SSD as SQLite Engine

Soyee Choi
SungKyunKwan University
Republic of Korea
ithdli@skku.edu

ACM Reference Format:
Soyee Choi. 2018. SSD as SQLite Engine. In *Proceedings of 2018 International Conference on Management of Data (SIGMOD'18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3183713.3173720

1 INTRODUCTION

As a proof-of-concept for the vision "SSD as SQL Engine" (**SaS** in short), we demonstrate that SQLite [4], a popular mobile database engine, in its entirety can run inside a real SSD development platform. By turning storage device into database engine, **SaS** allows applications to directly interact with full SQL database server running inside storage device. In **SaS**, the SQL language itself, not the traditional dummy block interface, will be provided as new interface between applications and storage device. In addition, since **SaS** plays the role of the unified platform of database computing node and storage node, the host and the storage need not be segregated any more as separate physical computing components.

Background During the last decade flash memory SSDs have relentlessly been replacing harddisks as the main storage because of several advantages such as fast latency, high IOPS/$ and low power consumption, especially in data centers running OLTP workloads [7]. Despite of its different characteristics from harddisk, short latency, and the consequent numerous opportunities for optimizations, however, SSDs are mostly used as faster harddisks. Therefore, the data-intensive computing of database servers are conducted in host, totally segregated from the SSD storage. Further, between the database engine and the storage are barriered by the heavy IO stacks of file system and OS kernel, which are the legacies developed in the era of slow harddisks and thus are not suitable to the era of SSDs.

Problems The dichotomy of host and storage in modern computer architecture has forced the data-intensive softwares to naturally segregate their data processing from the storage. When SSDs are used as the data storage, however, this design of segregating the data processing logic far from the storage is not desirable at least for three reasons: 1) the legacy IO stack overhead, 2) difficulties in vertical optimization, and 3) inefficiency in architectural and economical perspectives. We briefly explain each in turn. First, while the IO stack overhead in the existing file systems and kernels marginally contributes to the latency of data access in harddisks (e.g., less than 3%), it is commonly said to be very significant in SSDs with short data access latency (e.g., more than 30% in the case of SSD from Samsung) [10, 12]. Second, while there are many opportunities for vertical optimization between database engines and SSD [6, 8, 11], the intervening IO stacks combined with the dummy block IO interface make it extremely difficult to make them to be seamlessly integrated. Although an effective and practical vertical optimization solution is successfully embodied on top of a specific file system and kernel, it is still hard to become a generally available solution unless other file systems and kernels support the same implementation. Last, the physical separation of host and storage is not economical or architecturally elegant because bigger physical space is required to accommodate them, more power is consumed, and the storage interface device is necessary.

Motivation Any contemporary SSD itself is a computer equipped with powerful CPU with several cores and large memory, which is enough[1] to run enterprise-class database servers. Though challenging and looking radical, the offloading of a complete SQL database engine into SSD would be a promising approach to obviate the problems due to the barriers of file system and IO stack disturbing between database engine and storage. That is, the offloading moves database engine very close to the core firmware called FTL (flash translation layer) and the data in flash memory chips. This in turn enables that both of database engine and storage are tightly coupled through lightweight interfaces, thus bypassing the heavy IO stacks and also exploring various vertical optimizations between two layers much more easily and flexibly.

2 PROPOSED APPROACH AND UNIQUENESS

SSD as SQL Engine Based on the above motivation, we envision the idea of "SSD as SQL engine" (**SaS**). That is, SSD itself plays not only the conventiontional role of storage but also the role of SQL engine, and database applications in other computing nodes (e.g., Apache memcache) will now interact with **SaS** using the SQL interface over the network. When the idea of **SaS** is realized, the distinction of host and storage will be blurring and the segregated host and storage will be replaced by one **SaS** device. In this sense, the vision of **SaS** is an opposite extreme of the existing segregation of database processing from storage. However, it is a daunting task to realize the concept of **SaS** using enterprise-class database engine such as MySQL/InnoDB engine. For this reason, we have prototyped a simple version of **SaS**, "SSD as SQLite engine". As is detailed in Section 3, the SQLite database is a good candidate for prototyping **SaS** with moderate endeavour, for several reasons. First, its codebase is tiny enough to fit in rather limited DRAM in real SSDs. Second, because its IO is based on simple

[1]Of course, it has to be admitted that that the performance of CPU and DRAM inside SSD is not yet comparable to that in the high-end servers.

well-abstracted VFS interface [2], VFS can be implemented by extending FTL in SSD minimally. Third and most importantly, there exists a good sample of vertical optimization in **SaS** [8].

Related Work and Uniqueness In the sense that SaS offloads the database engine to the storage and comes with new interfaces, at least two types of the existing work are closely related to SaS: new SSD interfaces for database applications [8, 11] and the in-storage-processing approach (ISP in short) [3, 5, 9]. But, both approaches are common in that only a specific core module or functionality is offloaded so that the main body of database engine still reside in host side and has to interact with the storage device. Therefore, they still suffer from the problems of the segregated architecture of the computing node and the storage node. In contrast, by offloading an SQL engine in its entirety to the storage, **SaS** allows to completely bypass the IO stack overhead in the existing approaches and also enables to pursue further optimizations because SQL engine is moved to close to the storage.

3 SSD AS SQLITE ENGINE

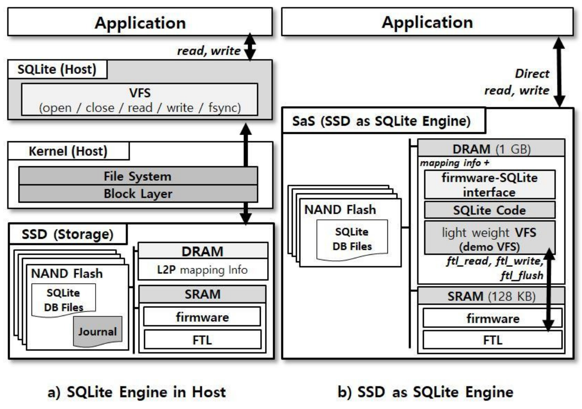

Figure 1: Two SQLite Processing Architectures

Implementation We have implemented a prototype of **SaS** using a real SSD, which has a Marvel controller with 400 MHz ARM9 processor, and the controller contains 128 KB SRAM to store the firmware and 1 GB DRAM to store metadata such as mapping tables. In our current implementation, the SQLite runtime binary and its heap memory reside in the DRAM area. The source code of a SQLite written in C (SQLite 3.13 amalgamation version) was offloaded down to the designated NAND flash memory area in the SSD. And, the boot loader of the SSD was modified to load the SQLite binary to a fixed DRAM location upon its booting. Thus, in our **SaS** prototype, the SQLite engine, rather than the FTL module, is mainly responsible for interfacing with the application using the SQL interface. The vanilla FTL in the SSD was extended to support the functionalities of a simple file system, that is, the `vfs` interface, so that the vanilla SQLite library is made easily portable to the SSD with minimal change. The resultant "SSD as SQLite Engine" architecture is presented in Figure 1(b), and, for comparison, the existing architecture of separated host and storage is also presented in Figure 1(a). Thanks to the vertical optimization exploiting the transactional atomicity in the extended FTL, our **SaS** can guarantee the transactional atomicity without resorting to the heavy journaling in the original SQLite. For this reason, note that the journal file is no more necessary in our **SaS** prototype. The SQLite buffer cache is now located in DRAM area, together with the L2P (logical-to-physical) address mapping information. The main goal in embodying the preliminary **SaS** prototype was the tight integration of the offloaded SQLite and the existing FTL firmware in the SSD. First of all, for SQLite to work, the `vfs` layer is a key OS abstraction heavily used in SQLite. Therefore, we had to implement `vfs`'s all the interfaces using the existing FTL layer in the SSD. Next, to support transactional atomicity without resorting to the costly journaling, the FTL firmware was extended to support atomic propagation of one or more pages to the flash memory chips, as in X-FTL [8].

Preliminary Evaluation Result In order to evaluate the effect of "SSD as SQLite Engine", we have evaluated its performance by running a simple update-intensive benchmark, AndroBench [1]. For comparison, we have also evaluated the performance of the SQLite on host using the same benchmark. The host machine used in the evaluation is a Linux system running on Intel core i7-860 2.8GHz processor. Our **SaS** prototype can complete the AndroBench benchmark 2x faster than the SQLite on host in terms of IO time. Also, in terms of the internal write amplification factor (WAF) measured while running the benchmark, our **SaS** prototype was 2.3x or more efficient than the SQLite on host. These results are consistent with the observation made in X-FTL work [8]. This impressive advantages of our **SaS** prototype can be explained mainly for three reasons. First, our **SaS** prototype can avoid the redundant journaling overhead so that it halves the amount of data to be written while executing the benchmark. Second, our **SaS** prototype bypasses the file system so that it does not cause additional IOs for file metadata which is not marginal in the SQLite on host. Finally, our **SaS** prototype does not suffer from the IO stack overhead while the original SQLite on host does.

4 SUMMARY AND FUTURE WORK

This paper proposes new radical database processing architecture, "SSD as SQL Engine" (**SaS**) and proves its feasibility by prototyping "SSD as SQLite Engine". In addition, through a primitive evaluation, we demonstrate its numerous benefits.

The preliminary work in this abstract is the very first step towards our vision of "SSD as SQL Engine". There are many challenging but intriguing technical issues ahead of us. To name a few:

- We will exploit further vertical optimizations which are now possible with the "SSD as SQL Engine" architecture.
- We will investigate whether new query processing and optimization techniques are possible under the **SaS** architecture.
- We will examine how to deal with concurrency. To support multiple users/processes, we have to support session management from **SaS** and each SQL interaction from different application has to be assigned with its transaction id.
- We will study how to design the SQL interfaces between applications and **SaS** device. For an instance, we have to work on how to effectively return a large amount of SQL result from the **SaS** prototype to applications.

ACKNOWLEDGEMENTS

This research was supported by IITP under the "SW StarLab" (IITP-2015-0-00314).

REFERENCES

[1] 2011. AndroBench (SQLite Benchmark). http://www.androbench.org/wiki/AndroBench. (2011).

[2] 2011. The SQLite OS Interface or VFS. https://sqlite.org/vfs.html. (2011).

[3] 2012. *White paper: A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server.* Technical Report. Oracle corp.

[4] 2017. Well-Known Users of SQLite. http://www.sqlite.org/\famous.html. (2017).

[5] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: A Framework for Near-data Processing of Big Data Workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 153–165. https://doi.org/10.1109/ISCA.2016.23

[6] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*.

[7] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh. 2014. Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases. In *Proceedings of the 40th SIGMOD International Conference on Management of Data (SIGMOD '14)*. 529–540.

[8] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Gi-Hwan Oh, and Changwoo Min. 2013. X-FTL: Transactional FTL for SQLite Databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, 97–108.

[9] Woods Louis, Istvan Zsolt, and Alonso Gustavo. 2014. Ibex—An Intelligent Storage Engine with Support for Advanced SQL Off-loading. *Proceedings of the VLDB Endowment* (2014), 963–974.

[10] D. Nellans M. Bjørling, J. Axboe and P. Bonnet. 2013. Linux block IO: introducing multi-queue SSD access on multi-core systems. In *Proceedings of the 6th International Systems and Storage Conference ACM.* p.22.

[11] Ravi Mayuram Kee. Yang-Suk Oh. Gihwan, Seo. Chiyoung and Lee. Sang-Won. 2016. SHARE Interface in Flash Storeage for Relational and NoSQL Database. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. 343–354.

[12] S.Swanson and A. Caulfield. 2013. *Refactor, reduce, recycle:Restructiong the I/O Stack for the Future of Storage.* Vol. 46. 52–59 pages.