A Flash-aware Random Write Optimized Database

Da Zhou
School of Information
Renmin University of China
Beijing 100872, China
Email: cadizhou@gmail.com

Xiaofeng Meng
School of Information
Renmin University of China
Beijing 100872, China
Email: xfmeng@ruc.edu.cn

***Abstract*—Solid State Drive (SSD), as new data storage media with low power consumption, high shock resistance and lightweight form, has been widely used in laptops during the past several years. The most attractive characteristic of SSD is its high random read speed because of no mechanical latency. However, poor random write performance becomes the bottle neck in wider applications. Random write is almost two orders of magnitude slower than both random read and sequential access. In our database prototype based on Oracle Berkeley DB (BDB), we propose to insert unmodified data into random write sequence in order to convert random writes into sequential writes, and then data sequence can be flushed at the speed of sequential write. Further, we optimize the write performance by reducing quantity of unmodified data to be inserted. In this demo, we visualize the progress and results when a write sequence is run on three database system. Besides this, we can select three kinds of benchmarks to run. After getting runtime of one benchmark on three database systems, we can replay their run progress in the same time. In this way, we can intuitively show the high performance of our database system.**

I. Introduction

SSD, as a new electronic storage device, is widely adopted in laptops during the past several years. This mainly benefits from its high read performance, especially random read performance. As we know, SSD does not have mechanical part like the magnetic head of HDD, therefore random read has similar speed with sequential read. This characteristic improves the read performance of system fundamentally. Besides this, SSD has other attractive characteristics, such as low power consumption, high shock resistance and lightweight form. All of these advantages make SSD as an outstanding data storage instead of HDD. However, the random write performance of SSD[4], especially small random write, is very poor in common. Random read and sequential access are faster than random write in two orders. The main reason is that small random writes often trigger erase operations. Erase operation is peculiar to flash memory which is storage media of SSD. In a word, erase operation has two characteristics: erase before rewrite and high cost. Besides this, large quantity of data need to be transferred during the course of erase operation. Existing database systems, such as Berkeley DB, are optimized according to characteristics of HDD, therefore they could not overcome the low random write performance of SSD.

In our database prototype, we propose a novel and efficient method to avoid random writes by converting random write sequence into sequential write sequence, therefore we can take full advantage of high sequential write speed. We novelly insert unmodified data items into the random write sequence, which locate between the lower and upper limit of the random write sequence. Finally we flush out the constructed sequential write sequence into SSD instead of original random write sequence. Compared with flushing random write sequence, writing the converted sequential write sequence has very low cost. Therefore, the write performance is enhanced obviously. When facing with data stream and low density, we propose cluster to improve write performance.

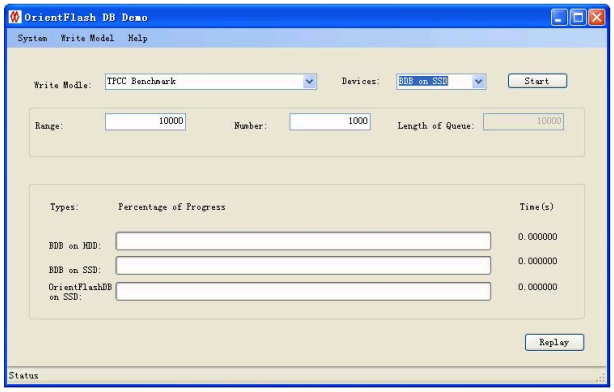

Fig. 1. Interface of Demo. Users can select different benchmarks to be run on different database systems. The runtime of different databases can be shown on the interface in the same time.

In this demo, we visualize the progress when a benchmark is executed on different databases as shown in Fig.1. We directly run Berkeley DB on HDD and SSD and our database prototype on SSD by selecting the *Devices*. When a benchmark is run on these three databases, the progress is visualized. Besides this, the results are shown on the interface in the same time. In this way, our demo provides an easy and intuitive way to show the performance our prototype system. In order to show the write performance dynamically, we replay the write progress of three databases in the same time by *Replay* button. It is easy to show the high performance of our DB prototype.

II. Demo Objectives

This demo visually shows the performance of Berkeley DB on HDD and SSD, and our prototype system on SSD under different benchmarks. Firstly, when a write sequence is executed in a database, this demo can show the progress of

978-0-7695-4048-1/10 $26.00 © 2010 IEEE
DOI 10.1109/MDM.2010.56

writing and record the runtime. Secondly, various benchmarks can be used to test the performance of databases by selecting $WriteModle$, such as random update model, TPCC, IOZone, TioBench and Postmark. Finally, we can replay the progress of one benchmark on three databases in the same time. According to this demo, the high write performance of our prototype can be shown intuitively.

III. System Architecture

In this section we firstly introduce the workflow of our demo, and then explain how our system optimize random write.

A. Workflow

There are two steps in our system. The first step generates random write sequence. In the second step, random write sequence is input into three different database systems as shown in Fig.2. The progress and time of writing are shown in the interface. After getting the runtime of all databases about the same benchmark, we can more intuitively show the comparison performance of databases by replaying the progress in the same time.

We can generate four types of benchmarks: random write model, TPCC, file system benchmarks and file IO benchmarks. As for random write model, users can input the value of parameters $range$ and $number$. They are used to decide the write times and range. As for TPCC, write sequence comes from the disk IO operations by running TPC-C on postgreSQL. We modify postgreSQL to record disk IO operations when executing the routines of writing data to disk. As for file system benchmarks and file IO benchmarks, we use blktrace[1] to trace the IO activities at the block level in order to get IO traffic of benchmarks to SSD in detail. After running benchmarks, we get the IO activities with help of blktrace.

B. Database prototype system

In prototype system, we improve write performance by converting random write sequence into sequential write sequence in order to take full advantage of high sequential write speed. We novelly insert unmodified data items into the random write sequence, which locate between the lower and upper limit of the random write sequence. We flush the converted write sequence instead of previous random write sequence. Therefore the cost of write will be reduced due to high sequential write performance.

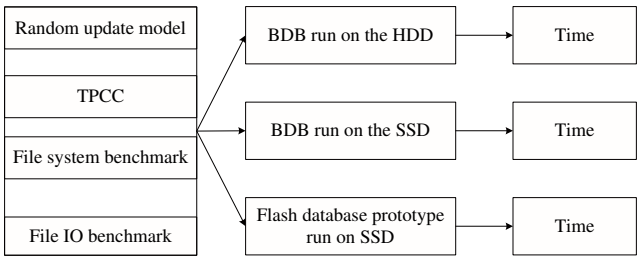

Fig. 2. The workflow of generating write sequence, selecting databases to be run and getting runtime.

Density is defined as the length ratio of random write sequence to converted sequential write sequence. Density is direct ratio to performance. In our method, we convert the random write sequence by inserting unmodified data items. The less the density is, the more unmodified data items are inserted, and the larger the cost of reading and flushing these data is. $Cluster$ is proposed to avoid low performance of our method when the density is less than the threshold. A sub-sequence is defined as a cluster if the density of it is larger than the threshold. Clusters obviously reduce the quantity of unmodified data to read, and then the costs of random read and sequential write decrease. Therefore, the performance of our method is higher.

Real applications usually generate writes continuously as a data stream. Write sequence from data stream has three characteristics: 1) New data items will be inserted into write sequence continually. 2) The density is usually not uniform. 3) The length of write sequence is limited by available cache. In this case, we first load random writes from data stream into write sequence, and then sort it when cache is consumed up. The number of write items is gotten from the parameter $Length$ as shown in Fig.1. Secondly clusters are generated from sorted write sequence. Lastly the clusters are flushed to SSD after being converted. This course cycles until the end of data stream.

IV. Demonstration

In our interface, we can select five benchmarks to run on three databases. Different benchmarks have different access modes. Therefore, the high performance of our system is more convinced. We can also input the number of $Length$ which is the length of write sequence.

A. TPCC Benchmark

As for TPCC, write sequence comes from the disk IO operations by running TPC-C on postgreSQL database system version 8.3.5. In the same way, we flush the write sequence into three databases. The length of write sequence varies from 10,000 to 100,000. According to Fig.3, BDB on SSD outperforms BDB on HDD only about 20%. This also shows Berkeley DB does not overcome the low random write performance of SSD. On the contrary, our system outperforms BDB on HDD about 100%, and BDB on SSD 40%. The high performance is obtained because our method converges random writes into high density clusters and flushes them in parallel.

B. Random write model

In this type of write model, every data items have the same frequency to be written. The parameter $range$ and $number$ is used to decide the range of data items and write times. The smaller the range is and the larger the number is, the higher performance our system is. Our system outperforms Berkeley DB from 50% to 150% at different $Length$ sequence.

C. PostMark

PostMark[2] is a new benchmark to measure performance for the ephemeral small-file regime used by internet software. We got the write sequence by running PostMark benchmark. After setting length of write sequence from 5,000 to 50,000, we run write sequence in three databases. Our system is faster than Berkeley DB by about 100% as a whole. When the length of write sequence decreases, the performance enhancement is more obvious. The reason is the IO activities are basically sequential in small sub-sequence, and then our database outperforms BDB about 300%.

D. IOzone

IOzone[5] is a file system benchmark tool. The benchmark generates and measures a variety of file operations, such as write, re-write, fwrite and aio_write. We got the write sequence by running IOzone. After setting length of write sequence from 1,000 to 10,000, our system is faster than BDB about 400% as a whole. The performance enhancement is very obvious. As for IOzone, the IO activities are random and converge in a limited range,such as 512MB. Therefore, every writes are random, and then the write cost of BDB is very high. However, in this scenario, converted sequential write sequence has high density. Therefore our method is taken full advantage of, and then performance is enhanced obviously.

E. Tiobench

Tiobench[3] is a multi-threaded file IO benchmark, which is used to measure file system performance in four basic operations: sequential/random read/write. Our system is faster than BDB about 300%. The reason is the multi-thread of Tiobench. IO activities are produced in wide area simultaneously because of multi-thread. Therefore, as for BDB, IO activities are random in wide area, and then the performance is very low. In OrientFlash DB, the random write sequence is divided into several sequential sequences as clusters. Therefore, the write performance is enhanced obviously.

F. Replaying

After running a write sequence in three databases, we can get the runtime of each databases. We can intuitively decide the performance of databases according to the results. In order to show the performance more obviously, we offer the *Replay* function. During the course of replaying, we dynamically simulate the progress of each databases according to their runtime. Because the runtime of our system is smallest, the progress bar of it progresses at highest speed as shown in Fig.4. According to the replay function, we can show the high performance of our system more intuitively.

V. Conclusion

SSD is applied more and more widely in laptops due to high random read speed. However, poor random write leads to the low performance of database on SSD. We novelly propose to convert random write sequence into sequential write sequence by inserting unmodified data items into write sequence. In this demo, we visualize the progress when a write sequence is run in three databases. Write sequence is gotten from different benchmarks. Besides this, we intuitively replay the write performance of different databases in the same time.

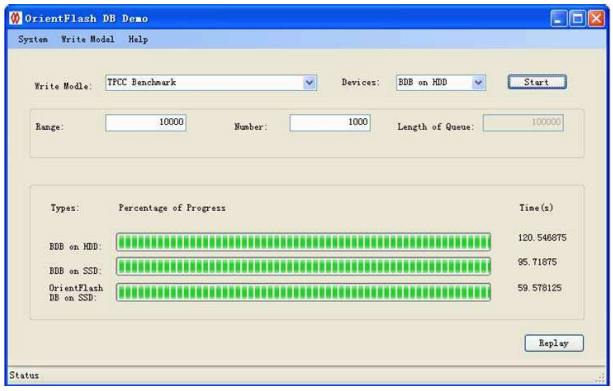

Fig. 3. The runtime of TPCC write sequence on three databases.

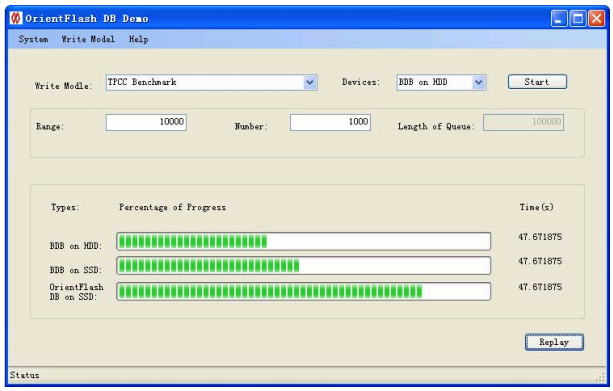

Fig. 4. Replaying the write progress of three databases in the same time after getting runtime of three databases on TPCC write sequence.

VI. Acknowledgment

We would like to thank Yinan Li and Prof. Qiong Luo for their help in the experiments. This research was partially supported by the grants from the Natural Science Foundation of China (No.60833005); the National High-Tech Research and Development Plan of China (No.2009AA011904); and the Doctoral Fund of Ministry of Education of China (No. 200800020002).

References

[1] J. Axboe, A. D. Brunelle, and N. Scott. blktrace(8) - linux man page, 2006. http://linux.die.net/man/8/blktrace.

[2] J. Katcher. Postmark: A new file system benchmark, 1997. http://communities.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp-tr3022.pdf.

[3] M. Kuoppala. Threaded i/o bench for linux, 2004. http://directory.fsf.org/project/tiobench/.

[4] Micron. Nand flash memory mt29f4g08aaa, mt29f8g08baa, mt29f8g08daa, mt29f16g08faa, 2007. http://download.micron.com/pdf/datasheets/flash/nand/4gb_nand_m40a.pdf.

[5] W. Norcott. Iozone filesystem benchmark, 2006. http://www.iozone.org/.