Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device

Hannes Payer [*]
University of Salzburg
hannes.payer@cs.uni-salzburg.at

Marco A.A. Sanvido
Hitachi Global Storage Technologies Research
marco.sanvido@hitachigst.com

Zvonimir Z. Bandic
Hitachi Global Storage Technologies Research
zvonimir.bandic@hitachigst.com

Christoph M. Kirsch [*]
University of Salzburg
christoph.kirsch@cs.uni-salzburg.at

Abstract

We propose a new type of heterogeneous storage device called Combo Drive, which comprises of a smaller-capacity low-latency solid-state disk drive (SSD) *concatenated* with a larger-capacity high-throughput hard disk drive (HDD). The overall cost of a Combo Drive, similar to a Hybrid Drive, is still dominated by the more capacious HDD. With Combo Drive, the performance advantages of both the SSD and the HDD are readily utilized by assigning the lower portion of the address space, which is already considered by many file systems as faster than the higher portion, to the SSD. Performance can be optimized further on file system level on the host side. In contrast, existing Hybrid Drives utilize non-volatile memory hierarchically as a cache transparent to the environment requiring complex cache coherence algorithms. We built a Combo Drive prototype and propose multiple heuristic optimization algorithms implemented in file-system-level optimizers. Performance measurements on the host side show that the prototype achieves system start up time and application launch time similar to an SSD alone while offering large capacity and low cost of an HDD.

Categories and Subject Descriptors D.4.2 [*Operating Systems*]: Storage Management, Heterogeneous Storage Media; D.4.3 [*Operating Systems*]: File System Management

General Terms Storage

Keywords SDD, HDD, file system management, heterogeneous storage media

[*] Supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design and the Austrian Science Fund No. P18913-N15.

1. Introduction

In recent months we have witnessed a flurry of activities related to storage devices based on solid-state memory, in particular, solid-state drives (SSD) [5]. For over 50 years, hard disk drives (HDD) have been the non-volatile memory of choice for a range of applications from personal computer (PC) clients to enterprise storage, due to their appealing ratio of cost versus performance [10]. However, due to rapidly falling prices of silicon Flash [7], SSDs are emerging as a boot drive choice for high-end mobile computers as well as non-volatile memory cache for enterprise storage subsystems.

Storage devices are characterized primarily by their capacity (measured in GB or TB of storage space), sequential read/write (R/W) data rate or throughput (measured in MB/s), and random access times or latency (typically measured in ms or μs). The key components of HDDs are magnetic media where information bits are stored and mechanically actuated R/W recording heads, as well as printed circuit boards containing the hard disk controller. Unlike HDDs, SSDs do not contain any movable parts, and typically have superior latency, potentially two or more orders of magnitude lower. This dramatic improvement in latency could lead to a revolutionary change, similar to replacement of magnetic tapes with disk drives, especially in environments that are intensive in demanding a large number of input/output operations of small blocks of data such as system start up, application launching, databases, and caching in enterprise storage applications.

The sustained read and write data rate performance of SSDs vary greatly, depending on the type of underlying Flash memory (single-level cell (SLC) or multiple-level cell (MLC)) and the level of the parallelization in the controller. The first generation of SATA-based SSDs in the market offered excellent latency (smaller than 0.1 ms) as expected, but suffered in throughput, especially write throughput (smaller than 20 MB/s), as a consequence of the attempt to come up with a low cost product. The second generation corrected throughput-related issues improving it to a range between 40 MB/s up to 250 MB/s (depending on cost and target market). We refer to the first generation as *low-performance SSD* and to the second generation as *high-performance SSD*. In general, a low-performance SSD is expected to have lower cost compared to a high performance SSD, as many compromises may be made in the Flash controller and type of Flash memory used. HDDs typically have throughput between 60 MB/s and 180 MB/s, but have relatively slower latency in the range between 3 and 20 ms. However, HDDs offer significantly better cost per capacity ratio ($/GB), and are typically 3-5 times lower in cost. This cost ratio makes an HDD a primary storage device in the majority of the PC client space, likely to remain undisplaced in that position as long as magnetic recording technology continues improving storage areal density.

In this study, we aim to combine SSD and HDD into a single heterogeneous storage device called *Combo Drive*. The capacity of the Combo Drive is equal to the sum of the capacity of the smaller and more expensive SSD, and the larger and cheaper HDD. This is in contrast to a Hybrid Drive, which utilizes a caching architecture, i.e., uses non-volatile memory to cache read and write requests to HDD, and where the total capacity is equal to the capacity of the magnetic media. On the Combo Drive, the sectors of the SSD are occupying the lower sector range, concatenated by the sectors of the HDD. The overall device cost is still dominated by the HDD cost. Data access on the Combo Drive can be optimized when file usage characteristics in combination with properties of HDD and SSD are considered.

The overall optimization problem is distilled into two distinct optimization problems depending on the type of SSD used in the Combo Drive: low performance (low latency, low throughput) or high performance (low latency, high throughput). In the case of low-performance SSD, we propose two different kinds of heuristic optimization algorithms: one which moves executable files and program libraries to the SSD, and moves remaining files to the HDD; and one which moves randomly accessed files to the SSD, contiguously accessed files to HDD, and mixed accessed files to the HDD (reflecting the fact that low-performance SSD has lower throughput than HDD). In the case of high-performance SSD, we move the most frequently used files to the SSD and move all other files to the HDD, as there is no concern about throughput differences.

We have benchmarked our heuristic algorithms with a dedicated file system optimizer: we run HDTunePro, and measured Windows XP start up time and Microsoft Word 2003 applications launch time. The outcome of the experiments confirm that with both low-performance and high-performance SSD the performance of the Combo Drive was similar to the best of breed, i.e., boot time and application launch time of the SSD, and capacity of the HDD. Since the overall capacity is dominated by the capacity of the HDD, the Combo Drive cost is substantially similar to the cost of the HDD.

2. Combo Drive

A *Combo Drive* is a storage media device that combines Flash memory as well as magnetic memory in a single storage device and is addressable through the standard storage interface protocols, e.g. ATA [20]. It concatenates both devices and provides a single contiguous storage space where the Flash memory is mapped to the beginning and the magnetic media is mapped to the end of that space.

More formally: A Combo Drive consists of Flash memory of size k clusters and magnetic media of size l clusters and appears to its environment (the operating system) as a single contiguous storage media with the following properties: (1) the total size of the Combo Drive is $n = k + l$ clusters (size of Flash memory plus size of magnetic media), (2) the Flash media is at the lower logical block addresses $[0, k-1]$ of the Combo Drive, and (3) the magnetic media is at the higher logical block addresses $[k, n-1]$ of the Combo Drive.

In contrast to Combo Drive, caching architectures like Hybrid Drives [15, 17, 13] use a large buffer of non-volatile Flash memory to cache data. A cache is transparent to the environment. Therefore, the size of the Hybrid Drive is the size of the magnetic media. A cache architecture has the advantage that the host side does not need to know that the Hybrid Drive is caching. Moreover, the HDD may be able to spin down while still servicing read and write request. A cache architecture is typically a complicated architecture, and its performance is heavily dependent upon the cache block replacement strategy. Moreover, since most of the writes as well as the cached reads are going to hit the cache, the write pressure on the Flash is accelerating wear-out.

A Combo Drive, on the other hand, is a simpler architecture that shifts the burden of finding the best allocation strategy to the host file optimizer. Moreover, since many standard file systems already work with the assumption that lower LBAs are faster than higher LBAs (see Section 5.1 for a description of HDD performance), having the SDD/Flash storage statically allocated at the lower address space of the Combo Drive already improves overall system performance, making the file system optimizer a nice-to-have but still optional feature. Since many file systems are already able to cache large portions of data, it would be in principle possible to spin down the HDD of the Combo Drive while keeping the SSD/Flash part of the Combo Drive running, making it arguably possible to reduce power consumption as much as in a Hybrid Drive (by caching in DRAM only the HDD LBAs while servicing the SSD LBAs).

A Combo Drive has the following immediate advantages: the total storage device cost per capacity ratio ($/GB) can minimize the 3-5 times higher Flash cost ($/GB) by increas-

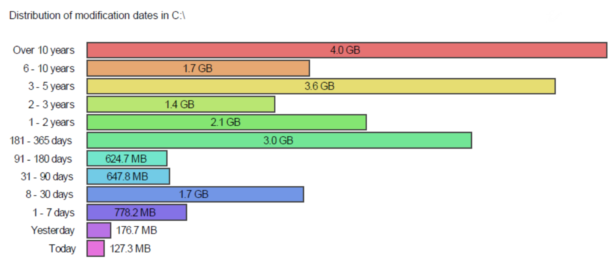


Figure 1. Typical disk usage

ing the magnetic memory capacity. In other words, a Flash-only storage device (SSD) of the same size would cost much more, whereas a Combo Drive can be made cost-competitive to a magnetic-media-only storage device (HDD). Since the cost of the Combo Drive depends indirectly on the ratio SSD to HDD storage space, the smaller this ratio is the more cost-competitive the Combo Drive will be. From an overall system performance point of view, we will show that by adopting user-space file system optimizations (cf. Section 3), a Combo Drive can utilize the advantages of both SSD and HDD.

Since overall system performance heavily depends on system workload, we focus on the phenomenon that, in a typical PC, not all files and data are created and used equally by the system. For some files we see an immediate performance advantage of storing them in the most performant but expensive media, whereas for the vast majority of files and data storing them in a low-performance, low-cost memory does not impact the overall system performance and is thus a better storage choice. For example, all rarely used files shall reside on a cheaper storage media. Figure 1 shows a typical corporate laptop hard disc drive access distribution. In this particular example only 1.1GB of the total 120GB HDD drive have been modified in the past week.

2.1 Implementations

A Combo Drive can in principle be implemented in three different ways. Concatenating two or more devices into a single logical storage device may entirely be done in software. A software-based approach, however, typically increases latency and host CPU utilization, may not be portable, and requires a separate SATA port for each device. A hardware-based Combo Drive, on the other hand, may be implemented by integrating raw Flash memory into a standard HDD. An integrated Combo Drive is likely to provide low latency and would only require a single SATA port on the host system but is difficult to build and does not allow exchanging the actual storage devices. Another hardware-based solution is a bridged Combo Drive which concatenates an SSD and an HDD via a SATA-to-2xSATA bridge chip. This approach may provide performance comparable with the integrated Combo Drive but is more flexible and also requires only a

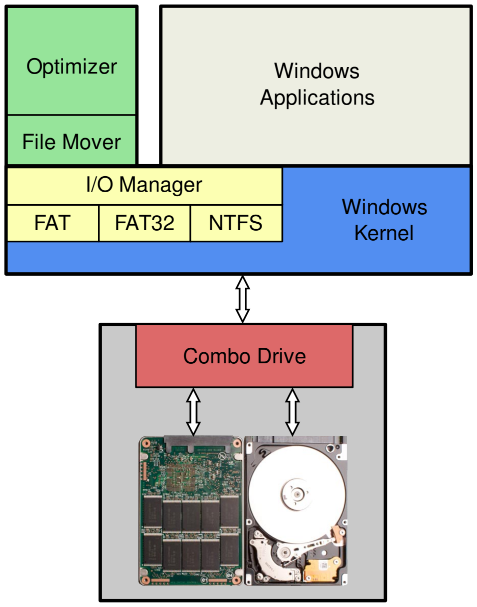


Figure 2. Combo Drive system

single SATA port on the host system. For our experiments, we built a prototype of a bridged Combo Drive.

2.2 Our Prototype

Our prototypical Combo Drive was build using the Silicon Image Sil5744 [19] bridge chip. While the chip is able to concatenate two SATA devices into a device that appears to the host transparently as a single entity, it also has many other operational modes such as RAID0, RAID1, etc. However, we only used its concatenation mode (BIG mode in Sil5744 terminology). The overhead introduced by the chip is negligible.

Figure 2 shows the components of the Combo Drive system. An SSD and an HDD are concatenated by the Sil5744 bridge chip in BIG mode. After concatenation both disks appear to Windows as a single disk with a single contiguous storage space. Windows can format the disk into NTFS, FAT, and FAT32. The file system optimizer runs in user space and executes its file and cluster moving operations depending on the optimization strategy and the state of the files on the Combo Drive. In the next section, we elaborate on how the file and cluster moving is implemented in user space.

3. Optimization Strategies

We discuss the experimental Combo Drive optimization strategies that have been implemented so far. We start by introducing the *Mover* library, which provides the basic set of routines required to implement the optimization strategies.

We distinguish *static* and *dynamic* optimization strategies. A static optimization strategy moves file sectors based on a predefined classification and independently of their ac-

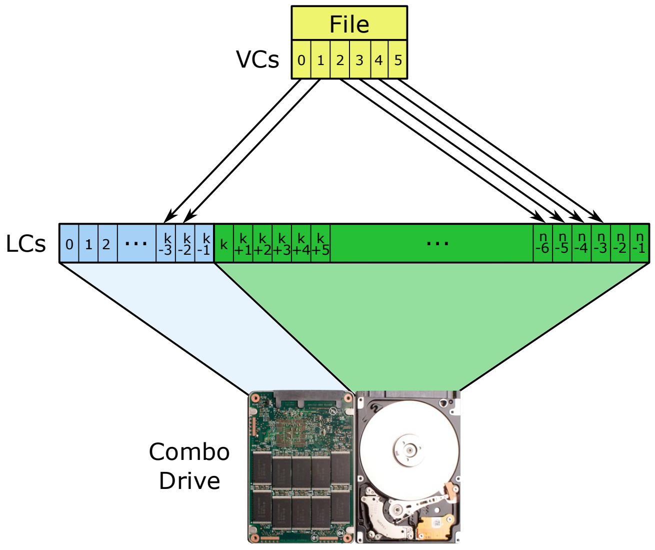

File
VCs
0 1 2 3 4 5
LCs
0 1 2 ··· k -3 k -2 k -1 k k +1 k +2 k +3 k +4 k +5 ··· n -6 n -5 n -4 n -3 n -2 n -1
Combo Drive

Figure 3. Virtual to logical cluster mapping

cess statistics to a specified disk. A dynamic optimization strategy obtains at runtime file usage statistics to move file sectors. In [18] file access patterns of several operating systems (including Windows NT) are examined. The authors showed that the vast majority of writes and 66% of reads are executed contiguously in Windows NT. Moreover, most of the traced random access reads refer to program executions. Just 3% of the files are both read and written. Note that similar results have also been obtained for BSD Unix [18].

3.1 Mover Library

The Mover library implements in user space the basic functionality for moving virtual clusters (VC) of files in an NTFS, FAT, and FAT32 file system to other free logical clusters (LC) of the disk. The mapping from virtual to logical clusters is shown in Figure 3. A virtual cluster number (VCN) represents the cluster within a file at a given file-cluster offset. A logical cluster number (LCN) designates the logical cluster to where the data is stored on disk. The cluster size is defined by the file system, e.g., the typical cluster size in NTFS [14] for disks larger then 2GB is 4KB which corresponds to eight 512-byte disk sectors.

The Mover library is implemented in C# and consists of two layers: a low-level I/O wrapper, which handles low-level Windows calls using the *DeviceIoControl* function of *kernel32.dll* and a higher-level interface providing easy-to-use functionality for implementing optimization strategies. The functionality of the low-level I/O wrapper includes getting the overall LC usage of the disk, getting the LCN to VCN mapping of a given file, and moving VCNs of a given file to other LCNs of the disk.

A file can be in one of six different states on the Combo Drive. It is in undefined state if it cannot be moved on the disk. Directories and files smaller then 1500 bytes are always

Disk	File Type
SSD	.exe, .dll, .sys, .msi, .cab, .drv, .jar, . . .
HDD	.jpg, .pdf, .wmv, .wma, .mp3, .java, .doc, .xls . . .

Table 1. File type classification

stored in the master file table to reduce fragmentation and are therefore in undefined state. Furthermore, a file can be stored in four different ways either contiguously, or else non-contiguously, on either SSD exclusively, or else on HDD exclusively. It is in mixed state if some of its LCs are stored on SSD and some on HDD. For example, the file in Figure 3 is in mixed state. Its first two virtual clusters are stored on the SSD at $[k_{-3}, k_{-2}]$, while its last four virtual clusters are stored on the HDD at $[n_{-6}, n_{-3}]$. The optimization strategies presented in the following section aim at placing files contiguously on either the SSD exclusively, or else on the HDD exclusively. Whenever a file cannot be moved contiguously in one piece to a disk due to disk fragmentation it is split up into smaller pieces that fit contiguously into free space of the target disk.

A user-space file system optimizer has advantages and disadvantages. Errors in a user-space program do not necessarily lead to system crashes and data inconsistency problems, which simplifies the development process. A disadvantage of the user-space design is that detecting potential for on-the-fly optimization comes with more overhead. For our prototype implementation we focused on Windows as target operating system with no other choice than a user-space implementation. A kernel-space implementation, for example, in Linux is future work. So far, we run the optimizer manually. A stand-alone optimization daemon, however, could perform file system optimizations automatically.

3.1.1 File Type Optimizer

The aim of the *File Type Optimizer* is to move files entirely to one of the two Combo Drive disks according to their expected rather than actual access characteristics distinguished by file type. This is a static optimization strategy since it does not consider the actual file usage at runtime.

Table 1 presents some examples of file types which under typical usage on Windows XP show better performance on SDD or HDD. Since executable files and libraries introduce more random access at load time and are rarely written, they perform better on SSD. Files that are contiguously read and/or contiguously written show better performance on HDD. We define the following optimization strategy for the File Type Optimizer:

STRATEGY 1. *Move executable files and program libraries to the SSD and move the remaining files to the HDD.*

Since the vast majority of disk capacity on PCs these days is used by user data (music, photos, videos, . . .) and a relatively small part is used by programs (exe, dll, sys, . . .), the low capacity of the SSD and the much higher capacity of the

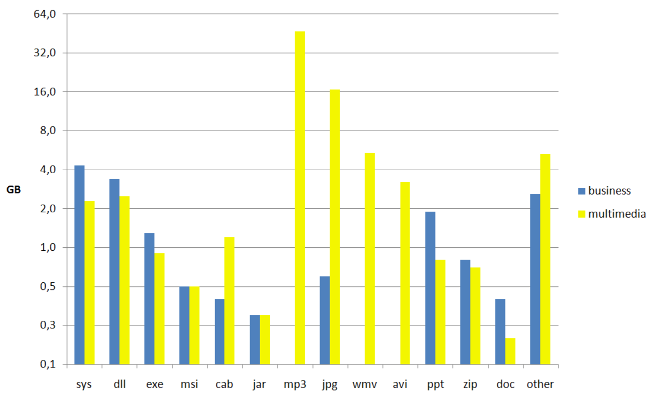


Figure 4. File type distribution

HDD of the Combo Drive fit the outcome of this optimization strategy. Figure 4 shows the distribution of file types on a typical business and multimedia home PC, respectively. Note that the number of executables and libraries is on both PCs almost the same and small in comparison to the user data on the multimedia home PC.

We allow the File Type Optimizer to apply its optimizations on files of a given directory, recursively starting from a given root directory or based on the entries of the Layout.ini file, which is administrated by Windows and can be found in the Windows/prefetch directory. In this file, Windows keeps track of recently and often used programs and files [11].

3.1.2 File Access Optimizer

The *File Access Optimizer* is a dynamic optimization strategy. File and sector access statistics are obtained using the Process Monitor Tool of Technet Microsoft [16], which logs the number of read and write calls on sector and file basis. We run the Process Monitor Tool periodically to find out whether a file is accessed contiguously, randomly, or both contiguously and randomly. The optimizer is applied after each period. We define the following optimization strategy for the File Access Optimizer:

STRATEGY 2. *Move files that are randomly accessed to the SSD, move files that are contiguously accessed to the HDD, and move files that are both randomly and contiguously accessed to the HDD.*

A refined File Access Optimizer which we left for future work could determine at runtime if parts of a file are either contiguously or else randomly accessed, and then move these parts to the corresponding disks.

For high-performance SSDs, which have read and write throughput comparable to HDDs, there is no need to move files that are contiguously accessed to the HDD. For these SSD types, we use the following optimization strategy in the File Statistics Optimizer:

STRATEGY 3. *Move the most frequently used files to the SSD and move all other files to the HDD.*

Again, we run the Process Monitor Tool periodically and, after each period, apply the optimizer based on the obtained access statistics.

3.2 Future Strategies

We are currently evaluating other possible optimization strategies. For example, a model-based optimizer may be able to search for files and sectors with a potential for improving overall system performance when moved using a detailed model of the Combo Drive. Besides performance, such an optimizer could also improve power consumption since SSD use in general less power than HDD for lack of any mechanical parts involved in the read/write process.

4. Related Work

Microsoft has introduced two solutions for Windows Vista, which are related to Combo Drive, called ReadyBoost and ReadyDrive [15]. ReadyBoost supports the use of non-volatile Flash storage devices to improve overall system performance. ReadyDrive enables Vista to use Hybrid Drives.

Some HDD manufacturers already announced Hybrid Drive Products [17]. Intel's Turbo Memory presented by Matthews et.al. [13] supports both ReadyBoost and ReadyDrive. Cache policies for performance improvements and power savings, and corresponding experimental results presented in this article show better performance and reduced power consumption. The authors observed that cache hits are counterproductive if data that is not critical to user experience is involved.

In [12, 2, 3, 4] different techniques are proposed to use Flash memory as non-volatile cache. The aim is to maintain blocks which are likely to be accessed in the near future in Flash memory. This allows to spin the disk down for longer periods and therefore reduces power consumption.

Kim et al. [8] showed that the combination of an HDD and an SSD results in an energy-efficient secondary storage solution for mobile platforms. The authors introduce a file placement technique, which optimizes power consumption. Experiments based on simulation show that significant reductions in energy consumption can be achieved using the file placement strategy. Benchmarks on real hardware are missing.

Wang et al. [23, 24] presented the Conquest file system, which supports HDDs extended with persistent RAM in which small files, file metadata, executables, and libraries are stored. Experiments showed that Conquest achieves better performance than purely disk-based file systems and similar performance as purely RAM-based file systems. Unlike Combo Drive, Conquest requires special host hardware and cannot be used with other file systems.

The difference between an architecture like the Hybrid Drive, which uses the SSD as cache with a predefined low-

Device		HDD	SSD	
		HGST 5K500	SSD1	SSD2
Avg. Latency (ms)	**Read**	18	0.1	0.1
	Write	18	0.1	0.1
Avg. Through-put (MB/s)	**Read**	53.0	64.2	109.5
	Write	53.0	12.3	75.3

Table 2. Characteristics of the disks used in the experiments

level cache policy, and the Combo Drive, which performs high-level file system optimizations, is somewhat symmetric to the difference between hardware-managed CPU cache management versus software-managed scratchpad memory [21, 1, 9, 22]. With scratchpad memory the problem of implementing a cache policy is moved into the compiler and/or runtime system, which may result in better performance, lower power consumption, and allows to adjust the caching scheme to the system requirements at runtime.

5. Experiments

The Combo Drive for our experiments consists of a 32GB SSD and a Hitachi Global Storage Technology (HGST) 5400rpm 500GB HDD. Thus the SSD part of the drive is about 6% of the total storage capacity. We used two different SLC-based SSDs, a low-performance SSD (referred to as SSD1) and a high-performance SSD (referred to as SSD2). The SSD1 came at a significantly lower price than the SSD2.

An overview of the performance details of the disks as measured by HDTunePro [6] is presented in Table 2. Both the SSD1 and SSD2 offer lower latency and higher read throughput in comparison to the HDD. The write throughput of the SSD2 is higher then the write throughput of the HDD, whereas the write throughput of the SSD1 is significantly lower. For our optimization experiments presented in Section 5.2 and Section 5.3, we only used the Combo Drive with SSD1 and the File Type Optimizer.

We ran all our experiments on a Shuttle PC with an Intel Pentium4 2.8GHz CPU and 512MB of RAM.

5.1 Combo Drive Performance

We used the HDTunePro [6] benchmark to measure the performance of the Combo Drive. HDTunePro executes two tests: (1) it reads/writes contiguously from/to the beginning of the disk (ID) to its end (OD) and measures its throughput, and (2) it reads/writes data of increasing seek distance to measure its latency (random access time). Figure 5 and 6 show the performance of the Combo Drive with SSD1 and SSD2, respectively. The SSD portion occupies the first 6% of the Combo Drive's total storage space. The left y-axes correspond to throughput, which is stated in MB/s, while the right y-axes correspond to latency stated in ms. The x-axes represent, for the throughput benchmark, the position on the disk (LBA) from ID to OD. For the latency benchmark, it represents the seek distance of increasing length. Note that throughput is shown by a line and latency by dots.

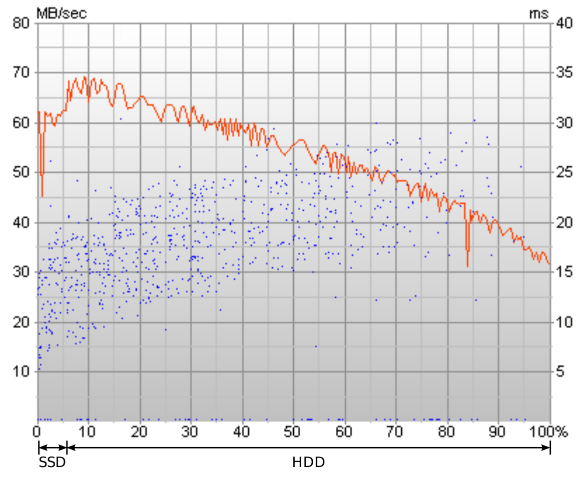
The throughput of the HDD decreases from ID to OD and is, on average, 55 MB/s for reading and 50 MB/s for writing. The maximum transfer rate is at the ID with about 69 MB/s for reading and writing. The SSD1 has good read throughput with 62 MB/s on average but its write throughput is low with 14 MB/s on average. The average read throughput of the Combo Drive with the SSD1 is 54 MB/s, cf. Figure 5(a). The average write throughput is 48 MB/s, cf. Figure 5(b). In comparison, the SSD2 provides an average read throughput of 90 MB/s and an average write throughput of 70 MB/s. Therefore, the average read throughput of the Combo Drive with the SDD2 is 55 MB/s, cf. Figure 6(a). The average write throughput is 50 MB/s, cf. Figure 6(b). Note that, for increasing seek distances, the latency on the HDD is increasing whereas the latency on the SDD is constant, independent of the seek distance. The dots at the bottom of Figure 5 and Figure 6 depict this constant fast latency for the SSDs.

5.2 Windows XP Startup

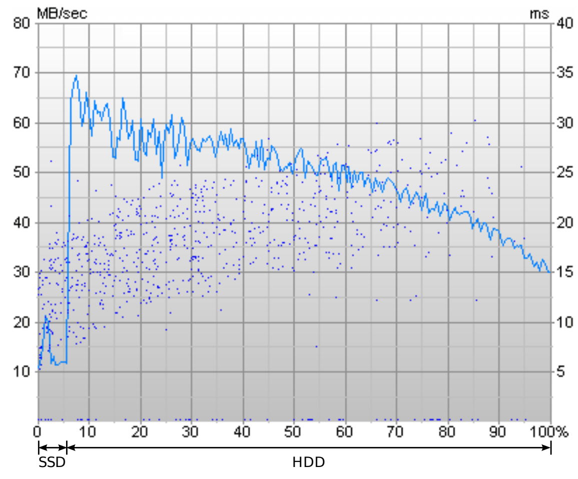
In this benchmark we measure the startup time of Windows XP. Note that the startup time consists of the BIOS boot time and the Windows boot time itself. The BIOS boot time was constantly 22s for all experiments. We always emptied the prefetch cache of Windows XP after booting the machine to get comparable results and repeated each testrun 20 times. The results of this experiment are shown in Figure 7 and represent the average startup time. The y-axis has logarithmic scale. *Unoptimized Combo Drive* represents the startup time after installing Windows on the Combo Drive without any modifications to the file layout. It took Windows XP 20s to boot. *SSD* means that all the data is on the SSD which results in a 17s Windows boot time. *c-HDD* and *a-HDD* represent the cases where all the data is either stored contiguously on the HDD or is in an aged state on the HDD, respectively. By aged state we mean that files are non-contiguously distributed over the HDD, which happens to files on a file system when Windows is running for months under typical usage. It took Windows 19s to boot up in the contiguous case and 224s in the aged case. Applying the File Type Optimizer (*File-Level Optimized Combo Drive*) results in a 17s Windows boot time. Both *SSD* and *File-Level Optimized Combo Drive* show the same performance but applying the File Type Optimizer results in a much smaller SSD utilization because just 13% of the Windows directory are moved to the SDD, whereas the other data is kept on the HDD. Moreover, the *File-Level Optimized Combo Drive* reduces the Windows XP startup time significantly by approximately 15% in comparison to the *Unoptimized Combo Drive*.

5.3 Microsoft Word 2003

In this benchmark we measure the startup time of Microsoft Word 2003. We ran this benchmark after a freshly booted Windows with an empty prefetch cache to get comparable

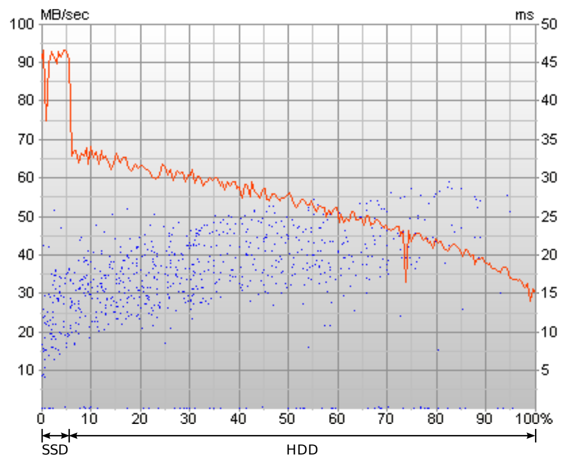


(a) eSATA Read

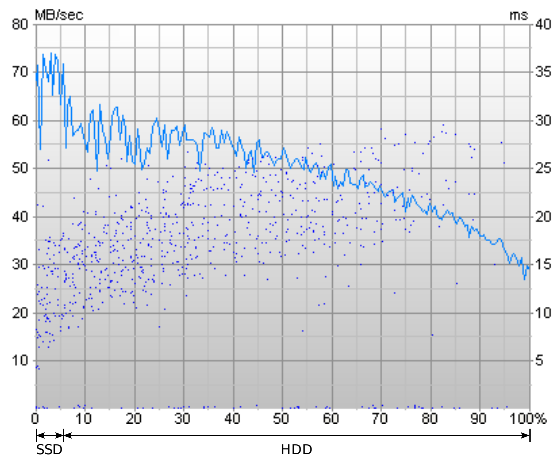


(b) eSATA Write

Figure 5. HDTune eSATA benchmark for Combo Drive with SSD1



(a) eSATA Read



(b) eSATA Write

Figure 6. HDTune eSATA benchmark for Combo Drive with SSD2

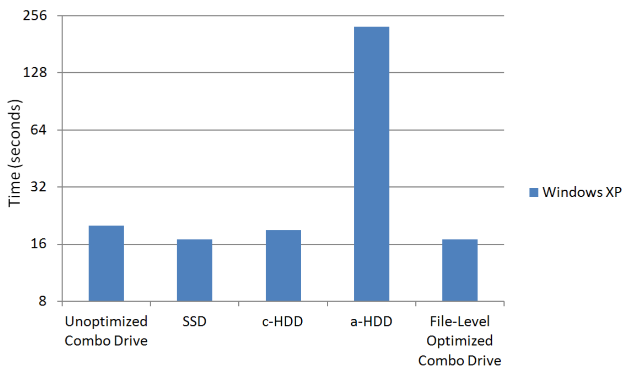


Figure 7. Windows XP startup benchmark

measurement results. We repeated these testruns 20 times. Note that we use the same abbreviations for the different testruns as introduced in the previous section. For *SSD* it took 2.82s to start Word. Having all data on HDD in *c-HDD* or *a-HDD* state it took 4.15s and 29.03s, respectively. Applying the File Type Optimizer (*File-Level Optimized Combo Drive*) results in 2.79s startup time. The File Type Optimizer moved just 46MB of the installed Microsoft Office 2003 package to the SSD. The results are shown in Figure 8. The y-axis has again logarithmic scale.

6. Conclusions

We built a research prototype of a novel heterogeneous storage device called Combo Drive, which comprises of a smaller-capacity low-latency solid-state disk drive (SSD) concatenated with a larger-capacity high-throughput hard disk drive (HDD). For different types of SSD (low performance: low latency but low throughput; or high performance: low latency as well as high throughput), we proposed

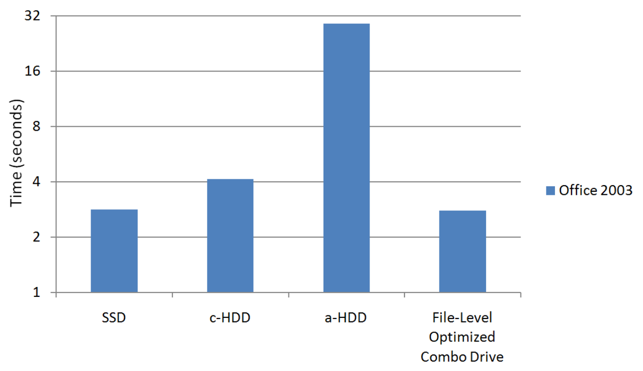


Figure 8. Microsoft Word 2003 benchmark

several heuristic algorithms implemented in a file-system-level optimizer, which redistributes files between SSD and HDD based on file type or file access patterns. We ran experiments with the prototype using HDTunePro and measured Windows XP startup time and Microsoft Word 2003 application launch time. In these benchmarks, our optimization strategies provide with our Combo Drive the low latency of an SSD in combination with the high capacity, high throughput, and low cost of an HDD.

References

[1] AVISSAR, O., BARUA, R., AND STEWART, D. An optimal memory allocation scheme for scratch-pad-based embedded systems. *Transactions on Embedded Computing Systems 1*, 1 (2002), 6–26.

[2] BISSON, T., AND BRANDT, S. A. Reducing energy consumption with a non-volatile storage cache. In *Proc. IWSSPS* (2005), IEEE.

[3] BISSON, T., BRANDT, S. A., AND LONG, D. D. A hybrid disk-aware spin-down algorithm with I/O subsystem support. In *Proc. IPCCC* (2007), IEEE, pp. 236–245.

[4] CHEN, F., JIANG, S., AND ZHANG, X. SmartSaver: turning flash drive into a disk energy saver for mobile computers. In *Proc. ISLPED* (2006), ACM, pp. 412–417.

[5] ENGADGET. Storage. `http://www.engadget.com/category/storage/`.

[6] HDTUNE. HD Tune Pro Version 3.10 . `http://www.hdtune.com`.

[7] INSPECTRUM. Nand flash prices. `http://www.insye.com/`.

[8] KIM, Y.-J., KWON, K.-T., AND KIM, J. Energy-efficient file placement techniques for heterogeneous mobile storage systems. In *Proc. EMSOFT* (2006), ACM Press, pp. 171–177.

[9] MAI, K., PAASKE, T., JAYASENA, N., HO, R., DALLY, W. J., AND HOROWITZ, M. Smart memories: a modular reconfigurable architecture. In *Proc. ISCA* (2000), ACM, pp. 161–171.

[10] MARCO A. A. SANVIDO, FRANK R. CHU, A. K. R. S. Nand flash memory and its role in storage architectures. In *Proceedings of the IEEE* (2008), vol. 96, IEEE, pp. 1864–1874.

[11] MARK RUSSINOVICH, S. D. *Microsoft Windows Internals*, 4 ed. Microsoft Press, 2005, pp. 458–462.

[12] MARSH, B., DOUGLIS, F., AND KRISHNAN, P. Flash memory file caching for mobile computers. In *Proc. HICSS* (1994), IEEE, pp. 451–460.

[13] MATTHEWS, J., TRIKA, S., HENSGEN, D., COULSON, R., AND GRIMSRUD, K. Intel®turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. *Trans. Storage 4*, 2 (2008), 1–24.

[14] MICROSOFT. The default cluster size for the NTFS and FAT file systems. http://support.microsoft.com/kb/314878.

[15] MICROSOFT. ReadyDrive and hybrid disk. `http://www.microsoft.com/whdc/system/sysperf/perfaccel.mspx`.

[16] MICROSOFT. Technet microsoft process monitor. `http://technet.microsoft.com`.

[17] PANABAKER, R. Hybrid Hard Disk & ReadyDriveTM technology: Improving performance and power for Windows Vista Mobile PCs. In *Proc. WinHEC* (2006). `http://www.microsoft.com/whdc/winhec/pres06.mspx`.

[18] ROSELLI, D., LORCH, J. R., AND ANDERSON, T. E. A comparison of file system workloads. In *Proc. ATC* (2000), USENIX, pp. 41–44.

[19] SILICON IMAGE. Sil5477 documentation. `http://www.siliconimage.com`.

[20] T13/1699D. *Information technology - AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS)*. ANSI INCITS, 2008.

[21] UDAYAKUMARAN, S., AND BARUA, R. Compiler-decided dynamic memory allocation for scratch-pad based embedded systems. In *Proc. CASES* (2003), ACM.

[22] UDAYAKUMARAN, S., DOMINGUEZ, A., AND BARUA, R. Dynamic allocation for scratch-pad memory using compile-time decisions. *Transactions on Embedded Computing Systems 5*, 2 (2006), 472–511.

[23] WANG, A.-I. A., KUENNING, G. H., REIHER, P. L., AND POPEK, G. J. The conquest file system: Better performance through a disk/persistent-RAM hybrid design. *Trans. Storage 2*, 3 (2006), 309–348.

[24] WANG, A.-I. A., REIHER, P. L., POPEK, G. J., AND KUENNING, G. H. Conquest: Better performance through a disk/persistent-RAM hybrid file system. In *Proc. ATEC* (2002), USENIX, pp. 15–28.