System Co-Design and Data Management for Flash Devices

Philippe Bonnet
IT University of Copenhagen
Denmark
phbo@itu.dk

Luc Bouganim
INRIA and University of Versailles
France
Luc.Bouganim@inria.fr

Ioannis Koltsidas
IBM Research, Zurich
Switzerland
iko@zurich.ibm.com

Stratis D. Viglas
School of Informatics, University of Edinburgh
United Kingdom
sviglas@inf.ed.ac.uk

ABSTRACT

Flash devices are emerging as a replacement for disks. How does this evolution impact the design of data management systems? While flash devices have been available for years, this question is still open. In this tutorial, we share two views on the development of data management systems for flash devices. The first view considers that flash devices introduce so much complexity that it is necessary to reconsider the strictly layered approach between storage system, operating system and data management system. The second view considers that data management systems should recognize the complexity of flash devices and leverage the characteristics of different classes of devices for different usage patterns. Throughout the tutorial, we will cover the data management stack: from the fundamentals of flash technology, through storage for database systems and the manipulation of flash-resident data, to query processing.

1. SYSTEM CO-DESIGN

Since the advent of Unix, the stability of disks characteristics and interface have guaranteed the timelessness of major database system design decisions, i.e., pages are the unit of IO; random accesses are avoided.

Today, the quest for energy proportional systems and the growing performance gap between processors and magnetic disk performance are pushing flash devices as replacements for disks. Indeed, flash devices rely on tens of flash chips wired in parallel that together can deliver hundreds of thousands accesses per second with low energy consumption. Flash devices embed a complex software called Flash Translation Layer (FTL) in order to hide flash chip constraints (erase-before-write, limited number of erase-write cycles, sequential page-writes within a flash block). A FTL provides address translation, wear leveling and strives to hide the impact of updates and random writes based on observed update frequencies, access patterns, temporal locality.

This trend towards flash devices has created a mismatch between the simple disk model that underlies the design of today's database systems and the complex flash devices of today's computers. This mismatch results in sub-optimal IO performance, which is costly both in terms of throughput and energy consumption. In fact, a tension exists between the design goals of flash devices and DBMS. Flash device designers aim at hiding the constraints of flash chips to compete with hard disks providers. They also compete with each other, tweaking their FTL to improve overall performance, and masking their design decision to protect their advantage. Database designers, on the other hand, have full control over the IOs they issue. What they need is a clear and stable distinction between efficient and inefficient IO patterns to produce a stable (re)design of core database techniques. They might even be able to trade increased complexity for improved performance and stable behavior across devices.

The goal of the first part of this tutorial is to offer database researchers and practitioners an insight into flash chip management as well as a survey of the constraints and opportunities it creates for database system or algorithm designers. We will stress the need for a tighter form of collaboration between database system, operating system and FTL to reconcile the complexity of flash chip management with the performance goals of a database system.

2. DATA MANAGEMENT

In the near future, commodity and enterprise-level hardware is expected to incorporate both flash Solid State Drives (SSDs) and magnetic disks as storage media. In light of this, fundamental principles of data management need to be revisited, as all existing database systems and algorithms have been designed for disks consisting of rotating platters.

However, the term SSD incorporates multiple classes of device. The only major common characteristic of all these devices is their excellent random read performance. The remaining characteristics range within more than two orders of magnitude across different devices. Some SSDs are more than an order of magnitude slower than disks at random writes, while other SSDs dominate disks in both random read and write throughput and latency. The most important

Proceedings of the VLDB Endowment, Vol. 4, No. 12

question to be answered is what is the best use of each class of device in a DBMS. Equally important is how this question can be answered automatically, by the DBMS itself, without administrator intervention. The answer also depends on the amount of main memory available and the number, size, rotational speed and RAID configuration of the underlying disks. Possible answers are (*a*) using the SSD as persistent storage, either in combination with disks or only by itself, (*b*) using the SSD as a read cache for the HDDs, as a write cache or as a combined read-write cache, (*c*) using the SSD as a transactional log, (*d*) using the disk as a log-structured write cache for the SSD, (*e*) using the SSD as a temporary buffer for specific query evaluation algorithms (e.g., sorting), and, of course, (*f*) any combination of the above.

The aim of the second part of the tutorial is to present the challenges that arise when flash technology is introduced in a database system context; the recent results in this fresh research area; and an outlook of existing problems and things to come.

3. REFERENCES

[1] D. Agrawal *et al.* Lazy-adaptive tree: An optimized index structure for flash devices. *Proc. VLDB Endow.*, 2(1):361–372, 2009.

[2] N. Agrawal *et al.* Design tradeoffs for SSD performance. In *USENIX Annual Technical*, pages 57–70, 2008.

[3] P. A. Bernstein *et al.* Hyder - a transactional record manager for shared flash. In *CIDR*, 2011.

[4] M. Bjørling *et al.* Performing sound flash device measurements: some lessons from uFLIP. In *SIGMOD*, pages 1219–1222, 2010.

[5] M. Bjørling *et al.* Understanding the energy consumption of flash devices with uFLIP. *IEEE Data Eng. Bull.*, pages 1–7, 2010.

[6] P. Bonnet and L. Bouganim. Flash device support for database management. In *CIDR*, 2011.

[7] L. Bouganim *et al.* uFLIP: Understanding flash IO patterns. In *CIDR*, 2009.

[8] M. Canim *et al.* An object placement advisor for DB2 using solid state storage. *Proc. VLDB Endow.*, 2(2):1318–1329, 2009.

[9] M. Canim *et al.* SSD bufferpool extensions for database systems. *Proc. VLDB Endow.*, 3(2):1435–1446, 2010.

[10] B. Debnath *et al.* Flashstore: High throughput persistent key-value store. *Proc. VLDB Endow.*, 3(2):1414–1425, 2010.

[11] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Comput. Surv.*, 37(2):138–163, 2005.

[12] J. Gray. Tape is dead, disk is tape, flash is disk. `http://research.microsoft.com/en-us/um/people/gray/talks/Flash_is_Good.ppt`, 2006.

[13] A. Gupta *et al.* DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS*, 2009.

[14] A. Holloway. Adapting database storage for new hardware. PhD Thesis. University of Wisconsin - Madison, 2009.

[15] X.-Y. Hu *et al.* Write amplification analysis in flash-based solid state drives. In *SYSTOR*, pages 10–18, 2009.

[16] H. Kim and S. Ahn. BPLRU: a buffer management scheme for improving random writes in flash storage. In *FAST*, 2008.

[17] J. Kim *et al.* A space-efficient flash translation layer for CompactFlash systems. *Trans. on Consumer Electronics.*, pages 366–375, 2002.

[18] Y. Kim *et al.* Flashsim: A simulator for nand flash-based solid-state drives. In *SIMUL*, pages 125–131, 2009.

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

[20] I. Koltsidas and S. D. Viglas. Designing a flash-aware two-level cache. In *ADBIS*, 2011.

[21] S.-W. Lee and B. Moon. Design of flash-based DBMS: An in-page logging approach. In *SIGMOD*, pages 55–66, 2007.

[22] S. Lee *et al.* LAST: locality-aware sector translation for NAND flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, 2008.

[23] S.-W. Lee *et al.* A log buffer-based flash translation layer using fully-associative sector translation. *Trans. on Embedded Comp. Sys.*, 2007.

[24] S.-W. Lee *et al.* A case for flash memory SSD in enterprise database applications. In *SIGMOD*, pages 1075–1086, 2008.

[25] A. Leventhal. Flash storage memory. *Commun. ACM*, 51(7):47–51, 2008.

[26] D. Ma *et al.* LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *SIGMOD*, pages 1–12, 2011.

[27] D. Narayanan *et al.* Migrating server storage to SSDs: analysis of tradeoffs. In *EuroSys*, pages 145–158, 2009.

[28] S. Nath and P. B. Gibbons. Online maintenance of very large random samples on flash storage. *Proc. VLDB Endow.*, 1(1):67–90, 2008.

[29] S. Nath and A. Kansal. FlashDB: Dynamic Self-Tuning Database for NAND Flash. In *IPSN*, pages 410–419, 2007.

[30] X. Ouyang *et al.* Beyond Block I/O : Rethinking Traditional Storage Primitives. In *IEEE HPCA*, pages 301–311, 2011.

[31] A. Rajimwale *et al.* Block management in solid-state devices. In *USENIX Annual Technical*, 2009.

[32] F. Shu and N. Obr. Data set management commands proposal for ATA8- ACS2. `http://www.t13.org/`, 2007.

[33] G. Soundararajan *et al.* Extending SSD lifetimes with disk-based write caches. In *FAST*, 2010.

[34] R. Stoica *et al.* Evaluating and repairing write performance on flash devices. In *DAMON*, pages 9–14, 2009.

[35] M. Stonebraker. Operating system support for database management. *Commun. ACM*, 24(7):412–418, 1981.

[36] D. Tsirogiannis *et al.* Query processing techniques for solid state drives. In *SIGMOD*, pages 59–72, 2009.

[37] X. Wu and A. N. Reddy. Exploiting concurrency to improve latency and throughput in a hybrid storage system. *MASCOTS*, pages 14–23, 2010.