International Journal of Engineering & Technology, 7(4.16) (2018) 293-296

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Column-Oriented Replication Management for Large-Volume Data Storages

Siwoo Byun[1]*

[1] *Dept. Software, Anyang University*
**Corresponding author E-mail: swbyun@anyang.ac.kr*

Abstract

The column-based database repository is a highly advanced model for big data analysis systems with its superior I/O performance. In order to improve write operations, traditional database systems utilize a block-oriented storage in which records of column attributes are placed continuously on the hard disk. However, for read-intensive data warehouse, column-oriented storage becomes a more appropriate model to exploit its excellent performance. In addition, flash SSDs using MLC memory have recently been recognized as a most suitable storage medium for high-speed data analysis systems.
This paper introduces the column-oriented model and proposes a new storage management scheme that utilizes cross-compression method for high-speed data warehouses. The proposed storage management scheme is implemented on two MLC SSDs and provides excellent performance and reliability even in high CPU and I/O workloads. The results of our performance evaluation show that the proposed storage management scheme is better than the conventional scheme in terms of column update throughput and response time.

Keywords: *Column-Data; Compression; Data Duplication; , Flash Replication; Mirroring; MLC SSD.*

1. Introduction

The column-oriented database (CDB) [1-4], which is rapidly emerging in the field of big data warehouses, is the opposite database of the traditional record-oriented storage. This means that the column-oriented database stores data vertically, so similar data are clustered together to be effective for compression and retrieval (Fig. 1).

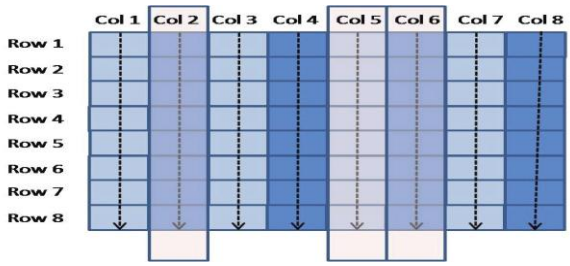

Fig. 1: Column-oriented Database

By default, column-oriented data of each column should be stored vertically when storing a table. For example, although a table logically consists of two dimensions of rows and columns, the physical data consists of a series of one-dimensional byte stream on the file storage and is linked to the operating system or buffer [5]. In other words, column-oriented database stores data vertically in cluster mode for easy compression and fast retrieval.
In the past, since hard disk was main storage for most database systems, characteristics of hard disk had been taken into account for horizontal record-based model which is the most efficient way to handle block I/O operations. But now with the large capacity and price competitiveness of high-performance, SSD has become a superior storage over traditional HDDs. In addition, modern data centers are moving from traditional hard disk to fast flash SSD for medium to large data servers [6,7].

2. Background

Recently, flash memory SSD is largely recognized as the preferred memory storage device of choice for consumer, industrial, and computing applications, with each market having its own performance, feature and density requirements. Actually, SSDs are vastly faster than traditional hard drives, because they can perform two to four times the input/output operations per second of a hard drive. Therefore, SSD has experienced tremendous growth in the last decade.
Two of the popular types of SSD architecture are: single-level cell (SLC) and multi-level cell (MLC). SLC flash memory stores one bit of data per memory cell and so a single voltage level is required for the device to detect one of the memory's two possible states. If a current is sensed, then the bit is stored as "0" (meaning "programmed"). If there's no current, then the bit is "1" (meaning "erased"). Due to this rather simple architecture, SLC is able to offer quick read and write capabilities, long-term durability and rather simply error correction algorithms. Its simplicity, though, does have one major downfall (besides its obvious limited programming density): SLC flash is considerably more expensive per bit when compared to MLC technologies. This is due to the fact that since each bit stores only one bit of data.
MLC is able to store at least two bits of data per memory cell and so multiple voltage levels are necessary to decipher between the memory's four possible states. The different states per memory cell are recognized as "00", "01", "10", and "11" and they range in value from fully programmed ("00") to partially programmed ("01"), partially erased ("10") and fully erased ("11"). Two advantages of this increase in bits per cell are that the memory's

capacity is greater and production costs are reduced. As a result, MLC can be sold at a much lower price-per-bit than SLC and also offer twice the density.
Due to the complexity of its architecture, though, MLC's programming/reading capabilities are much slower than SLC. That is, SLC is much faster than MLC, and several times more durable. Also, because multiple levels of voltage are needed in order to read between the memory's four states, there is greater power consumption and wear on the cells [8].
Many new SSDs offer much better speed than hard disk drives since there are no moving parts in SSD. However, it also could suffer from a sudden performance degradation caused by flash segment cleaning overhead called wear-levelling process for preparing next write operation.
SSD performance and endurance are related. Generally, the poorer the performance of a drive, the shorter the lifespan. That's because the management overhead of an SSD is related to how many writes and erases to the drive take place. The more write/erase cycles there are, the shorter the drive's lifespan. General SSD could last between five and 10 years [9].
Both SLC and MLC Flash memory are similar in their base design. However, MLC Flash devices cost less and allow for higher storage density, while SLC Flash devices provide faster write performance and greater reliability. Therefore, it is ideal that CDB exploits cheap MLC storage with the fast write performance of SLC.
In this respect, this study attempts to make the most of the strength of the column-oriented database and to reflect the advantages of high-speed MLC SSDs. In addition, we propose an efficient replication scheme that can ensure optimal performance and reliability for large column-oriented databases despite the high variability of CPU and I/O workloads.

3. Proposed storage management scheme

3.1 Storage Management for column-oriented Databases

The column-oriented database is compressed and stored in lower storage devices for processing large amounts of data at high speed, and the input/output operation is performed at high speed in memory after restoring compressed data as needed. To effectively and reliably support this database, advanced storage system should avoid failure of the data access rather than post data recovery.
In particular, the column-oriented database uses a very large database and its contents are intended for semantic analysis such as business strategy and smart marketing, so it is complex and performs urgent tasks such as decision making. As a result, time overhead of re-analysis after data recovery and reloading is too large to pay. Therefore, data replication is critical to avoid data failures and unnecessary recovery in advance. This means that the recovery of storage media in large column-oriented databases is complex and time-consuming, and therefore requires a method of storing at least one more copy (replica) of the data in a storage system to increase the reliability of the database.
As a traditional way to increase reliability, most database system uses simple physical mirroring which has the same data disks. In disk mirroring, the write operation ends when both writes of each disk device are fully completed. In this study, we also use the term redundancy which is similar to the mirroring concept but redundancy does not have to be completely physically identical.

3.2 Cross-compressed duplication control scheme

In this study, the idea of our scheme called Cross Compressed Duplication Control (CrossCompDC) is that two copies have the same data, but it stores the original uncompressed data segment and compressed data segment by turns, as shown in Fig. 2. In addition, if one of the two copies is replicated, the replication process is returned as asynchronous, rather than waiting for the other copy to be replicated. The other copy can be completed in CPU idle time. Of course, theoretical stability of our scheme is lower than physical mirroring in which both of the two copies should be replicated at the same time. However, since the column-oriented database always keeps the same data in RAM and sends the data to permanent storage, the data reliability is practically guaranteed at all times through keeping the data in both RAM and MLC.

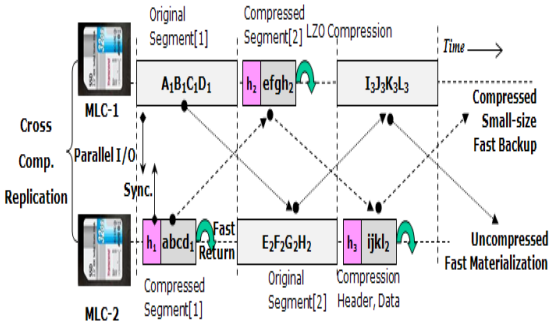

Fig. 2: Architecture of Cross Compressed Duplication Control Scheme

The compression method used for the cross compressed duplication is LZO [10]. LZO is easy to modify because its source is simple and is published as the GPL. Our duplication scheme manages both compressed copy and non-compressed copy, whereas traditional mirroring scheme physically maintains two same compressed copies. If the CPU is busy due to other operational tasks and thus the decompression speed drops severely[11], the non-compressed copy can be read without decompression overhead to increase user response.
In addition, if full backups are required, user can select either compressed version or non-compressed version. In general, compressed version would be faster in terms of backups speed, but non-compressed version would be needed for interface with commercial databases that are different from column-oriented databases or for data exchanges and format conversion.
In summary, our scheme provides flexible backup options and superior performance for column-oriented storage. Since the compressed data can be read sequentially via the backup link, the actual read speed is the same as in compressed mirroring. However, our scheme consumes more storage space than compressed mirroring scheme because cross-compression should store uncompressed data as well.
Another important issue to consider in improving the performance of column-oriented database is variability of CPU and I/O workloads. As compared to typical online databases, the column-oriented data warehouse creates a concentrated peak load but CPU loads are significantly reduced in idle time. If we exploit this variability which is generally seen as a disadvantage, we can devise a very effective storage system. For example, if storage I/Os are very busy and CPU resource are sufficient, it is more efficient to make compressed segment rather than uncompressed segment.
However, due to the nature of the column-oriented database, CPU resources are difficult to provide to compression process under CPU intensive workloads. In this case, it is faster and more efficient to just store non-compressed segments without compressing them and not taking CPU resources for user's data analytics operations. In worst case that many users' operations are focused on a MLC-SSD, this could lead to severe performance degradation. Under the extreme I/O congestion, it is more efficient to take full advantage of CPU resources for data compression to reduce I/O workload.
In addition, cross compression of column segments contribute to fast backup and rapid loading such as materialization [12]. As shown in Fig. 2, the compressed backup streams lead to faster backups without compression delay. Similarly, by following the links of the non-compressed segment, the column data can be copied directly into RAM memory without decompression delay.

In summary, our scheme effectively handles the load variability in column-oriented database, thereby enhancing the system performance through cross compression.

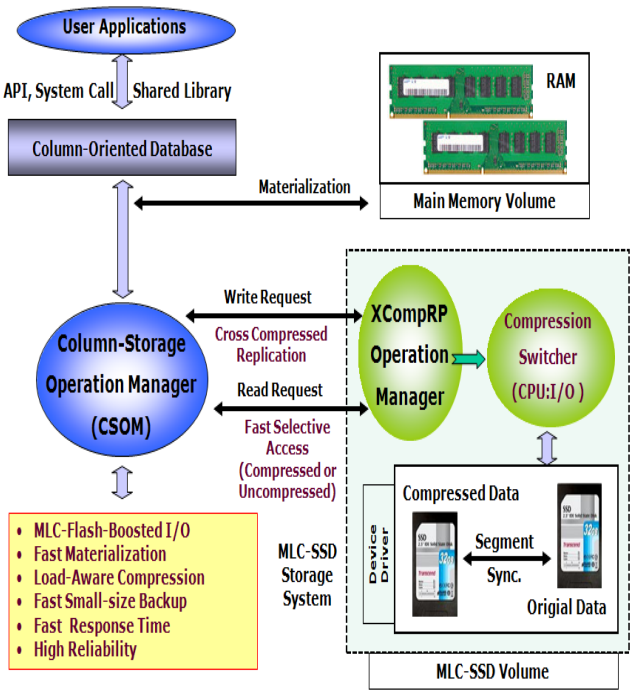

Fig. 3: Architecture of Proposed System

In the proposed system (Fig. 3), if I/O requests from user applications are received in a column-oriented database system, they are passed through the Column-Storage Manager to CrossCompDC duplication manager. The duplication manager decides the compression or non-compression mode after checking CPU and I/O workloads.

In order to optimize the storage performance of the column-oriented database, the compression characteristics of the column data locality can be used. That is, a traditional management method called data polarization which divides one table column into an active (hot) or non-active (cold) columns according to the frequency of update operations. For simplicity, our proposed scheme stores both compressed and uncompressed versions without column data polarization.

4. Performance evaluation

The storage management schemes compared to proposed scheme(CrossCompDC) are a single non-compression storage scheme (SglNoComp), a single compression storage scheme (SglComp), and a compressed mirroring scheme(CompMirrorDC) which is the most common replication scheme.

4.1 Experimental environment

The demo system used 16G RAM, I7-CPUs, and 64bit MS-Windows OS. In addition, a total of 800 million records were deployed for simulation database and CSimAPI [13] was used for performance analysis. The CSIM modules generate a constant load of desired I/O requests, so after generating a certain number of column storage operations per second, the response and performance of the issued request are measured.

In this experiment, key evaluation indicators are column-segment update throughput and its response time. The update throughput means how many storage operations per second have been processed and its response time refers to the time it takes to complete the request after the storage operation begins.

4.2 Preliminary tests

In order to determine the performance differences caused by CPU resources, preliminary tests were conducted as shown below before the full-scale experiment.

On the graph, Comp_CPU-n% is the storage performance curve of column segments with n % of CPU support for compression. The horizontal axis in Fig. 4 represents the storage operation load on the column-segment storage system, and the vertical axis represents the number of column-segmented storage operations at that load.

This preliminary experiment showed that the variability in CPU resources has a significant impact on system storage performance. This means that the non-compression method can be more effective if there is insufficient CPU resource. This variability is an important variable in the column-oriented database environment, where more and more CPU resources are used for semantic analysis.

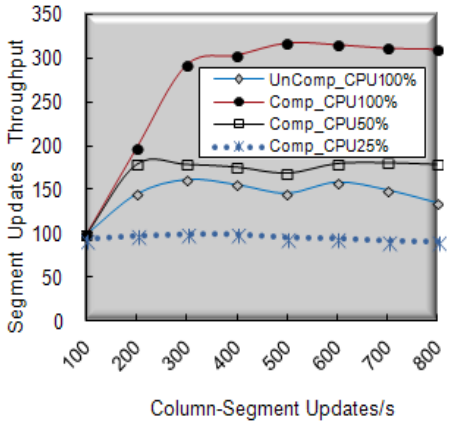

Fig. 4: Throughput by CPU allocation

4.3 Performance test and comparison

This experiment is intended to analyze how the number of column update operations affects the storage system performance. In Fig. 5, the overall update throughput shows that the proposed scheme, CrossCompDC, is higher than the SglNoComp, SglComp and CompMirrorDC.

At low intervals, the workloads caused by CPU and MLC I/O are adequately accommodated in the system without performance degradation. However, as the number of update operations increases gradually, the update operations are overfilled in the work queue, which slows down overall storage systems.

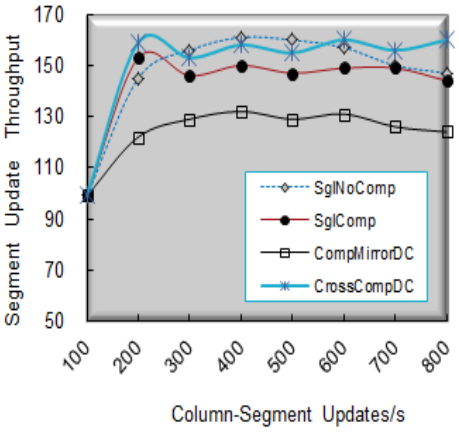

Fig. 5: Throughput of Column Updates

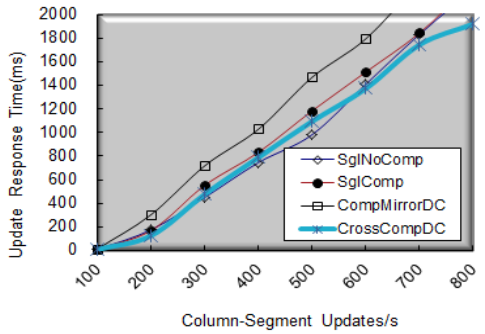

Fig. 6: Response Time of Column Updates

However, under the same workload conditions, CrossCompDC performed relatively better than other schemes. On average, CrossCompDC showed at least 23% more throughput than the traditional CompMirrorDC scheme. In particular, the throughput in the peak interval was 29% higher than CompMirrorDC. This is because CrossCompDC scheme overcomes storage update congestions by switching two MLC write operations to one MLC operation at that time.

That is, CrossCompDC scheme actively responded to the variability of CPU and MLC I/O, and completed congested update operations faster than the CompMirrorDC. On the other hand, the CompMirrorDC does not shorten the overall response time because the two compressed copies has to be stored in MLC at the same time.

When comparing four schemes in Fig. 6, the order of response time was consistent with the order of performance shown in Fig. 5. In low intervals, the processing performance and response time of SglComp were superior to SglNoComp in which the column-segment was compressed within a single MLC SSD. However, the performance of SglComp is lower than SglNoComp after the mid interval. This is why SglComp were suffering from the higher workload which causes a heavier CPU compression, while SglNoComp does not have the CPU overhead for compression. In the term of the response time in Fig. 6, CrossCompDC has at least 51% better than traditional schemes in the entire intervals.

5. Conclusion

The column-oriented storage is an advanced storage model suitable for high-speed analysis system with large volumes of data. In this study, we blended the high-speed MLC storage device into a column-oriented data model, and proposed a new storage management scheme utilizing cross-compressed duplication.

The proposed model is deployed on two MLC SSDs, and the cross compression of column segments provides superior update performance and data reliability under the high CPU and I/O workloads. In addition, the proposed scheme has shown to be better than traditional disk mirroring schemes by means of the performance evaluation based with more than 100 million data.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07044418).

References

[1] Ahn S. & Kim. K. (2013), A Join Technique to Improve the Performance of Star Schema Queries in Column-Oriented Databases, *Journal of Korean Institute of Information Scientist and Engineers* 40:3, 209-219.

[2] Byun S. (2017), Design of Efficient Index Management for Column-based Big Databases, *International Journal of Internet of Things and Big Data* 2:1, 59-64.

[3] Abadi D., Boncz A. & Harizopoulos S., "Column-oriented Database Systems," *Proceedings of the VLDB*, (2009) Lyon, France, 24-28 .

[4] Harizopoulos S., Liang V., Abadi D. J., & Madden S., "Performance tradeoffs in read-optimized databases, *Proceedings of the VLDB*, (2006), 487-498.

[5] Halverson A., Beckmann J., & Naughton J. (2006), A comparison of c-store and row-store in a common framework, Technical Report, UW Madison Department of CS, TR1566.

[6] Byun S. (2017), Shadow Indexing Scheme Using Hybrid Memory for Column-Based Datawarehouses, *INFORMATION* 20:11, 8125-8132.

[7] Kim H. & Noh H. (2015), Hybrid Main Memory Systems Using Next Generation Memories Based on their Access Characteristics, *Journal of KIISE* 42:2, 183-189.

[8] Li, Y. B. He, R. J. Yang, Luo Q., & Yi K., "Tree indexing on solid state drives," " *Proceedings of the VLDB 3:1*, (2010), 1195-1206.

[9] Lu, N., Choi I, & Kim S. (2012), A PRAM based block updating management for hybrid solid state disk, *IEICE Electronics Express* 9:4, 320-325.

[10] LZO, "LZO Professional data compression", (2018), available online: http://www.oberhumer.com/products/lzo-professional/

[11] Madhushree G, Kavitha V. N., Arpitha S. M., Latha S. S., & ArunBiradar (2018), ACO Technique for Reducing Energy Consumption in Wireless Sensor Network, *International Journal of Computing, Communications and Networking* 7:2, 42-47.

[12] Abadi D., Myers D., DeWitt D., & Madden S. (2006), Materialization strategies in a column-oriented DBMS, MIT CSAIL Technical Report, MIT-CSAIL-TR-2006-078.

[13] Csim, "Getting Started:CSIM 20 Simulation Engine (C Version)", (2018), available online: https://static1.squarespace.com/statc/56eb309fe321407d6a06998a/t/5824f21c46c3c4041b461eeb/1478816285097/Getting_Started-C.pdf.