SSD-HDD-Hybrid Virtual Disk in Consolidated Environments

Heeseung Jo[1,*], Youngjin Kwon[1], Hwanju Kim[1], Euiseong Seo[2], Joonwon Lee[3], and Seungryoul Maeng[1]

[1] Korea Advanced Institute of Science and Technology (KAIST),
335 Gwahangno, Yuseong-gu, Daejeon, Korea
[2] Ulsan National Institute of Science and Technology (UNIST)
100 Banyeon-ri, Eonyang-eup, Ulju-gun, Ulsan, Korea
[3] SungKyunKwan university,
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do, Korea
[1] {heesn, yjkwon, hjukim, maeng}@camars.kaist.ac.kr,
[2] euiseong@unist.ac.kr, [3] joonwon@skku.edu

Abstract. With the prevalence of multi-core processors and cloud computing, the server consolidation using virtualization has increasingly expanded its territory, and the degree of consolidation has also become higher. As a large number of virtual machines individually require their own disks, the storage capacity of a data center could be exceeded. To address this problem, copy-on-write storage systems allow virtual machines to initially share a template disk image. This paper proposes a hybrid copy-on-write storage system that combines solid-state disks and hard disk drives for consolidated environments. In order to take advantage of both devices, the proposed scheme places a read-only template disk image on a solid-state disk, while write operations are isolated to the hard disk drive. In this hybrid architecture, the disk I/O performance benefits from the fast read access of the solid-state disk, especially for random reads, precluding write operations from the degrading flash memory performance. We show that the hybrid virtual disk, in terms of performance and cost, is more effective than the pure copy-on-write disks for a highly consolidated system.

Keywords: Consolidation, Virtual machine (VM), Copy-on-write (CoW), Hybrid storage.

1 Introduction

Virtualization enables multiple operating systems to run on a single physical machine, and server consolidation systems using virtualization have expanded their territory

[*] Please note that the LNCS Editorial assumes that all authors have used the western naming convention, with given names preceding surnames. This determines the structure of the names in the running heads and the author index.

significantly, especially in large-scale computing systems or cluster systems. This trend is based on the effort to lower the management cost, which is one of the primary factors for server hosting centers or the server market. With server consolidation, fewer physical machines are needed to run the same number of servers, thus saving power and space. These factors are directly related to the total cost of ownership; it is known that 50-70% of reduction could be possible [1]. Moreover, the virtualized system is also advantageous due to the availability and manageability of servers.

Storage virtualization has been less focused than other resources, such as memory and CPU, since disks have better density and are easily sharable via network attached storage. The introduction of cloud computing, however, makes efficient storage virtualization more relevant in terms of disk capacity. Cloud environments allow thousands of cloud users to store their own contents and privately view their storage. With more virtual machines (VMs), one VM will require more storage space due to operating systems and applications that become richer and larger. Therefore, the capacity requirements for storages are expected to grow exponentially. Data centers serving a large-scale VM farm cannot extend their storage infinitely, since the cost of doing so is not inexpensive, after taking into account ownership costs such as maintenance, cooling, and space. Since traditional sharing-based storage cannot deal with this requirement, many data centers are unable to afford the storage capacity for private disks required by cloud users.

Two representative approaches have been developed to relieve the burst requirement of storage in virtualized environments: copy-on-write (CoW) storage and content addressable storage (CAS) [2, 3]. First, CoW storage enables multiple VMs to initially share a template disk image. This mechanism allows read-only sharing by isolating any write attempts from the template disk image. This approach was adopted in QCOW [4], CoWNFS [5], and Parallax [6]. Second, CAS uses a content-based address to access a disk block. This mechanism does not require even template disk image sharing, but incurs computational overheads. Although these two approaches significantly reduce disk footprints, they do not improve the disk I/O performances of those mechanisms.

This paper presents a hybrid CoW virtual disk that combines the solid-state disk (SSD) and hard disk drive (HDD) within a highly consolidated system. The SSD-HDD-hybrid virtual disk (HVD) uses SSD for read-only template storage, whereas privately written data are stored in HDD. HVD gains high disk I/O performance from the fast read operations of SSD, especially for random reads. Since the read operations of consolidated VMs are multiplexed, a sequential read stream of each VM could be broken, and thereby realized as small random reads. Further, the isolation of write operations from SSD eliminates drawbacks from write I/O, such as erase-before-write and wear-out. Our evaluation results indicate that the hybrid architecture of HVD outperforms HDD-only or SDD-only storage. For several real workloads, HVD shows more than 40% performance enhancement and does not suffer from the heavy write, which is the main weakness of SSD.

The rest of this paper is organized as follows: Section 2 describes the design and implementation of HVD and discusses related challenging issues. Section 3 presents the evaluation results of HVD compared with pure storage by using several micro-benchmarks and real workloads. Finally, we summarize the paper and present a future direction in Section 4.

2 Hybrid Virtual Disk (HVD)

This section describes the overall architecture of HVD. First, we give a brief description of the virtualized environments using CoW storage. Then, we present the hybrid architecture of HVD and its implementation. Finally, we illustrate migrating data between SSD and HDD, a challenging issue for HVD architecture.

2.1 CoW Storage in Virtualized Systems

In virtualized environments, the CoW mechanism over virtual disks has been prevalent due to its efficient use of disk and easy management of snapshot [12, 13, 6]. The CoW mechanism is a well-known technique that allows multiple entities to share a resource, until a write attempt occurs to the shared resource; once written, the shared resource is copied to a newly allocated space for the private use of the resource. In this manner, a CoW disk enables multiple VMs to share a template disk image while presenting each VM with the private view of its own storage. In addition, the CoW disk can support fast snapshots by preserving metadata for the current disk image.

The CoW disk is compelling in consolidated environments for three reasons. First, many VMs typically run the same operating systems and applications, especially in cluster-based systems, which provide replicated services for reliability and load balancing [5]. In this system, multiple VMs can share a template disk image that contains common operating systems and applications in a CoW manner, thereby reducing disk footprints. Second, as the degree of consolidation has grown considerably, a virtualized data center could accommodate many more servers than a native data center. Since each server at least requires a system image from which to boot, a large number of servers may exceed the storage capacity [14]. The CoW disk can effectively relieve this increased requirement of storage capacity. Finally, snapshot is a frequent operation used to control the history of a virtual disk for reliability. As cloud computing has emerged in large-scale consolidated environments, reliability is now a more important concern to cloud users. The efficient snapshot functionality of the CoW disk enables fast backup and recovery of storage.

2.2 SSD-HDD-Hybrid Design

To maximize the advantages and to minimize the drawbacks of SSD, we introduce the HVD for virtualized environments. In HVD, the read-only templates of VM disk images are stored on SSD to support fast read operations. On the other hand, the privately written blocks of a VM are placed on HDD. This design is inspired by the asymmetric I/O characteristic of SSD; the write operation is slow and varied, whereas the read operation is fast and uniform.

SSD is currently an emerging storage device for server systems to enhance the disk I/O performance [9, 10]. SSD is a NAND flash memory-based storage device that is expected to replace HDD in the near future because of its versatile features, such as non-volatility, solid-state reliability, low power consumption, shock resistance, and high cell densities [7, 8, 11]. SSD supports high read performance, especially for random reads, since it does not include HDD-like mechanical parts that incur seek and rotational delays. SSD, however, has several weak points caused by the nature of NAND flash memory. One is the *erase-before-write* characteristic that a page, which is the basic unit of read and write operations, should be erased before being rewritten in the same location. The erase operations can only be performed on a block, which is larger than a page. Therefore, SSD shows slow and non-uniform write latency. Another limitation is the wear-out problem. Unfortunately, each block in flash memory has a limited number of erase/write cycles, and data in a block become unreliable if the block reaches this limit. The current limit for single-level cell NAND flash memory is approximately 100,000 erase/write cycles.

Considering these features of SSD, the SSD-HDD-hybrid scheme has several advantages. First, HVD supports fast read accesses to a template disk image, which typically contains rich applications, libraries, and common data contents. HVD improves user experiences by boosting the startup of applications and the loading of libraries. In addition, random read accesses to a template disk image benefit from SSD. Since multiple sequential read streams from guest VMs are fairly multiplexed, each stream might be broken into small random read operations, which result in the poor performance of HDD. SSD provides better latency for the broken random reads. Next, isolating writes from SSD eliminates the aforementioned problems induced from write operations. As HVD preserves a template disk image on SSD from write operations, SSD does not suffer from wear-out and overheads for erase-before-write. Finally, HVD allows for cost-effective storage, in terms of performance and capacity. Since SSD is more expensive than HDD with the same capacity, pure SSD-based storage might not be an affordable option to store large amounts of private data of VMs. HVD requires SSD capacity only for template disk images, making our approach more cost-effective.

2.3 Implementation

We implemented two versions of HVD: HVD based on *cowloop* [15] and HVD based on Parallax. Our approach to hybridizing SSD and HDD for a CoW block device can be applied with low reengineering costs. Moreover, our approach is also advantageous in terms of transparency. It can be provided to upper layers without any modifications due to block level implementation.

Cowloop is a simple and lightweight block device used to support the CoW behavior. Figure 1 shows the HVD implementation overview based on cowloop. A VM uses the template disk image as read-only, and when the VM updates blocks, the write operations are forwarded to its cowfile, which stores the privately written blocks. If a block is written once, the next access to the block is forwarded to its cowfile. For example, block 1 of VM1 is read from a template disk image, and block 6 that is written before is read from the cowfile1. For HVD, we place the template disk image on SSD, and use HDD as cowfile storage.

On the other hand, Parallax is a novel distributed storage system for Xen VMs [20] and supports the CoW mechanism to reduce the required storage size. Furthermore, Parallax provides many features, such as network access, snapshot, and the efficient

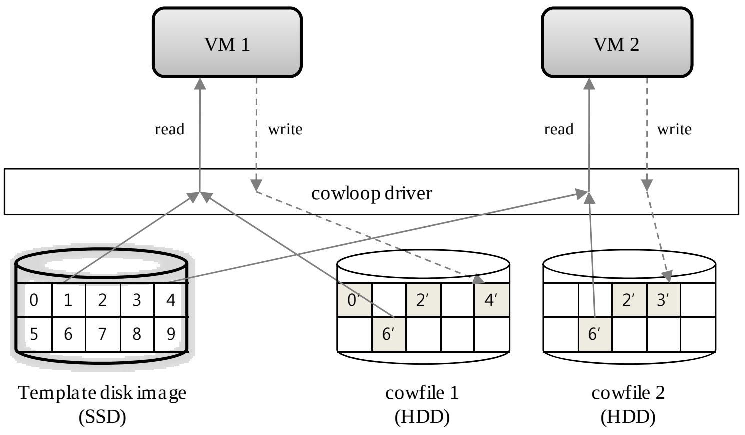

Fig. 1. The HVD implementation overview based on cowloop.

lock mechanism. Our Parallax version of HVD spontaneously inherits these features. To support the CoW behavior, Parallax uses a radix tree that translates the logical block number (LBN) from a VM to the physical block number (PBN). If a VM updates a block, the related radix tree nodes are created, and their leaf node possesses the PBN. In addition to PBN, the entry of a leaf node indicates whether a data block is read-only or written via a bit flag. For HVD, we add a 1 bit locator flag that denotes whether a block resides in SSD or HDD.

2.4 Migration between SSD and HDD

The current placement policy of HVD has optimization chances to migrate data between SSD and HDD. In cases where a file is first modified and frequently read afterward, this file is obtained from HDD without the benefit of SSD. Such write-once read-many blocks can be migrated to SSD so that better read performance is achieved. There are various possible methods to identify migratable blocks at different levels of hierarchy.

First, users can specify rules that reflect their preferences. For example, many configuration files or static web contents (e.g. /etc, html, or web image files) are initially modified and primarily read for the rest of their lifetimes. In this case, a user can define that such files should always reside in SSD. This rule-based approach should collaborate with the file system to inform HVD of blocks in which a specified file is located. While requiring user intervention, this method can directly write a specified file to SSD without migration.

Second, the file system can identify write-once read-many files by monitoring modification and access times stored in the metadata. This monitoring-based method enables frequently read files, after being written, to be migrated to SSD without user intervention. This method, however, requires a monitoring daemon in each guest VM.

Third, HVD maintains read access frequency for each block stored once in a location of HDD during a certain period. When detecting a frequently read block,

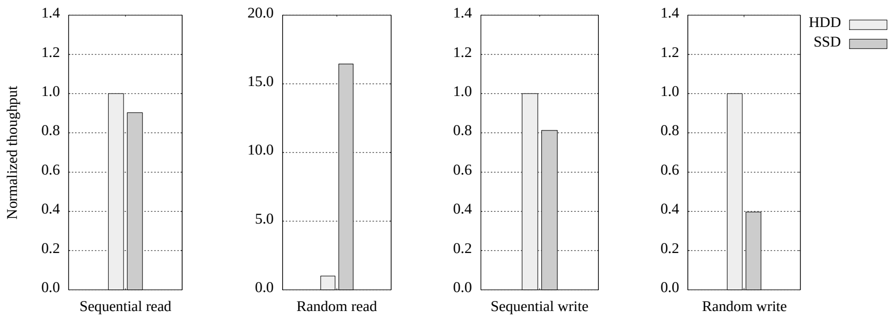

Fig. 2. The raw I/O performance comparison between HDD and SSD.

HVD migrates this block to SSD. This method is guest VM-agnostic, so that no guest-level daemon is required. On the other hand, the block-level approach redundantly manages metadata for each block in order to maintain access frequency.

3 Evaluation

In this section, we evaluate the performance aspect of HVD. Our storage system is implemented on Xen-3.2.3 with a para-virtualized Linux 2.6.18 kernel for the x86 architecture. The machine for Xen has an Intel Core2 Duo 2.33 GHz CPU with 2 GB of RAM. The memory size of a driver VM, which is in charge of I/O device accesses and contains HVD, is configured to 512 MB, and that of each guest VM is set to 128 MB. All tests are performed on a local storage to exclude network overhead.

In all evaluations, we used the cowloop version of HVD, since Parallax has several functions including a garbage collector and a locking mechanism in addition to the CoW features. Although Parallax is a more sophisticated virtual disk, we suppose that the cowloop version of HVD clearly shows the performance gain from our hybrid approach to exclude the impact of additional features, except the CoW mechanism. To demonstrate the impact of our hybrid approach, we evaluate HVD in comparison with the pure CoW disks: cowloop-HDD and cowloop-SSD.

Raw device performance. Seagate barracuda with 7200 RPM [18] and Samsung SSD [19] are used in all evaluations. These storage devices are selected for a reasonable performance comparison. Figure 2 shows the raw performances of HDD and SSD, which are used in this evaluation for several types of disk operations (sequential read/write, random read/write), and the results are normalized to HDD. We performed the tests using *sysbench* [16] on a native machine. As depicted, except in the case of the random read, HDD performs better than SSD. In the case of the

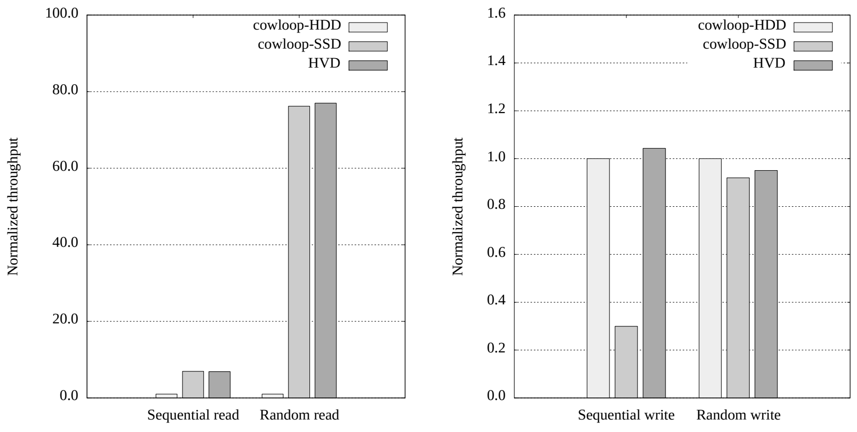

Fig. 3. The I/O performance of cowloop-HDD, cowloop-SSD, and HVD for each operation type.

random read, however, SSD shows better performance than HDD by the multiple of sixteen[†].

Micro-benchmark. Figure 3 shows the evaluation result of micro-benchmark using sysbench. All the tests are performed on a guest VM, and all I/O operations are delivered to each storage device through a driver VM. The tests are performed for sequential read/write and random read/write, and the y-axis shows the normalized throughput.

In the case of the read operation, SSD shows significant effects, especially for the random read. For the sequential read, unlike a native environment, both cowloop-SSD and HVD indicate higher throughput than cowloop-HDD. While HDD maximizes the sequential read performance for a burst read, a driver VM interrupts read operations, breaking burstness and thus reducing HDD read performance.

For the write operation, the performance of cowloop-HDD and HVD is similar as expected. The sequential write operation of HDD is much faster than that of SSD due to the erase-before-write characteristic of SSD. In the case of random write, HDD shows little higher performance than SSD, since HDD incurs seek and rotational overheads.

Real workloads. With regard to real workload evaluations, we performed four workloads: the booting of VMs, the online transaction processing (OLTP), the decompression, and the data writing. The first two are read-intensive workloads, and the decompression is read/write mixed with a ratio of 1.5. The last data writing is a write-intensive workload. The VM configurations for all the tests are the same as that

[†] A recent high-end SSD for server environments outperforms HDD for all disk operations, but our current experiments are not conducted with a high-end SSD for a fair comparison.

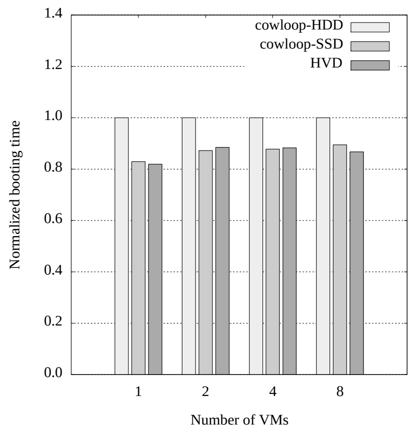

Fig. 4. The booting time of VMs.

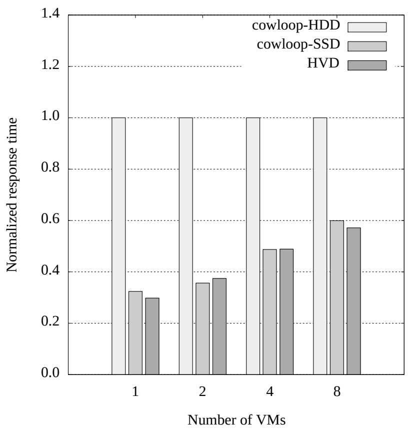

Fig. 5. The response time of OLTP.

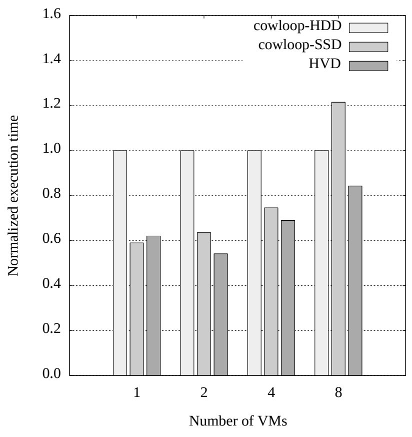

Fig. 6. The execution time of decompression.

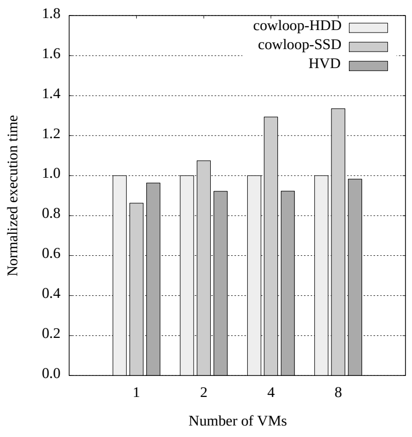

Fig. 7. The execution time of data writing.

of the micro-benchmark test. We evaluate each workload as increasing the number of VMs. Figure 4 shows the normalized booting time of guest VMs. The performance gain of HVD is not considerable, since the booting sequence of a VM involves only a little amount of I/O operations. The next test is the online transaction processing with Mysql [17] and sysbench, which requests approximately 130 database transactions per second. The evaluation results are illustrated in Figure 5, and the y-axis is the normalized average response time. The response time of cowloop-SSD and HVD are 30-60% less than that of cowloop-HDD.

On the other hand, the evaluation result of the decompression is presented in Figure 6. This workload decompresses the source code of Xen and Linux. The notable situation occurs when the number of VMs is eight. The execution time of cowloop-

SSD is longer than that of cowloop-HDD, since SSD shows the slowest operation latency for heavy random writes, due to erase-before-write. The same case is more clearly shown in the write-intensive workload, the data writing. As presented in Figure 7, cowloop-SSD results in a longer execution time when the number of VMs is more than one. All the results illustrate that HVD has higher disk performance than cowloop-SSD and cowloop-HDD, especially when a large number of VMs are consolidated. More significantly, outperforming pure SSD means that HVD is more cost-effective for server consolidation workloads.

4 Conclusion and Future Work

This paper presents a hybrid virtual disk that makes possible the efficient combination of SSD and HDD within consolidated environments. We derive the performance benefit from fast random reads of SSD by locating a read-only template disk image in SSD, while written data are stored in HDD. This placement policy intensifies the advantages of SSD, avoiding overheads caused by write operations. The contribution of this work is that the hybrid CoW storage is obviously advantageous, in terms of performance and cost, for server consolidation workloads, especially for those in which sequential operations might be broken into small random ones.

As future work, we plan to implement sophisticated migration techniques between SSD and HDD. We expect that the identification of write-once read-many data is a crucial concern for the migration work. In addition, we also consider using SSD as a cache that temporarily stores the written blocks from guest VMs. There are lots of related work including the five-minute rule [21], and we will evaluate them in the virtualization environment and HVD. Efficient migration will make the hybrid virtual disk approach more successful for virtualized environments.

References

1. http://www.vmware.com
2. Sean Rhea, Russ Cox, Alex Pesterev. Fast, Inexpensive Content-Addressed Storage in Foundation. In *Proceedings of the 2008 USENIX Annual Technical Conference*.
3. Anthony Liguori, Eric Van Hensbergen. Experiences with Content Addressable Storage and Virtual Disks. In *Proceedings of the Workshop on I/O Virtualization (WIOV '08)*, 2008.
4. M. McLoughlin. The QCOW image format. http://www.gnome.org/~markmc/qcow-image-format.html.
5. Evangelos Kotsovinos, Tim Moreton, Ian Pratt, Russ Ross, Keir Fraser, Steven Hand, and Tim Harris. Global-scale service deployment in the XenoServer platform. In *Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS '04)*. San Francisco, California, December 2004.
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 Eurosys*, 2008.
7. F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber. Storage alternatives for mobile computers. In *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI)*, pages 25–37, November 1994.

8. C. Park, J. Seo, D. Seo, S. Kim, and B. Kim. Cost-efficient memory architecture design of nand flash memory embedded systems. In *Proceedings of the 21st International Conference on Computer Design (ICCD '03),* pages 474–480, October 2003.
9. Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: Using Flash Memory to B uild Fast, Power-efficient Clusters for Data-intensive Applications. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems 2009.*
10. Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In Proceedings of *the 14th International Conference on Architectural Support for Programming Languages and Operating Systems 2009.*
11. J. Kang, H. Jo, J. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In *Proceedings of the International Conference on Embedded Software (EMSOFT),* pages 161-170, October 2006. ISBN 1-59593-542-8.
12. VMware, Inc. VMware VMFS product datasheet. http://www.vmware.com/pdf/vmfs_ datasheet.pdf.
13. M. McLoughlin. The QCOW image format. http://www.gnome.org/~markmc/qcow-image-format.html.
14. A. Warfield, R. Ross, K. Fraser, C. Limpach, and S. Hand. Parallax: Managing storage for a million machines. In *Proceedings of 10th Hot Topics in Operating Systems*, May 2005.
15. http://www.atcomputing.nl/Tools/cowloop/
16. http://sysbench.sourceforge.net/
17. http://www.mysql.com/
18. http://www.seagate.com/staticfiles/docs/pdf/marketing/po_barracuda_7200_12.pdf
19. http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=161&partnum=MCCOE64G5MPP#component01
20. 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 19th ACM Symposium on Operating Systems Principles*, pages 164-177, 2003.
21. Goetz Graefe. The Five-Minute Rule 20 Years Later. Communications of the ACM, Vol. 52, No. 7, Pages 48-59.