KIPS Tr. Software and Data Eng.
Vol.4, No.9 pp.371~376 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.9.371

A Selective Compression Strategy for Performance Improvement of Database Compression

Ki-Hoon Lee†

ABSTRACT

The Internet of Things (IoT) significantly increases the amount of data. Database compression is important for big data because it can reduce costs for storage systems and save I/O bandwidth. However, it could show low performance for write-intensive workloads such as OLTP due to the updates of compressed pages. In this paper, we present practical guidelines for the performance improvement of database compression. Especially, we propose the SELECTIVE strategy, which compresses only tables whose space savings are close to the expected space savings calculated by the compressed page size. Experimental results using the TPC-C benchmark and MySQL show that the strategy can achieve 1.1 times better performance than the uncompressed counterpart with 17.3% space savings.

Keywords: Database Compression, Performance Improvement, OLTP, TPC-C

데이터베이스 압축 성능 향상을 위한 선택적 압축 전략

이 기 훈†

요 약

사물인터넷 (IoT)은 데이터의 양을 상당히 증가시킨다. 데이터베이스 압축은 저장 시스템 비용과 I/O 대역폭을 절약할 수 있기 때문에 빅데이터에 있어서 중요하다. 그러나 데이터베이스 압축은 압축된 페이지에 대한 업데이트로 인해 OLTP와 같은 쓰기 집중적인 워크로드에 대해 낮은 성능을 보일 수 있다. 본 논문에서는 데이터베이스 압축의 성능 향상을 위한 실용적 가이드라인을 제시한다. 특히, 압축 페이지 크기에 의한 계산으로부터 예상되는 공간 절약과 거의 같은 공간 절약을 보이는 테이블들만을 압축하는 SELECTIVE 전략을 제시한다. TPC-C 벤치마크와 MySQL을 이용한 실험을 통해 SELECTIVE 전략이 압축하지 않는 방법에 비해 1.1배 높은 성능을 보이면서 17.3%의 공간을 절약한다는 것을 보였다.

키워드: 데이터베이스 압축, 성능 향상, OLTP, TPC-C

1. Introduction

Compression is especially important for big data because we can substantially reduce database storage cost and save I/O bandwidth. However, it is generally known that database compression is not suitable for write-intensive workloads like OLTP [1,2]. The main reason is that updates of compressed pages incur many page splits, which require exclusive locks [1]. Furthermore, updates require page reorganization and recompression, which consume CPU cycles.

Due to the complexity of real systems, the tuning of database compression for OLTP workloads is difficult and time-consuming. It depends on many factors such as the compressed page size, workload characteristics, tables to compress, and storage device types [3]. In this paper, we provide comprehensive guidelines for tuning database compression. Our guidelines do not need any database engine modifications, and thus, can offer benefits to the millions of already-deployed systems. One of the most important guidelines, the SELECTIVE strategy, is that we should compress only tables whose space savings are similar to the expected space savings calculated by the

※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2015R 1C 1A 1A02036517).
※ 이 논문은 2014년도 광운대학교 교내 학술연구비 지원에 의해 연구되었음.
† 정 회 원 : 광운대학교 컴퓨터공학과 조교수
Manuscript Received : July 28, 2015
Accepted : August 24, 2015
* Corresponding Author: Ki-Hoon Lee(kihoonlee@kw.ac.kr)

compressed page size. We focus on MySQL because it is the world's most widely used open-source DBMS with millions of users. We conduct extensive experiments using the TPC-C benchmark [4] and MySQL InnoDB [3], and the results show that by applying the guidelines we can reduce storage space without performance degradation.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces database compression. Section 4 proposes compression strategies, and Section 5 presents an experimental evaluation of the proposed compression strategies and discussions. Finally, Section 6 provides some concluding remarks.

2. Related Work

Poess and Potapov [2] have presented compression techniques used in Oracle for read-only workloads. Bhattacharjee et al. [5] have introduced index compression techniques used in DB2. Ordulu and Tolmer [6] have proposed the adaptive padding method to reduce the compression overhead for insert-intensive workloads. Recently, Lee [1] has improved the performance of database compression for OLTP workloads by avoiding lock contention. These techniques need modification of DBMS engines. In contrast, our work focuses on tuning database compression without any engine modification. Thus, our work is easily applicable to the millions of already-deployed MySQL database systems.

3. Database Compression

A table in MySQL InnoDB is stored as a B^+-tree index [3]. InnoDB uses zlib [7] to compress tables and indexes [3]. The page size of an uncompressed table is 16KB, but a compressed table can use a smaller page size (1KB, 2KB, 4KB, or 8KB) [3]. The same compressed page size is used for a table and all of its indexes [3]. When a page cannot be compressed to the compressed page size, InnoDB splits the page and recompresses the splitted pages. The optimal size of the compressed page depends on the data type and distribution [3]. If the compressed page size is larger than the actual size of the compressed data in the page, some space is wasted, but the compressed page is rarely splitted and recompressed for updates [3]. If the compressed page size is too small, updates may incur page splits and recompression, resulting in low concurrency, more space consumption, and more CPU overhead [3].

4. Compression Strategies

We propose two strategies for choosing which tables to compress: *ALL* and *SELECTIVE*. We define the *space savings* (*SS*) according to literature [2, 8] in Formula (1).

$$SS = \frac{uncompressed\ size - compressed\ size}{uncompressed\ size} \times 100 \quad (1)$$

The ALL strategy simply compresses all the tables. The SELECTIVE strategy compresses only tables whose space savings T_{ss} are close to the expected space savings CP_{ss} calculated by the compressed page size. Here, *CP* denotes "*C*ompressed *P*age". This means that most of the uncompressed pages of those tables successfully compress to the compressed page size without incurring page splits. We say T_{ss} is *δ-close* to CP_{ss} if $(1-\delta)CP_{ss} \leq T_{ss} \leq CP_{ss}$ where $0 \leq \delta \leq 1$. For example, when the compressed page size is 8KB and the uncompressed page size is 16KB, i.e., CP_{ss} = 50%, the SELECTIVE strategy chooses tables whose space savings $45\% \leq T_{ss} \leq 50\%$ when $\delta = 0.1$.

According to Lee [1], if data is too densely stored in compressed pages, page splits will be often incurred by updates. Page splitting lock the whole index in exclusive mode and lead to low concurrency. Suppose that an update of a record increases the compressed size of the record. Due to the increased size of the record, a compression failure likely happens if the compressed page is already almost full before updating. In this case, we should perform updates with exclusive lock because page splitting can occur. The SELECTIVE strategy reduces page splits for updates by sacrificing space savings. Since it compresses only tables that are compressed well, enough free space for the update is usually available in the compressed page, and updates do not incur page splits in most cases. Furthermore, the SELECTIVE strategy has less compression/decompression overhead because it compresses a smaller number of tables than the ALL strategy.

5. Experimental Evaluation

5.1 Experimental Setup

TPC-C [4] is a standard benchmark simulating real online transaction processing (OLTP) workloads. The workload of TPC-C is composed of two read-only and three read-write transactions, which together provide many concurrent random reads and writes to the storage device. Table 1

shows the mix of the transaction types. Besides the standard mix in Table 1, which is a write-intensive workload, we also configure a read-intensive workload, in which the read-only transactions are dominant.

Table 1. Transaction Types in TPC-C Benchmark

Transaction type	I/O property	Standard mix(%)	Read-intensive mix(%)
New-Order	read-write	45	1
Payment	read-write	43	1
Delivery	read-write	4	1
Stock-Level	read-only	4	48.5
Order-Status	read-only	4	48.5

We use DBT-2 [9], which is an open-source implementation of the TPC-C specification. We use a workload of 1,000 warehouses with 20 database connections, 10 terminals per warehouse, and the duration of 7,200s after 10,000s warming-up period. The Key and Thinking time is set to zero in order to measure the maximum performance. According to Zaitsev [10], we add a secondary index to the NEW_ORDER table on columns (no_w_id, no_d_id).

We run DBT-2 on top of MySQL Community Server 5.5.38. The buffer pool size is set to 2GB. To minimize the interference by data caching at the OS layer, we use direct I/O (O_DIRECT). To assure a controlled setting, we disable MySQL binary logging, which keeps track of all updates to the database. Unless specially stated, we use the default compressed page size 8KB for experiments.

We measure space savings in Formula (1), database loading time ratio in Formula (2), transactions-per-minute (TPM) ratio in Formula (3). For the loading time ratio, smaller is better, but for the TPM ratio, larger is better. Hereafter, A denotes the ALL strategy, S the SELECTIVE strategy, and N no compression. For the S strategy, we set δ to 0.1.

$$loading\ time\ ratio = \frac{loading\ time\ with\ comp.}{loading\ time\ without\ comp.} \quad (2)$$

$$TPM\ ratio = \frac{TPM\ with\ comp.}{TPM\ without\ comp.} \quad (3)$$

Our experimentation platform is a 3.4GHz Intel Core i7-2600K quad-core processor with 4GB of main memory, a Samsung 830 Series 128GB SSD (256MB cache and SATA III (6 Gb/s) interface) and a Western Digital Caviar Black WD1002FAEX HDD (1TB, 7200RPM, 64MB cache, and SATA III (6 Gb/s) interface). The operating system used for the basic performance evaluation in Section 5.2.1 is Windows 7 64bit and that used for all other experiments is Ubuntu 12.04 64bit.

5.2. Experimental Results

5.2.1. Basic Performance Results

To understand the effect of storage devices (HDD and SSD) on the performance of database compression, we measure the basic performance of the devices. Tables 2 and 3 show the results, which are measured using the CrystalDiskMark benchmark [11] version 3.0.1c. Compared with HDD, SSD shows higher sequential and random bandwidth and much higher concurrent random I/O performance (BS = 4KB, QD = 32). For HDD, the relative bandwidth gap between sequential I/O and random I/O is quite large, because HDD has the inevitable mechanical latency, which dominates the access time for a small data page (e.g., 4KB) [12].

Table 2. Sequential I/O Performance

Device	Read (MB/s)	Write (MB/s)
SSD	486.2	260.8
HDD	132.6	130.6

Table 3. Random I/O Performance (BS = Block Size, QD = Queue Depth)

Device	BS=512KB (MB/s)		BS=4KB (MB/s)		BS=4KB, QD=32 (MB/s)	
	Read	Write	Read	Write	Read	Write
SSD	334.3	250.7	23.9	82.1	318.9	118.3
HDD	44.7	76.5	0.6	1.2	1.2	1.1

5.2.2. Space Savings of Each Table

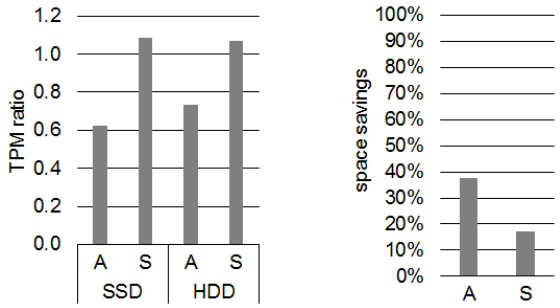
Table 4 shows the space savings of each table in TPC-C. The table size includes the size of the indexes created on the table. For the S strategy, we compress only ORDER-LINE, ORDER, and HISTORY because their space savings is greater than 45%.

Table 4. The Space Savings of Each Table

Table	Uncompressed size (MB)	Compressed size (MB)	Space savings
STOCK	36,904	24,488	34%
ORDER-LINE	**26,308**	**13,296**	**49%**
CUSTOMER	21,592	15,804	27%
ORDER	**3,056**	**1,536**	**50%**
HISTORY	**2,404**	**1,208**	**50%**
NEW-ORDER	456	260	43%
ITEM	18	12	33%
DISTRICT	9	5	44%
WAREHOUSE	0.20	0.13	35%

5.2.3. Results for the Standard Workload

We first present experimental results for the standard mix of transactions, which is write intensive, then results for the read-intensive workload in the next section. Fig. 1 and 2 show the influence of compression strategies. The *S* strategy shows 1.1 times higher TPM than the uncompressed counterpart (*N*) with 17.3% space savings as in Fig. 1(a) and 1(b). Compared with the *A* strategy, the *S* strategy shows lower space savings but up to 1.7 times higher TPM.



(A) (B)

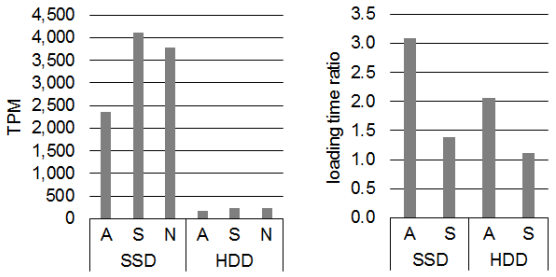
Fig. 1. TPM Ratio (A) and Space Savings (B) with Different Strategies

Table 5 shows major time-consuming sub-operations for executing a transaction. Lock wait and physical file I/O take most of the transaction response time (61% to 83%), and the compression and decompression time takes a small portion. We see that the decreasing order of lock wait time is *A*, *S*, and *N*, and that of file I/O time is the opposite. The lock wait time of the *A* strategy is 8.3 to 9.5 times longer than that of the *S* strategy due to more page splits, and thus, the benefits of file I/O time savings are eaten up by the additional lock waits.

Table 5. Average Elapsed Time Per Transaction

Device	Time (ms)	A	S	N
SSD	transaction	509	292	316
	lock wait	333	40	5
	file I/O	63	143	188
	comp. + decomp.	83	13	N/A
HDD	transaction	7,246	4,968	5,322
	lock wait	4,714	496	252
	file I/O	1,332	3,424	3,919
	comp. + decomp.	982	138	N/A

Fig. 2(a) shows that the TPMs of SSD are orders of magnitude higher than those of HDD since SSD has superior random I/O performance.



(A) (B)

Fig. 2. TPM (A) and Loading Time Ratio (B) with Different Strategies

Fig. 2(b) shows that, in terms of loading time, the *S* strategy shows comparable performance (1.1 to 1.4 times slower) to no compression *N* and up to 2.2 times better performance than the *A* strategy. This is because the *A* strategy has more compression overhead. During database loading, page splits do not incur lock waits since we sequentially load tables, and compression time becomes a large portion as in Table 6.

Table 6. Loading Performance

Device	Measure	A	S	N
SSD	loading time (s)	24,075	10,868	7,815
	comp. time (s)	15,028	3,090	N/A
	CPU usage	80%	57%	38%
HDD	loading time (s)	32,045	17,304	15,525
	comp. time (s)	17,284	3,617	N/A
	CPU usage	38%	22%	11%

To see the effect of the compressed page size, we vary the compressed page size with 4KB and 8KB. For the *S* strategy, we should choose tables whose space savings are close to 75% when the compressed page size is set to 4KB, but there is no table satisfying the condition. The space savings of all tables are less than 67%. Thus, the *S* strategy is not available for 4KB. Fig. 3 and 4 show that the page size of 4KB, which has more compression overhead, shows similar space savings, similar TPM, and worse loading performance compared with the page size of 8KB.

5.2.4. Results for the Read-intensive Workload

Fig. 5 shows experimental results for the read-intensive workload. If the workload is dominated by reads, rather than updates, lock contention is not a bottleneck anymore. Since the *A* strategy saves more I/O bandwidth than the *S* strategy, it shows higher TPM for the read-intensive workload for SSD. For HDD, however,

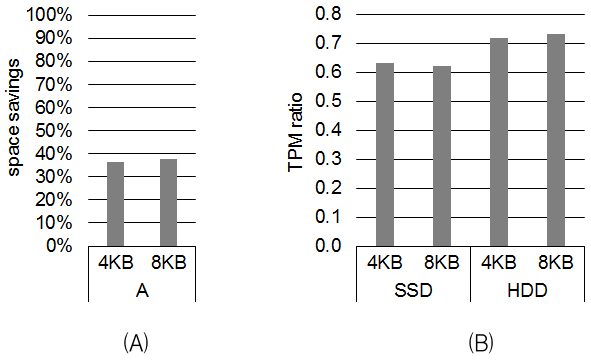


Fig. 3. Space Savings (A) and TPM Ratio (B) with Different Page Sizes (Only for the *A* Strategy)

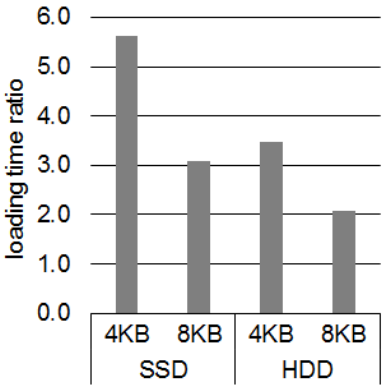


Fig. 4. Loading Time Ratio with Different Page Sizes (Only for the A Strategy)

the *A* strategy shows similar TPM to the *S* strategy due to poor random I/O performance of HDD. SSD has no seek time and rotational delay, but HDD has. Compression can reduce the transfer time, but cannot reduce the seek time and rotational delay, which take most of the I/O time in case of HDD. The *S* strategy shows comparable TPM to no compression for both write-intensive and read-intensive workloads, i.e., is less sensitive to the workload type.

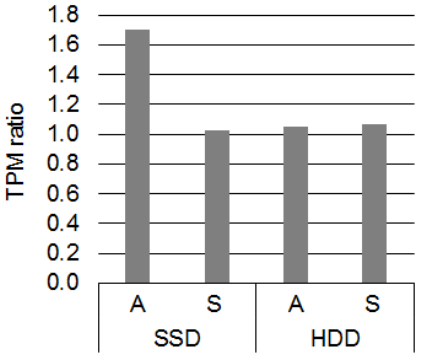


Fig. 5. TPM Ratio for the Read-Intensive Workload

5.3 Discussions

Practical guidelines for performance improvement of database compression are as follows.

- For write-intensive workloads, use the SELECTIVE strategy to reduce storage space without performance degradation.
- For read-intensive workloads on SSD, use the ALL strategy.
- If the type of workloads is unknown or unclear, use the SELECTIVE strategy because it shows comparable performance to no compression for both write-intensive and read-intensive workloads.
- Although the I/O characteristics of SSD are very different from those of HDD, database compression is also effective for SSD.
- Smaller compressed page sizes degrade loading performance without improving space savings and throughput for both SSD and HDD.

6. Conclusions

We have provided a set of useful guidelines for tuning database compression based on an extensive experimental evaluation. Especially, the SELECTIVE strategy makes database compression highly desirable for both write-intensive and read-intensive workloads. Experimental results show that the SELECTIVE strategy achieves 1.1 times better performance than the uncompressed counterpart with 17.3% space savings for the TPC-C benchmark.

References

[1] K.-H. Lee, "Performance Improvement of Database Compression for OLTP Workloads," *IEICE Trans. on Information and Systems*, Vol.E97-D, No.4, pp.976-980, Apr., 2014.

[2] M. Poess and D. Potapov, "Data Compression in Oracle," In *Proc. the Int'l Conf. on Very Large Data Bases* (*VLDB*), pp.937-947, Sept., 2003.

[3] MySQL 5.5 Reference Manual, https://dev.mysql.com/doc/refman/5.5/en/

[4] Transaction Processing Performance Council (TPC), TPC BENCHMARK C, Standard Specification, Revision 5.11, Feb., 2010.

[5] B. Bhattacharje et al., "Efficient index compression in DB2 LUW," In *Proc. the Int'l Conf. on Very Large Data Bases* (*VLDB*), pp.1462-1473, Aug., 2009.

[6] N. Ordulu and J. Tolmer, "InnoDB Compression: Present and Future," *Percona Live MySQL Conference*, Apr., 2013.

[7] zlib, http://zlib.net

[8] M. A. Roth and S. J. V. Horn, “Database Compression,” *ACM SIGMOD Record*, Vol.22, No.3, pp.31-39, Sept., 1993.
[9] Database Test 2 (DBT2), http://sourceforge.net/apps/mediawiki/osdldbt
[10] P. Zaitsev, “MySQL/InnoDB Performance, Server and Schema,” *MySQL Users Conference*, Apr., 2004.
[11] CrystalDiskMark, http://crystalmark.info/?lang=en
[12] E.-M. Lee, S.-W. Lee, and S. Park, “Optimizing Index Scans on Flash Memory SSDs,” *ACM SIGMOD Record*, Vol.40, No.4, pp.5−10, Dec., 2011.

Ki-Hoon Lee
e-mail : kihoonlee@kw.ac.kr
He has received B.S. (2000), M.S. (2002), and Ph.D. (2009) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). He is currently an assistant professor of Department of Computer Engineering at Kwangwoon University.