchmark for small file and metadata-intensive workloads. It was designed to emulate Internet applications such as e-mail, netnews, and e-commerce.

All results in this paper were collected on an AMD system with a 2.3GHz processor. Benchmark I/O was performed to a dedicated 500GB SATA drive. The Linux 2.6.24 kernel was used throughout. Two different VMMs were evaluated: a Xen 3.2 VMM and the VMWare 2.0 Server VMM. To minimize the effect of memory caching, virtual machines were configured with a minimal 256MB of memory, and the VMM was limited to 256MB of memory in addition to the memory required by the virtual machines. For the three virtual machine tests, the system was configured with 1GB of memory. While the system had 2GB of memory installed, memory in addition to the limits above was unused by the operating system. Each virtual machine was allocated 120GB of the 500GB dedicated drive. The virtual disks were allocated as contiguous space on the physical disk to maximize the seek time when performing I/O operations on different virtual disks. Benchmarks were run in Xen DomU virtual machines, and the disk I/O schedulers used in both Dom0 and DomU were varied.

2.2 Key Results

Figure 1 shows the throughput comparison for the different schedulers using a Xen VMM. The I/O scheduling algorithms shown along the lower axis are the schedulers running in Dom0. The different shaded bars reflect the I/O scheduling algorithms running in the DomU guests. CFQ16 is a tuned version of the default Linux CFQ scheduler. All error bars shown in this paper reflect a 95% confidence interval. Dramatically different I/O throughput, measured in transactions per second, can be seen using different scheduling algorithms. The worst case combination provides only 40% of the throughput of the best case. The results from the two different benchmarks show that the Noop scheduler provides the highest throughput for any Dom0 scheduler,

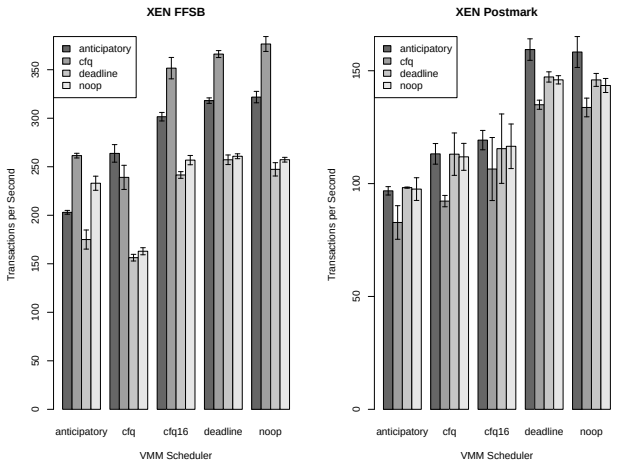

Figure 1: Throughput Comparisons on Xen

while the best of DomU scheduler depends on the workload being run.

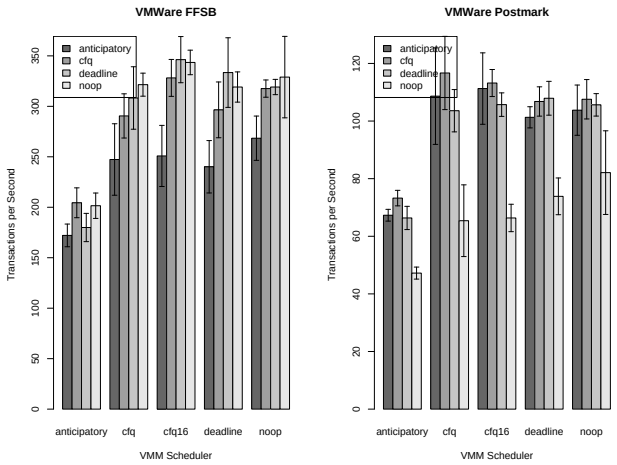

Figure 2: Throughput Comparisons on VMWare Server

In order to try to generalize the results beyond a single VMM implementation, the tests were repeated using the VMWare Server VMM. VMWare and Xen use completely different virtualization technologies, including different virtual disk device implementations. The VMWare Server version was used, since it runs on top of a Linux host operating system, and allows for the scheduler in the VMM layer to be varied. Clearly, a more comprehensive analysis would include the VMWare ESX server and VMFS filesystem, however in this particular scenario that would have prevented varying the VMM disk scheduler. Figure 2 shows the benchmarks run with three VMWare virtual machines and the results are similar to those with the Xen VMM.

One significant observation is that different schedulers in the guest operating system are optimal for different workloads. The results show that the CFQ scheduler is the preferred choice in the guest OS for the FFSB workload, regardless of the VMM scheduler. In the best case, CFQ provides a 17% improvement over the Anticipatory scheduler. For PostMark, the Anticipatory scheduler demonstrates the best performance, with an 18% improvement over CFQ. Further, it should be noted that the default Linux configuration is to run the CFQ scheduler both in the VMM and the guest. For the FFSB benchmark in Xen, using the Noop scheduler in the VMM provides a 60% performance improvement over this configuration. Using the Anticipatory scheduler in the guest and the Noop scheduler in the VMM provides a 72% throughput advantage over the default configuration using the Postmark benchmark. These results are remarkably consistent across the Xen and VMWare environments tested.

The two benchmarks used in this study have quite different characteristics. FFSB is heavily threaded, utilizing 128 threads in the configuration used. The PostMark benchmark, by comparison, utilizes a single thread that mimics the I/O patterns of common Internet software. The results indicate that PostMark benefits significantly from the Anticipatory scheduler's assumption that the process is likely to deliver additional I/O operations soon. In fact, since PostMark is a purely I/O bound application, this is exactly its behavior. The heavily threaded FFSB benchmark, by comparison, is adversely impacted by the Anticipatory schedule's behavior.

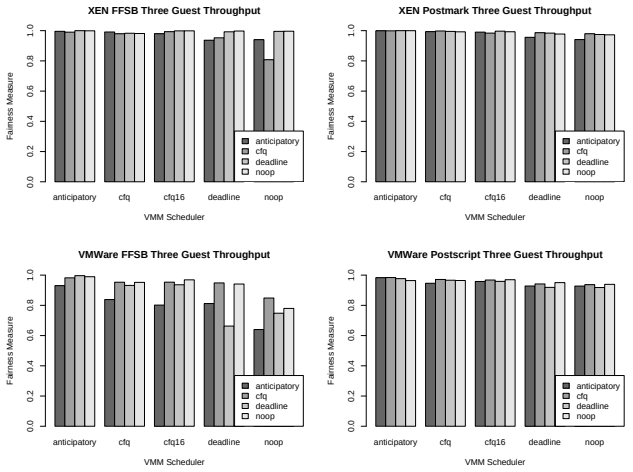

Figure 3: Fairness

When running multiple virtual machines, an important metric is "fairness". In other words, do each of the virtual machines receive equal access to resources? Figure 3 shows Jain's Fairness measure for throughput running three virtual machines using both Xen and VMWare VMMs. While all combinations of VMM and guest schedulers are generally fair, the *least* fairness is shown using the Noop scheduler in Dom0 and the VMWare VMM. This indicates that the improvement in throughput may come at the expense of reduced fairness.

In summary, our evaluation produced the somewhat surprising conclusion that *there is no benefit in throughput from performing additional I/O reordering in the VMM*. That is, at the layer closest to the physical device. This observation is true in our environment, which utilized a traditional disk drive, and we expect it to hold true in other I/O environments, such as flash and SAN storage, that are less impacted by seek time. While our environment consisting of a single traditional disk drive is an extreme case that should have emphasized he impact of a VMM level seek-reducing scheduler, benefits from scheduling in the VMM were not demonstrated. In the workloads and VMMs studied, performing minimal I/O scheduling in the VMM produced the highest throughput, while choosing disk scheduler appropriate to the workload in the guest operating system does provide benefit. At the same time, more complex scheduling in the VMM *can* provide better fairness between multiple guest operating systems.

2.3 Examining the Linux I/O Schedulers

To understand the results presented above, it is useful to examine the four block I/O schedulers present in the Linux kernel. Disk I/O schedulers perform two basic operations: merging and sorting. Merging recognizes adjacent I/O requests and combines them, reducing the number of I/O requests, and associated interrupts, DMA operations, etc, that must be exchanged with the device. In a virtualized environment, it reduces the number of transitions between the guest operating system and the VMM, which are frequently the source of much of the overhead in virtualized environments[1]. Sorting arranges pending I/O requests in block order to minimize the distance that the disk heads have to move on the physical disk since seeks are the most expensive part of physical disk I/O[4]. Pratt and Heger[13] do a thorough evaluation of the different Linux I/O schedulers and their effect on different workloads.

The Noop scheduler is implemented using a simple FIFO queue and performs only basic merging and sorting. It assumes that I/O performance will be optimized in the block device. It is particularly appropriate for devices such as Storage Area Networks (SANs) that have much more knowledge of the underlying physical devices than the operating system.

The Completely Fair Queuing (CFQ) scheduler is the default scheduler in the Linux 2.6 kernel. CFQ works by placing synchronous requests submitted by processes into per-process queues and allocating time-slices for handling I/O operations from each queue. The length of the time slice is referred to as the "quantum". When utilized by a VMM, each virtual machine is treated as a "process". The quantum can be tuned, and had a significant effect on performance in a virtualized environment. In our results above, the bar labeled "CFQ" reflects results using the default configuration. Changing the "Quantum" parameter in the VMM from the default of 4 to 16 (CFQ16) provided up to 21% improvement in throughput. Further evaluation of the behavior of the CFQ scheduler using the Linux `blktrace` utility helped explain the results shown. While work conserving, the CFQ scheduler dispatches fewer I/O operations to the physical device at a time than does the Noop scheduler. The Xen virtual machine implementation dispatches at most 32 I/O requests from each DomU to Dom0 concurrently. In the test environment evaluated here the number of concurrent I/O

operations supported by the physical device is greater than 32. Our analysis showed that on average the CFQ scheduler dispatched only 6 I/Os to the physical device in the VMM, while the Noop scheduler dispatches the maximum possible. Further, the I/Os dispatched to the physical device often complete out of order, indicating that the SATA drive itself is performing I/O optimization through caching or I/O reordering.

The Deadline elevator uses a deadline algorithm to minimize I/O latency for a given I/O request. A standard elevator algorithm is used, unless the request at the head of one of the FIFO queues grows older than an expiration value. In that case the scheduler begins servicing expired requests. The results shown in this study show the Deadline scheduler behaving almost identically to the Noop scheduler. It seems likely that the deadlines imposed by the scheduler are never exceeded, resulting in the Deadline scheduler devolving to the same behavior as the Noop scheduler.

The Anticipatory elevator assumes that processes typically perform multiple I/O operations within a short time duration, and that these I/O requests are likely to be physically close. Upon receiving an I/O request from a process it introduces a short delay before dispatching the I/O to attempt to aggregate and reduce disk seek operations. A normal work-conserving I/O scheduler switches to servicing I/O from an unrelated process if there is no I/O operation immediately available that is locally close to the most recent I/O. The Anticipatory scheduler sacrifices a small amount of working time to increase I/O locality. The Anticipatory scheduler is the only scheduler which, rather than "work conserving", is "seek-conserving". It is surprising that the Anticipatory scheduler did not demonstrate better performance when used in the VMM layer. We expected that waiting to gather as many I/O requests from a single virtual machine as possible would improve performance.

3. DISCUSSION AND FUTURE DIRECTIONS

The rapid evolution and adoption of virtualization technologies in both academia and industry has led to an increased focus on optimizing virtual machine execution[3][7][11]. There has been a significant amount of investigation around optimizing processing[2][10][18][19], memory[9][14][20], and I/O operations[17], though much of the work related to virtual machine I/O optimization has focused on networking[1][5][12]. Relatively little work has been done in the area of disk I/O optimization[15]. Since storage systems are often the system bottleneck in systems running I/O intensive applications, the effect of scheduling on overall system performance can be significant. One critical area of future investigation is the extent to which utilizing I/O scheduling in a guest operating system can minimize the overhead of running in a virtualized environment. For example, the Anticipatory scheduler delays issuing I/O operations to optimize for the case that subsequent I/O operations may be related. It seems reasonable that such an approach could be used to reduce the number of guest/VMM transitions and thus reduce the virtualization overhead. We have demonstrated that the benefits of the Anticipatory scheduler are very dependent on the workload, however developing a scheduler that is domain-transition minimizing, rather than seek-minimizing should provide a general performance advantage.

Another area of investigation is the optimal scheduler for the VMM layer. While an optimal scheduler is dependent on the workloads involved, different VMMs share many characteristics. Virtual disks tend to be large, contiguous storage regions. Minimizing the number of seeks between virtual disks should provide a benefit. Seelam and Teller[15] propose a "Scheduler of Schedulers" that runs in the VMM and focuses on fairness and isolation. They demonstrate improved fairness at the cost of throughput. Their results, which show execution time rather than throughput, agree with our results that the Noop scheduler used in the VMM provides both minimum execution time and least fairness. It seems likely that a virtualization-optimized scheduler would do better than the Noop scheduler for both throughput and fairness.

In a virtualized environment the VMM has additional insight into the overall performance of the system. The most interesting area of future research appears to be the extent to which the scheduler in the VMM and the scheduler in the guests can cooperate. The Cello disk scheduling framework[16] proposed a two level disk scheduling architecture where the top level is application specific and the lower level mediates between multiple upper levels. This model is very similar to the multiple schedulers present in a virtualized environment and some of the techniques described in that work appear to be applicable to a virtualized system. Zhang and Bhargava[21] propose a self-learning disk scheduler which also has techniques that could be exploited to intelligently choose the schedulers running in a virtual machine. Their technique focuses on workload input, such as request size and think time. Providing additional input from the VMM has the potential to improve the choice of scheduler.

The objective of this study was to determine if additional investigation in the area of disk scheduling and virtualization is warranted. Since it appears there is significant performance advantages to be gained from appropriate scheduling in a virtualized environment, an obvious next step would be to examine other commonly virtualized operating systems, such as Microsoft Windows.

Finally, as more virtualization moves to "cloud" environments with non-traditional storage, and enterprise environments with Storage Area Networks, we anticipate that the different storage back-ends will warrant further study of optimal I/O scheduling strategies. The evaluations performed in this paper studied a traditional computing system. Additional evaluation needs to be performed to determine if the results hold up in more complex environments, such as the Amazon Elastic Compute Cloud and and virtualization oriented filesystems such as VMFS.

4. CONCLUSIONS

We have demonstrated that disk I/O scheduling is far from obsolete in a virtualized system. There are a number of interesting specific conclusions from the evaluation performed in this paper. Specifically:

- The choice of I/O scheduler in the *guest* operating system of a virtualized environment should be made based on the workload being executed. Significant throughput advantages can be obtained by selecting the appropriate scheduler. Gains of up to 18% were demonstrated in the evaluation performed in this paper.
- Using the environments and workloads evaluated here, the best choice of I/O scheduler in a VMM appears to

be a minimal scheduler. The Linux Noop scheduler provided almost universally the best throughput, regardless of VMM, workload, or guest operating system I/O scheduler.

- More advanced schedulers, such as the Completely Fair Scheduler, *do* provide better fairness when running multiple virtual machines at the cost of throughput and latency.

In conclusion, using the default Linux schedulers does not appear to be the optimal configuration for maximizing throughput in a virtualized environment. The evaluation in this paper demonstrated up to a 72% throughput increase by selecting the appropriate schedulers.

The environments in which virtualization is being executed continues to evolve. Data storage in cloud computing and Flash storage are becoming increasingly important, and SAN storage is ubiquitous in the enterprise environment. While it may initially appear that disk I/O scheduling is obsolete in environments utilizing these technologies, with all scheduling decisions left to the underlying storage technology, this study suggests that in a virtualized environment there is still a benefit to making intelligent decisions about the order in which the operating system delivers I/O requests to a storage back end.

5. REFERENCES

[1] Vineet Chadha, Ramesh Illiikkal, Ravi Iyer, Jaideep Moses, Donald Newell, and Renato J. Figueiredo. I/o processing in a virtualized platform: a simulation-driven approach. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 116–125, 2007.

[2] L. Cherkasova and R. Gardner. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *USENIX Annual Technical Conference*, April 2005.

[3] Ulrich Drepper. The cost of virtualization. *Queue*, 6(1):28–35, 2008.

[4] H. Frank. Analysis and optimization of disk storage devices for time-sharing systems. *J. ACM*, 16(4):602–620, 1969.

[5] Sriram Govindan, Arjun R. Nath, Amitayu Das, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 126–136, 2007.

[6] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, Digital Equipment Corporation, September 1984.

[7] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Antfarm: tracking processes in a virtual machine environment. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 1–1, 2006.

[8] J. Katcher. Postmark: A new file system benchmark. Technical Report Technical Report 3022, Network Appliance Inc., 1997.

[9] Jacob Faber Kloster, Jesper Kristensen, and Arne Mejlholm. On the feasibility of memory sharing. Department of Computer Science,Aalborg University, June 2006.

[10] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. Pre-virtualization: slashing the cost of virtualization. Technical Report Technical Report PA005520, National ICT Australia, October 2005.

[11] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *First ACM/USENIX Conference on Virtual Execution Environments*, Chicago, Illinois, USA, June 2005.

[12] Diego Ongaro, Alan L. Cox, and Scott Rixner. Scheduling i/o in virtual machine monitors. In *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 1–10, 2008.

[13] Stephen Pratt and Dominique Heger. Workload dependent performance evaluation of the linux 2.6 i/o schedulers. In *Proceedings of the Linux Symposium*, volume 2. Ottawa Linux Symposium, 2004.

[14] Martin Scwidefsky, Huburtus Franke, Ray Mansell, Himanshu Raj, Damian Osisek, and JonHyuk Choi. Collaborative memory management in hosted linux environments. In *Proceedings of the Linux Symposium, Volume 2*, Ottawa, Canada, July 2006.

[15] Seetharami R. Seelam and Patricia J. Teller. Virtual i/o scheduler: a scheduler of schedulers for performance virtualization. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 105–115, New York, NY, USA, 2007. ACM.

[16] Prashant Shenoy and Harrick M. Vin. Cello: A disk scheduling framework for next generation operating systems. In *In Proceedings of ACM SIGMETRICS Conference*, pages 44–55, 1997.

[17] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the 2001 USENIX Annual Technical Conference*, Boston, Massachusetts, USA, June 2001.

[18] Ananth I. Sundararaj, Ashish Gupta, and Peter A. Dinda. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, pages 47–58, 2005.

[19] Volkmar Uhlig, Joshua LeVasseur, Espen Skoglund, and Uwe Dannowski. Towards scalable multiprocessor virtual machines. In *3rd Virtual Machine Research and Technology Symposium (VM'04)*, pages 43–56, May 2004.

[20] C. Waldspurger. Memory resource management in vmware esx server. In *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[21] Yu Zhang and Bharat Bhargava. Self-learning disk scheduling. *IEEE Trans. on Knowl. and Data Eng.*, 21(1):50–65, 2009.