Bulletin of the Technical Committee on

Data Engineering

December 2010 Vol. 33 No. 4

IEEE Computer Society

Letters

Letter from the Editor-in-Chief . . . . . *David Lomet* 1
Letter Announcing TC Chair Election Results . . . . . *Carrie Walsh* 2
Letter from the Outgoing TCDE Chair . . . . . *Paul Larson* 2
Letter from the Special Issue Editor . . . . . *Peter Boncz* 3

Special Issue on Data Management using Modern Storage Hardware

Trends in Storage Technologies . . . . . *E. Eleftheriou, R. Haas, J. Jelitto, M. Lantz, H. Pozidis* 4
Implementing an Append-Only Interface for Semiconductor Storage . . . . . *C. W. Reid, P. A. Bernstein* 14
Designing Database Operators for Flash-enabled Memory Hierarchies . . . . . *G. Graefe, S. Harizopoulos, H. Kuno, M. A. Shah, D. Tsirogiannis, J. L. Wiener* 21
Flash in a DBMS: Where and How? . . . . . *M. Athanassoulis, A. Ailamaki, S. Chen, P. B. Gibbons, R. Stoica* 28
Storage Class Memory Aware Data Management . . . . . *B. Bhattacharjee, M. Canim, C. A. Lang, G. A. Mihaila, K. A. Ross* 35
Accelerating In-Page Logging with Non-Volatile Memory . . . . *S.-W. Lee, B. Moon, C. Park, J.-Y. Hwang, K. Kim* 41
uFLIP: Understanding the Energy Consumption of Flash Devices . . . . . *M. Bjørling, P. Bonnet, L. Bouganim, B. Jònsson* 48

Conference and Journal Notices

ICDE 2011 Conference . . . . . back cover

Editorial Board

Editor-in-Chief

David B. Lomet
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
lomet@microsoft.com

Associate Editors

Peter Boncz
CWI
Science Park 123
1098 XG Amsterdam, Netherlands

Brian Frank Cooper
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043

Mohamed F. Mokbel
Department of Computer Science & Engineering
University of Minnesota
Minneapolis, MN 55455

Wang-Chiew Tan
IBM Research - Almaden
650 Harry Road
San Jose, CA 95120

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

TC Executive Committee

Chair

Paul Larson
Microsoft Research
One Microsoft Way
Redmond WA 98052, USA
palarson@microsoft.com

Vice-Chair

Calton Pu
Georgia Tech
266 Ferst Drive
Atlanta, GA 30332, USA

Secretary/Treasurer

Thomas Risse
L3S Research Center
Appelstrasse 9a
D-30167 Hannover, Germany

Past Chair

Erich Neuhold
University of Vienna
Liebiggasse 4
A 1080 Vienna, Austria

Chair, DEW: Self-Managing Database Sys.

Anastassia Ailamaki
École Polytechnique Fédérale de Lausanne
CH-1015 Lausanne, Switzerland

Geographic Coordinators

Karl Aberer (**Europe**)
École Polytechnique Fédérale de Lausanne
Batiment BC, Station 14
CH-1015 Lausanne, Switzerland

Masaru Kitsuregawa (**Asia**)
Institute of Industrial Science
The University of Tokyo
Tokyo 106, Japan

SIGMOD Liason

Christian S. Jensen
Department of Computer Science
Åarhus University
DK-8200 Arhus N, Denmark

Distribution

Carrie Clark Walsh
IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
CCWalsh@computer.org

Letter from the Editor-in-Chief

TCDE Chair Election

There is a letter from the IEEE Computer Society Sr. Program Specialist Carrie Walsh reporting on the results of the recent election for TCDE Chair. The election was held in accordance with IEEE Computer Society procedures. Voting was held open from September 15 through December 1. The Computer Society collected votes via their election web site. Previously, a nominating committee had asked for nominations from TCDE members. Elections such as these are important for two reasons: (1) it is an opportunity for members to express their preferences for leadership positions; and (2) these positions matter as decisions made by TC Chairs do influence the effectiveness of a TC and its program of activities.

On My Election as TCDE Chair

It is an honor to be nominated to serve as Chair of the TC on Data Engineering, and now to be elected. I believe that jobs like this matter to the health and the relevance of an organization.

There are two main activities of the TCDE. (1) We sponsor the International Conference on Data Engineering, one of the premier database conferences. I will work with ICDE Steering Committee Chair Calton Pu in the on-going effort to preserve and enhance the quality of ICDE. (2) We publish a quarterly bulletin, the Data Engineering Bulletin. This informal publication is one of the most higly cited publications in the database field. At least for a while longer, I will continue as Bulletin editor in addition to serving as TC Chair.

I thank the nominating committee for giving me the opportunity to serve as TC chair, and the TC voters for electing me to this office. And I thank my predecessor and colleague, Paul Larson, for his fine job as TC Chair. My intention is to continue in his footsteps.

The Current Issue

Hardware platforms have changed substantially since database systems first made their appearance in the late 1960's. Many have commented that perhaps we should be revisiting DBMS architecture in light of these hardware changes. For a very long time (the last 20 years or so), such revisiting did not lead to new DBMS architecture. The fundamental technical issue continued to be the three orders of magnitude that separated main memory access from disk access. Processor speed improved, and processor cache utilization became more important, but mostly this produced local algorithmic changes. Recent processor changes are now substantial enough to trigger larger changes, but changes in storage technology have really brought database architecture to the forefront.

This issue focuses on persistent random access memory (PRAM), with access characteristics that are much closer to those of main memory technology. Interestingly, it was the consumer marketplace that made this class of memory economically viable, not the database or even the server business more broadly. There are several flavors of PRAM, flash, phase change, etc. How database architecture is impacted may well depend on which technology ultimately dominates. But any of them will require re-architecting systems to best exploit them. Some are block oriented, some require an erase cycle before they can be written, etc. They may have a limited number of writes, so "wear leveling" may be important.

The current issue covers several approaches to exploiting the new PRAM technologies plus an article focused on underlying PRAM technology. The thing about architecture is that it makes one re-think almost everything. So in the same way that storage has caused a re-thinking of architecture, architecture engenders new collections of techniques to enable the architectures. The issue contains some of the very latest thinking from both industry and academia. It's an exciting topic and the issue serves as a terrific gateway to it. I want to thank Peter Boncz for his fine job of handling this issue as its editor and bringing it to fruition.

David Lomet
Microsoft Corporation

Letter Announcing TC Chair Election Results

Subject: New TC on Data Engineering Chair
Date: Thu, 4 Nov 2010 09:47:28 -0700
From: Carrie C Walsh <CCWalsh@COMPUTER.ORG>
To: tc-election@COMPUTER.ORG

Dear TCDE Members:

The polls for the TCDE Chair election have closed, and the votes have been tallied and reviewed.

David Lomet has been elected TC Chair with 100% of the total votes.

There were no write-in candidates.

Our congratulations to Dr. Lomet on being elected TCDE Chair!

Sincerely yours,
Carrie

Carrie Walsh
IEEE Computer Society

Letter from the Outgoing TCDE Chair

It is time to hand over the chairmanship of the IEEE-CS Technical Committee on Data Engineering. I took over from Erich Neuhold in 2007. I have had the pleasure of working with an oustanding TCDE Executive Committee comprised of Calton Pu, Thomas Risse, Erich Neuhold, Guy Lohman, Karl Aberer, Masaru Kitsuregawa, David Lomet and Yannis Ioannides.

TCDE has continued to sponsor several conferences and workshops related to data engineering. Our flagship conference, the International Conference on Data Engineering (ICDE), has continued to grow in quality and attendance. Its workshop program has become very successful with eight to ten workshops co-located with the main conference. We have also, with the help of our excellent Editor in Chief Dave Lomet, continued the publication of the quarterly Bulletin on Data Engineering.

I would like to thank the members of the Executive Committee for their contributions and all members of the Technical Committee on Data Engineering for their support. I wish the new Chairman David Lomet all the best for the future and thank him for his willingness to take on the leadership of TCDE.

Paul Larson
Microsoft Corporation

Letter from the Special Issue Editor

There will always be an interplay between architecting a software system and the hardware this system runs on. This interplay is particularly evident in data management, and has always fascinated me personally and inspired me in my research. Therefore, I have made this the theme of this bulletin, which David Lomet graciously asked me to put together.

In the past half century, the storage hardware scene has been fully dominated by magnetic disk and tape; both of which depend on physically moving parts. Recently, solid state storage has gained advantages over these magnetic technologies. NAND flash, in particular, has combined the technological ability to adhere to an aggressive silicon scaling road map with impressive economies of scale, fuelled by the worldwide adoption of digital photography and other multimedia consumer electronics devices. It is now being rapidly adopted in the performance-sensitive part of the data storage market, highlighting the constraints that magnetic technologies were imposing on data management solutions, especially regarding high random access latency, but also concerning energy efficiency. NAND flash, however, is just one of the technology contenders in this space, which is known as "Storage Class Memory" (a concept originating from IBM Research). This special issue examines the question which storage hardware technologies are on the future road map, and what would be their consequences for future data management systems.

The direct trigger for pursuing this particular topic in this issue was the excellent keynote that Evangelos Eleftheriou gave at the sixth International Workshop on Data Management for Modern Hardware (DAMON) at SIGMOD/PODS 2010 in Indianapolis, USA. This keynote talk was very well-attended and well-received and covered in detail the road maps of magnetic tape, magnetic disk, a host of solid state technologies. All this information, and more, you find now written up in the introductory paper. Conclusions are that a Storage Class Memory (SCM) will emerge as the storage hardware of choice for performance-critical applications, but that magnetic technologies are not dead; in fact, a healthy future for tape is to be expected. Phase Change Memory (PCM) could dominate the SCM segment in the medium- or long-term.

Complementing the opening piece, which focuses exclusively on the hardware road map, I invited contributions from data management researchers who had worked on exploiting modern storage hardware:

- Three papers contain results on data management on NAND flash. Bernstein et al describe the Hyder system, which argues for purely log-based approach to provide transactional storage for high-performance distributed systems. The team of authors from HP Labs describe work on FlashScan/FlashJoin, and introduce new ideas for spilling merge-algorithms; showing that with flash one can avoid multiple passes to magnetic disk. The combined Intel/EPFL team summarise their work on using cheap flash for logging, and introduces new work in supporting mixed OLAP/OLTP workloads. The idea is to direct OLTP modifications to a delta structure maintained on flash, which OLAP queries merge quickly using a new Materialized Sort-Merge (MaSM) algorithm.

- Two papers in this issue look at the software consequences of technologies beyond flash: the IBM team outlines its vision on Storage Class Memory and in particular Phase Change Memory, and the Korean team looks at the particular attractions of Phase Change Memory and in-page logging.

- The final paper, by the original authors of the best-paper-award-winning uFLIP benchmark (at CIDR 2009), investigates the energy efficiency of current Solid State Drives (SSDs); showing that remarkable difference exist, which can only be brought to light using an experimental approach.

Let me hereby thank all authors for their efforts, and express my hope that you will enjoy reading the resulting material.

Peter Boncz
CWI
Amsterdam

Trends in Storage Technologies

E. Eleftheriou, R. Haas, J. Jelitto, M. Lantz and H. Pozidis
IBM Research – Zurich, 8803 Rüschlikon, Switzerland

Abstract

The high random and sequential I/O requirements of contemplated workloads could serve as impetus to move faster towards storage-class memory (SCM), a technology that will blur the distinction between memory and storage. On the other hand, the volume of digital data being produced is increasing at an ever accelerating pace. Recent technology demonstrations indicate that low-cost, low-power tape storage is well positioned to scale in the future and thus serve as archival medium. In this paper, we will look at the implications of these technology extremes, namely, SCM and tape, on the storage hierarchy.

1 Introduction

One of the challenges in enterprise storage and server systems is the rapidly widening gap between the performance of the hard-disk drives (HDD) and that of the remainder of the system. Moreover, trying to narrow this gap by increasing the number of disk spindles has a major impact on the energy consumption, space usage, and cost. Given that major improvements in HDD latency are not expected, research and development efforts have shifted towards semiconductor memory technologies that not only complement the existing memory and storage hierarchy but also reduce the distinction between memory (fast, expensive, volatile) and storage (slow, inexpensive, nonvolatile).

In the past, Flash memory technology has been driven by the consumer market alone. The current cost per GB and latency characteristics of NAND Flash make it an interesting candidate to bridge the widening performance gap in enterprise storage systems. In particular, the introduction of multi-level cells (MLC) further decreased the cost as compared to single-level cells (SLC), albeit at the expense of reliability and performance. The absence of moving parts in NAND Flash enhances the ruggedness and latency, and reduces the power consumption, but its operation and structure pose specific issues in terms of reliability and management overhead, which can be addressed in the hardware or software of a Flash controller. Although NAND Flash has already established itself in the memory and storage hierarchy between DRAM and HDDs, it still cannot serve as a universal memory/storage technology. In other words, it does not fully qualify as a storage-class memory (SCM) technology [1]. There are various technologies that could eventually qualify as SCM which combine the performance benefits of solid-state memories with the low cost and the large permanent storage capabilities of magnetic storage. These include ferroelectric, magnetic, phase-change, and resistive random-access memories, including perovskites and solid electrolytes, as well as organic and polymeric memories [2]. It is widely believed that the prime candidate of all these technologies to become SCM is phase-change memory (PCM).

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

On the other hand, the volume of digital data being produced is growing at an ever increasing pace. According to a recent International Data Corporation (IDC) study, 800 exabytes of data were created in 2009 [3]. In the future, this already staggering volume of data is projected to increase at an annual growth rate of more than 60%, faster than the expected growth of the total storage capacity worldwide. Moreover, new regulatory requirements imply that a larger fraction of this data will have to be preserved for extended periods of time. All of this translates into a growing need for cost-effective digital archives. Despite the significant progress of HDD technology over the past years, magnetic tape still remains the least expensive long-term archiving medium. In a recent technology demonstration by IBM in collaboration with Fujifilm, an areal density of 29.5 Gb/in^2 stored on a new BaFe medium was shown, which translates into a potential tape cartridge capacity of 35 terabytes [4]. Moreover, an analysis of the limits of current tape technology suggests that the areal density of tape can be pushed even further towards 100 Gb/in^2, leading to cartridge capacities in excess of 100 terabytes [5]. This clearly indicates that tape remains a very attractive technology for data archiving, with a sustainable roadmap for the next ten to twenty years, well beyond the anticipated scaling limits of current HDD technology. Moreover, the new long-term file system (LTFS) that has been introduced in the LTO (Linear-Tape-Open) generation-5 tape-drive systems allows efficient access to tape using standard and familiar system tools and interfaces. LTFS allows the tape cartridge to be self-describing and self-contained, making tape look and work like other storage solutions.

In this paper, we will first briefly look at today's memory storage requirements from a workload perspective. Next we will discuss HDD technology trends and various solid-state memory technologies that potentially could serve as SCM. Then, we will focus on PCM, the prime contender for SCM, review its current status and technological challenges and also discuss its impact on the system memory and storage hierarchy. We will also consider the other extreme: tape storage. Specifically, we will provide a short overview of the role of tape in the storage hierarchy and where we see the main advantages of tape. We will then briefly refer to the recent world record areal density study of 29.5 Gb/in^2, which indicates that tape has a viable roadmap for the foreseeable future. We will also discuss the recent introduction of LTFS into tape systems and its wide implications for the growing market of archival applications. Finally, we will close by discussing the implications of these two technology extremes, namely, SCM and tape, on the tiered storage hierarchy.

2 Optimized Workloads

There is a fundamental trend towards designing entire systems such that they are optimized for particular workloads, departing from the traditional general-purpose architecture. The typical system, with standard CPUs consisting of a small number of identical cores with a common set of accelerators and relying on a memory and storage hierarchy mainly composed of DRAM and HDDs, has reached its limits in terms of delivering competitive performance improvements for an increasingly diverse set of workloads: future systems will be built out of increasingly heterogeneous components.

From a CPU-level perspective, technology scaling will allow 50 billion transistors to be put on a single chip in approximately five CPU generations, whereas the chip area and total power consumed will remain similar to current levels, namely, $\sim$500 mm^2 and $\sim$200 W. By exploiting the plethora of transistors available, we expect an increasing heterogeneity with dedicated fixed-functions cores (e.g., decryption, XML parsing, pattern matching), programmable cores with domain-tailored instruction sets, or even reconfigurable logic (FPGA-like).

The memory subsystem is becoming one of the most expensive parts of the system, with virtualization being one of the key factors fuelling the push towards larger amounts of memory. Such subsystems will likely be a mix of fast, expensive memory (like DRAM) and a slower, more cost-efficient memory, such as PCM. Mixed memory systems could be built in which the DRAM serves as hardware cache for the PCM memory or DRAM and PCM could both form a contiguous memory environment in which the OS or memory controller determines the optimal placement of data.

From a storage-level perspective, not all data is needed equally often or quickly. The coarse distinction between on-line, near-line, and off-line data naturally maps to heterogeneous storage tiers using the most appropriate technology in terms of access performance and cost. Automated tiering methods already offer the capability to relocate data between tiers as access patterns evolve. Solid-state nonvolatile technologies such as Flash have started to displace HDD usage both in on-line high-performance storage and in mobile computers. HDDs together with the data deduplication technologies are poised to increasingly target near-line applications, such as backup and restore, "encroaching" on the traditional stronghold of tape. On the other hand, with the introduction of LTFS, tape appears to be well positioned to offer complete system-level archive solutions. It is deemed that the low energy consumption and volumetric efficiency together with a file-level interface represent a huge growth opportunity for tape in archival as well as off-line applications.

In summary, the drive for increased performance, coupled with improved cost and power efficiency, is already leading to a greater exploitation of the heterogeneity at the CPU, memory and storage levels. We expect that further performance increases will be delivered by fully application-optimized hardware.

Let us now look at key applications and their workloads, and focus on streaming analytics as well as active archiving. Streaming analytics refers to applications that process large volumes of mainly unstructured streaming data to provide critical information rapidly. This may apply to continuous portfolio risk evaluation, real-time analysis of call data-records, etc. (for examples, see [6]). Such applications require a system designed to sustain the continuous processing of ephemeral data, with systems built out of special pre-processing frontend engines that take disparate data streams at wire rates as input, filter and analyze these data streams very efficiently, and provide the highest-value outputs to a back-end computational layer. This will be achieved by a combination of heterogeneous and dedicated processing engines complemented by a customized memory and storage hierarchy that can accommodate the throughput of such data streams while buffering a sufficient amount for processing purposes in a cost-effective fashion.

At the other extreme of the application spectrum, active archiving addresses the need to efficiently store and access all the data created by an organization. The objective is to continue to be able to leverage information even after it has been stored for good. This may apply to reprocessing or searching data that was expensive to collect in the light of new algorithms or when searching for new artifacts, such as in the case of oil-field inspections or clinical studies, or data that cannot be recreated, such as bank transactions kept for compliance reasons.

An active archive application makes all data always available without taking up expensive primary disk capacity, but instead spreading such data across multiple tiers of storage in a transparent fashion to the user (except for the variation in access performance) [7]. We expect that this will be achieved by using a mix of cache and storage technologies composed of PCM, Flash, HDDs, and tape.

3 HDD Technology Trends

The cost of storing data using HDDs is primarily governed by the achievable areal recording density. Over the past 30 years, HDD capacity and areal densities have increased at compound annual growth rates (CAGR) of about 60%, resulting in a similar exponential decrease in cost per byte of storage. In the 1990's HDD areal density grew at a CAGR of up to 100%. However, during the transition from longitudinal to perpendicular recording, this rate slowed to values as low as $\sim$20%, and currently is around 30%.

The spectacular improvement in HDD capacity has been achieved by continually scaling the size of an individual bit cell, concomitant with steadily improving the various building blocks of the drive. Unfortunately, there is a fundamental physical limitation to how far this scaling can be continued because of thermal stability effects in the storage medium, known as the super paramagnetic limit, that occurs when the magnetic energy stored in a bit becomes comparable to the thermal energy in the environment.

Currently there are two proposed technologies to extend magnetic recording beyond the super-paramagnetic

limit: bit-patterned media and heat-assisted magnetic recording (HAMR). The idea of bit-patterned media is to pattern the media into magnetically isolated, single domain islands, such that each bit consists of a single "grain", which can be significantly larger than the 8 nm grain size in use today. Implementing this idea requires the development of a low-cost, large-area nano-scale patterning technique, such as nano-imprint lithography, and also necessitates changes in the way track-follow servo and timing are performed in the drive. The idea of HAMR is to use a medium with a large magnetic anisotropy and to locally reduce the magnitude of the magnetic field required to write to the medium by applying a very short and localized heat pulse with a laser. Challenges associated with this idea include the cost of the laser and the waveguide technology required to deliver the heat pulses, as well as problems associated with keeping the heat localized in the metallic storage medium, which is a good heat conductor.

To date, areal densities of over 800 Gb/in^2 have been demonstrated in the lab using perpendicular recording [8], and at the time of writing, the highest areal density shipping in a product was $\sim$739 Gb/in^2 [9]. The highest capacity HDDs currently available are 1.5 TB in the 2.5$''$ form factor and 3 TB in the 3.5$''$ form factor. The 3-TB drives contain either four 750-GB platters or five 600-GB platters. The cost of a 3-TB drive is around $250, which translates to about 8 cent/GB. However, the total costs of ownership (TCO) of disk-based storage systems are significantly higher than this because of the additional cost of power, cooling, redundancy, etc.

Conventional perpendicular recording is expected to extend to areal densities of up to $\sim$1 Tb/in^2 using exchanged-coupled composite (ECC) media. At the current rate of scaling, this areal density will be reached in about one to two years. To scale beyond this, other technologies, such as patterned media and/or HAMR, will be required.

In stark contrast to the continued exponential improvement in HDD areal densities, other attributes, such as power consumption, data rate and access times, have recently been improving at much more modest rates or in some case not at all. For example, data rate is primarily determined by the linear density and the rotation rate of the drive. Figure 1 shows a plot of the maximum sustained data rate versus time, where it can be seen that since 2000, data rates have increased by a factor of only about 2 to 3. This stagnation is due to a significant slowdown in linear density scaling and the only marginal increase in rotation rates in this period. The latter are limited by the radius of the platters and the mechanical components of the drive. Recently even a trend to decrease rotation rates to reduce power consumption emerges.

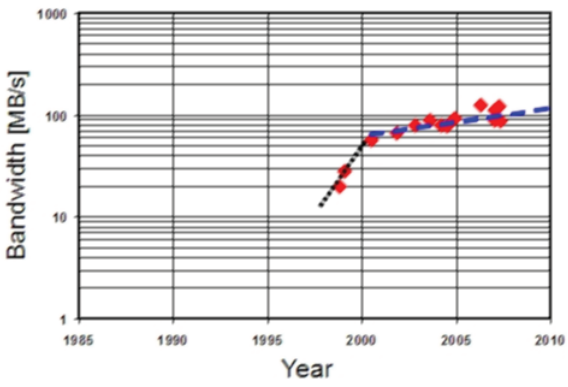


Figure 1: Max sustainable rate. Adapted from [33].

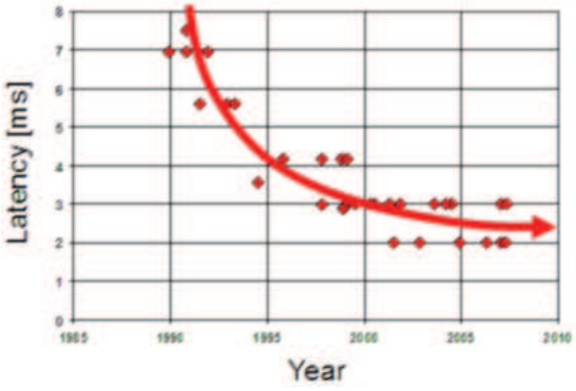


Figure 2: HDD latency. Adapted from [33].

Latency is the time interval between receipt of a read command and the time the data becomes available. It is dominated by the seek time and the disk rotation rate (typically in the range of 5400 -Ű- 15000 rpm). In modern HDDs, access times range from around 15 ms to only a few milliseconds, depending on the class and the power consumption of the drive. Figure 2 shows a plot of latency decrease versus time, where again it can be

seen that improvements since the year 2000 have been only marginal. This apparent floor results from practical limitations of the mechanical components used for rotating the disks and positioning the read-write heads.

The number of read/write operations that can be performed per second (IOPS) is given by the inverse of the latency. The stagnation in latency improvements thus means that also IOPS performance gains have stagnated. In fact, because of the continued exponential increase in capacity and the stagnation of latency, the IOPS per GB have actually been decreasing exponentially. The only way to improve the IOPS performance is to increase the number of spindles operated in parallel, which has the obvious disadvantages of a linear increase in system costs and power consumption.

The power consumption of an HDD depends on the operations performed and varies with the type of HDD. Number and size of the platters, rotation rate, interface type, data rate and access time all impact power consumption. For example, in a 2005 study of 3.5″ disks, the power consumption was found to vary from as much as ∼30 W during spin-up to 5 — 10 W during idle, with an average of approx. 10 W for intensive work loads [10]. More recently, HDD manufacturers have begun to produce more "eco-focused" drives in which the power consumption during read/write operations for a 3.5″ drive is on the order of 5 W and can be as low as ∼2 W for a 2.5″ drive. One strategy to reduce power in a large array of disks is to spin down unused disks, a strategy known as MAID (massive array of idle disks). However, this concept has not yet become very successful, most likely because of the large latency penalty and the reliability concerns associated with continually powering disks on and off.

4 Flash Technology and Storage Class Memory (SCM) Trends

Today's memory and storage hierarchy consists of embedded SRAM on the processor die and DRAM as main memory on one side, and HDDs for high-capacity storage on the other side. Flash memory, in the form of solid-state-disks (SSD), has recently gained a place in between DRAM and HDD, bridging the large gap in latency (∼10^5 times) and cost (∼100 times) between them. However, the use of Flash in applications with intense data traffic, i.e., as main memory or cache, is still hampered by the large performance gap in terms of latency between DRAM and Flash (still ∼1000 times), and the low endurance of Flash (10^4 Ű– 10^5 cycles), which deteriorates with scaling and more aggressive MLC functionality. MLC technology has become the focus of the Flash vendors because they target the huge consumer-electronics market, where the low cost per gigabyte of MLC plays a very important role, but it suffers from endurance and latency issues, which could be problematic for enterprise-class applications. For example, at the 32-nm technology node and 2 bits per cell, it is expected that the standard consumer-grade MLC will offer a write/erase endurance of approx. 3,000 cycles, which clearly will not suffice for enterprise-class storage applications. On the other hand, an enterprise-grade MLC with higher cost per gigabyte could offer a write/erase endurance of 10^4 to 3×10^4, albeit with a slower programming latency of approx. 1.6 ms. These limitations of the MLC technology necessitate the use of more complex error-correction coding (ECC) schemes and Flash management functions, which, depending on the workload, could improve the reliability and hide the latency issues to a certain extent — but certainly not to full satisfaction.

Moreover, as we go down on the technology node, these issues will be further aggravated, and new challenges will have to be resolved. For example, the stringent data-retention requirements, in particular for enterprise-storage systems, impose a practical limit for the thickness of the tunnel oxide. Another challenge in the scaling of floating-gate NAND is floating-gate interference. To resolve this issue, a charge-trapping layer has been proposed as an alternative technology to the floating gate [2]. In general, it was believed for a long time that by moving to charge-trapping storage it would be possible to scale at least to the 22-nm lithography generation. However, recently a very promising trend towards stacking memory cells in three dimensions in what is called 3D memory technology has emerged, and leading NAND Flash memory manufacturers are already pursuing it [11]. Of course, this 3D memory technology will not truly have an impact on reliability, endurance and latency, but it will offer much larger capacities at even lower cost in the future. For all these reasons, NAND Flash is not

expected to become an SCM technology in general.

Scaling issues are also critical for other solid-state memories, such as SRAM and DRAM. Specifically, SRAM suffers from signal-to-noise-ratio degradation and 10x leakage increase with every technology node, and DRAM faces a continuous increase of the refresh current.

Hence, there is a large opportunity for new solid-state nonvolatile memory technologies with "universal memory" characteristics. These technologies should not only extend the lifetime of existing memories, but also revolutionize the entire memory-storage hierarchy by bridging the gap between memory (fast, expensive, volatile) and storage (slow, inexpensive, permanent). The requirements of this new family of technologies called SCM [1] are nonvolatility, solid-state implementation (no moving parts), low write/read latency (tens to hundreds of nanoseconds), high endurance (more than 10^8 cycles), low cost per bit (i.e., between the cost per bit of DRAM and Flash), and scalability to future technology nodes.

Many new nonvolatile solid-state memory technologies have recently emerged. The objective has not only been to realize dense memory arrays and show a viable scalability roadmap, but also to achieve a performance superior to that of Flash memory in many aspects. The catalog of new technologies is very long, and they may be broadly categorized into charge-trap-based, capacitance-based and resistance-based memories. Charge-trap based memories are basically extensions of the current floating-gate-based Flash and, while offering advantages in reliability, suffer from the same drawbacks that afflict Flash technology, namely, low endurance and slow writing speeds. Capacitance-based memories, in particular ferroelectric memories (FeRAM), exhibit practically infinite cycling endurance and very fast read/write speeds, but are hampered by short retention times and, even more importantly, their limited scaling potential up to the 22-nm node [12].

Resistance-based memories encompass a very broad range of materials, switching mechanisms and associated devices. Following the International Technology Roadmap for Semiconductors (ITRS), one may categorize resistance-based memories into nanomechanical, spin torque transfer, nanothermal, nanoionic, electronic effects, macromolecular and molecular memories [12]. Of these technologies, those that have received more attention by the scientific community and the semiconductor industry and are thus in a more advanced state of research and/or development, are spin torque transfer, nanoionic and thermal memories. We will take a closer look at these technologies next.

Spin-torque transfer memory (STTRAM) [13]–[15] is an advanced version of the magnetic random access memory (MRAM) in which the switching mechanism is based on the magnetization change of a ferromagnetic layer induced by a spin-polarized current flowing through it. The most appealing features of STTRAM are its very high read/write speed, on the order of 10 ns or less, and its practically unlimited endurance. Important challenges are overcoming the small resistance range (between low and high resistance), which limits the possibility of MLC storage, and achieving adequate margins not only between read and write voltages but also between write and breakdown voltages for reliable operation, especially at high speeds.

Nanoionic memories [16]–[21] are characterized by a metal-insulator-metal (MIM) structure, in which the "metal" typically is a good electrical conductor (possibly even dissimilar on the two sides of the device) and the "insulator" consists of an ion-conducting material. Typical insulator materials reported so far include binary or ternary oxides, chalcogenides, metal sulfides, and even organic compounds. The basic switching mechanism in nanoionic memories is believed to be the combination of ionic transport and electrochemical redox reactions [19]. Most of these technologies—with very few exceptions—are still in a very early stage of research, with many of their interesting features derived from projections or extrapolations from limited experimental data. As is generally known, the actual issues associated with a particular technology will likely only manifest themselves in large demonstration device test vehicles, so that it may well be that the road to widespread adoption of nanoionic memories is still a long one.

The best known thermal memory is phase-change memory (PCM). This discussion focuses on PCM, mainly because of the very large activity around it and the advanced state of development it has reached, allowing credible projections regarding its ultimate potential. The active material in PCM is a chalcogenide, typically involving at least two of the materials Ge, Sb and Te, with the most common compound being $Ge_2Sb_2Te_5$

or, simply, GST. The active material is placed between two electrically conducting electrodes. The resistance switching is induced by the current flowing through the active material, which causes a structural change of the material due to Joule heating. Phase-change materials exhibit two meta-stable states, namely, a (poly)-crystalline phase of long-range order and high electrical conductivity and an amorphous phase of short-range order and low electrical conductivity. Switching to the amorphous phase (the RESET transition) is accomplished by heating the material above its melting temperature followed by ultra-fast quenching, whereas the crystalline phase (SET transition) is reached by heating the materials above its crystallization temperature and subsequent annealing. The RESET transition necessitates high current, but this current has been shown to scale linearly with the technology node as well as decrease significantly in confined memory cell architectures [22]. The RESET transition is fast, typically less than 50 ns in duration, whereas the SET transition is on the order of 100 ns, although very fast materials exhibiting sub-20-ns switching times have been reported [23].

PCM scores well in terms of most of the desirable attributes of a SCM technology. In particular, it exhibits very good endurance, typically exceeding 10^8 cycles, excellent retention, and superb scalability to sub-20-nm nodes and beyond. Most importantly, these characteristic numbers have been measured on large prototype devices and thus provide confidence regarding the true performance of the memory technology. On a smaller-scale device level, PCM has been shown to possess all the necessary characteristics of a SCM technology. Specifically, sub-20-ns SET switching times have been reported with doped SbTe materials [23]. Furthermore, an impressive device has been fabricated at the 17-nm design rule at $4F^2$ size, with further scaling prospects not limited by lithography but only by the material film thickness [24]. The same device showed an extrapolated cycling endurance exceeding 10^{15} cycles. The ultimate scaling limits of phase change in chalcogenide materials provide an indication regarding the future scaling of PCM. In a recent study, GST films that are a mere 2 nm thick have been shown to crystallize when surrounded by proper cladding layers [25].

Apart from the necessary RESET current reduction and SET speed improvement discussed above, a significant challenge of PCM technology is a phenomenon known as (short-term) resistance drift: The resistance of a cell is observed to drift upwards in time, with the amorphous and partially-amorphous states drifting more than their crystalline counterparts. This drift is believed to be of electronic nature, manifests itself as noise, and seriously affects the reliability of MLC storage in PCM because of the reduced sensing margin between adjacent tightly-packed resistance levels. Therefore, effective solutions of the drift issue are a key factor of the cost competitiveness of PCM technology and thus of its suitability as SCM.

In summary, PCM is the only one of numerous emerging memory technologies that has evolved from the basic research stage to the advanced development and late prototyping stage without encountering any fundamental roadblocks. Advanced prototype PCM chips that at least partially meet the requirements for SCM already exist today [26, 27], and new and exciting device demonstrations have shown tremendous potential for further improvement. These developments render PCM the leading technology candidate for SCM today, with the potential to play an extended role in the memory and storage hierarchy of future computing systems.

5 Tape Technology Trends

Tape systems constitute an integral part of current tiered storage infrastructures. They are especially suited for low-cost, long-term storage of data as well as for backup and disaster recovery purposes. Tape technology offers several important advantages, including energy savings, security, lifetime, reliability and cost. Moreover, in such applications, the main drawback of tape, its slow access time, does not have a major impact on the system., Once data has been recorded in tape systems, the medium is passive; it simply sits in a rack and no power is needed. Compared with similar disk-based systems, a tape-based archive consumes approximately 290 times less power [28]. In terms of security, once data has been recorded and the cartridge removed from the access system, the data is inaccessible until the cartridge is reinstalled in the active system. Security is further enhanced by drive-level encryption, which was introduced in Linear Tape Open generation-4 drives (LTO-4) and

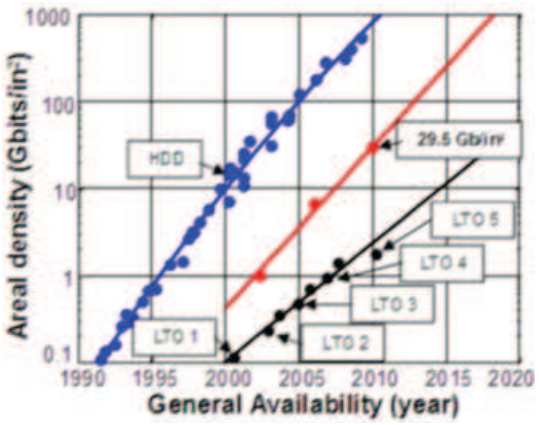


Figure 3: Figure 3. Recording of areal density of HDD and tape products and tape demonstrations. Adapted from [5].

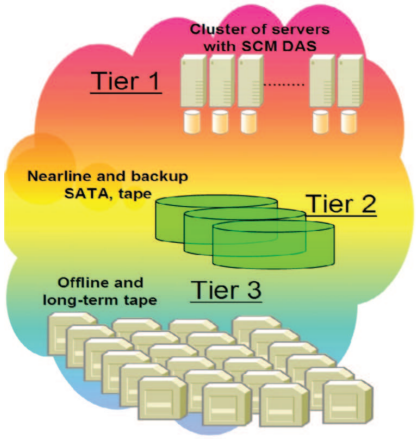


Figure 4: Tiered storage.

is also standard in enterprise-level tape drives. The tape medium has a lifetime of 30+ years; however, this is rarely taken advantage of because of the rapid advances in tape hardware and the cost savings associated with migration to higher-capacity cartridges. In terms of reliability, LTO-4 tape has a bit error rate that is at least an order of magnitude better than that of a SAS HDD [29]. Moreover, the fact that tape media is removable and interchangeable, means that, in contrast to HDDs, mechanical failure of a tape drive does not lead to data loss because a cartridge can simply be mounted in another drive. All of the above advantages contribute to the major net advantages of tape, which are cost and reliability. Estimates of the savings of the total cost of ownership of a tape backup system, relative to HDD backup, range from a factor of three to more than 20, even if developments such as data deduplication are taken into account. In archival applications, where no effective use of deduplication is possible, cost savings can be even higher.

Figure 3 compares the evolution of the areal densities of HDD and tape products over time, including recent tape areal density demonstrations. The plot indicates that even though the gap between the areal densities of HDD and tape products has remained essentially constant in recent years, tape areal density demonstrations exhibit a slope of about 60% CAGR, indicating the potential of reducing the areal density gap between tape and HDD. Insight into how this was achieved can be gained by comparing the bit aspect ratios (BAR) of tape drives and HDDs. The typical BAR (bit width to bit length) in recent HDD technology demonstrations is on the order of 6:1 [30] and, as mentioned in Section 3, currently there are HDD products on the market that operate at linear densities of more than 700 Gb/in^2. In contrast, the latest LTO-5 tape drives operate at an areal density of 1.2 Gb/in^2 with a BAR of about 123:1 [31]. This comparison indicates that there is considerable room to reduce the track width in tape systems. The potential to continue to scale tape areal densities was recently experimentally confirmed in a world record tape areal density recording demonstration of 29.5 Gb/in^2, performed jointly by IBM and Fujifilm [4]. What is clear from this density demonstration is that there is ample room to continue scaling the areal density and cartridge capacity of tape systems for at least the next 10 years. Moreover, a recent study shows that further improvements in technology may increase this density to 100 Gb/in^2, indicating the potential to continue scaling tape for many years to come without a fundamental change of the tape-recording paradigm [5]. Finally, considering that the areal density of HDDs is currently scaling at a CAGR of $\sim$30% and that tape has the potential to continue scaling at 40% or more, the cost advantages of tape over HDD may become even greater.

One of the obstacles to more widespread adoption of tape in the past has been the difficulty of using tape in

a general or stand-alone context. Hard disks provide random access to data and generally contain a file index managed by a file system. These files can be accessed by means of standard sets of application programming interfaces (APIs) using various operating systems and applications. Tape, in contrast, is written in a linear sequential fashion typically using a technique called “shingling" which provides backward write compatibility, but also implies that new data can only be appended and that previously written areas can only be reclaimed if the entire cartridge is reclaimed and rewritten. In traditional tape systems, an index of the files written on a given cartridge is usually only kept in an external database managed by an application such as a proprietary back-up application. The need to access an external database to retrieve data renders data on tape much less portable and accessible than with alternative storage methods, such as a HDD or a USB drive.

To address these deficiencies, a new long-term file system (LTFS) has recently been introduced into the LTO-5 tape-drive systems to enable efficient access to tape using standard and familiar system tools and interfaces [32]. LTFS is implemented using the dual-partition capabilities supported in the new LTO-5 format. A so-called index partition is used for writing the index, and the second, much larger partition for the data itself. This new file system makes files and directories show up on the desktop with a directory listing. Users can “drag and drop” files to and from tape and can run applications developed for disk systems. In library mode, the content of all volumes in the library can be listed and searched without mounting the individual cartridges. All these features help reduce tape, file management and archive costs and eliminate the dependency on a middleware layer. Hence the cost per GB stored is reduced. In addition, tape becomes cross-platform-portable (Linux*, Mac*, Windows*), enabling and facilitating the sharing of data between platforms. These features enable significant new use cases for tape, such as video archives, medical images, etc.

Considering the cost advantages of tape over other storage solutions, the demonstrated potential for the continued scaling of tape-cartridge capacity and cost per GB as well as the increasing usability of tape provided by advances such as the LTFS, tape appears set to play an important role in the exploding market for archival data storage solutions.

6 Implications of SCM and Tape on Tiered Storage Hierarchy Ű– Conclusion

Figure 4 shows how SCM and tape will affect the tiered storage hierarchy. The advent of SCM as well as the continuous density scaling of the low-cost tape technology will impact the storage tiering hierarchy, fitting the workloads envisaged in future applications. Specifically, it is expected that, for the on-line tier, the direct-attached storage (DAS) model will resurface in the form of server clusters with low-latency SCM storage in each server. This will provide the high performance required by IO-limited workloads by leveraging the huge internal bandwidth necessary for streaming analytics, for instance. The co-location of computation workload and corresponding data sets will in turn reduce the need for caching. The current benefits in provisioning and utilization of virtualized SAN-attached storage will need to be leveraged in the new DAS model as well.

For the near-line tier, both low-cost SATA disks and tape will compete to support backup and active archive applications, whereas for the off-line tier, which primarily targets archiving applications, power-efficient high-capacity, low-cost tape will be the greenest technology and the ultimate insurance policy for the enterprise.

*Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both. IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both. Other product and service names might be trademarks of IBM or other companies.

References

[1] R. F. Freitas, W. W. Wilcke. Storage-Class Memory: The Next Storage System Technology. *IBM J. Res. Develop.* 52(4/5), 439–447 (2008).

[2] G. W. Burr *et al.* Overview of Candidate Device Technologies for Storage-Class Memory. *IBM J. Res. Develop.* 52(4/5), 449–464 (2008).

[3] 2010 Digital Universe Study: A Digital Universe Decade Ű Are You Ready? http://gigaom.files.wordpress.com/2010/05/2010-digital-universe-iview_5-4-10.pdf

[4] G. Cherubini *et al.* 29.5 Gb/in^2 Recording Areal Density on Barium Ferrite Tape. *IEEE Trans. Magn.* 47(1) (January 2011, in press) DOI: 10.1109/TMAG.2010.2076797; also in *Digest of Magnetic Recording Conf. "TMRC,"* San Diego, CA, pp. 29–30 (August 2010).

[5] A. J. Argumedo *et al.* Scaling Tape-Recording Areal Densities to 100 Gb/in^2. *IBM J. Res. Develop.* 52(4/5), 513-527 (2008).

[6] http://www.netezza.com/documents/whitepapers/streaming analytic white paper.pdf

[7] Active Archive Alliance whitepapers: http://www.activearchive.com/common/pdf/TimeValue WP US 4229.pdf http://www.activearchive.com/common/pdf/ActiveArchiveWPInterSect360201004.pdf

[8] Naoki Asakawa. TDK Achieves World's Highest Surface Recording Density of HDD. *Nikkei Electronics*, Sept. 30, 2008, http://techon.nikkeibp.co.jp/english/NEWS EN/20080930/158806/

[9] http://www.conceivablytech.com/2106/products/hard-drives-get-new-record-density-where-is-the-limit/

[10] HDD Diet: Power Consumption and Heat Dissipation. http://ixbtlabs.com/articles2/storage/hddpower.html

[11] M. Kimura. 3D Cells Make Terabit NAND Flash Possible. *Nikkei Electronics Asia*, Sept. 17, 2009.

[12] *International Technology Roadmap for Semiconductors, Emerging Research Devices*, 2009 edition.

[13] J-G. Zhu. Magnetoresistive Random Access Memory: The Path to Competitiveness and Scalability. *Proc. IEEE* 96, 1786–1797 (2008).

[14] T. Kishi *et al.* Lower-Current and Fast Switching of a Perpendicular TMR for High Speed and High Density Spin-Transfer-Torque MRAM. In *Proc. IEDM 2008*, pp. 1–4 (2008).

[15] M. Hosomi *et al.* A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin RAM. In *2005 IEDM Technical Digest*, p. 459 (2005).

[16] A. Asamitsu, Y. Tomioka, H. Kuwahara, Y. Tokura. Current-Switching of Resistive States in Colossal Magnetoresistive Oxides. *Nature* 388, 50–52 (1997).

[17] M. N. Kozicki, M. Yun, L. Hilt, A. Singh. Applications of Programmable Resistance Changes in Metal-Doped Chalconides. *Electrochem. Soc. Proc.* 99-13, 298-309 (1999).

[18] A. Beck, J. G. Bednorz, C. Gerber, C. Rossel, D. Widmer. Reproducible Switching Effect in Thin Oxide Films for Memory Applications. *Appl. Phys. Lett.* 77(1), 139–141 (2000).

[19] R. Waser, M. Aono. Nanoionics-based Resistive Switching Memories. *Nature Materials* 6, 833–840 (2007).

[20] R. Meyer, et al. Oxide Dual-Layer Memory Element for Scalable Non-Volatile Cross-Point Memory Technology. In *Proc. NVMTS 2008*, pp. 54-58 (2008).

[21] J. Borghetti *et al.* "Memristive" Switches Enable 'Stateful' Logic Operations via Material Implication. *Nature* 464(8), 873–876 (2010).

[22] D. H. Im *et al.* A Unified 7.5nm Dash-Type Confined Cell for High Performance PRAM Device. In *Proc. Int'l. Electron Devices Meeting (IEDM) 2008*, San Francisco, CA, pp. 211–214 (2008).

[23] B. Cheong *et al.* Characteristics of Phase Change Memory Devices Based on Ge-doped SbTe and its Derivative. In *Proc. European Phase Change and Ovonics Symposium (E*PCOS)* (2007).

[24] I. S. Kim. High Performance PRAM Cell Scalable to Sub-20nm Technology with below 4F^2 Cell Size, Extendable to DRAM Applications. In *2010 Symp. on VLSI Technology Digest of Technical Papers*, pp. 203–204 (2010).

[25] R. E. Simpson *et al.* Toward the Ultimate Limit of Phase Change in $Ge_2Sb_2Te_5$. *Nano Lett.* 10(2), 414–419 (2010).

[26] F. Bedeschi *et al.* A Bipolar-Selected Phase Change Memory Featuring Multi-Level Cell Storage. *IEEE J. Solid-State Circuits* 44(1), 217–227 (2009).

[27] K.-J. Lee *et al.* A 90 nm 1.8 V 512 Mb Diode-Switch PRAM with 266 MB/s Read Throughput. *IEEE J. Solid-State Circuits* 43(1), 150–161 (2008).

[28] D. Reine, M. Kahn. Disk and Tape Square Off Again — Tape Remains King of the Hill with LTO-4. *Clipper Notes* (2008). www.clipper.com/research/TCG2008009.pdf

[29] H. Newman. The Tape Advantage: Benefits of Tape over Disk in Storage Applications. White paper, Instrumental (2008).

[30] Z. Bandic, R. H. Victora. Advances in Magnetic Data Storage Technologies. *Proc. IEEE* 96(11), 1749–1753 (2008).

[31] https://www.bluestoragemedia.com/External/BigBlueBytes/LTO5

[32] D. Pease *et al.* The Linear Tape File System. In *Proc. 2010 IEEE 26th Symp. on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, pp. 1–8 (2010).

[33] R. Freitas, W. Wilcke, B. Kurdi. Storage Class Memory Technology and Use. *Proc. FAST '08*, 26–29, (2008).

Implementing an Append-Only Interface for Semiconductor Storage

Colin W. Reid, Philip A. Bernstein
Microsoft Corporation
Redmond, WA, 98052-6399
{philbe, colinre}@microsoft.com

Abstract

Solid-state disks are currently based on NAND flash and expose a standard disk interface. To accommodate limitations of the medium, solid-state disk implementations avoid rewriting data in place, instead exposing a logical remapping of the physical storage. We present an alternative way to use flash storage, where an "append"İ interface is exposed directly to software. We motivate how an append interface could be used by a database system. We describe bit rot and space reclamation in direct-attached log-structured storage and give details of how to implement the interface in a custom controller. We then show how to make append operations idempotent, including how to create a fuzzy pointer to a page that has not yet been appended (and therefore whose address is not yet known), how to detect holes in the append sequence, and how to scale out read operations.

1 The Case for a Log-Structured Database

Most database systems store data persistently in two representations, a database and a log. These representations are based in large part on the behavior of hard disk drives, namely, that they are much faster at reading and writing pages sequentially than reading and writing them randomly. The database stores related pages contiguously so they can be read quickly to process a query. Writes are written sequentially to the log in the order they execute to enable high transaction rates.

Inexpensive, high-performance, non-volatile semiconductor storage, notably NAND flash, has very different behavior than hard disks. This makes it worthwhile to consider other storage interfaces than that of hard disks and other data representations than the combination of a database and log. Unlike hard disks, semiconductor storage is not much faster at sequential operations than at random operations, so there is little performance benefit from laying data out on contiguous pages. However, semiconductor storage offers enormously more random read and write operations per second per gigabyte (GB) than hard drives. For example, a single 4GB flash chip can perform on the order of 10,000 random 4KB reads per second or 5,000 random writes per second. So, in a shared storage server cluster with adequate I/O bandwidth, 1TB of flash can support several million random reads per second and could process those requests in parallel across physical chips. This compares to about 200 random reads for a terabyte of hard drive capacity. However, per gigabyte, the raw flash chips are more than ten times the price of hard drives.

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Given the large number of random reads that are available with flash storage, the main benefit of storing data in two representations disappears. However, if data is to be stored one way, which representation is better, a database that is updated in place or a log? We advocate a log, for several reasons.

Flash has two properties that encourage writing sequentially. The first property is that a flash page cannot be updated in-place. It must be erased before it can be over-written (or "programmed" in flash terminology). In typical NAND flash chips the erase operation wipes dozens or hundreds of pages at a time, and blocks other operations on a large portion of the chip for about 2 ms. Therefore, it is best to erase pages within storage regions that will not be needed for other latency-sensitive operations, such as a random read to service a database cache miss. Second, erase operations cause flash to wear out. Flash can be erased a limited number of times, roughly 10,000 to 100,000 times depending on the technology being used. Thus, flash needs "wear-leveling"İ to ensure all blocks are erased at about the same rate.

The problems of slow erase cycles and wear leveling are solved by the use of log-structured storage. In this design, an update is appended after the last written page of storage. To recycle storage capacity, reachable data on the first non-empty chip C is collected and appended to the updatable end of storage so that C can be erased. This design minimizes the impact of slow erase cycles by ensuring that a chip is erased only when it has no data that is worth reading from it. It also solves the problem of wear leveling because all chips are erased the same number of times.

Another benefit of log-structured storage arises when parallel processors need to synchronize their random updates to storage. The overhead of this synchronization is relatively expensive when the update itself is cheap, as it is with flash memory. Log-structured storage can offer a self-synchronizing, atomic append operation that allows many processors to independently store data at will, with no need for another synchronization process. One problem with log-structured disk-based storage is that it is hard to maintain physical contiguity of related data. This is important because sequential reads are faster than random reads. But this property does not hold for flash. In fact, when using many flash chips as components of a storage system, optimal response times are achieved when related data is spread across multiple chips and retrieved in parallel.

2 The Hyder System

Given these benefits of log-structured storage, we have designed a new database system, called Hyder, which uses an append-only log as its one and only storage medium. It is a data-sharing system, where many servers read from and write to shared storage. Its main feature is that it scales out without partitioning the database or transactions. We briefly sketch Hyder's use of log-structured storage here, as motivation for the mechanisms that support it in the rest of this paper. The architecture is described in detail in [1].

In Hyder, each transaction T executes in the normal way on some server S, reading and writing shared data items. Hyder uses multiversion storage, so T executes against the latest committed database state D known to S, which is defined by the last committed transaction in the log that contributed to D. When T writes a data item, its update is stored in a cache that is private to T. When T has finished computing, S gathers the after-images of T's updates in a log record, L. L also contains a pointer P to the earlier log record that identifies D (the database state that T read). S appends L to the log and broadcasts it to all other servers. The shared log defines a total order of all log records. Thus, all servers have a copy of the log consisting of the same sequence of log records.

When a server S′ receives L, it needs to determine whether T committed. Conceptually speaking, it does this by checking whether there is a log record in between P and L that contains a committed transaction's update that conflicts with T, where the definition of conflict depends on T's isolation level. (This is only conceptual; the actual algorithm works differently and is much more efficient.) If there is no such conflicting committed update, then T has committed, so S′ incorporates T's updates into its cached partial-copy of the last-committed database state. Otherwise, T has aborted, so S′ ignores T's updates . If S′ = S (i.e., S′ is where T originally executed) then it informs T's caller of T's outcome.

Since all servers see the same log, for every transaction T they all make the same decision whether to commit or abort T, without any server-to-server communication. The only point of arbitration between servers is the operation to append a record to the log, which occurs only once per multi-step transaction.

Notice that a server must continually roll forward the log in order to have a cached copy of a recent database state against which it can run newly-submitted transactions. Since log records describe both committed and aborted transactions, it is only through the roll forward process that a server knows which are which and hence knows the content of the last-committed state.

Hyder was partly inspired by work of Dave Patterson and Jim Gray that latency lags bandwidth [4] and bandwidth lags capacity [2]. While physical laws limit latency to light speed, bandwidth can continue to grow, and raw storage capacity is increasingly cheap. So eventually, the capacity cost of append-only storage will decrease to the point of irrelevance, the bandwidth cost of sending information about all transactions to all servers will be less significant, and the latency benefit of a single synchronization per transaction will become more and more obvious. These trends are as relevant to multicore handheld devices as to large clusters, and they can be harnessed at both extremes with a simple abstraction for log-structured storage.

3 A Log-structured Storage Abstraction for Reliable Interconnects

Direct-attached solid-state storage, as in a phone or standalone server, is typically very reliable and can be accessed without message loss. A log-structured storage abstraction for such devices requires only a few trivial operations. The Append operation takes a page of data as an argument, durably appends it to storage, and returns the address at which it was stored. Other operations are GetLastPage (which returns the last page written by an Append), GetSpecificPage (which takes a page-frame address as a parameter), and Erase (which erases all of the page frames within a given storage unit, such as a flash chip).

A small device with direct-attached storage and a reliable interconnect is likely to have only a single storage controller. The controller can track a small amount of state, such as the address of the last appended page, in volatile memory. On power failure, the in-memory state can be consistently flushed to non-volatile storage before the device stops functioning. This greatly simplifies the implementation of the GetLastPage operation, avoiding a more expensive search for the last page during restart.

3.1 Bit Rot

The volatility of semiconductor storage is a continuum, not a binary distinction. “Non-volatile" memories such as NAND flash and phase-change memory do actually degrade over time, and “volatile" memories like DRAM do actually store data without power between their refresh cycles. In all these memories, data needs to be refreshed or rewritten occasionally to prevent it from degrading.

In high availability systems with large amounts of long-lived data, it is usually necessary to scrub storage periodically in order to detect partial failures before they become unrecoverable. In other words, modern semiconductor storage devices periodically read data, and then rewrite it, even if it is static and only rarely accessed by an application. This can be done blindly, by continually scrubbing and refreshing all storage capacity regardless of whether it contains interesting information, or more intelligently, by processing only the storage containing data that is actually useful.

The latter option requires some interaction with higher layers of the system in order to determine which data is still useful. It provides an important opportunity to consolidate, defragment, compress and otherwise optimize the stored representation of important data. It is also an opportunity to free up storage capacity that is no longer useful.

3.2 Space Reclamation

The major downside of append-only storage is that it fills up. Since semiconductor storage capacity is still a fairly valuable resource, a useful system must be able to reclaim space and reuse it to store new information.

The task is very much like garbage collection in memory-managed programming environments like .NET and Java. There are essentially two types of garbage collectors in those types of environments: mark-sweep collectors and copying collectors [3].

In a sense, block storage systems that target disks today are using a form of mark-sweep collection. Applications allocate and release blocks of storage, and a file system or database storage manager marks pages and attempts to allocate blocks of storage on demand from the free pool.

Instead, it is possible to reclaim storage using a copying collector. In this scheme, which has well-understood benefits and tradeoffs in the managed-memory world, reachable data from a sparsely populated region of memory is simply copied densely to an empty region. Append-only storage is well-suited to this approach, particularly if the erase-block size of the device is relatively large, as it is in NAND flash. Copying collection frees large blocks of storage that can be efficiently erased. It also has the advantage of automatically scrubbing and refreshing storage to mitigate bit rot. Reachable information from an old segment of the log is simply re-appended by the application to a new one, and the old segment is then erased and recycled. This requires a small logical-to-physical remapping table of log segments, but is otherwise very straightforward. Such practices are common in log-structured file systems today [5].

This approach can result in unnecessary copying if very large contiguous sections of the log contain long-lived static information. To address that, applications can maintain two logs: one for data that has been copied once, and one for new data. This is similar to the generational garbage collection that is commonly used in managed-memory runtimes.

4 Log-structured Shared Storage over Unreliable Interconnects

The same four Append, GetLastPage, GetSpecificPage, and Erase operations can support log-structured storage in locally distributed compute clusters. As long as the storage devices are able to continue serving their stored pages reliably, transient errors such as lost messages can be accommodated by slightly augmenting Append.

4.1 Making Append Idempotent

The Append operation stores a page of data and returns the address at which it was stored. By not specifying the address at which to write the page, servers do not have to coordinate the regions of storage that they intend to modify. This "just-in-time synchronization" is useful in any parallel environment, regardless of the interconnect protocol's reliability, and the storage unit thus provides a dense log of append operations.

A problem arises if a caller (i.e., a server) sends an Append operation to a controller and does not receive a reply, either because the operation did not execute or because the reply message got lost. In the former case, the caller should re-execute the Append. In the latter case, the caller needs to get another copy of the reply. The problem is that it does not know which to do. It needs help from the storage controller to take the appropriate action, such as to get another copy of the reply.

To solve this problem, we design Append to be idempotent, which means that executing an Append operation multiple times has the same effect as executing it just once. Each page that is sent to the controller has a unique key. After the controller processes an Append and before it returns the appended page's address to the caller, it stores the page's key and storage address in a table that is local to the controller. We call it the Append Result table. The controller uses the Append Result table to cope with duplicate Append operations. When the controller receives an Append operation, it first checks to see whether the key of the page to be appended is in its Append Result table. If so, it replies with the address in the Append Result table and does not re-execute

the Append. If not, then it executes the Append operation, which is safe to do since it has not executed the operation previously. Therefore, if a server does not receive a reply to an Append, it is safe for it to resend the same Append operation to the controller.

The key must be generated in such a way that there is negligible probability that two different pages that are received by the controller within a certain time period have the same key. The time period is the amount of time it takes to execute N Append operations, where N is the number of entries in the Append Result table. This ensures that the controller will not be fooled into thinking that two different pages are the same because they have the same key. For example, the key could be a 128-byte globally-unique identifier, commonly known as a GUID, a cryptographic hash of the appended state, or a sequence number. Or, if the storage controller caches payloads of recently appended pages anyway, the key could even be the entire content of the page.

Since Append is idempotent, a server can increase the likelihood that the operation will execute successfully by sending it multiple times. For example, if there are two communication links that connect the server to the storage controller, then the server can eagerly send the Append on both links. This has less latency than retrying after a timeout, since the server can proceed based on the reply from whichever operation executes first. It also decreases the chances that the server will fail to receive a reply, since both Append operations would have to malfunction for that to occur.

A problem with the above technique for making Append idempotent is that the Append Result table has a fixed size. New entries in the Append Result table need to overwrite the oldest ones. Therefore, if a server resends an Append a long time after sending its initial Append, then it is possible that the result of the first Append was overwritten in the Append Result table before the storage controller receives the second Append. In this case, the controller would incorrectly conclude that the second Append did not previously execute and it would execute the operation again.

4.2 Bounding the Storage Address

We solve this problem by including a “concurrency bound" parameter to the Append operation. The concurrency bound defines a range of pages where the page must be appended. If the first page in the range is not in the Append Result table, then the operation returns an exception, because it cannot tell whether the Append is a duplicate. The operation also returns an exception if there are no erased pages in this range, which means that all of the requested page frames are full and cannot be written by the Append. Since storage is append-only, the controller can implement this simply by comparing the address of the last page frame it wrote (which is in the Append Result table) to the top of the concurrency bound.

A good way express the concurrency bound is as a page-frame address and an extent. The page-frame address is the next page frame address beyond the last written page frame seen by the caller (e.g., in the reply to a previous Append). The extent is the number of page frames in the concurrency bound.

A concurrency bound can be used to store a soft pointer to a page whose address is not yet known. For example, consider a more reliable distributed storage architecture where a multi-page stripe $\{P_1, , P_n\}$ needs to be stored atomically on several log-structured storage devices. One way to do this is, after the pages have been appended, to store a commit recordİ C that contains pointers to those pages. For recovery purposes, it is helpful to have a pointer from each P_i to C to make it easy to determine whether the set of pages that included P_i was committed and which other pages are part of its atomically-written set. However, since C's location can only be known after it is appended to the store, its location cannot be known before P_i is written. Instead, a concurrency bound for C can be stored in each P_i, which limits the search required to find C.

4.3 Broadcast

In some applications of log-structured storage, it is useful to have Append operations and their results be made known to all servers. This is important in Hyder, where all servers need to know the content of the tail of the

log. It may be useful in other applications that need to maintain coherent server caches.

A server can easily make its Appends known to other servers by broadcasting the Append to other servers at the same time it sends the Append to the storage controller. The other servers also need to know the storage address where the appended page is applied. That information is only known after the controller executes the Append and returns the address to the caller. The caller could broadcast that result to the same servers to which it sent the Append. Although this works, it is inefficient because it introduces extra delay, since the server must wait until it receives the result from the controller and then send the address to other servers.

A better solution is to have the controller broadcast the result of the Append to all servers. This avoids the extra message delay of first sending the result to the server that issued the Append and then having the server broadcast it to other servers. It is especially efficient if the network that connects the servers to the shared storage controller has a built-in broadcast capability.

Broadcast is realistic over common connectionless protocols like UDP, because a 4KB flash page fits comfortably in an Ethernet jumbo frame. (However, this is not true across public networks where the maximum transfer unit is typically 1.4KB.) Since pages are totally ordered by their storage addresses, we do not need TCP to help maintain an ordered session to storage. We can therefore take advantage of the network's broadcast capabilities.

4.4 Filtering Broadcasts

It is frequently unnecessary for all storage operations to be broadcast to all servers. In this case, it is beneficial to include a parameter to the Append that tells the controller whether or not to broadcast the result to all servers. The server performing the Append determines whether a page is worth broadcasting, and if so it indicates this in the broadcast parameter and uses the broadcast facility to transmit the message. Otherwise the page is sent to storage via a unicast mechanism, and the result is unicast back to only the originating server. In the Hyder system, the broadcast parameter is used to differentiate large transaction-specific values from the transaction metadata. Alternatively, the response to a unicast append can still be broadcast, in order to make it easier for other servers to monitor the state of the log.

4.5 Detecting Holes

Replaying the log occurs in two ways. At startup, a server uses the GetLastPage operation to retrieve a page from the log. Every page can contain application information about checkpoints or other interesting log positions from which to replay, and the server can use that information to start requesting each page in sequence beginning at some checkpoint. As discussed below, this process can be heavily optimized.

At some point during replay the server will reach the end of the stored log. However, if the rest of the system is still appending, the log will continue to grow. During replay, when the server encounters its first end-of-log response to a GetPage operation, it begins monitoring for broadcasts from other servers and from the log itself.

As with any other broadcast communication, this introduces the possibility of missing a logged page. A server that has caught upİ and is replaying the log from the live broadcasts will normally need to detect these holes in the log. This is trivial when all pages are broadcast to all servers, because there will be a gap in the sequence of page numbers. The replaying server can simply request the missed page from storage.

However, broadcast filtering makes hole-detection more difficult. The replaying server needs to determine whether the gap in the page number sequence is due to a missed message or simply because the page was never intended to be broadcast.

One way to tell the difference is to have the storage controller track the highest stored page that was flagged for broadcast. It can then include that value in its response to the next Append that is flagged for broadcast, thus producing a linked list of broadcast messages. Whenever a server receives a broadcast from the controller, it

checks this value to ensure that all broadcasts have been received. If it detects a hole, it must replay part of the log to fill in the hole.

4.6 Speed Reading

Reading from a log can be scaled out arbitrarily, because the content of an append-only log is by definition immutable, and therefore the content can be shared, copied and cached without concern for cache coherence. A page number is a unique identifier for the content of a particular page, and the page is permanently identified by that number. So any number of servers can capture a mapping of page numbers to pages, in any form that they find useful, and can keep their cache relatively up-to-date by monitoring broadcasts. Even application-specific functions over the data in the cached pages can in turn be cached and invalidated efficiently.

One convenient way to cache the log is to index the pages that were marked for broadcast, and to keep the payloads of recent such pages in memory. In this way it becomes possible for a server to cheaply enumerate the broadcast pages in some recent region of the log. Such an index provides an excellent way to speed up log replay: when a server encounters a hole in the log, it simply asks any neighboring server to provide the missing information. This avoids having to request each page from the storage controller. The neighboring server can normally respond to such requests instantly, with a dense representation of the missing pages that had been originally intended for broadcast. This same facility can be used when replaying the log after catching up, or when recovering after a restart.

Likewise, it is trivial to implement a distributed hash table that caches frequently accessed log pages. When a server needs to fetch a page to perform a computation, it first requests it from one or two of the neighbors that are responsible for caching pages in the appropriate hash bucket. This allows a large system based on appendable storage to scale out indefinitely for read-only operations.

4.7 Conclusion

By briefly summarizing the kinds of mechanisms and data structures designed for Hyder, our goal was to show that interesting systems can be built over an append-only interface for semiconductor storage. In fact, such systems can run much more efficiently on modern storage devices if the devices stop trying to pretend they are disks. They fundamentally are not, and the impact on the storage stack is as significant as that last major storage hardware shift, from tape to disk.

References

[1] Bernstein, Philip A., Colin W. Reid, and Sudipto Das: Hyder – A Transactional Record Manager for Shared Flash. 5th Conf. on Innovative Database Systems (CIDR 2011), to appear.

[2] Gray, Jim: Long Term Storage Trends and You, research.microsoft.com/en-us/um/people/gray/talks/io_talk_2006.ppt

[3] Jones, Richard and Rafael Lins. Garbage Collection: Algorithms for Automatic Dynamic Memory Management. John Wiley and Sons, July 1996.

[4] Patterson, David A.: Latency Lags Bandwith. Commun. ACM 47(10): 71-75 (2004)

[5] Rosenblum, Mendel and John K. Ousterhout: The Design and Implementation of a Log-Structured File System. ACM Trans. Computer Systems 10(1), 26-52 (1992).

Designing Database Operators for Flash-enabled Memory Hierarchies

Goetz Graefe[‡] Stavros Harizopoulos[‡] Harumi Kuno[‡]
Mehul A. Shah[‡] Dimitris Tsirogiannis[§] Janet L. Wiener

[‡]Hewlett-Packard Laboratories, Palo Alto, CA, USA
{firstname.lastname}@hp.com

[§]Microsoft, Madison, WI, USA
dimitsir@microsoft.com

Abstract

Flash memory affects not only storage options but also query processing. In this paper, we analyze the use of flash memory for database query processing, including algorithms that combine flash memory and traditional disk drives. We first focus on flash-resident databases and present data structures and algorithms that leverage the fast random reads of flash to speed up selection, projection, and join operations. FlashScan and FlashJoin are two such algorithms that leverage a column-based layout to significantly reduce memory and I/O requirements. Experiments with Postgres and an enterprise SSD drive show improved query runtimes by up to 6x for queries ranging from simple relational scans and joins to full TPC-H queries. In the second part of the paper, we use external merge sort as a prototypical query execution algorithm to demonstrate that the most advantageous external sort algorithms combine flash memory and traditional disk, exploiting the fast access latency of flash memory as well as the fast transfer bandwidth and inexpensive capacity of traditional disks. Looking forward, database query processing in a three-level memory hierarchy of RAM, flash memory, and traditional disk can be generalized to any number of levels that future hardware may feature.

1 Introduction

As flash memory improves in price, capacity, reliability, durability, and performance, server installations increasingly include flash-based solid state drives (SSDs). However, the transition to a memory hierarchy that includes SSDs requires us to reexamine database design decisions. SSDs perform random reads more than 100x faster than traditional magnetic hard disks, while offering comparable (often 2-3x higher) sequential read and write bandwidth. Database workloads may not benefit immediately from SSDs, however, because databases have traditionally been designed so as to avoid random I/O. Instead, traditional data structures, algorithms and tuning parameters favor sequential accesses, which are orders of magnitude faster than random acccesses on hard disk drives (HDDs).

The present paper analyzes the use of flash memory for database query processing, including algorithms that combine flash memory and traditional disk drives. We review relevant prior work in Section 2. We consider how to leverage flash for selection, projection, and join operations in Section 3, and propose new techniques for sorting in a multi-level memory hierarchy in Section 4. Finally, we offer conclusions in Section 5.

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Prior Work

Work addressing SSDs has thus far focused on quantifying instant benefits from the fast random reads of SSDs [11, 15] and addressing their slow random writes [10, 12, 16, 19]. Our focus is unique in that we investigate query processing techniques that use flash to improve the performance of complex data analysis queries, which are typical in business intelligence (BI) and data warehousing workloads.

Several recent studies have measured the read and write performance of flash SSDs [2, 15, 18]. uFLIP [2] defines a benchmark for measuring sequential and random read and write performance and presents results for 11 different devices. Of particular note, they identify a "startup phase" where random writes may be cheaper on a clean SSD (since no blocks need to be erased) but quickly degrade as the disk fills. Polte et al. perform a similar study using less sophisticated benchmarks, but focus filesystems running on SSDs. They show the degraded mode is 3-7X worse than the "startup" phase but still an order of magnitude faster than current HDDs [18].

Graefe [7] reconsiders the trade-off between keeping data in RAM and retrieving it as needed from non-volative storage in the context of a three-level memory hierarchy in which flash memory is positioned between RAM and disk. The cited paper recommends disk pages of 256KB and flash pages of 4KB to maximize B-tree utility per I/O time. Interestingly, these page sizes derive retention times of about 5 minutes, reinforcing the various forms of the "five-minute rule".

Both Myers [15] and Lee et al. [11] measure the performance of unmodified database algorithms when the underlying storage is a flash SSD. Myers considers B-tree search, hash join, index-nested-loops join, and sort-merge join, while Lee et al. focus on using the flash SSD for logging, sorting, and joins. Both conclude that using flash SSDs provides better performance than using hard disks.

A number of database algorithms have been designed especially for flash characteristics. These generally emphasize random reads and avoid random writes (whereas traditional algorithms try to avoid any random I/O). Lee et al. [10] modify database page layout to make writing and logging more efficient. Clementsen and He [3] improve I/O performance with a fully vertically partitioned storage structure that stores each column separately on either the HDD or SSD. Ross [19] proposes new algorithms for counting, linked lists, and B-trees that minimize writes. Nath and Gibbons [16] define a new data structure, the B-file, for maintaining large samples of tables dynamically. Their algorithm writes only completely new pages to the flash store. They observe that writes of pages to different blocks may be interleaved efficiently on flash SSDs. Li et al. [12] use a B-tree in memory to absorb writes, together with several levels of sorted runs of the data underneath the tree. This structure uses only sequential writes to periodically merge the runs and random reads to traverse the levels during a search. Finally, Koltsidas and Viglas [9] consider hybrid SSD/HDD configurations of databases systems and design a buffer manager that dynamically decides whether to store each page on a flash SSD or a hard disk, based on its read and write access patterns. However, they assume all page accesses are random.

3 Fast Scans and Joins for SSD-only Databases

In the context of workloads that include complex queries, we modify traditional query processing techniques to leverage the fast random reads of flash SSDs. We have three goals: (1) avoid reading unnecessary attributes during scan selections and projections, (2) reduce I/O requirements during join computations by minimizing passes over participating tables, and (3) minimize the I/O needed to fetch attributes for the query result (or any intermediate node in the query plan) by performing the fetch operation as late as possible.

We advocate a column-based layout such as PAX [1] within each database page. We used a PAX-based page layout to implement *FlashScan*, a scan operator that reads from the SSD only those attributes that participate in a query. FlashScan proactively evaluates predicates before fetching additional attributes from a given row, further reducing the amount of data read.

Building on FlashScan's efficient extraction of needed attributes, we introduce *FlashJoin*, a general pipelined

join algorithm that minimizes accesses to relation pages by retrieving only required attributes, as late as possible. FlashJoin consists of a binary join kernel and a separate fetch kernel. Multiway joins are implemented as a series of two-way pipelined joins. Our current implementation uses a hybrid-hash join algorithm [20]; any other join algorithm may be used instead. Subsequently, FlashJoin's fetch kernel retrieves the attributes for later nodes in the query plan as they are needed, using different fetching algorithms depending on the join selectivity and available memory. FlashJoin significantly reduces the memory and I/O needed for each join in the query.

3.1 FlashScan Operator

Figure 1 shows the page organization of both NSM and PAX for a relation with four attributes. In general, for an n-attribute relation, PAX divides each page into n *minipages*, where each minipage stores the values of a column contiguously. When a page is brought into main memory, the CPU can access only the minipages needed by the query [1].

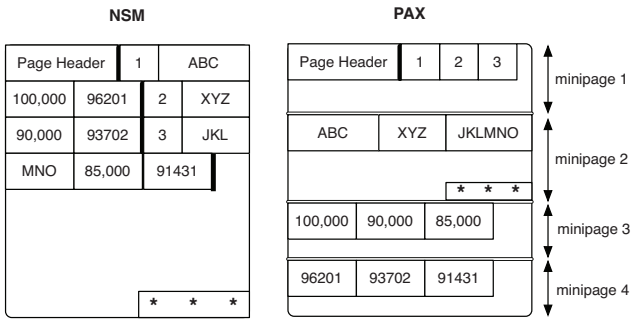


Figure 1: Disk pages in NSM (left) and PAX (right) storage layout of a relation with four attributes.

Data transfer units in main memory can be as small as 32 to 128 bytes (the size of a cacheline), but a database page is typically 8K to 128K. With SSDs, the minimum transfer unit is 512 to 2K bytes, which allows us to selectively read only parts of a regular page. *FlashScan* takes advantage of this small transfer unit to read only the minipages of the attributes that it needs. As discussed in [21], column-store systems can enjoy performance benefits similar to FlashScan on SSDs, but our techniques are easier to integrate in existing row-based database systems than building a column-store from scratch.

3.2 FlashJoin Operator

FlashJoin is a multi-way equi-join algorithm, implemented as a pipeline of stylized binary joins. Like FlashScan, FlashJoin avoids reading unneeded attributes and postpones retrieving attributes until needed. Each binary join in the pipeline is broken into two separate pieces, a *join kernel* and a *fetch kernel*, each of which is implemented as a separate operator, as shown in Figure 2. The join kernel computes the join and outputs a join index. Each join index tuple consists of the join attributes as well as the row-ids (RIDs) of the participating rows from base relations. The fetch kernel retrieves the needed attributes using the RIDs specified in the join index. FlashJoin uses a *late materialization* strategy, in which intermediate fetch kernels only retrieve attributes needed to compute the next join and the final fetch kernel retrieves the remaining attributes for the result.

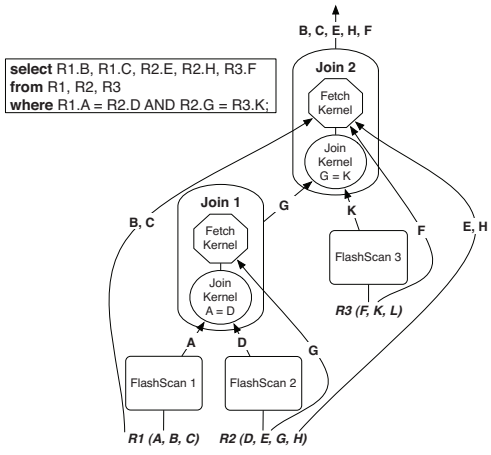


Figure 2: Execution strategy for a three-way join when FlashJoin is employed.

The *join kernel* leverages FlashScan to fetch only the join attributes needed from base relations. The join kernel is implemented as an operator in the iterator model. In general, it can implement any existing join algorithm: block nested loops, index nested loops, sort-merge, hybrid hash, etc. We explore here the characteristics of a join kernel that uses the

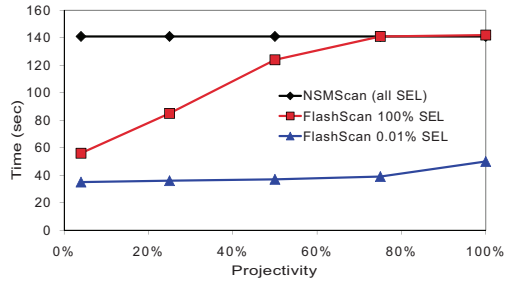


Figure 3: Scans on SSDs: FlashScan is faster than traditional scan when few attributes (small projectivity) or few rows (small selectivity – SEL) are selected.

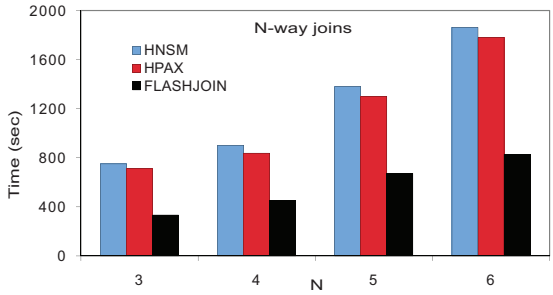


Figure 4: Multi-way joins on SSDs: FlashJoin is at least 2x faster than hybrid hash join over either traditional row-based (HNSM) or PAX layouts.

hybrid-hash algorithm [4]. This kernel builds an index (hash table) on the join attribute and RID from the inner relation and probes the index in the order of the outer. Compared to a traditional hash join, this join kernel is more efficient in two important ways. First, the join kernel needs less memory, thus many practical joins that would otherwise need two passes can complete in one pass. Second, when the join kernel spills to disk, materializing the runs is cheaper. Although we have only explored hybrid-hash, we believe a sort-merge kernel would offer similar results because of the duality of hash and sort [5]. Different strategies for the fetch kernel along with various optimizations can be found in [21]. FlashJoin is inspired by Jive/Slam-joins [13] (those algorithms were limited to two-way non-pipelined joins), and its late materialization strategy bears similarities to radix-cluster/decluster joins [14] which were optimized for main-memory operation.

3.3 Evaluation inside PostgreSQL

We implemented the proposed techniques inside PostgreSQL. In addition to incorporating a new scan and join operator, we modified the buffer manager and the bulk loader to support both the original and PAX layouts. We experimented with an enterprise SSD drive using synthetic datasets and TPC-H queries. Figure 3 compares the performance of FlashScan to Postgres' original scan operator (NSMScan) as we vary the percentage of tuple length projected from 4% to 100% (experimentation details are in [21]). FlashScan (100% selectivity, middle line) is up to 3X faster than the traditional scan for few projected attributes as it exploits fast random accesses to skip non-projected attributes. For low percentages of selectivity (0.01%, bottom line), FlashScan consistently outperforms the traditional scan by 3-4x, as it avoids reading projected attributes that belong to non-qualifying tuples. The original scan reads all pages regardless of selectivity.

Figure 4 shows the performance improvements when running multiway joins in Postgres using our FlashJoin algorithm. FlashJoin uses the drive more efficiently than hybrid-hash join over both traditional row-based (NSM) and column-based (PAX) layouts, by reading the minimum set of attributes needed to compute the join, and then fetching only those attributes that participate in the join result. By accessing only the join attributes needed by each join, FlashJoin also reduces memory requirements, which is beneficial in two ways: it decreases the number of passes needed in multi-pass joins (hence speeding up the join computation), and it frees up memory space to be used by other operators (hence leading in improved overall performance and stability in the system).

4 Sorting in a Memory Hierarchy

Sorting is a fundamental function used by database systems for a large variety of tasks. Sorting algorithms and techniques thus have a significant impact upon database performance. Informed by prior work such as [5, 6, 17, 8], we now outline techniques for sorting efficiently in database servers with flash memory in addition to traditional disk drives. The SSD advantage is access time; their disadvantages are capacity and cost/capacity.

In other words, very large operations require “spilling" from SSD to traditional disk, as do large databases if economy is a factor, i.e., the cost to purchase, house, power, and cool the computer system. On the other hand, data transfer to and from SSD storage is efficient even if the unit of transfer (or page size) is fairly small. Whereas optimal page sizes for traditional disk drives are in the 100s of KB, optimal page sizes for SSD are at most in the 10s of KB. In the case of an external merge sort with a fixed amount of RAM, runs on SSD storage permit a much higher merge fan-in than runs on traditional disks.

4.1 Using Flash Memory Only

If the input volume is too large for a traditional in-memory sort yet sufficiently small to fit on the available flash memory, a traditional external merge sort is the appropriate algorithm, with only minor modifications. Figure 5 illustrates how run generation with flash as external storage is more similar than different from run generation with a traditional disk as external storage.

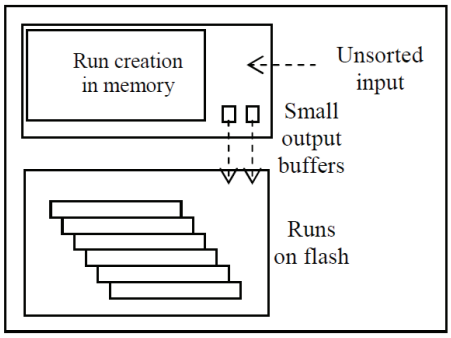


Figure 5: Runs on flash.

Note that due to the fast latency, there is little incentive to employ large transfers or pages. Small pages, e.g., 4 KB, are entirely appropriate. As a second consequence of short access latencies, the importance of double-buffered I/O diminishes. For example, a disk operation of 1 MB takes about 16.7 ms (10 ms + 1 MB ÷ 150 MB/s), equivalent to 33 million cycles in a processor running at 2 GHz. A flash disk operation of 4 KB takes merely 0.041 ms (0.025 ms + 4 KB ÷ 250 MB/s) or 81,000 cycles, a difference of almost three orders of magnitude. This advantage of flash becomes even more apparent as SSD technology improves, offering bandwidth increasingly faster than that of HDD.

4.2 Run Generation with Disk and Flash

If the input size exceeds the available capacity of flash memory, runs on disk must be employed. For best use of both flash and disk, initial runs on flash can be merged to form initial runs on disk. The size of runs on disk will be similar to the available capacity of flash memory.

The page size on flash and disk may differ. In fact, it should differ to exploit the fast access latency of flash devices and the fast bandwidth of disk devices. Thus, the allocation of memory between merge input and merge output will be different from traditional merge operations. For example, half of the available memory might be dedicated to buffering merge output in order to enable large disk transfers. Nonetheless, the small page size on flash may still permit a fairly high merge fan-in.

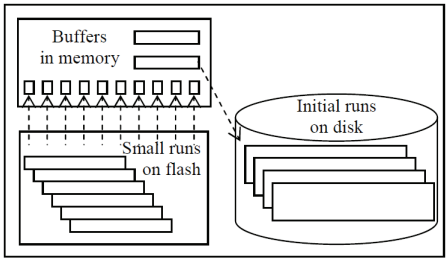


Figure 6: Merging from flash to disk.

Figure 6 illustrates run generation on disk by means of merging memory-sized runs gathered on flash memory. The most important aspect is the difference in page sizes appropriate for the different access latencies on flash memory and disk. Due to small input buffers, the fan-in is quite high even for a fairly small memory and thus the initial runs on disk are much larger than the available memory and the small runs on flash.

The merge bandwidth is limited by the slower of input and output. Two techniques will typically be required for matched bandwidth and thus best overall performance. First, the flash memory must remain far from full such that wear leveling can remain effective without excessive overhead for creation of free space (also a problem in log-structured file systems [OD 89], which are quite similar). In other words, flash capacity must be traded

for performance. In fact, some vendors of flash hardware seem to do this already, i.e., some devices contain actual capacity much larger than their externally visible storage capacity, and some devices make less storage space available than is built in. It might create a point of contention whether the internal or the external capacity should be the stated storage capacity. Second, to match input and output bandwidth, the ratio of the number of flash devices versus the number of disks must be adjusted to reflect their relative read and write bandwidths. Organization of flash devices as the array can be hidden within a hardware device (i.e., a single SATA device contains an array of flash chips) or may be provided by the operating system or the database management system.

4.3 Merging with Disk and Flash

The final sort output is produced merging all runs on disk. Ideally, the availability of flash memory is exploited such that the merge operation benefits from both the fast access latency of flash memory and the fast transfer bandwidth of traditional disks.

If the merge operation from flash to disk may be viewed as “assembling" large disk pages from small pages on flash, merging runs on disk may be viewed as the opposite, i.e., “disassembling" large disk pages. Specifically, large disk pages are moved from disk to flash memory, if necessary via memory (if no disk-to-flash DMA is available). This move exploits the fast disk bandwidth and requires flash arrays and internal organization with matching bandwidth.

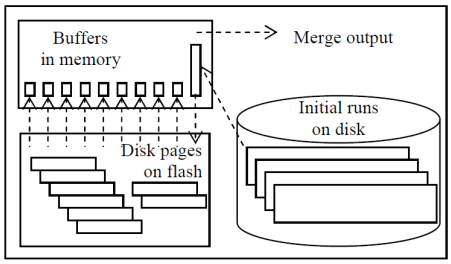


Figure 7: Merging from disk via flash.

These large pages on flash are divided into small pages and those are merged with a high fan-in, thus exploiting the fast access latency of flash memory. In order to make this work, the large pages on disk must indeed be organized as groups of small pages, i.e., each small page must contain an appropriate page header, indirection information for variable- length records, etc.

Figure 7 illustrates sort output formed from runs on disk. A large buffer in memory may be required to move large disk pages to flash memory. The actual merge operation in memory fetches small pages from flash, and replenishes flash memory by fetching additional large pages from disk.

5 Conclusions

SSDs constitute a significant shift in hardware characteristics, comparable to large CPU caches and multi-core processors. In this paper, we explore how for database systems, SSDs change not only power efficiency but also query execution efficiency for business intelligence and other complex queries.

We first demonstrated the suitability of a column-based page layout for accelerating database scan projections and selections on SSDs, through a prototype implementation of *FlashScan*, a scan operator that leverages the columnar layout to improve read efficiency, inside PostgreSQL. We presented *FlashJoin*, a general pipelined join algorithm that minimizes memory requirements and I/Os needed by each join in a query plan. FlashJoin is faster than a variety of existing binary joins, mainly due to its novel combination of three well-known ideas that in the past were only evaluated for hard drives: using a column-based layout when possible, creating a temporary join index, and using late materialization to retrieve the non-join attributes from the fewest possible rows. We incorporated the proposed techniques inside PostgreSQL. Using an enterprise SSD, we were able to speed up queries, ranging from simple scans to multiway joins and full TPC-H queries, by up to 6x.

We next explored how adding flash to the memory hierarchy impacts external merge sort. We chose to investigate sorting because it is prototypical for many query processing algorithms. We consider both the case where the input data fits into flash, as well as how flash can participate in external merge sort. With regard to

the latter, we first explore using flash to store runs, then using flash to merge runs. Balancing the bandwidths of input and output devices during the merge process, challenging even when using just HDD, is complicated by the addition of SSD. Due to the lack of mechanical seeking, accesses are fast and inexpensive. In addition, read and write bandwidths differ greatly in SSDs whereas they are quite similar in HDDs. For these reasons, bandwidth balancing strategies must be reconsidered in the face of SSD.

Our work has focused so far on storage formats and query execution algorithms that can take advantage of the performance characteristics of SSDs. One new consideration for query optimization is that many query execution plans that were not appropriate for HDDs become appropriate for SSDs. The fast access times of SSD shift the break-even point between table scans and index retrieval using both secondary and primary indexes. For example, if index-to-index navigation out-performs a table scan only if less than 1% of 1% of a table is needed, flash devices might shift the break-even point to 1%. In other words, many more queries can use indexes and thus a small concurrency control foot print that in the past would use table scans. Future research and industrial practice will show the extent to which this is true, and whether it might affect the feasibility and efficiency of real-time data warehousing with continuous trickle loading.

References

[1] A. Ailamaki, D. J. DeWitt, and M. D. Hill. Data page layouts for relational databases on deep memory hierarchies. *The VLDB Journal*, 11(3), 2002.

[2] L. Bouganim, B. Jonsson, and P. Bonnet. uflip: Understanding flash IO patterns. In *Proc. CIDR*, 2009.

[3] D. S. Clementsen and Z. He. Vertical partitioning for flash and hdd database systems. *J. Syst. Softw.*, 83(11):2237–2250, 2010.

[4] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood. Implementation techniques for main memory database systems. *SIGMOD Rec.*, 14(2):1–8, 1984.

[5] G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[6] G. Graefe. Implementing sorting in database systems. *ACM Comput. Surv.*, 38(3):10, 2006.

[7] G. Graefe. The five-minute rule 20 years later (and how flash memory changes the rules). *Commun. ACM*, 52(7):48–59, 2009.

[8] G. Graefe and P.-Å. Larson. B-tree indexes and cpu caches. In *ICDE*, pages 349–358, 2001.

[9] I. Koltsidas and S. D. Viglas. Flashing up the storage layer. *Proc. VLDB*, 1(1):514–525, 2008.

[10] S.-W. Lee and B. Moon. Design of flash-based dbms: an in-page logging approach. In *Proc. SIGMOD '07*, pages 55–66, New York, NY, USA, 2007.

[11] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A case for flash memory ssd in enterprise database applications. In *Proc. SIGMOD '08*, pages 1075–1086, New York, NY, USA, 2008.

[12] Y. Li, B. He, Q. Luo, and K. Yi. Tree indexing on flash disks. In *Proc. ICDE*, 2009.

[13] Z. Li and K. A. Ross. Fast joins using join indices. *VLDB J.*, 8:1–24, 1999.

[14] S. Manegold, P. Boncz, N. Nes, and M. Kersten. Cache-conscious radix-decluster projections. In *Proc. VLDB*, 2004.

[15] D. Myers. On the use of NAND flash memory in high-performance relational databases. *MIT Msc Thesis*, 2008.

[16] S. Nath and P. B. Gibbons. Online maintenance of very large random samples on flash storage. *Proc. VLDB Endow.*, 1(1):970–983, 2008.

[17] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. B. Lomet. Alphasort: A cache-sensitive parallel external sort. *VLDB J.*, 4(4):603–627, 1995.

[18] M. Polte, J. Simsa, and G. Gibson. Enabling enterprise solid state disks performance. In *Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, 2009.

[19] K. A. Ross. Modeling the performance of algorithms on flash memory devices. In *Proc. DaMoN '08*, pages 11–16, New York, NY, USA, 2008.

[20] L. D. Shapiro. Join processing in database systems with large main memories. *ACM Trans. Database Syst.*, 11(3):239–264, 1986.

[21] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query processing techniques for solid state drives. In *Proc. SIGMOD*, pages 59–72, 2009.

Flash in a DBMS: Where and How?

Manos Athanassoulis[†] Anastasia Ailamaki[†] Shimin Chen[⋆] Phillip B. Gibbons[⋆] Radu Stoica[†]

[†]Ecole Polytechnique Fédérale de Lausanne [⋆]Intel Labs Pittsburgh

Abstract

Over the past decade, new solid state storage technologies, with flash being the most mature one, have become increasingly popular. Such technologies store data durably, and can alleviate many handicaps of hard disk drives (HDDs). Nonetheless, they have very different characteristics compared to HDDs, making it challenging to integrate such technologies into data intensive systems, such as database management systems (DBMS), that rely heavily on underlying storage behaviors. In this paper, we ask the question: Where and how will flash be exploited in a DBMS? We describe techniques for making effective use of flash in three contexts: (i) as a log device for transaction processing on memory-resident data, (ii) as the main data store for transaction processing, and (iii) as an update cache for HDD-resident data warehouses.

1 Introduction

For the past 40 years, hard disk drives (HDDs) have been the building blocks of storage systems. The mechanics of HDDs rotating platters dictate their performance limitations: latencies dominated by mechanical delays (seeks and rotational latencies), throughputs for random accesses much lower than sequential accesses, interference between multiple concurrent workloads further degrading performance [25], etc. Moreover, while CPU performance and DRAM memory bandwidth have increased exponentially for decades, and larger and deeper cache hierarchies have been increasingly successful in hiding main memory latencies, HDD performance falls further and further behind. As illustrated in Table 1, HDDs' random access latency and bandwidth have improved by only 3.5X since 1980, their sequential bandwidth lags far behind their capacity growth, and the ratio of sequential to random access throughput has increased 19 fold.

New storage technologies offer the promise of overcoming the performance limitations of HDDs. Flash, phase change memory (PCM) and memristor are three such technologies [7], with flash being the most mature. Flash memory is becoming the de facto storage medium for increasingly more applications. It started as a storage solution for small consumer devices two decades ago and has evolved into high-end storage for performance sensitive enterprise applications [20, 21]. The absence of mechanical parts implies flash is not limited by any seek or rotational delays, does not suffer mechanical failure, and consumes less power than HDDs. As highlighted in Table 1, flash-based solid state drives (SSDs) fill in the latency and bandwidth gap left by HDDs. SSDs also provide a low ratio between sequential and random access throughput[1]. The time to scan the entire device sequentially is much lower than modern HDDs, close to what it was in older HDDs.

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]Random access throughput for SSDs is computed using a mix of reads and writes. If only reads are performed then sequential and random access throughputs are roughly equal.

Table 1: Comparison of Hard Disk Drives (1980, 2010) and Flash Drives (2010)

Device & Year	Capacity (GB)	Cost ($)	Cost/MB ($)	Random Access Latency (ms)	Random Access Bandwidth (MB/s)	Sequential Access Bandwidth (MB/s)	Sequential BW / Random BW	Device Scan (s)
HDD 1980	0.1	20000	200	28	0.28	1.2	4.3	83
HDD 2010	1000	300	0.0003	8	0.98	80	81.6	12500
SSD 2010	100	2000	0.02	0.026	300	700	2.3	143

Sources: Online documents and presentations [13], vendor websites and other sources [18].

Flash devices, however, come with certain limitations of their own. There is an asymmetry between random reads and random writes [3, 6, 23], with the latter incurring a performance hit due to the specifics of the flash technology [2, 6]. Newer SSDs [16, 19] mitigate this performance hit, but random writes still have a negative impact on the future performance of the device. Interestingly, sequential writes not only offer good performance but in many cases repair device performance after extensive random writes [26]. Finally, flash cells wear out after 10K–100K writes to the cell.

The differences between HDDs and SSDs are particularly important in Database Management Systems (DBMS) because their components (query processing, query optimization, query evaluation) have been tuned for decades with the HDD characteristics in mind. Specifically, random accesses are considered slow, sequential accesses are preferred, and capacity is cheap. Thus, achieving successful integration of flash requires revisiting DBMS design. Currently, flash usage in DBMS follows two trends, resulting either in flash-only systems or in hybrid flash-HDD systems. Flash-only systems benefit greatly from flash-friendly join algorithms [29], indexes [1, 8, 23, 24], and data layout (this paper). Hybrid systems use flash judiciously to cache either hot or incoming data or for specific operations [9, 14].

In this paper we describe techniques for using flash to optimize data management in three different settings, covering DRAM-resident, SSD-resident, and HDD-resident data stores. Each technique represents an example of how flash can be used within current systems to address the limitations of HDDs. First, we consider transaction processing on a memory-resident data store, and show how to use flash for transactional logging, dramatically improving transaction throughput (Section 2.1). Second, a straightforward way to use flash is to replace all the HDDs with SSDs without changing the DBMS software. In this scenario flash addresses the performance limitations of HDDs but poses new challenges because excessive random writes degrade performance and wear out the flash storage prematurely. To overcome these challenges, we present a technique that leaves a DBMS unchanged and yet avoids random writes on flash (Section 2.2). Third, we show how flash can be used as a performance booster for data warehouses stored primarily on HDDs. Specifically, we describe techniques for buffering data updates on flash such that (i) for performance, queries process the HDD-resident data without interference from concurrent updates and (ii) for correctness, the query results are adjusted on-the-fly to take into account the flash-resident updates (Section 3). Finally, we conclude by highlighting techniques and open problems for other promising uses of flash in data management (Section 4).

2 Efficient Transaction Processing Using Flash Devices

In this section we describe how flash can be used effectively in the context of online transaction processing (OLTP). We show that the absence of mechanical parts makes flash an efficient logging device and that using SSDs as a drop-in replacement for HDDs greatly benefits from flash-friendly data layout.

2.1 Using Flash for Transactional Logging

Synchronous transactional logging, in which log records are forced to stable media before a transaction commits, is the central mechanism for ensuring data persistency and recoverability in database systems. As DRAM capacity doubles every two years, an OLTP database that was considered “large” ten years ago can now fit into main memory. For example, a database running the TPCC benchmark with 30 million users requires less than 100GB space, which can easily fit into the memory of a server (64–128GB of memory). In contrast,

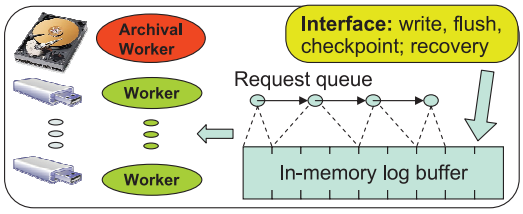


Figure 1: FlashLogging architecture: exploiting an array of flash devices and an archival HDD for faster logging and recovery.

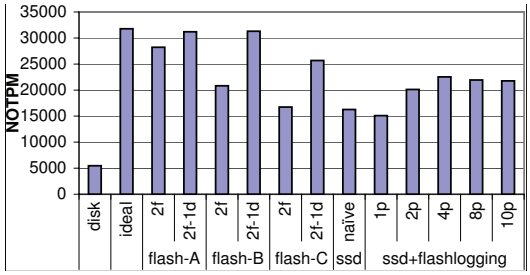


Figure 2: FlashLogging TPCC performance.

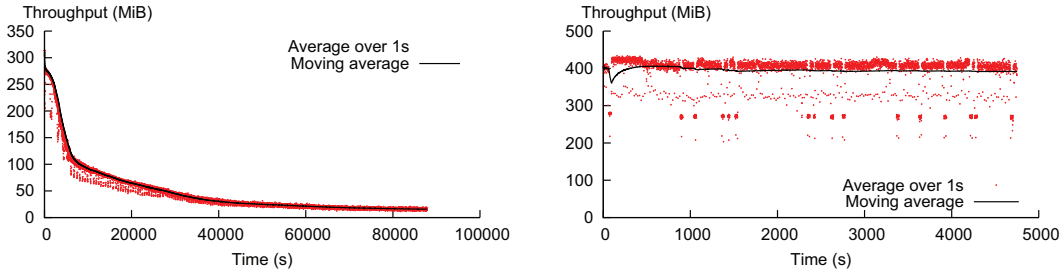
synchronous logging requires writing to stable media, and therefore is becoming increasingly important to OLTP performance. By instrumenting a MySQL-InnoDB running TPCC, we find that synchronous logging generates small sequential I/O writes. Such write patterns are ill-suited for an HDD because its platters continue to rotate between writes and hence each write incurs nearly a full rotational delay to append the next log entry. Because flash supports small sequential writes well, we propose FlashLogging [10] to exploit flash for synchronous logging.

FlashLogging Design. Figure 1 illustrates the FlashLogging design. First, we exploit multiple flash devices for good logging and recovery performance. We find that the conventional striping organization in disk arrays results in sub-optimal behavior (such as request splitting or skipping) for synchronous logging. Because our goal is to optimize sequential writes during normal operations and sequential reads during recovery, we instead propose an *unconventional* array organization that only enforces that the LSNs (log sequence numbers) on an individual device are non-decreasing. This gives the maximal flexibility for log request scheduling. Second, we observe that write latencies suffer from high variance due to management operations (such as erasures) in flash devices. We detect such outlier writes and re-issue them to other ready flash devices, thus hiding the long latencies of outliers. Third, the flash array can be implemented either with multiple low-end USB flash drives, or as multiple partitions on a single high-end SSD. Finally, our solution can exploit an HDD as a near-zero-delay archival disk. During normal processing, flash-resident log data is flushed to the HDD once it reaches a predefined size (e.g., 32KB). Logging performance is improved because the HDD can also serve write requests when all the flash drives are busy.

Performance Evaluation. We replace the logging subsystem in MySQL/InnoDB with FlashLogging. Figure 2 reports TPCC throughput in new order transactions per minute (NOTPM), comparing 14 configurations. "Disk" represents logging on a 10k rpm HDD, while "ideal" enables the write cache in "disk" (violating correctness) so that small synchronous writes achieve almost ideal latency. We evaluate three low-end USB flash drives (A, B, and C) from different vendors; "2f" uses two identical flash drives, and "2f-1d" is "2f" plus an archival HDD. We also employ a single high-end SSD either directly with the original logging system ("naive"), or with FlashLogging while using multiple SSD partitions (1–10 partitions) as multiple virtual flash drives. From Figure 2, we see that (i) FlashLogging achieves up to 5.7X improvements over traditional (HDD-based) logging, and obtains up to 98.6% of the ideal performance; (ii) the optional archival HDD brings significant benefits; (iii) while replacing the HDD with an SSD immediately improves TPCC throughput by 3X, FlashLogging further exploits the SSD's inner parallelism to achieve an additional 1.4X improvement; and (iv) when compared to the high-end SSD, multiple low-end USB flash drives achieve comparable or better performance at much lower price.

2.2 Transparent Flash-friendly Data Layout

Using flash as persistent storage medium for a DBMS often leads to excessive random writes, which impact flash performance and its predictability. In Figure 3(a) we quantify the impact of continuous 4K random writes



(a) Random writes (b) Random read/write mix (50%-50%) using A/P

Figure 3: Experiments with a FusionIO SSD as (a) drop-in replacement of HDDs and (b) using Append/Pack.

on FusionIO [16], a high-end SSD. We find that the sustained throughout is about 16% of the advertised random throughput. At the same time, sequential writes can fix random read performance [26].

Transforming Temporal Locality To Spatial Locality. Viewing SSDs as append logs helps us exploit the good sequential write bandwidth when writing. Consequently, temporal locality is transformed to spatial locality and thus sequential reads are transformed to random reads. This transformation helps overall performance, because unlike HDDs, flash offers virtually the same random and sequential read performance.

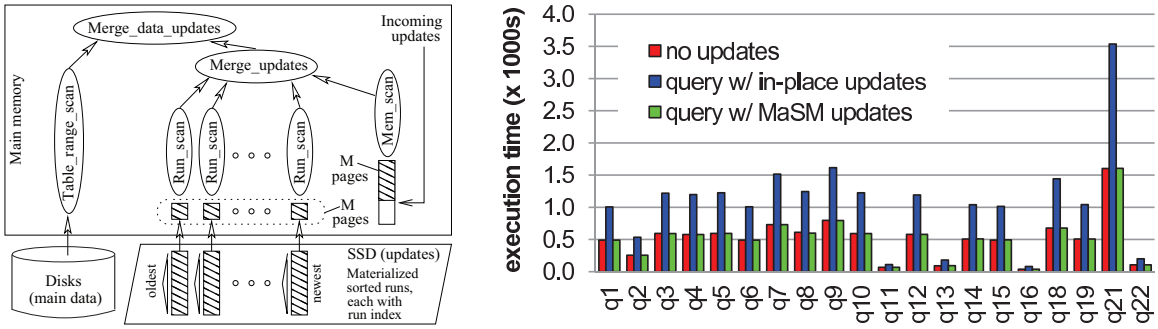
Append/Pack (A/P) Design. The A/P data layout [26] treats the entire flash device as a log. The dirty pages that come out of the buffer pool to be stored persistently are appended in the flash device and an index is kept in memory maintaining the mapping between the database page id and the location in the log. The A/P layer exports to the overlying device a transparent block device abstraction. We implement a log-structure data layout algorithm that buffers the dirty pages and writes them in blocks equal to the erase block size in order to optimize write performance. A/P organizes, as well, a second log-structured region of the device that is used for the packing part of the algorithm. During the operation of the system, some logical pages are appended more than once (i.e., the pages are updated, having multiple versions in the log). In order to keep track of the most recent version of the data, we invalidate the old entry of the page in the in-memory data structure. When the log is close to full with appended pages, we begin *packing*, that is, we move valid pages from the beginning of the log to the second log-structure assuming these pages are not updated for a long period (*cold pages*). The premise is that any (write-)hot page should be invalidated and all remaining pages are assumed to be cold. The cold pages are appended in a second log-structure to ensure good sequential performance and, thus, fast packing of the device.

Experimentation and Evaluation. We implement A/P as a standalone dynamic linked library which can be used by any system. The A/P library offers the typical *pwrite* and *pread* functions but manipulates writes as described above. We experimented with a PCIe Fusion ioDrive card [16]. The flash card offers 160GB capacity, and can sustain up to 700MB/s read bandwidth and up to 350MB/s sequential write bandwidth. Serving a TPCC-like workload using the A/P design, we maintain stable performance (in a read/write mix) achieving the max that the device could offer (400MB/s throughput as a result of combining reads and writes), as shown in Figure 3(b). When compared to the performance of a TPCC system on the same device but without A/P, we achieved speedups up to 9X [26].

3 Flash-enabled Online Updates in Data Warehouses

In this section, we investigate the use of flash storage as a performance booster for data warehouses (DW) stored primarily on HDDs, because for the foreseeable future, HDDs will remain much cheaper but slower than SSDs. We focus on how to minimize the interference between updates and queries in DWs.

While traditional DWs allow only offline updates at night, the need for 24x7 operations in global markets and the data freshness requirement of online and other quickly-reacting businesses make concurrent online updates



(a) MaSM algorithm using 2M memory

(b) TPCH performance with online updates

Figure 4: MaSM design and performance. ($M = \sqrt{\|SSD\|}$)

increasingly desirable. Unfortunately, the conventional approach of performing updates in place can greatly disturb the disk-friendly access patterns (mainly, sequential scans) of large analysis queries, thereby slowing down TPCH queries by 1.5–4.0X in a row-store DW and by 1.2–4.0X in a column-store DW [4]. Recent work studied *differential updates* for addressing this challenge in column store DWs [17, 27]. The basic idea is (i) to cache incoming updates in an in-memory buffer, (ii) to take the cached updates into account on-the-fly during query processing, so that queries see fresh data, and (iii) to migrate the cached updates to the main data when the buffer is full. While significantly improving performance, these proposals require large in-memory buffers in order to avoid frequent, expensive update migrations. But dedicating a large amount of memory solely to buffering updates significantly degrades query operator performance, because less memory is available for caching frequently accessed structures (e.g., indices) and intermediate data (e.g., in sorting, hash-joins).

Exploiting Flash to Cache Updates. Our proposal follows the differential updates idea discussed above, but instead of using a large in-memory buffer, exploits SSDs (and a small amount of memory) to cache incoming updates. The goal is to minimize online updates' performance impact on table range scan operations, which are the major query pattern in large analytical DWs [5]. Unfortunately, naively applying the prior differential update techniques [17, 27] in the flash-augmented setting slows down table range scans by up to 3.8X [4]. Instead, we observe that the merging operation is similar to an outer join between the main data residing on HDDs and the updates cached on SSDs. To facilitate the merging, we sort the cached updates in the same order as the main data. The memory footprint is reduced by using two-pass external sorting: generating sorted runs then merging them. This requires $M = \sqrt{\|SSD\|}$ pages of memory to sort $\|SSD\|$ pages of cached updates. Moreover, because a later query should see all the updates that an earlier query has seen, we can materialize sorted runs and reuse them across multiple queries. We call the resulting algorithm *materialized sort-merge (MaSM)*.

Figure 4(a) illustrates a MaSM algorithm using $2M$ memory. It uses an M-page in-memory buffer to cache incoming updates. When the buffer fills, updates are sorted to generate a materialized sorted run of size M on the SSD. A table range scan will merge up to M materialized sorted runs and the in-memory updates. When the SSD is close to full, it is likely that there exist cached updates for every data page. Therefore, MaSM migrates updates to the main HDD-resident data store by sequentially writing back the merging outcome of a full table scan. Moreover, by attaching the commit timestamp to every update record, MaSM can correctly support concurrent queries, migration, and incoming updates, as well as transactions based on snapshot isolation or two-phase locking [4]. Furthermore, we also designed a more sophisticated MaSM algorithm that reduces the memory footprint to M pages by incurring extra SSD I/Os [4].

MaSM Evaluation. We implement MaSM in a prototype row-store DW. We compare MaSM and prior approaches using both synthetic data and TPCH [4]. We report the latter here. Figure 4(b) shows the TPCH performance with online updates. We record the disk traces when running TPCH queries on a 30GB database in a commercial row store. (Query 17 and 20 did not finish in 24 hours.) Then we replay the disk accesses as

the query workload on our prototype. For MaSM, we use 1GB flash space, 8MB memory, and 64KB sized SSD I/Os. In Figure 4(b), there are three bars for every query: query time without updates (left), with in-place updates (middle), and with updates using MaSM (right). We see that in-place updates incur 1.6–2.2X slowdowns. In contrast, the MaSM algorithm achieves almost the same performance (within 1%) as the queries without updates, providing fresh data with negligible overhead.

4 Conclusions and Open Problems

Increasingly popular in mainstream computing, flash devices present new performance vs. price tradeoffs for data management systems. In this paper, we examined three aspects of traditional relational database management systems (RDBMS) for answering the question: Where and how can flash be exploited in a DBMS? In particular, we studied the use of flash in transactional logging in memory-resident OLTP systems, flash-friendly data layout in flash-resident OLTP systems, and flash as an update cache in HDD-resident data warehousing systems. Experimental results showed that our proposed techniques make effective uses of flash in a DBMS.

There are a number of other opportunities for exploiting flash in data management, including:

- *Flash-Only Data Warehouses.* Given its high performance, low energy consumption, and decreasing price, flash-based SSDs have been considered as the main storage for DWs [28]. In such settings, the fast random accesses of flash may significantly benefit database data structures and query processing algorithms (e.g., joins [29]). On the other hand, because of flash's poor random write performance, our solution for online updates may still be desirable for converting random in-place updates into sequential writes.
- *Exploiting Flash Beyond Traditional RDBMS.* More generally, it is interesting to study the use of flash for improving data management solutions beyond traditional RDBMS, such as data stream management, data management in the cloud, key-value stores [14], sensor networks [23], approximate query processing, and so on. For example, Chen *et al.* [11] proposed a non-blocking join algorithm, PR-Join, that achieves nearly optimal performance by exploiting SSDs as temporary storage for spilling intermediate data. PR-Join is shown to support efficiently both online aggregation and stream processing.
- *Alternative Memory Hierarchies with Flash.* Given the very different characteristics of flash compared to both DRAM and HDDs, it can be beneficial to use flash in alternative memory/storage hierarchy organizations. For example, Mesnier *et al.* [22] proposed differentiated storage services that intelligently cache high-priority I/O blocks on SSDs for better QoS. Canim *et al.* [9] showed the benefits to traditional RDBMS of including flash as a caching layer between main memory and HDDs.
- *Emerging Byte-Addressable Non-Volatile Memory Technologies.* Several emerging non-volatile memory technologies are byte-addressable with access latencies comparable to DRAM, and endurance much better than flash [7]. Among them, phase change memory (PCM) is the most promising to be ready for commercial use in the near future. Because the performance of PCM falls inbetween flash and DRAM, there have been proposals to use PCM both in SSDs and as main memory (replacing DRAM) [15]. PCM-based SSDs provide a similar block-level interface and better read/write performance than flash-based SSDs. The main characteristic difference is that PCM does not require erases before writing and therefore its random write performance is close to its sequential write performance. PCM-based main memory promises to bring more profound changes to computer systems because of the fine-grained non-volatility and the unique read/write characteristics of PCM (e.g., writes are much slower and power-hungry than reads). Thus, it is important to investigate the impact of such changes on data management systems [12].

These opportunities make for interesting future work.

References

[1] D. Agrawal, D. Ganesan, R. K. Sitaraman, Y. Diao, and S. Singh. Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices. *VLDB*, 2009.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. *USENIX*, 2008.

[3] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo. Characterizing the Performance of Flash Memory Storage Devices and Its Impact on Algorithm Design. *WEA*, 2008.

[4] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. Towards Efficient Concurrent Updates and Queries in Data Warehousing. *Technical Report, EPFL-REPORT-152941*, 2010.

[5] J. Becla and K.-T. Lim. Report from the First Workshop on Extremely Large Databases (XLDB 2007). In *Data Science Journal*, 2008.

[6] L. Bouganim, B. T. Jónsson, and P. Bonnet. uFLIP: Understanding Flash IO Patterns. *CIDR*, 2009.

[7] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy. Overview of Candidate Device Technologies for Storage-Class Memory. *IBM J. of R&D*, 2008.

[8] M. Canim, G. A. Mihaila, B. Bhattacharjee, C. A. Lang, and K. A. Ross. Buffered Bloom Filters on Solid State Storage. *ADMS*, 2010.

[9] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang. SSD Bufferpool Extensions for Database Systems. *PVLDB*, 2010.

[10] S. Chen. FlashLogging: Exploiting Flash Devices for Synchronous Logging Performance. *SIGMOD*, 2009.

[11] S. Chen, P. B. Gibbons, and S. Nath. PR-Join: A Non-blocking Join Achieving Higher Early Result Rate with Statistical Guarantees. *SIGMOD*, 2010.

[12] S. Chen, P. B. Gibbons, and S. Nath. Rethinking Database Algorithms for Phase Change Memory. In *CIDR*, 2011.

[13] M. Dahlin. Technology Trends. http://www.cs.utexas.edu/~dahlin/techTrends.

[14] B. Debnath, S. Sengupta, and J. Li. FlashStore: High Throughput Persistent Key-Value Store. *VLDB*, 2010.

[15] E. Doller. Phase Change Memory and its Impacts on Memory Hierarchy. http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf, 2009.

[16] FushionIO. The FusionIO Drive Data Sheet. http://community.fusionio.com/media/p/459.aspx.

[17] S. Héman, M. Zukowski, N. J. Nes, L. Sidirourgos, and P. A. Boncz. Positional Update Handling in Column Stores. *SIGMOD*, 2010.

[18] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, 2007.

[19] Intel. X-25E Technical Documents. http://www.intel.com/design/flash/nand/extreme/technicaldocuments.htm.

[20] S.-W. Lee, B. Moon, and C. Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. *SIGMOD*, 2009.

[21] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A Case for Flash Memory SSD in Enterprise Database Applications. *SIGMOD*, 2008.

[22] M. Mesnier, S. Hahn, and B. McKean. Making the Most of Your SSD: A Case for Differentiated Storage Services. *FAST WIPs*, 2009.

[23] S. Nath and P. B. Gibbons. Online Maintenance of Very Large Random Samples on Flash Storage. *VLDB Journal*, 2010.

[24] S. T. On, H. Hu, Y. Li, and J. Xu. Lazy-Update B+-Tree for Flash Devices. *Mobile Data Management*, 2009.

[25] J. Schindler, A. Ailamaki, and G. R. Ganger. Lachesis: Robust Database Storage Management Based on Device-specific Performance Characteristics. *VLDB*, 2003.

[26] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki. Evaluating and repairing write performance on flash devices. *DaMoN*, 2009.

[27] M. Stonebraker et al. C-Store: A Column-Oriented DBMS. *VLDB*, 2005.

[28] Teradata. Teradata Extreme Performance Appliance: The World's First Solid State Data Warehouse Appliance for Hyper-analytics. http://www.teradata.com/t/newsrelease.aspx?id=12282, October 2009.

[29] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query Processing Techniques for Solid State Drives. *SIGMOD*, 2009.

Storage Class Memory Aware Data Management

Bishwaranjan Bhattacharjee
IBM T. J. Watson
Research Center
bhatta@us.ibm.com

Mustafa Canim*
The University of Texas
at Dallas
canim@utdallas.edu

Christian A. Lang
IBM T. J. Watson
Research Center
langc@us.ibm.com

George A. Mihaila
IBM T. J. Watson
Research Center
mihaila@us.ibm.com

Kenneth A. Ross
IBM T. J. Watson and
Columbia University
kar@cs.columbia.edu

Abstract

Storage Class Memory (SCM) is here to stay. It has characteristics that place it in a class apart both from main memory and hard disk drives. Software and systems, architectures and algorithms need to be revisited to extract the maximum benefit from SCM. In this paper, we describe work that is being done in the area of Storage Class Memory aware Data Management at IBM. We specifically cover the challenges in placement of objects in storage and memory systems which have NAND flash (one kind of SCM) in a hierarchy or in the same level with other storage devices. We also focus on the challenges of adapting data structures which are inherently main memory based to work out of a memory hierarchy consisting of DRAM and flash. We describe how these could be addressed for a popular main memory data structure, namely the Bloom filter.

1 Introduction

Database systems make extensive use of main memory. There is a whole range of products where the data is main memory resident only. These include the IBM SolidDB [15], Oracle TimesTen [18], etc. Even in systems where the data is disk based, like DB2 LUW [7] and Oracle 11g [12], main memory is used for various operations to achieve acceptable performance. These operations include caching, locking, sorting, hashing, aggregations and bitmap operations. Due to the high number of transactions supported, as well as the type of operations being performed, main memory is always at a premium on a database system. Database systems have implemented a variety of ways to balance the demand on main memory, including putting in features like Self Tuning Main Memory [16, 6]. These help in getting the best usage for the limited main memory available.

The advent of Storage Class Memory [8], like flash and Phase Change Memory (PCM), opens up the possibility that with smart algorithmic changes [13], one could think of extending data structures and operations that are inherently main memory limited to work out of a memory hierarchy consisting of DRAM supplemented with

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*The majority of this work was completed while the author was an intern at IBM T. J. Watson Research Center

SCM. SCM has good random access capability that existing storage layers in the hierarchy below DRAM lack. Current SCM like flash, provide this capability in block based fashion and if existing main memory algorithms can be modified to take advantage of this, then it is possible to extend them to work beyond main memory.

In current disk based database systems a major bottleneck is the IO capability of the disks. Transaction processing systems tend to stress the IO subsystem with their short running point queries and updates. These result in random IO, which is a major drawback of hard disks. With their moving arms, a hard disk is very limited in the number of IO operations per second (IOPS) that it can produce. Even in the case of Data Warehousing, with customers running multiple streams of queries, ingests and deletes, the IO capability of the storage system tends to be a major pain point.

In these situations, SCM based or supplemented storage solutions could help. Examples include the SSD based Teradata Extreme Performance Appliance [17] or the Oracle Exadata [11] or IBM Smart Analytics Systems [14]. Given that customers may not be able to afford to place all their data on SCM based storage due to its cost, a key challenge that needs to be addressed is what data or objects should be placed in this costlier medium so that customers can get the best value for their money. In the case of database systems, the data or objects in question could be whole objects like relational tables, indexes, materialized views. Alternatively, one could selectively store pieces (e.g., pages) of these objects.

In this paper, we describe work that is being done in the area of Storage Class Memory aware Data Management at IBM. We specifically cover the challenges in placement of objects in storage and memory systems that have flash or SCM in a hierarchy or in the same level with other storage devices. We also focus on the challenges of adapting data structures that are inherently main memory based to work out of a memory hierarchy consisting of DRAM and flash. We describe how these could be addressed for a popular main memory data structure, namely the Bloom filter. We also briefly look into how these challenges and solutions will have to adapt to future SCM technologies like Phase Change Memory.

The rest of the paper is organized as follows: in Section 2 we present both a static and a dynamic approach for selective data placement in a heterogenous storage environemnt containing SCM in addition to hard disk drives; in Section 3 we describe how Bloom filters can take advantage of SCM; we conclude in Section 4.

2 Data Placement for Heterogeneous Storage

Given a collection of storage devices with vastly different performance and cost characteristics, such as hard drives and SCM, one important issue is deciding on the physical placement of the data, to achieve good performance at a reasonable cost. At one extreme, placing all data on the faster device (SCM) will enable the best performance, at a high cost. At the other extreme, keeping all data on hard drives will be the least expensive, but it may exhibit poor performance, especially if the workload causes substantial random access to the disk. So the question is whether one can obtain significant improvements in performance at a reasonable cost by placing only a fraction of the database on the SCM. To achieve such improvements, there must be some locality of access that can be exploited. Fortunately, except for extreme workloads that scan all data, such locality does exist in practice.

One may consider either a static or a dynamic approach to the data placement problem. One can either make data placement choices as part of the physical database design stage, or dynamically re-locate data as needed in the production stage. In the remainder of this section we will describe our experience with both approaches and discuss their relative benefits and drawbacks.

2.1 Static Object Placement

In order to make informed decisions about data placement, one needs to know something about the workload that will be executed on the database. However, with the typical database having hundreds of objects (tables, indexes,

materialized views, etc.), deciding on the optimal location of each object can quickly become overwhelming even for an experienced database administrator. In order to assist the DBA in this task, we implemented an Object Placement Advisor (OPA) tool [4], that generates placement recommendations based on profiling information obtained after running a calibration workload on a test instance of the database.

Specifically, the profiling information used includes, for each database object O_i, the total time spent reading and writing disk pages belonging to that object, distinguishing among random and sequential access. Using these statistics, the OPA tool estimates the total potential benefit B_i (access time savings) were this object to be placed on to SCM instead. The potential benefit is calculated given the ratio between the throughput of the SCM device relative to the hard drive for each type of access (random vs. sequential, and read vs. write). Assuming the size S_i of each object is known, or can be estimated (for the production database), the object placement problem becomes an instance of the well-known Knapsack problem, for which good heuristics exist.

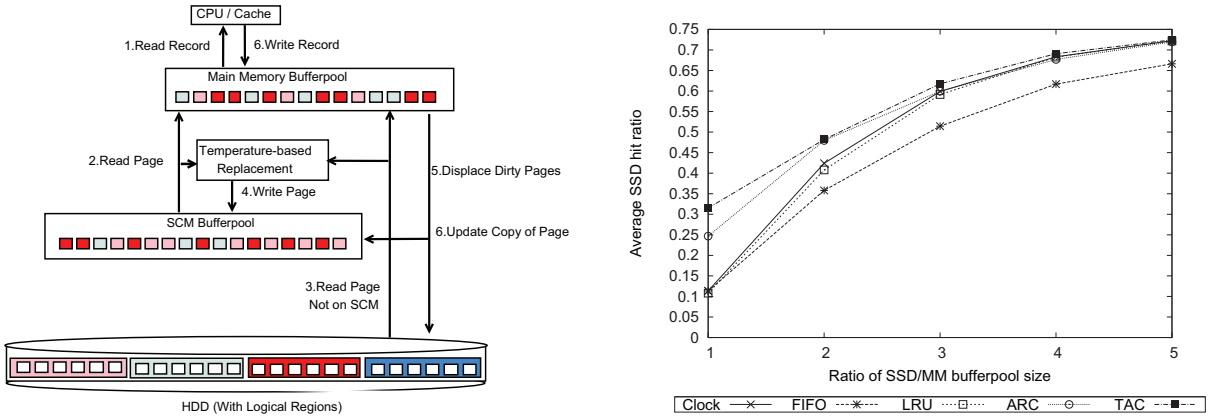
In our implementation, we used a Greedy heuristic, which delivers very competitive solutions in practice, and runs in linear time in the number of objects. Specifically, the Greedy heuristic iteratively places objects on the SCM in descending order of their average *access density* (access time savings per unit of storage) until there is no space left for the next object. To validate this approach, we implemented the recommendations computed by the OPA on the TPC-C benchmark and we compared the throughput (tpm-C) with the base database (all data on hard drive). The database layout suggested by OPA managed to deliver 10x more transactions per minute with 36% of the data placed on a consumer-grade Samsung SSD delivering 5000 IOPS (compared to about 120 random IOPS for the hard drive). For comparison, we also implemented a naive placement strategy (all indexes on SCM, all tables on magnetic disk), which achieved only a 2x throughput improvement with 23% of the data stored on SCM.

The static placement solution works best when the workload is relatively stable over time, but if the workload shifts significantly the DBA needs to re-run the OPA tool periodically and potentially move objects across storage devices to implement changes in the placement recommendations. Another limitation of this scheme is that it doesn't allow *partial* placement of objects on different devices. This means that large objects that do not fit in their entirety on the SCM will never be considered. Both of these limitations are lifted by the dynamic scheme described next.

2.2 Dynamic Temperature-Aware Bufferpool Extension

In order to enable a more flexible and adaptive placement scheme we conceptually move SCM up in the storage hierarchy, to an intermediate level between the magnetic disk(s) and main memory, effectively using it to house an extension of the main memory bufferpool [5]. Thus, the SCM-resident bufferpool extension acts like a *second-level* page cache, while the main memory bufferpool is still the primary cache. Our system architecture is summarized in Figure 1(a).

Clearly, such an arrangement has the potential to overcome both the workload adaptability and the partial placement limitations exhibited by the static scheme. However, since the access patterns for a second-level cache are quite distinct from those of a primary cache, traditional replacement policies are not best suited for governing page admission and eviction decisions. Instead, we base these decisions on the "temperature" of pages, a notion inspired from the access density used in the static scheme. Conceptually, the current temperature of a given disk page is the total time spent reading that page from disk. Since maintaining the precise access time of each database page is expensive, we group a small number of adjacent pages (currently between 8 and 64) into *regions* and maintain only the average temperature of each region. Also, instead of actually measuring the time spent, we estimate it based on the type of access (random vs. sequential) and the disk performance rating. Thus, for any given page we approximate its current temperature by the average temperature of its region. Whenever a main memory bufferpool page miss occurs, we consult the SCM bufferpool extension, and, if the page is currently cached there, we serve it (item 2); otherwise, we read it from the hard drive (item 3), and based on its region temperature, decide whether we should cache it in the SCM bufferpool. A page p is only cached in the SCM



(a) System Architecture

(b) SCM bufferpool hit ratio comparison

Figure 1: Dynamic Bufferpool Extension.

bufferpool if its temperature exceeds that of the coldest page q currently cached there, in which case q will be evicted and p stored in its place (item 4). This way, for any given workload, the SCM bufferpool content will automatically converge to the hottest set of K pages, where K is the SCM capacity in pages. To account for changes in the workload, the temperatures of all regions are periodically multiplied with an aging factor (by default 0.8). This aging mechanism effectivelly reduces the relative importance of older accesses in favor of new accesses. Queries may cause records to be modified (item 6), resulting in pages marked "dirty" in the main memory bufferpool. Whenever a dirty page is displaced from the main memory bufferpool (item 5) it is written to the hard disk (HDD) and updated in the SCM (item 6), if present. Every time a page is read from either the SCM (item 2) or HDD (item 3), its identifier is also fed into the temperature-based replacement logic so that the temperature information is updated.

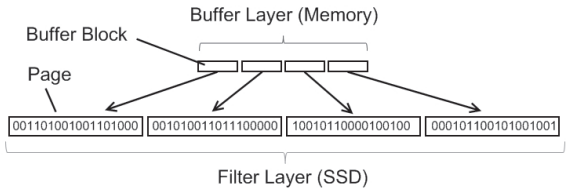
In order to compare the temperature-aware caching policy (TAC) with other replacement policies, we benchmarked them against a TPC-C [19] database (150 warehouses). Specifically, we compared TAC with the following page replacement policies: least recently used (LRU), first in first out (FIFO), clock replacement, and adaptive replacement (ARC) [9]. With this scaling factor the database occupies a total of 15 GB of disk space. For the memory bufferpool, 368 MB (2.5% of the database size) of space is used while running the transactions of 20 clients. The results for different ratios between the size of the SCM bufferpool and the main memory bufferpool are shown in Figure 1(b).

As the results prove, TAC outperforms the other replacement algorithms for all SCM bufferpool sizes. The difference in hit ratios is more pronounced for the case when the SCM bufferpool size is smaller (e.g. 1 or 2 times the size of the main memory bufferpool). This is due to the selective caching property of TAC (not caching pages colder than the coldest page cached), which reduces cache pollution. For large SCM bufferpool sizes, all algorithms (except for FIFO) perform about the same.

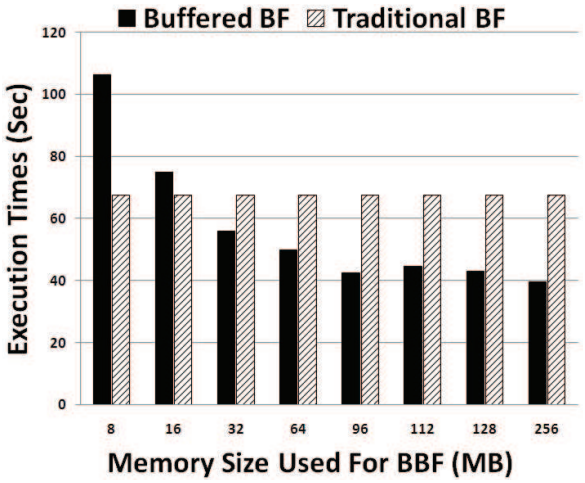
3 Extending Main Memory Data Structures using SCM

Besides the bufferpool, a database system employs main memory for a variety of other operations like locking, sorting, hashing, aggregations and bitmap operations. In this section, we show how the Bloom filter [2], can be redesigned to take advantage of a memory hierarchy consisting of main memory and SCM [3]. Database systems have used Bloom filters for index ANDing [1], join processing [10], selectivity estimation [20], and statistics collection [1, 20].

A traditional Bloom filter (TBF) consists of a vector of β bits, initially all set to 0. To update the filter,



(a) Structure



(b) Execution Times for Different Buffer Sizes

Figure 2: Buffered Bloom Filter.

k independent hash functions $h_1, h_2, ..., h_k$ all with range $\{1, ..., \beta\}$ are used. For each element $R \in S$, the bit locations $h_i(R)$, $i \in \{1, 2, ..., k\}$, are set to 1. To check if an element e is in S, the bit locations $h_i(e)$, $i \in \{1, 2, ..., k\}$, are checked. If any of them is not set to 1, we conclude that e is not an element of S. Otherwise, we assume that e is an element of S, with a certain probability that the assumption is wrong. If we conclude that the element is in the set although it is not, that element is called a false positive. The false positive rate is directly proportional to the compression ratio. Thus, in order to lower the false positive rate, a larger amount of memory is required.

We now present a Buffered Bloom Filter (BBF) algorithm that runs on a memory hierarchy consisting of DRAM and SCM. This algorithm reduces the main memory requirement and the false positive rate without any performance loss. Given that flash is one order of magnitude slower that main memory, a straight forward moving of the filter to flash will result in significant performance loss. To compensate for the performance loss, we defer the read and write operations during the execution of both build and probe phases. With this improvement, SSD page accesses are reduced by up to three orders of magnitude depending on the buffer size. We also divide the Bloom filter into virtual sub-filters to improve the locality of the bit operations. This reduces the CPU cache misses.

The hierarchical structure of the BBF is depicted in Figure 2(a). The upper layer (called the *buffer layer*) is stored in main memory while the lower layer (the *filter layer*) is stored in the SSD. The former is used to buffer the auxiliary information pertaining to the deferred reads and writes. The latter is used to store the filter's bit vector. The filter is divided into virtual pages such that each page keeps the bit vector of a sub-filter. If the filter is divided into δ pages, the buffered Bloom filter can be considered as a combination of δ separate traditional Bloom filters. Each sub-filter has a dedicated buffer block.

We experimentally examined the impact of buffer size on execution time of the BBF while specifying a upper bound of 0.25% on the false positive rate. For these experiments, the number of build and probe records was kept at 50 Million. We used 2 hash functions for the BBF with a filter size of 256 MB (on SSD) and page size of 2MB. For the TBF we used 3 hash functions with a filter size of 128 MB (in DRAM). The execution times of each run for different buffer sizes are given in Figure 2(b). The BBF runs faster than the TBF provided that the buffer size is at least 32 MB of memory space. We also measured the false positives in an experiment setting where we used 32 MB of memory space for BBF and 64 MB for TBF. We observed that TBF yields at least an order of magnitude more false positives than BBF does.

4 Conclusion

In this paper we have described our work in the exploitation of SCM for database systems. We presented some of our early results in placement of objects in storage and memory systems that have SCM in a hierarchy or in the same level with other storage devices. We also examined some of the challenges of adapting data structures which are inherently main memory based to work out of a memory hierarchy consisting of DRAM and SCM.

While our approaches have been validated using flash devices, similar principles would be applicable (with slightly different tuning parameters) for other SCM devices like Phase Change Memory. Certain potential features of PCM, such as byte-addressability and the absence of an erase cycle, may make PCM superior to NAND flash for some workloads.

References

[1] A. Balmin, T. Eliaz, J. Hornibrook, L. Lim, G. M. Lohman, D. E. Simmen, M. Wang, and C. Zhang. Cost-based optimization in DB2 XML. *IBM Systems Journal*, 45(2):299–320, 2006.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Comm. ACM*, 13(7):422–426, 1970.

[3] M. Canim, G. A. Mihaila, B. Bhattacharjee, C. A. Lang, and K. A. Ross. Buffered Bloom filters on solid state storage. In *First Intl. Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS*10)*, 2010.

[4] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang. An object placement advisor for DB2 using solid state storage. *Proc. VLDB Endow.*, 2(2):1318–1329, 2009.

[5] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang. SSD bufferpool extensions for database systems. *PVLDB*, 3(2):1435–1446, 2010.

[6] B. Dageville and K. Dias. Oracle's self-tuning architecture and solutions. *IEEE Data Eng. Bull.*, 29(3):24–31, 2006.

[7] DB2 for Linux, UNIX and Windows. `www-01.ibm.com/software/data/db2/linux-unix-windows`.

[8] R. F. Freitas and W. W. Wilcke. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development*, 52(4-5):439–448, 2008.

[9] N. Megiddo and D. S. Modha. Outperforming LRU with an adaptive replacement cache algorithm. *Computer*, 37(4):58–65, 2004.

[10] J. K. Mullin. Optimal semijoins for distributed database systems. *IEEE Trans. Software Eng.*, 16(5):558–560, 1990.

[11] A technical overview of the Sun Oracle Exadata storage server and database machine. Internet Publication, September 2009. `www.oracle.com/us/solutions/datawarehousing/039572.pdf`.

[12] Oracle 11g database. `www.oracledatabase11g.com`.

[13] K. A. Ross. Modeling the performance of algorithms on flash memory devices. In *DaMoN*. ACM, 2008.

[14] IBM Smart Analytics System. `www-01.ibm.com/software/data/infosphere/smart-analytics-system/`.

[15] SolidDB product family. `www-01.ibm.com/software/data/soliddb/`.

[16] A. J. Storm, C. Garcia-Arellano, S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in db2. In *VLDB*, pages 1081–1092. ACM, 2006.

[17] Teradata extreme performance appliance. `teradata.de/t/assets/0/206/280/e30a3614-7a93-450d-9506-6eb6e461d0a2.pdf`.

[18] Oracle TimesTen in-memory database. `www.oracle.com/us/products/database/timesten/index.html`.

[19] TPC-C, On-Line Transaction Processing Benchmark. `www.tpc.org/tpcc/`.

[20] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Bloom histogram: Path selectivity estimation for xml data with updates. In *VLDB*, pages 240–251. 2004.

Accelerating In-Page Logging with Non-Volatile Memory

Sang-Won Lee[†] Bongki Moon[‡] Chanik Park[§] Joo-Young Hwang[§] Kangnyeon Kim[†]

[†]School of Info. & Comm. Engr.
Sungkyunkwan University
Suwon 440-746, Korea
{swlee,kangnuni}@skku.edu

[‡]Dept. of Computer Science
University of Arizona
Tucson, AZ 85721, U.S.A.
bkmoon@cs.arizona.edu

[§]Samsung Electronics Co., Ltd.
San #16 Banwol-Ri
Hwasung-City 445-701, Korea
{ci.park,jooyoung.hwang}@samsung.com

Abstract

A great deal of research has been done on solid-state storage media such as flash memory and non-volatile memory in the past few years. While NAND-type flash memory is now being considered a top alternative to magnetic disk drives, non-volatile memory (also known as storage class memory) has begun to appear in the market recently. Although some advocates of non-volatile memory predict that flash memory will give way to non-volatile memory soon, we believe that they will co-exist, complementing each other, for a while until the hurdles in its manufacturing process are lifted and non-volatile memory becomes commercially competitive in both capacity and price. In this paper, we present an improved design of In-Page Logging (IPL) by augmenting it with phase change RAM (PCRAM) in its log area. IPL is a buffer and storage management strategy that has been proposed for flash memory database systems. Due to the byte-addressability of PCRAM and its faster speed for small reads and writes, the IPL scheme with PCRAM can improve the performance of flash memory database systems even further by storing frequent log records in PCRAM instead of flash memory. We report a few advantages of this new design that will make IPL more suitable for flash memory database systems.

1 Introduction

Flash memory has been the mainstream of solid state storage media for the past decade and is being considered a top alternative to magnetic disk drives [1, 5, 10, 11]. Recently, other types of non-volatile memory such as phase-change RAM (PCRAM) (also known as storage class memory) have also been actively studied, developed and commercialized by industry leading manufacturers [2, 3, 8, 14]. While NAND type flash memory is page addressable and used for block storage devices, PCRAM is byte addressable and can be used much like DRAM. Furthermore, PCRAM can be written (or programmed) in place without having to erase the previous state and exhibits much lower read latency than flash memory.

In this paper, we revisit the In-Page Logging (IPL) scheme that has been proposed as a new storage model for flash-based database systems [9] and elaborate how the IPL scheme can accelerate database systems further by utilizing PCRAM for storing log records. The byte-addressability of PCRAM allows for finer-grained writing of log records than flash memory, thereby reducing the time and space overhead for both page read and write

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

operations. The results of preliminary performance evaluation will be presented to illustrate the potential benefit of IPL adapted to hybrid storage systems equipped with both flash memory and PCRAM.

2 Flash Memory vs. Non-Volatile Memory

2.1 Flash Memory

Although there are two types of flash memory, namely NOR and NAND, we are interested in NAND-type flash memory, because NAND-type flash memory is the one that is used as a storage medium for a large quantity of data. NOR-type flash memory is byte addressable and is mainly used to store program codes. Hereinafter, we use the term flash memory to refer to NAND-type flash memory.

The unit of a read or write operation in flash memory is a page of typically 2K or 4K bytes. Since flash memory is a purely electronic device without any mechanical part, it provides much higher and more uniform random access speed than a magnetic disk drive, as shown in Table 2.

Media	Access time		
	Read	Write	Erase
Magnetic Disk†	12.7 ms (2KB)	13.7 ms (2KB)	N/A
NAND Flash‡	75 μs (2KB)	250 μs (2KB)	1.5 ms (128KB)
PCRAM¶	206 ns (32B)	7.1 μs (32B)	N/A
DRAM§	70 ns (32B)	70 ns (32B)	N/A

†Disk: Seagate Barracuda 7200.7 ST380011A;
‡NAND Flash: Samsung K9F8G08U0M 16Gbits SLC NAND [15];
¶PCRAM: Samsung 90nm 512Mb PRAM [8];
§DRAM: Samsung K4B4G0446A 4Gb DDR3 SDRAM [16]

Table 2: Access Speed: Magnetic disk vs. NAND Flash vs. PCRAM vs. DRAM

One of the unique characteristics of flash memory is that no data item can be updated by overwriting it in place. In order to update an existing data item stored in flash memory, a time-consuming erase operation must be performed before overwriting for an entire block of flash memory containing the data item, which is much larger (typically 128 or 256 KBytes) than a page. Dealing with the erase-before-write limitation of flash memory has been one of the major challenges in developing solid state storage devices based on flash memory.

Most contemporary storage devices based on flash memory come with a software layer called *Flash Translation Layer (FTL)* [4]. The key role of an FTL is to redirect a write request from the host to an empty (or clean) area in flash memory and manage the address mapping from a logical address to a physical address for an updated data item. The use of an FTL hides the erase-before-write limitation of flash memory and makes a flash memory storage device look like a conventional disk drive to the host.

For the past few years, advances in the solid state drive (SSD) technology have made flash memory storage devices as a viable alternative to disk drives for large scale enterprise storage systems. Most enterprise class flash memory SSDs are equipped with parallel channels, a large overprovisioned capacity, and a large on-drive DRAM cache. Combined with advances in the FTL technology, this new SSD architecture overcomes huddles in small random write operations and provide significant performance advantages over disk drives [5, 10].

2.2 Byte-Addressable Non-Volatile Memory

Non-charge-based non-volatile memory technologies have recently been under active development and commercialization by industry leading manufacturers. Unlike charge storage devices such as flash memory, these non-

volatile memory technologies provide memory states without electric charges [6]. Examples of these memory technologies are ferroelectric RAM (FeRAM), magnetic RAM (MRAM) and phase-change RAM (PCRAM). Among those, PCRAM is considered a leading candidate for the next generation byte-addressable non-volatile memory.

PCRAM devices use phase change material for a cell to remember a bit. The phase change material can exist in two different states, amorphous and crystalline, which can be used to represent zero and one. Switching between the two states can be done by application of heat at different temperature ranges for different durations [6]. PCRAM devices can be programmed to any state without having to erase the previous state. Due to repeated heat stress applied to the phase change material, PCRAM has a limited number of programming cycles. However, PCRAM is considered to have greater write endurance than flash memory by a few orders of magnitude [7]. Furthermore, in contrast to NAND type flash memory, PCRAM need not operate in page mode and allows random accesses to individual bytes like DRAM does.

It is reported in the literature that the read and write latencies of PCRAM are approximately an order of magnitude greater than those of DRAM [7]. As is shown in Table 2, however, contemporary PCRAM products do not deliver the promised performance as yet particularly for write operations. While PCRAM takes only about 8 ns to read a two-byte word, it takes as long as one μs to write a two-byte word. A similar disparity in read and write speeds is observed in other PCRAM products as well [13].

Given the unique characteristics of PCRAM, we expect that PCRAM and flash memory will co-exist complementing each other for a while until the manufacturing cost of PCRAM is reduced to a level comparable to that of flash memory. As will be discussed in Section 3, we are interested in the potential of PCRAM that would make the In-Page Logging (IPL) scheme more efficient. Specifically, PCRAM allows for smaller units of writes, and the write time is approximately proportional to the amount of data to transfer.[1] These traits make PCRAM an excellent candidate for a storage medium for log data managed by the IPL scheme.

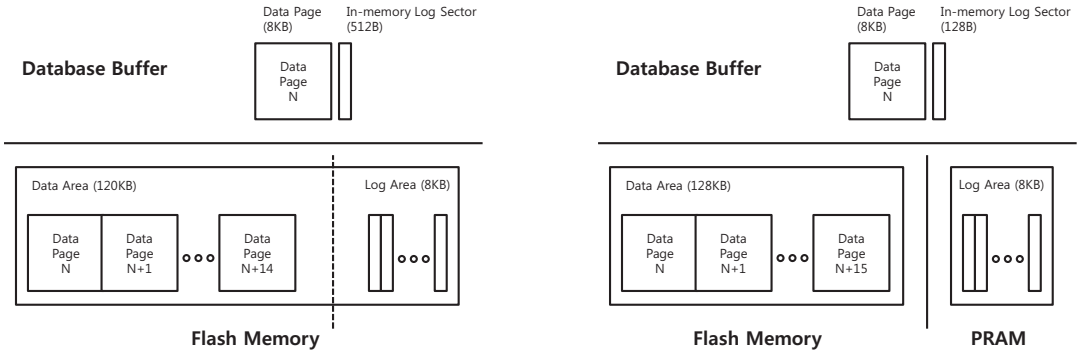
3 In-Page Logging with Non-Volatile Memory

In this section, we present a new implementation of IPL that utilizes non-volatile memory such as PCRAM to further improve its performance. For a hybrid storage device equipped with PCRAM as well as flash memory, IPL can boost its performance by taking advantage of higher access speed and byte-addressability of PCRAM for writing and reading small log records. We will review the IPL design and discuss the implications that the current technological trends of flash memory have on its performance. We will then present the new IPL implementation for hybrid storage devices.

3.1 IPL for Flash Only

It is quite common in the OLTP applications that most page frames become dirty in the buffer pool by small changes because updates are usually fairly small and distributed across a large address space. When a dirty page is about to be evicted from the buffer pool, a conventional buffer manager would write the entire dirty page back to a secondary storage device, even though the actual change amounts to only a small fraction of the page. The key idea of IPL is, as depicted in Figure 1(a), that only the changes made to a dirty page be written to a flash memory storage device in the form of log records without writing the dirty page itself [9]. Since the amount of changes is likely to be much smaller than the page itself, the IPL scheme can reduce the absolute amount of writes substantially, improve the overall write latency, and lengthen the lifespan of flash memory storage devices.

[1] For both read and write, the access time of PCRAM is not entirely proportional to the number of bytes to access, as there is an initial latency of about 78 ns for each operation [8]. Similarly, there is an initial latency of about 30 ns for DRAM [16]. The access times of PCRAM and DRAM shown in Table 2 include the latencies.



(a) Flash-only IPL

(b) IPL with PCRAM

Figure 1: IPL with Flash-only vs. IPL with Flash and PCRAM

The effectiveness of the IPL scheme, however, is being countered by the current trend in NAND flash memory technology towards a larger unit of I/O. For MLC-type NAND flash memory whose smallest unit of write is a page, it is impossible for IPL to reduce the absolute amount of writes because propagating even a few log records will require writing a full flash memory page. The situation becomes much easier for IPL when it comes to SLC-type flash memory. SLC-type NAND flash memory supports *partial programming* that allows a limited number of partial writes to be performed on the same page without erasing it. Thus, for example, a 2KB page can be written by four separate (512 byte) sector write operations instead of a single page write operation [12, 15].

While the effectiveness of IPL may not be compromised too much with SLC-type flash memory devices that are popular in enterprise database applications, there still remains a concern about the granularity of writes performed by the IPL scheme. As is discussed above, when a dirty page is about to be evicted, the amount of log records that need to be propagated for the page is usually very small. (It was in the range of 50 to 150 bytes in the TPC-C workloads we had observed.) This implies that write amplification could be further reduced if log records were written in a granularity smaller than even a sector. Obviously such a fine-grained writing is not feasible with any contemporary flash memory device, but it can be done very efficiently with byte-addressable non-volatile memory like PCRAM. Moreover, with PCRAM, the time taken to perform a write operation is almost linearly proportional to the amount of data to write. In contrast, this is not the case with NAND flash memory, because programming a sector takes the same amount of time as programming a full page, and they differ only in time taken to transfer a different number of bytes over the channel.

3.2 IPL for Flash and Non-Volatile Memory

Under the new design of IPL with PCRAM, as shown in Figure 1(b), regular data pages are stored in flash memory, while the corresponding log records are stored in PCRAM. Unlike the *flash-only* IPL, log records are not physically co-located with their corresponding data pages any longer, because data pages and log records reside in two separate storage components. Nonetheless, a flash memory block storing data pages must be associated with its log area in PCRAM efficiently. For simplicity, we assume that each data block in flash memory has a fixed length log area in PCRAM such that the address of a log area for a given data block can be determined by a direct mapping in constant time. Similar to the flash-only IPL, a merge operation needs to be carried out when the log area of a data block runs out of free space. Each log record in PCRAM log area is applied to its corresponding data page, and all the current data pages in the data block are written to a different clean data block.

In comparison, the new IPL design has the following advantages over the flash-only IPL. This is essentially attributed to the fact that the way PCRAM is used by the new IPL design for storing log records matches well with the characteristics of PCRAM.

- With byte-addressable PCRAM, log records can be flushed from the buffer pool in a finer grained fashion without increasing the write latency. In fact, this allows us to utilize the log area more efficiently, because the amount of log data a dirty page carries before being evicted is typically much smaller than a 512 byte sector (in the range of 50 to 150 bytes). This will in turn lead to much less frequent merge operations. Note that the size of an in-memory log sector is also reduced to 128 bytes in Figure 1(b).

- With PCRAM storing log data, the average latency of flushing log records from the buffer pool will be reduced considerably. This is because the write speed of PCRAM is (or will be) higher than that of flash memory. This effect will be amplified by the fact that the PCRAM write time is almost linearly proportional to the amount of data to write and the size of a write will be approximately one fourth of a 512 byte sector on average.

- PCRAM is faster than flash memory for small reads by at least an order of magnitude. Besides, when the current version of a data page needs to be computed, its log records can be fetched from PCRAM simultaneously while the old version of the data page is read from flash memory. Thus, the read overhead by the new IPL can be reduced to a negligible extent.

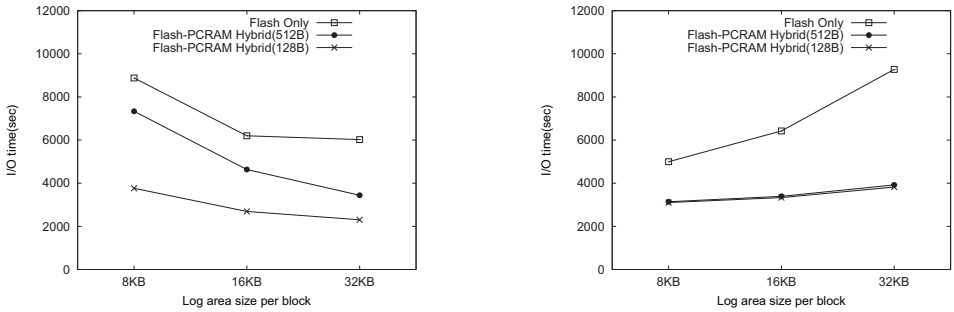
In addition to the performance aspects stated above, this new design makes IPL more pragmatic with existing flash memory solid state drives. With PCRAM being used to store log records, there is no need for partial programming or writing in a unit smaller than a page with flash memory SSDs any longer. Therefore, regardless of its type - SLC or MLC - any flash memory SSD can be used as a storage device for a database system with the IPL capability.

4 Preliminary Performance Evaluations

This section reports preliminary experimental results from a trace-driven simulation to demonstrate the effectiveness of the flash-PCRAM IPL design. We implemented an IPL module to the B^+-tree based Berkeley DB and ran it to obtain trace data for the simulation. Five million key-value records were inserted in random order, and the key-value records were retrieved again in random order. The size of each record was in the range of 40 to 50 bytes, and the database size was approximately 360 MB. The page size was 8 KB, and the buffer size was set to 20 MB. The trace obtained from the run included all the IPL related operations such as page reads, page writes, log sector writes, block merges and erasures.

In the experiment, we used the performance parameters of flash memory and PCRAM given in Table 2. Note that, in the case of flash-only IPL, the latency of a 512 byte sector write was set to 213 μs instead of 250 μs, because the amount of data to transfer over the channel was one fourth of what a full page write would do. When the size of an in-memory log sector was set to 512 bytes, the amount of valid log data flushed by a single write operation was 135 bytes on average, which was a little larger than the average size of a log write observed in the TPC-C benchmark (a little less than 100 bytes) [9]. Note that the size of a physical log write was still 512 bytes by flash-only IPL (as well as flash-PCRAM IPL (512B) that will be shown below).

Figure 2 shows the simulated runtime of the flash-only IPL and the flash-PCRAM IPL for the trace. In the figure, we plotted two cases of the flash-PCRAM IPL with an in-memory log sector of different sizes: 512 bytes and 128 bytes. Thus, in the latter case, log data were flushed to PCRAM more often in finer grained manner. In Figure 2(a), for the random insertion workload, the flash-PCRAM IPL (512B) outperformed the flash-only IPL 17 to 43 percent. This performance improvement was gained mostly by PCRAM whose latency for reading log data was shorter than that of flash memory. On top of that, by changing the size of an in-memory log sector from



(a) Random Insertion (b) Random Search

Figure 2: IPL Performance: Flash-only vs. Hybrid (5M records)

512 bytes to 128 bytes, the simulated runtime of flash-PCRAM IPL was reduced further up to almost 50 percent. Such a high percentage of reduction was due to the higher utilization of log areas achieved by finer-grained log writes, which in turn reduced block merge operations by more than a factor of two.

In Figure 2(b), for the random retrieval workload, the two cases of flash-PCRAM IPL yielded almost identical performance because the high speed of PCRAM read operations made the overhead of reading log data insignificant. For the same reason, the flash-PCRAM IPL outperformed the flash-only IPL with by a significant margin, because the overhead of reading log data from flash memory was not negligible and increased as the size of a log area increased.

5 Conclusions

There are many promising non-volatile memory technologies under active development. They are being considered yet another serious alternative to disk drives as much as NAND type flash memory is. In this paper, we propose a new In-Page Logging (IPL) design that uses PCRAM, a leading candidate of non-volatile memory technologies, as a storage medium for log records. The low latency and byte-addressability of PCRAM can make up for the limitations of flash-only IPL. The preliminary evaluation results show that the proposed design can accelerate the performance of IPL significantly even with the contemporary PCRAM products that do not deliver the promised performance for write operations as yet. This work provides a model case of hybrid storage design based on flash memory and PCRAM.

Acknowledgment

This work was partly supported by MEST, Korea under NRF Grant (No.2010-0025649) and NRF Grant (No.2010-0026511). This work was also sponsored in part by the U.S. National Science Foundation Grant IIS-0848503. The authors assume all responsibility for the contents of the paper.

References

[1] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang. SSD Bufferpool Extensions for Database Systems. *Proceedings of the VLDB Endowment*, 3(2), 2010.

[2] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better I/O Through Byte-Addressable, Persistent Memory. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[3] R. F. Freitas. Storage Class Memory: Technology, Systems and Applications (invited talk). In *Proceedings of ACM SIGMOD*, 2009.

[4] Intel Corp. Understanding the Flash Translation Layer (FTL) Specification. Application Note AP-684, Dec. 1998.

[5] Intel Corp. OLTP performance comparison: Solid-state drives vs. hard disk drives. Test report, Jan. 2009.

[6] International Technology Roadmap for Semiconductors. Process Integration, Devices, and Structures. 2009 Edition.

[7] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 2–13, June 2009.

[8] K.-J. Lee et al. A 90 nm 1.8 V 512 Mb Diode-Switch PRAM With 266 MB/s Read Throughput. *Solid-State Circuits, IEEE Journal of*, 43(1):150–162, Jan. 2008.

[9] S.-W. Lee and B. Moon. Design of Flash-Based DBMS: An In-Page Logging Approach. In *Proceedings of ACM SIGMOD*, June 2007.

[10] S.-W. Lee, B. Moon, and C. Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 863–870, 2009.

[11] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A Case for Flash Memory SSD in Enterprise Database Applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1075–1086, 2008.

[12] Micron Technology, Inc. NAND Flash 101: An Introduction to NAND Flash and How to Design It into Your Next Product (Rev. B). Technical Note TN-29-19, Apr. 2010.

[13] Numonyx. Omneo P8P 128-Mbit Parallel Phase Change Memory. Data Sheet 316144-06, Apr. 2010.

[14] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the 36th annual international symposium on Computer architecture (ISCA)*, 2009.

[15] Samsung Electronics. 2G x 8 Bit NAND Flash Memory (K9F8G08U0M). Data sheet, 2007.

[16] Samsung Electronics. 4Gb A-die DDR3L SDRAM (K4B4G0446A). Data sheet(rev. 1.01), Nov. 2010.

uFLIP: Understanding the Energy Consumption of Flash Devices

Matias Bjørling
IT University of Copenhagen
Copenhagen, Denmark

Philippe Bonnet
IT University of Copenhagen
Copenhagen, Denmark

Luc Bouganim
INRIA and U.Versailles
Le Chesnay, France

Bjørn Þór Jònsson
Reykjavik University
Reykjavik, Iceland

Abstract

Understanding the energy consumption of flash devices is important for two reasons. First, energy is emerging as a key metric for data management systems. It is thus important to understand how we can reason about the energy consumption of flash devices beyond their approximate aggregate consumption (low power consumption in idle mode, average Watt consumption from the data sheets). Second, when measured at a sufficiently fine granularity, the energy consumption of a given device might complement the performance characteristics derived from its response time profile. Indeed, background work which is not directly observable with a response time profile appears clearly when energy is used as a metric. In this paper, we discuss the results from the uFLIP benchmark applied to four different SSD devices using both response time and energy as metric.

1 Introduction

Energy efficiency is emerging as a major issue for data management systems, as power and cooling start to dominate the cost of ownership [2]. The key issue is to tend towards energy proportionality [1]—systems whose energy consumption is a linear function of the work performed, ideally with no energy consumed in idle mode. In this context, Tsirogiannis et al. [9] have argued that flash devices are very promising components. But, are there significant difference across flash devices? How much does power consumption depend on the IO patterns that are submitted? How stable are power measurements for a given device? Basically, how can we reason about the energy consumption of flash devices? This is the topic of this paper.

The first issue is: *How to measure energy efficiency?* At the scale of a data center[1], a few metrics have been introduced to measure and compare energy efficiency (e.g., Power Usage Effectiveness = total power for the facility / power for the data center). At the scale of a server or its components, the relevant metrics are the electrical power—expressed in Watts (W)—, or the energy—expressed in Joules (J)—drawn by the system. In this paper, we advocate measurements with a high temporal resolution (100 MHz sampling rate) that allow us to reason about the behavior of a flash device.

Copyright 2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] See e.g., http://www.google.com/corporate/green/datacenters/measuring.html for some reference measurements

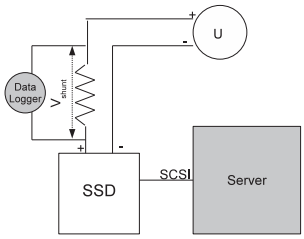


Figure 1: Diagram of the current shunt insertion set-up for measuring SSD energy consumption.

The second issue is: *What to measure?* We have designed a benchmark, called uFLIP, to cast light on all relevant usage patterns of current, as well as future, flash devices [5, 3]. uFLIP is a set of nine micro-benchmarks based on IO patterns, or sequences of IOs, which are defined by 1) the time at which the IO is submitted, 2) the IO size, 3) the IO location (logical block address, or LBA), and 4) the IO mode, which is either read or write. Each micro-benchmark is a set of experiments designed around a single varying parameter, that affects either time, size, or location. In [5], we measured and summarized the response time for individual IOs. In this paper, we focus on the energy profile of each experiment. While energy consumption cannot be traced to individual IOs, we can associate energy consumption figures to IO patterns, which helps us understand further the behavior of the devices.

Previous work covers different aspects of flash devices energy consumption: e.g., Tsirogiannis et al. focused on server power break-down resulting in key insights about the balance between idle and active modes, as well as CPU, hard disks and flash-based SSDs [9]; Mohan et al. [6] devised an analytical model for the power consumption of a flash chip (not a complete flash device); and Park et al. [7] focused on the design of low-power flash-based solid state drives (SSD). The study most similar to ours in its goals was conducted by Seo et al. [8]. They performed a few micro- and macro-benchmarks on three SSD devices as well as two hard drives. The micro-benchmarks focus on sequential reads, random reads, sequential writes and random writes as a function of IO size; while the macro-benchmarks are derived from file systems benchmarks. They use a multimeter to read the power value at 1KHz. The results exhibit a similar behavior between the three SSDs, which differs from the HDDs' behavior. The energy profile of each SSD corresponds to its throughput profile. The authors suggest that background work is specially important to explain the energy cost of random writes, but no details are given. In contrast, our study focus on a complete range of micro-benchmarks (the uFLIP benchmark), with a sampling rate of 100 MHz and a thorough study of the cases where the energy profile does not correspond to the performance profile. We show that there exist significant differences among flash devices in terms of energy profile.

Our contribution is the following: (1) we describe a set-up for measuring energy consumption of flash devices at high resolution; (2) we present the result of the uFLIP benchmark, using energy as a metric, for four flash-based solid state drives (SSD); and (3) we derive some lessons learned from our experiments.

2 Energy Measurement Set-Up

Our goal is to reason about power consumption at a fine granularity. For our measurements, we rely on shunt insertion: we measure the voltage across a shunt, which is proportional to the current flowing through the flash device. We use an oscilloscope equipped with a data logger to sample the voltage. The diagram in Figure 1 illustrates our set-up. The current shunt is installed on the high-side of an independent power source to guarantee stable voltage, the flash device being connected to the server via the data lines of the SATA connector.

We use a 1 Ohm resistor (±5% error) as a shunt, and we sample voltage at 100 MHz, i.e., at the order of ten μsec, which allows us to oversample the flash SSD IOs that are performed in tens of μsec. This does

SSD	Capacity (GB)	Idle (W)	Active (W)	Mean Idle Measured (W)	Std Dev Idle Measured
Memoright	32	1	2,5	0,96	0,3
MTron	16	0,5	2,7	0,99	0,05
Intel X25-E	32	0,6	2,4	0,54	0,05
Intel X25-E	64	0,6	2,6	0,57	0,05

Table 3: Capacity, advertised power characteristics (from the data sheet) and measured consumption in idle state

not give us the energy consumption of each IO, but we are working at the appropriate resolution to reason about the evolution of energy consumption in time. We obtain the energy consumed between two consecutive measurements E_t in Joule as follows: $E_t = (V_{shunt}(t)/R) * (U - V_{shunt}(t)) * \Delta$, where $V_{shunt}(t)$ is the voltage that we measure with the oscilloscope at instant t (in V), R is the resistor (in Ohm), U is the constant voltage provided by the power supply and Δ is the time elapsed between two consecutive measurements. We obtain the energy spent during an experiment by summing the relevant E_t.

We cannot rely on the response time measured in uFLIP to determine the duration of a run, as some work might be performed by the SSD after the last IO has completed. Indeed, this is one of the phenomenon we are interested in. We thus have to determine the beginning and end of a run from the energy trace. To do so, we first characterize the idle state for each device as a baseline. For each uFLIP run, we start collecting data approximately one second before the first IO is submitted and a few seconds after the last IO has completed to capture eventual asynchronous behavior. We use robust statistics to detect significant deviations from the idle state.

3 Results

We analyzed energy measurements from four different SSDs (see Table 1). The MTron SSD is from the first generation of devices on the market, while the device from Memoright and the two devices from Intel are more recent. Based on our work with uFLIP, we know that it is very hazardous to generalize our results to all SSDs. At best, these SSDs are representatives from interesting classes of flash devices.

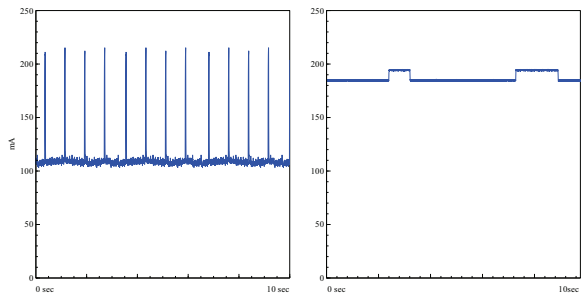


Figure 2: Intel SSD (left) and Memoright (right) in idle state

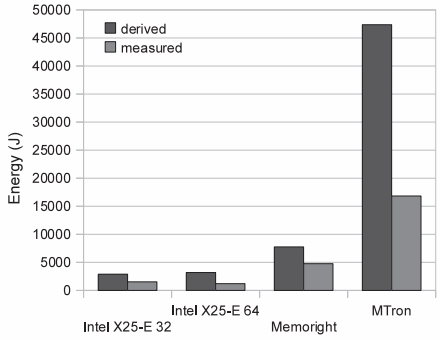


Figure 3: Total power consumption by device for the uFLIP benchmark

3.1 Idle State

In idle state, a flash device is not serving any IOs. An ideal device would not consume any energy in that state. In practice, flash devices consume some energy when idle, but much less than hard disks which make

them attractive in the contextof energy efficient systems [9, 2]. The product data sheet give a typical power consumption in idle mode.

As explained in Section 2, we measure current draw in idle mode for all devices in order to establish a baseline for our uFLIP measurements. These results are summarized in Table 1. We make two observations. First, the measured results are not far from the figures from the data sheets (except for the MTron). Second, we observe that the Intel and Memoright devices actually perform regular work in idle mode, as Figure 2 illustrates, while the MTron does not (not shown).

3.2 uFLIP Energy Results

Figure 3 presents an overview of the total energy consumed while executing the uFLIP benchmark (1074 runs of hundreds of IOs). We compare the energy consumption actually measured with an energy estimate derived as the time it takes uFLIP to complete multiplied by the active power figure from the data sheet (see Table 1). We make two observations. First, the derived results are off by a factor of two or three. The derived energy consumed is consistently higher than the measured energy. Second, the energy consumed for the execution of the benchmark varies by an order of magnitude between the most efficient device in our study (Intel X25-E 64GB) and the least efficient (MTron).

uFLIP is composed of 9 micro-benchmarks. The most straightforward micro-benchmark (called *granularity*) concerns the performance of baseline IO patterns: sequential reads (SR), sequential writes (SW), random reads (RR), random writes (RW) as a function of IO size. We detail the results for this micro-benchmark in the rest of this section. Note that the scale of the y-axis is different for each device.

Figure 4 (a) and (b) show the response time and energy profiles for the Intel X25-E 64 GB. The results are similar for both Intel devices in our study, so we only show one model. With the Intel SSDs, the RR pattern utilizes more power than all other patterns—write included. It confirms the measurement based on response time and shows that write performance is obtained at the cost of suboptimal reads. Indeed, the high energy consumption for RR reveals a significant amount of work, which is not dictated by the characteristics of the underlying flash chips.

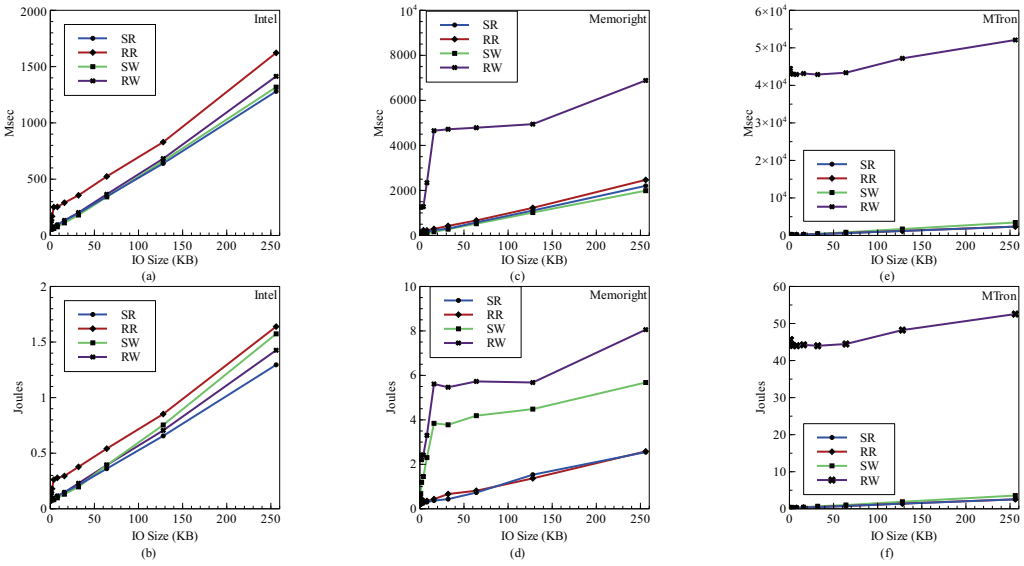


Figure 4: Time and Energy Profiles for the Intel X25-E 64GB (a),(b); Memoright (c),(d) and MTron (e),(f) for baseline patterns

Figure 4 (c) and (d) show the response time and energy profiles for the Memoright SSD. We can see that RW require additional work compared to other operations. While the energy required by SW is 50% lower than the energy required by RW, response time is five times faster. Obviously, the Memoright performs some work to hide the latency of SW. While SW response time is proportional to IO size, the energy profile reveals that energy-efficient SW should be done at the largest granularity.

Figure 4 (e) and (f) show the response time and energy profiles for the MTron SSD. Three interesting characteristics emerge from these graphs. First, the cost of sequential and random reads is very similar in terms of time and energy. Second, SW uses both more time and power than reads. Third RW are very slow but increase slowly with IO size. This means that even if the disk only writes 512 bytes, it is still doing nearly the same work as it would have for writing 256 KB.

This simple micro-benchmark exhibits significant differences between three classes of flash devices. In the next section, we explore further what the energy profiles tell us about the flash devices.

SSD	Memoright		MTron		Intel	
Category	SW	RW	SW	RW	SW	RW
Granularity	GWP		NS	S	S	None
Alignment	Ex		S		None	
Locality	GWP		NS	S	None	GWP
Partitioning	Ex		NS	GWP	None	S
Order	GWP		NS	GWP	None	GWP
Parallelism	Ex		NS		S	
Mix	GWP		S		None	GWP
Pause	Not applicable					
Bursts	Not applicable					

Table 4: Category of background work that depends on the submitted IO patterns. Ex - Extended, GWP - Grows with parameter, NS - Not significant, S - Small are the type of background work observed on the 9 uFLIP micro-benchmarks (see [5] for a complete description of these micro-benchmarks).

3.3 Device Characterization

One of the benefits of the energy profile is that it can reveal asynchronous activity, i.e., background work whose influence on response time is indirect. We identify three forms of background work[2]:

1. Background work independant of the activity of the SSD. We have seen in Section 3.1 that the Intel and Memoright devices exhibited a form of regular background work in idle state, while the MTron SSD does not.

2. Background work performed systematically whenever IOs are performed, regardless of the nature of those IOs—in our measurements this appears as a tail on each measurement. The MTron and MemoRight exhibit such a tail on every run throughout the benchmark, while the Intel devices do not. Figure 5 shows these tails on an example run (SR baseline pattern).

3. Background work that depends on the submitted IO patterns. A close look at the results from the numerous experiments revealed background work based on the submitted IO pattern. A first observation is that no device performed additional background work for experiments where only read operations where performed. We thus focused on experiments that contain writes and we classified the background work

[2]We refer the interested readers to [4] for more details about background work.

we observed in five categories: Extended (*Ex* - several seconds independantly of the micro-experiment parameter); Small (*S* - subsecond duration of background work independantly of the micro-experiment parameter); Grows with parameter (*GWP* - duration of background work increases regularly as a function of the micro-experiment parameter), and not significant (*NS*). Table 2 summarizes the type of background work per device and per micro-benchmark (see [5] for a complete description of each micro-benchmark).

Interestingly, Table 2 explains some of the behavior we observed in Figure 4 for the granularity micro-benchmark. On Figure 4 (b), we can see that, for the Intel device, the energy cost of SW increases faster than the cost of other operations. Indeed, SW trigger background work, while RW do not. On Figure 4 (d), for the Memoright device, we observed that the energy cost of SW is much higher than what could be deduced from the response time profile. Table 2 shows that both SW and RW trigger background work that grows with the IO size. A closer look reveals that the background work performed for SW lasts consistently much longer than the background work triggered by RW as illustrated in Figure 6 for an IO size of 128 KB.

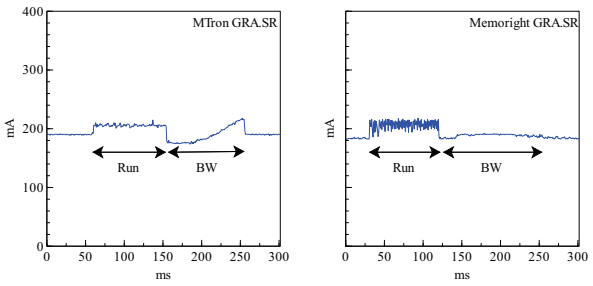


Figure 5: Systematic background work (BW) for MTron (left) and Memoright (right)

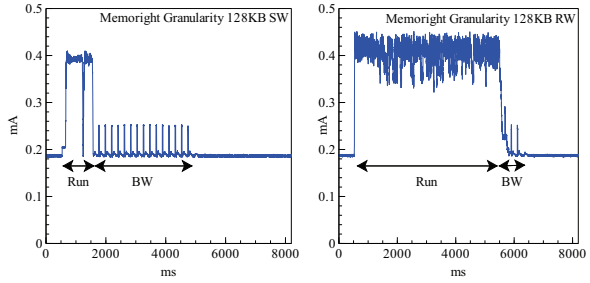


Figure 6: Background work (BW) for the granularity benchmark on Memoright SSD for SW (left) and RW (right) with an IO size of 128 KB.

3.4 Discussion

Measure. Do not simulate.. Flash devices are complex devices with layers of undocumented software and hardware. In particular, the Flash Translation Layer that is responsible for block management, wear leveling and error correction is a black box for DBMS designers. There is no straightforward way to define flash device model neither in terms of response time [3], nor in terms of energy consumption. The results from this section show that measuring response time and deriving energy based on the power figures from the data sheet is not an accurate method for estimating power consumption. Until we understand the characteristics of flash devices, the only way to reason about their energy consumption is to measure it.

Energy profiles reveal interferences. The energy profiles might reveal patterns of background work—which may or may not depend on the submitted IOs. This is very useful information for us in the context of the uFLIP benchmark. Indeed, we can rely on this information to plan the 1074 runs of a uFLIP execution so that we avoid interferences and minimize the pause in between runs. As a complement to the methodology we presented in [5], we can measure the energy consumption for baseline patterns and deduce the type of pause that we must introduce: minimal pause for read runs, pause depending on the nature of the run so that we can guarantees that there is no interference in between write runs.

No energy proportionality. The current generation of SSD is not energy-proportional. First, the devices that we studied consume energy in idle mode—some devices even perform systematic background work in idle mode. Second, flash devices implement trade-offs that require extra energy for some IO patterns (e.g., RW on

MTron and Memoright or RR on Intel). Third, all devices implement a form of background work that is not compatible with energy proportionality. When that is said, the Intel devices come pretty close.

4 Conclusion

Our goal with this study was to get a better understanding of flash devices. We devised a set-up to perform high-resolution energy measurements. We ran the uFLIP benchmark. Our measurements cast some lights on the nature of background work, on the accuracy of energy estimates, and on the significant differences between classes of flash devices. Whether flash devices will evolve into energy proportional devices is an open issue. At least, flash devices should have a well-defined energy profile that can be leveraged to build energy efficient data management systems.

References

[1] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, 2007.

[2] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[3] Matias Bjørling, Lionel Le Folgoc, Ahmed Mseddi, Philippe Bonnet, Luc Bouganim, and Björn Þór Jónsson. Performing sound flash device measurements: some lessons from uFLIP. In *SIGMOD Conference*, pages 1219–1222, 2010.

[4] Matias Bjørling Energy Consumption of Flash-based Solid State Drives Technical report, IT University of Copenhagen, 2010.

[5] Luc Bouganim, Björn Þór Jónsson, and Philippe Bonnet. uFLIP: Understanding flash IO patterns. In *CIDR*, 2009.

[6] Vidyabhushan Mohan, Sudhanva Gurumurthi, and Mircea R. Stan. Flashpower: A detailed power model for nand flash memory. In *DATE*, pages 502–507, 2010.

[7] Jinha Park, Sungjoo Yoo, Sunggu Lee, and Chanik Park. Power modeling of solid state disk for dynamic power management policy design in embedded systems. In Sunggu Lee and Priya Narasimhan, editors, *Software Technologies for Embedded and Ubiquitous Systems*, volume 5860 of *Lecture Notes in Computer Science*, pages 24–35. Springer Berlin / Heidelberg, 2009.

[8] Euiseong Seo, Seon-Yeong Park, and Bhuvan Urgaonkar. Empirical analysis on energy efficiency of flash-based SSDs. In *HotPower*, 2008.

[9] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 231–242, New York, NY, USA, 2010. ACM.

Call for Participation

The **27th IEEE International Conference on Data Engineering** addresses research issues in designing, building, managing, and evaluating advanced data-intensive systems and applications. It is a leading forum for researchers, practitioners, developers, and users to explore cutting-edge ideas and to exchange techniques, tools, and experiences. The mission of the conference is to share research solutions to problems of today's information society and to identify new issues and directions for future research and development work.

Keynotes

- Anastasia Ailamaki (EPFL, Switzerland)
- Johannes Gehrke (Cornell University, USA)
- Georg Gottlob (Oxford University, UK)

Workshops

- 6th International Workshop on Self Managing Database Systems (SMDB 2011)
- 1st International Workshop on Managing Data Throughout its Lifecycle (DaLi 2011)
- 2nd International Workshop on Graph Data Management: Techniques and Applications (GDM 2011)
- 3rd Workshop on Hot Topics in Software Upgrades (HotSWUp 2011)
- 1st International Workshop on Data Engineering Applications in Emerging Areas: Health Care Informatics and Energy Informatics (DEAEA 2011)
- 2nd International Workshop on Data Engineering meets the Semantic Web (DESWeb 2011)
- 1st International Workshop on Data Management and Analytics for Semi-Structured Business Processes (DMA4SP)
- ICDE 2011 PhD Workshop

Venue

ICDE 2011 will be held in the Hannover Congress Centrum, Hannover, Germany. Hannover is the capital of the federal state of Lower Saxony (Niedersachsen), Germany. Hannover is known as a trade fair city (e.g. CeBIT) but also for their Royal Gardens of Herrenhausen and many other places of interest.

For more information please regular visit the conference web site:

http://www.icde2011.org/

Microsoft hp YAHOO! LABS

IEEE computer society

Leibniz Universität Hannover

L3S

DFG

IEEE Technical Committee on Data Engineering

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398