Jurnal
Teknologi

Full Paper

OLTP PERFORMANCE IMPROVEMENT USING FILE-SYSTEMS LAYER COMPRESSION

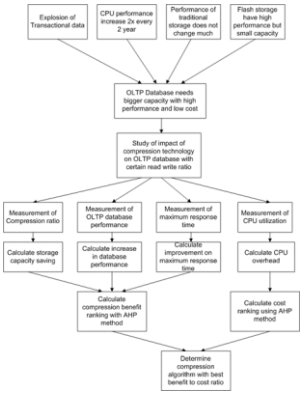
Suharjito*, Adrianus B. Kurnadi

Computer Science, Binus Graduate Program, Bina Nusantara University, Jakarta, Indonesia

Article history
Received
4 June 2016
Received in revised form
4 January 2017
Accepted
10 March 2017

*Corresponding author
suharjito@binus.edu

Graphical abstract



Abstract

Database for Online Transaction Processing (OLTP) application is used by almost every corporations that has adopted computerisation to support their operational day to day business. Compression in the storage or file-systems layer has not been widely adopted for OLTP database because of the concern that it might decrease database performance. OLTP compression in the database layer is available commercially but it has a significant licence cost that reduces the cost saving of compression. In this research, transparent file-system compression with LZ4, LZJB and ZLE algorithm have been tested to improve performance of OLTP application. Using Swing-bench as the benchmark tool and Oracle database 12c, The result indicated that on OLTP workload, LZJB was the most optimal compression algorithm with performance improvement up to 49% and consistent reduction of maximum response time and CPU utilisation overhead, while LZ4 was the compression with the highest compression ratio and ZLE was the compression with the lowest CPU utilisation overhead. In terms of compression ratio, LZ4 can deliver the highest compression ratio which is 5.32, followed by LZJB, 4.92; and ZLE, 1.76. Furthermore, it is found that there is indeed a risk of reduced performance and/or an increase of maximum response time.

Keywords: OLTP database, performance, file-systems layer compression

1.0 INTRODUCTION

The popularity of the internet and online commerce has resulted in the explosion of the number of electronic transactions. The increase of the size of transactions data results in a huge increase of storage requirements and its associated costs including space, electricity and cooling. Two techniques that are often used to reduce data size are compression and deduplication. Both techniques result in the increase of CPU utilisation traded with saving of storage capacity that has to be physically provided. This CPU utilisation overhead for compression and decompression is one of the deterrents that prevents the use of compression for database. [1] The Other effects of compression is that it will reduce the amount of I/O that has to be done by the application and this opens the possibility that compression besides saving the storage space might also improve performance.

To be able to increase application performance, the amount of time saving for I/O operations must be bigger than the additional time needed to do compression or decompression of data. There have been several studies on the effect of compression implemented on database layer or storage layer. The effect of filesystems compression on database performance on the other hand has not been much studied academically to the best of writer's knowledge.

Several studies on the effect of compression to database performance have been done before. However, most of them focus on compression implementation at the database layer itself [2], [8], [9], [10] or at the storage layer [5], [6], [11], [12]. Only one

study [13] focuses on compression implementation at the filesystems layer.

Compression implementation at the database layer has a unique benefit. It can also give benefits to database backup and database replication process, but it will only work for that specific database and sometimes translate to a higher software licence cost or a complex modification of the database software. Whereas compression implementation at the filesystem level will work on any database and in the case that the filesystems are available on several operating systems, the applicability of the study will be higher.

With regard to the type of the database for related works, some used database that is not widely used commercially or they did not mention the database type. One of the earliest studies [1] discusses compression implementation for Scientific and Statistical database, compression benefits and disadvantages and then compares several suitable compression algorithms for the type of data in Scientific and Statistical Database.

The second related work still deals with compression implementation at the database layer for a generic database [2] and discusses the query algorithm and compression characteristics that can enable query processing without decompression and its effect on I/O performance, transaction processing and query processing. Performance comparison in the second study is done only through theoretical calculation for hybrid hash join.

The third related work [3] also still discusses compression implementation at database layer. It discusses the characteristics of a compression algorithm that can increase database performance. The compression must be fast, and fine grained. The study uses TPC-D benchmark to shows that light weight compression can increase most query performance, in the extreme case up to 2 times and that performance is only reduced for some update operations. The database used for this study was also not named.

The fourth related work [4] explains how H-HIBASE compression implemented in the storage layer can increase performance for all kinds of query operations compared to DHIBASE and uncompressed Oracle 10g database. The last study [5] is the only one that discusses the impact of transparent compression at the filesystem layer (ZFS) to the performance of a datawarehouse application. This last study employs the widely used Oracle database combined with ZFS filesystems compression and SwingBench to show up to 92% performance improvement of compressed database. The last study uses Sales History scenario in Swingbench which is an OLAP scenario and the algorithm compared was LZJB, ZLE and GZIP. The study also compares the performance improvement with Oracle Advanced Compression Option (ACO) which is implemented at the database layer. The advantages of this study are that it can be directly applied since Oracle database is widely used and ZFS filesystem is available from various operating systems such as Oracle Solaris, Linux and FreeBSD. Although the study has successfully shown the performance improvement for data warehouse workload, it was limited in the sense that it did not compare against Oracle Hybrid Columnar Compression, which was claimed to have a much higher compression ratio and performance improvement for OLAP workload.

This study aims to extend the usability and applicability of the previous research by testing OLTP database workloads. OLAP workloads benchmark using Swingbench will only test the decompression impact on performance while an OLTP workload that has various read/write ratios will test the combination of both compression and decompression impact to database performance. The comparison of compression algorithm is changed to LZ4, LZJB and ZLE. GZIP was not tested in our study because it was the compression with the highest CPU overhead in the previous study. LZ4 is a new compression algorithm in ZFS that was newly incorporated in Oracle Solaris 11.3 operating systems used for the experiment.

Our study also aims to interpret three technical measurement results namely compression ratio, transaction per second and maximum response time into business benefit variables like storage savings, increased productivity, and SLA improvement. The business benefits will be ranked using Analytic Hierarchy Process and compared with the cost ranking for each scenario. With three read/write ratios, it is also possible to see correlation between read/write ratios and the level of performance improvement. There is also risky that with certain read/write ratios, the compression might have a negative impact on performance in terms of transaction per second or the increase in maximum response time. Our study will be able to find out which algorithm is the most suitable for a given scenario and which algorithm is the safest to be used in case we do not have knowledge of read/write ratio or application characteristics.

2.0 METHODOLOGY

This study is a contrived study using data measured from a laboratory experiment. The research framework of this study is shown in Figure 1.

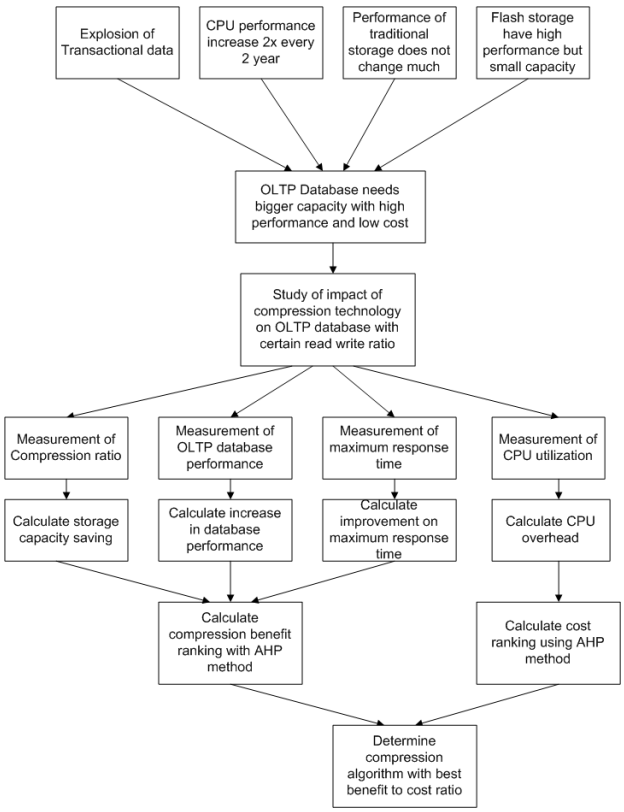


Figure 1 Research framework of this study

2.1 Experiment Environment Setup

Experiment on the impact of file-system compression was done using the following environment:

- SPARC T4-4 server logical domain with the specification as follows :
  - 4 core SPARC T4 2.85 GHz
  - 32 GB RAM
  - 450 GB HDD for OS
  - 450 GB HDD for data
- Solaris 11.3 operating systems
- Oracle Database 12c
- Swing-bench version 2.5.971

The server used actually had 4 x 8 core SPARC T4 CPU and 256 GB, but it was find out early that 4 core was already very powerful and it was easier to measure CPU utilisation when the number of core was reduced. The memory allocated to the logical domain was also reduced to reduce cache effect at ZFS file-system. The logical domain was set with bare metal I/O so there was no I/O overhead.

After the operating systems has been installed, we need to create user, group and project settings as the requirements of Oracle Database installation. We also need to create several ZFS file-systems with different compression settings. Before creating a ZFS file-system, we need to create a ZFS pool with the following command:

```
# zpool create dpool cxtxdx
```

The command above created a zfs pool called dpool from the disk cxtxdx. After the pool was created, we created several file-systems with the following commands:

```
# zfs create dpool/baseline
# zfs create dpool/zle
# zfs create dpool/lzjb
# zfs create dpool/lz4
```

To set appropriate compression algorithm to the file-systems, we used the following commands:

```
# zfs set compression=zle dpool/zle
# zfs set compression=lzjb dpool/lzjb
# zfs set compression=lz4 dpool/lz4
```

The benchmark tool Swing-bench was created by an Oracle UK employee named Dominic Giles. It was created to provide a realistic benchmark to test Oracle RAC [6]. Swing-bench was chosen as the benchmark tool for this study because it is often used to benchmark Oracle database performance both for vendors sponsored the white paper [7] [8] and also for academic research papers published in international conference especially about database performance on virtualised environment [9] [10] and [11]. Swing-bench software has four built-in benchmark scenarios as shown in Table 1.

Table 1 Swing-Bench Benchmark Scenarios

Benchmark	Description	Read/Write Ratio
Order Entry	Classic Order Entry Benchmark. TPC-C Like	60/40
Calling Circle	Telco based self-service application	70/30
Stress Test	Simple Insert / Update / Delete / Select Benchmark	50/50
Sales History	DSS benchmark	100/0

This experiment uses three Swing-bench OLTP benchmark scenarios to provide data: Order Entry with 60/40 read-write ratio, Calling Circle with 70/30 read-write ratio and Stress Test with 50/50 read-write ratio.

2.2 Data Collection Method

Swing-bench is not only a benchmark tool, it also comes with wizards to create the schema and populate data required for each benchmark scenario. The wizard can be run interactively with GUI to select such parameters as database network address, username, password, scale of data size, location of data file and other parameters, The wizard can also be run in lights out mode by providing command line parameter or referring to an xml configuration file.

We did some trial run with some parameter settings with each scenario to check if all data especially CPU

utilisation could easily be measured. Based on the trial run, the following statements are the parameter settings for each scenario:

- Order Entry scenario was set with 1 GB data size and 10 minutes runtime
- Calling Circle scenario was set with 10 GB raw data size and 5 minutes runtime
- Stress Test scenario was set with 10 GB raw data size and 5 minutes runtime
- All scenarios were set with 500 users

To ensure that the compression ratio and other measurable results can be compared between compression algorithms, an initial data for the benchmark must only be generated once for each scenario. The compression ratio is only valid when we compare exactly the same data. The wizard for each scenario should then be used to populate the data located in the dpool/baseline file-systems.

Order Entry benchmark and Calling Circle benchmark have its own setup wizard named oewizard and ccwizard, but Stress Test scenario uses the same schema and data population as Order Entry. Stress Test only differs from Order Entry in the type of transactions and its relative weight during the benchmark run.

Before the benchmark was run, compression ratio was measured and the data file along with Oracle spfile and control file should be backed up. The compression ratio for a given ZFS file-system can be measured with this command:

```
# zfs get compressratio dpool/filesystemname
```

To run the benchmark Swing-bench provides three options, we can use swing-bench, mini-bench and char-bench. The first is a full blown GUI, where we can set benchmark duration, number of users and so on, mini-bench is a minimalist GUI for the same purpose and char-bench uses command line options to provide parameters for the benchmark. All options used xml configuration files to set the transactions that will be run and its weight during the benchmark. We will use char-bench to run the benchmark because it can provide more detailed output including transaction dump and CPU utilisation monitoring result to output files.

The type of transaction and its relative weight for Order Entry benchmark that we used can be seen in Table 2.

Table 2 Order Entry Transactions and Weight

Transaction Name	Weight	Enabled
Customer Registration	15	TRUE
Update Customer Details	10	TRUE
Browse Products	40	TRUE
Process Orders	5	TRUE
Browse Orders	5	TRUE
Sales Rep Query	2	FALSE
Warehouse Query	2	FALSE
Warehouse Activity Query	2	FALSE

Order Entry has some transactions type that by default is not enabled. It can be used if we want to increase the read ratio, but for this study we left the settings as default. The type of transactions and its weight settings for Calling Circle and Stress Test are shown in Table 3 and Table 4.

Table 3 Calling Circle Transactions and Weight

Transaction Name	Weight	Enabled
New Customer	25	TRUE
Update Customer Details	100	TRUE
Retrieve Customer Details	50	TRUE

One notable difference of Calling Circle scenario is that during data population there is a setting to specify how much transactions should be prepared. We prepared 8000 transactions that took approximately 5 minutes to run.

Table 4 Stress Test Transactions and Weight

Transaction Name	Weight	Enabled
Insert Transaction	15	TRUE
Simple Select	40	TRUE
Update Transaction	30	TRUE
Delete Transaction	10	TRUE

After the benchmark was run for the baseline we need to shut down the database and restore the backup to another file-system with compression. Oracle database should be set to NOARCHIVELOG mode so that after the restore, the transactions from the first benchmark run will not be replayed.

2.3 Data Analysis Method

To analyze the result for each scenario we used Analytic Hierarchy Process employed to make decisions when we have several criteria. Analytic Hierarchy Process (AHP) was developed by Saaty [22].

AHP decomposed a decision objective into several criteria that can be decomposed further to sub criteria. Each criterion will be assigned ranking which was calculated from how many times a criterion was more important than the other criteria. The alternatives that will be chosen for the decision will also be ranked for each criterion.

For our work, the decision goal or objective is to choose which algorithm has the most benefit for each benchmark scenario. There are four criteria in making the decision:

- Storage saving is derived from compression ratio
- Productivity increase is derived from transaction per second improvement
- Improvement of the SLA is derived from the improvement of maximum response time
- Compression cost is derived from CPU utilisation overhead

From the four criteria above, we separate the first three as benefit criteria and the last as the cost criterion. Early when AHP was first implemented people tended to lump together positive and negative criteria together. However, it was recognized that positive and negative priorities in nature are not directly comparable. [13] By grouping positive criteria together, we will be able to rank the benefit of each scenario without the cost. Of course, we then calculate cost ranking separately and later can create benefit to cost ratio ranking for each scenario.

3.0 RESULTS AND DISCUSSION

The data results from each benchmark scenario were based on different data, so they were analyzed separately by, first, using AHP method. AHP method decomposed decision making process into a hierarchy of objective, criteria and alternatives which can be seen in Figure 2.

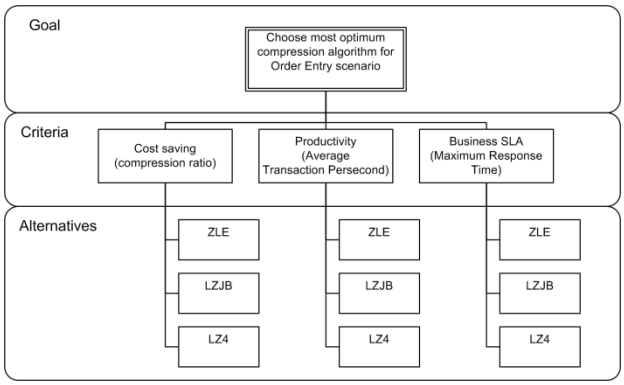


Figure 2 Analytic Hierarchy Process diagram

First, we need to determine relative importance of the benefit criteria. The relative importance of the criteria is subjective. In this research, based on experience, the writer's subjective judgment is that storage saving has equal importance to performance improvement and business SLA. The relative importance sum of all criteria is 1.

After the ranking of the criteria has been decided, we need to calculate the relative importance of the alternatives for each criterion using pair wise comparison matrices. Because the matrices values were taken from the measurement of each alternative value for related criteria, the resulting ranking will be consistent. AHP can combine both subjective and objective factor in the decision making process.

After calculating the highest ranking for benefit and benefit to cost ratio for each scenario, then we can do cross scenario comparison to see if there is any correlation between the results and the change of read/write ratio and other characteristics of each scenario.

3.1 Result and Analysis from Order Entry scenario

The benchmark results between the baseline and the three compression algorithms for Order Entry scenario shows increased performance as shown in Table 5.

Table 5 Order Entry Transaction Performance

Compression Algorithm	Transactions finished in 10 minutes	Average Transaction per second	Performance improvement
Baseline	162226	269.48	
ZLE	167773	279.16	3.59%
LZJB	191789	318.59	18.22%
LZ4	207503	342.98	27.27%

Storage saving comparison between the baselines and compressed for Order Entry scenario is shown in Table 6.

Table 6 Order Entry Compression Ratio

Compression Algorithm	Compression Ratio	Storage Saving
Baseline	1	
ZLE	1.69	40.83%
LZJB	3.02	66.89%
LZ4	3.76	73.40%

Improvement of maximum response time for Order Entry is shown in Table 7. Notice that LZJB compression algorithm improves maximum response time, while the other two compression algorithms make the maximum response time worse.

Table 7 Order Entry Max Response Time Improvement

Compression Algorithm	Maximum Response Time (ms)	Improvement	Relative Response Speed
Baseline	79862	0%	100%
ZLE	80625	-0.95%	99.05%
LZJB	39870	100.31%	200.31%
LZ4	95932	-16.75%	83.25%

CPU utilisation overhead for Order Entry scenario is shown in Table 8. Notice that CPU overhead of LZ4 algorithm is far greater than the other compression algorithms.

Table 8 Order Entry CPU Utilization Overhead

Compression Algorithm	System CPU	User CPU	CPU overhead
Baseline	2	6	
ZLE	3	6	12.50%
LZJB	4	6	25.00%
LZ4	7	16	187.50%

From the data in Table 5, Table 6 and Table 7 we calculated relative benefit ranking of compression

algorithm for performance improvement, storage saving, and SLA improvement using pair-wise comparison matrices. We used the relative response speed column in Table 8 to avoid negative comparison values in comparison.

The compression with the highest benefit ranking for Order Entry scenario can be calculated by multiplying alternatives ranking for each criteria with the criteria ranking, as shown in (1) below:

$$\begin{matrix} \text{ZLE} \\ \text{LZJB} \\ \text{LZ4} \end{matrix}\begin{vmatrix} 0.225431 & 0.073146 & 0.258879799 \\ 0.369313 & 0.371231 & 0.523535715 \\ 0.405256 & 0.555623 & 0.217584486 \end{vmatrix} \times \begin{vmatrix} 0.333 \\ 0.333 \\ 0.333 \end{vmatrix} \begin{matrix} \text{Saving} \\ \text{Performance} \\ \text{Response} \end{matrix} = \begin{vmatrix} 0.18563311 \\ 0.42093855 \\ 0.39242834 \end{vmatrix} \begin{matrix} \text{ZLE} \\ \text{LZJB} \\ \text{LZ4} \end{matrix} \quad (1)$$

Based on the calculation, in Order Entry scenario LZJB algorithm has the highest benefit, followed by LZ4 and ZLE. Cost ranking for Order Entry scenario can be calculated using comparison matrix with data that was seen in Table 8.

For Order Entry scenario the compression with the highest cost ranking is LZ4 algorithm, followed by LZJB and ZLE. The calculation result of the benefit to cost ratio and comparison chart for it is shown in Figure 3.

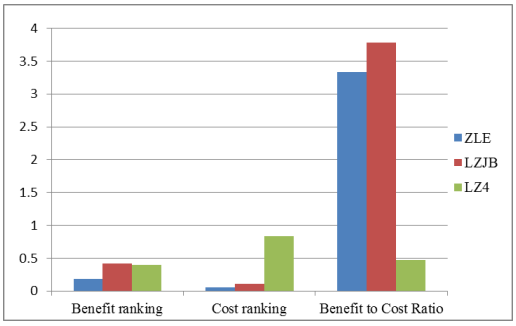


Figure 3 Order Entry Benefit to Cost ranking

The compression with the highest benefit to cost ranking is LZJB algorithm followed by ZLE and LZ4. LZ4 algorithm comes last in the ranking mainly because of its high CPU utilisation.

3.2 Result and Analysis from Calling Circle Scenario

The benchmark results between the baseline and the three compression algorithms for Calling Circle scenario show increased performance as shown in Table 9.

Table 9 Order Entry CPU Utilization Overhead

Compression Algorithm	Average Transaction per second	Performance improvement
Baseline	55.1	
ZLE	61.39	11.42%
LZJB	82.33	49.42%
LZ4	80.44	45.99%

Based on Table 5 and Table 9, we can see that on Calling Circle transaction is heavier and more CPU intensive than Order Entry scenario, shown by the lower average transaction per second achieved. Storage saving comparison between baseline database and compressed database for Calling Circle is shown in Table 10.

Table 10 Calling Circle Compression Ratio

Compression Algorithm	Compression Ratio	Storage Saving
Baseline	1	
ZLE	1.76	43.18%
LZJB	4.92	79.67%
LZ4	5.32	81.20%

Data in Calling Circle scenario are more compressible, shown by the higher compression ratio achieved compared to compression ratio in Order Entry scenario as shown in Table 6 and Table 10. The improvement of maximum response time for Calling Circle scenario is shown in Table 11.

Table 11 Max Response Time Improvement

Compression Algorithm	Maximum Response Time (ms)	Speed Improvement	Relative Response Speed
Baseline	26019		
ZLE	22613	15.06%	
LZJB	20758	25.34%	
LZ4	18481	40.79%	

As shown in Table 11, all compression algorithms in Calling Circle scenario improve maximum response time. Therefore, we do not need to use relative response speed. We will be able to use speed improvement column for Response Time Improvement comparison matrix because there is no negative or zero value. CPU utilisation overhead for Calling Circle scenario is shown in Table 12.

Table 12 Calling Circle CPU Utilization Overhead

Compression Algorithm	System CPU	User CPU	CPU overhead
Baseline	2	16	
ZLE	3	17	11.11%
LZJB	4	24	55.56%
LZ4	6	23	61.11%

From data in Table 10, Table 11 and Table 12, we calculated relative benefit ranking of compression algorithm for performance improvement, storage saving and SLA improvement using pair-wise comparison matrices.

Compression algorithm with the highest benefit ranking of Calling Circle scenario can be calculated by multiplying alternative ranking for each criterion with the criteria ranking as shown in (2):

$$\begin{matrix}\text{ZLE}\\ \text{LZJB}\\ \text{LZ4}\end{matrix}\begin{vmatrix}0.2116148 & 0.106898811 & 0.185490824\\ 0.390443519 & 0.462604137 & 0.312107402\\ 0.397941681 & 0.430497051 & 0.502401774\end{vmatrix} \times \begin{vmatrix}0.333\\ 0.333\\ 0.333\end{vmatrix} \begin{matrix}\text{Saving}\\ \text{Performance}\\ \text{Response}\end{matrix} = \begin{vmatrix}0.167833477\\ 0.387996634\\ 0.443169888\end{vmatrix} \begin{matrix}\text{ZLE}\\ \text{LZJB}\\ \text{LZ4}\end{matrix} \quad (2)$$

Based on the calculation result, on Calling Circle scenario, LZ4 algorithm has the highest benefit, followed by LZJB and ZLE. The cost ranking of Calling Circle scenario can be calculated using comparison matrix with data from Table 11.

On the Calling Circle scenario the compression algorithm with the highest cost ranking is LZ4, followed by LZJB and ZLE. The calculation of benefit to cost ratio and comparison chart for it, is shown in Figure 4.

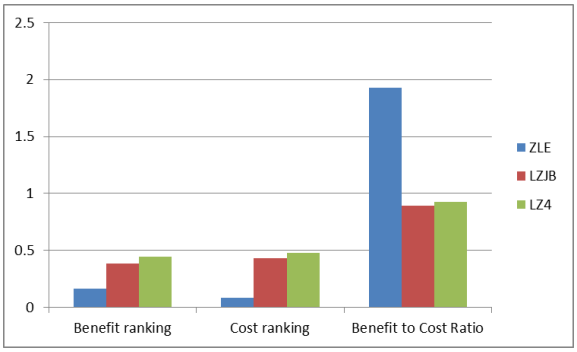


Figure 4 Calling Circle Benefit to Cost ranking

The compression algorithm with the highest benefit to cost ranking on Calling Circle scenario is ZLE algorithm followed by LZ4 and LZJB. ZLE algorithm comes first in this ranking mainly because of its very low CPU overhead.

3.3 Result and Analysis from Stress Test scenario

Stress Test is the benchmark with the lightest type of transactions shown by the highest average transaction per second achieved, but it is also the benchmark with the highest write ratio which results in the lowest performance improvement as shown in Table 13.

Table 13 Stress Test Transaction Performance

Compression Algorithm	Average Transaction per second	Performance Improvement	Relative Performance to Baseline
Baseline	8762.64	0%	100%
ZLE	9216.66	5.18%	105.18%
LZJB	9089.19	3.73%	103.73%
LZ4	8612.65	-1.71%	98.29%

LZ4 caused the decrease in performance in Stress Test scenario. We had to use Relative Performance value to avoid the negative comparison later. Storage saving comparison between baselines and compressed for Stress Test is shown in Table 14.

Table 14 Stress Test Compression Ratio

Compression Algorithm	Compression Ratio	Storage Saving
Baseline	1	
ZLE	1.58	36.71%
LZJB	2.94	65.99%
LZ4	3.49	71.35%

As usual, LZ4 achieved the highest storage saving followed by LZJB and ZLE. Improvement of maximum response time for Order Entry is shown in Table 15.

Table 15 Stress Test Max Response Time Improvement

Compression Algorithm	Maximum Response Time (ms)	Speed Improvement	Relative Response Speed
Baseline	4978257	0%	100%
ZLE	8282371	-39.89%	60.11%
LZJB	4003496	24.35%	124.35%
LZ4	2487597	100.12%	200.12%

In Stress Test scenario, ZLE algorithm achieved the highest performance improvement but made the maximum response time worse. We had to use Relative Response Speed in the comparison matrix. CPU utilisation overhead for Stress Test scenario is shown in Table 16.

Table 16 Stress Test CPU Utilization Overhead

Compression Algorithm	System CPU	User CPU	CPU overhead	Relative CPU utilization
Baseline	6	8	0%	100%
ZLE	6	8	0%	100%
LZJB	7	8	7.14%	107.14%
LZ4	9	7	14.29%	114.29%

LZ4 algorithm consistently showed the highest CPU utilisation overhead followed by LZJB and ZLE. CPU overhead for ZLE is so small it measured zero. We used Relative CPU utilisation to avoid comparison with zero. From data in Table 14, Table 15 and Table 16, we calculated relative benefit ranking of compression algorithm for performance improvement, storage saving and SLA improvement using pair-wise comparison matrices.

Compression algorithm with the highest benefit ranking for Calling Circle scenario can be calculated by multiplying alternatives ranking for each criterion with the criteria ranking as shown in (3)

$$\begin{matrix}\text{ZLE}\\ \text{LZJB}\\ \text{LZ4}\end{matrix}\begin{vmatrix}0.210916403 & 0.34238281 & 0.15630038\\ 0.379143924 & 0.33766276 & 0.323339747\\ 0.409939673 & 0.31995443 & 0.520359873\end{vmatrix} \times \begin{vmatrix}0.333\\ 0.333\\ 0.333\end{vmatrix} \begin{matrix}\text{Saving}\\ \text{Performance}\\ \text{Response}\end{matrix} = \begin{vmatrix}0.236296665\\ 0.346368762\\ 0.416334573\end{vmatrix} \begin{matrix}\text{ZLE}\\ \text{LZJB}\\ \text{LZ4}\end{matrix} \quad (3)$$

Based on the calculation, on Stress Test scenario, LZ4 algorithm had the highest benefit, followed by LZJB and ZLE. Cost ranking for Calling Circle can be calculated using comparison matrix with data from Table 16. For Calling Circle scenario the compression with the highest cost ranking is LZ4, followed by LZJB and ZLE. Comparison chart for cost ranking and benefit ranking can be seen in Figure 5.

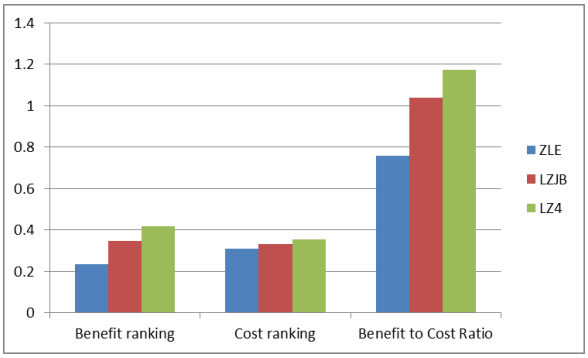


Figure 5 Stress Test Benefit to Cost ranking

The compression with the highest benefit to cost ranking for Stress Test was LZ4 followed by LZJB and ZLE. This might be because when the transaction types were very light and the cost differences will not change the benefit ranking.

3.4 Results Comparison across Scenario

We compared the results between scenario to see if there was any correlation between read/write ratio and the amount of performance improvement.

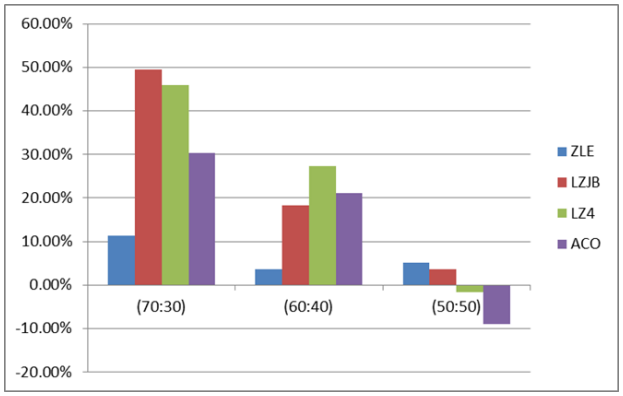


Figure 6 Performance Improvement All Scenario

Comparison Chart of Performance Improvement for All Scenario in Figure 6 shows trends of decreasing performance improvement for almost all compression algorithms when the write ratio increase. The exception is ZLE which improves between 60/40 to 50/50 read/write ratio. This is consistent with similar studies [13] which showed that with an increase in write ratio will require compression and decompression process more data thus increasing CPU usage and need more response time.

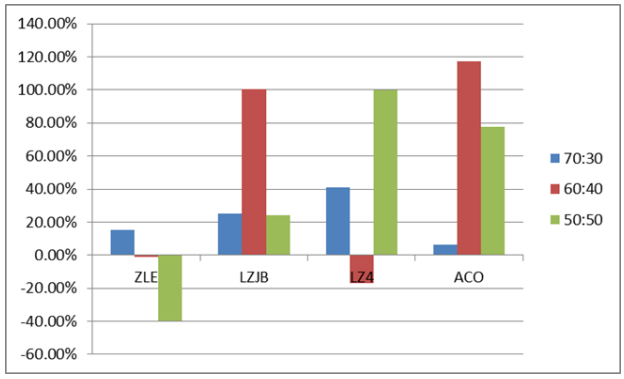


Figure 7 Maximum Response Time Improvement All Scenario

From the comparison chart in Figure 7, we can see that LZJB is the only compression that did not cause any increase in maximum response time. Meanwhile, from Figure 6, we can see that LZJB compression also never caused any performance decrease in this study. This observation makes LZJB was the safest compression algorithm to choose from the three algorithms, in case we have not enough knowledge about read-write ratio or other characteristics of the application accessing the database. Compression in the filesystem layer has the advantage that can be used for a variety of databases, not only for the Oracle database, but also do not require license fees. However, the compression in the database layer (ACO) also has unique advantages that can be used during the process of backup and data replication.

4.0 CONCLUSION

The results of the benchmark and its analysis have shown that compression at the file systems layer can improve OLTP database performance with the following things to note:

- Performance improvement tends to be higher for OLTP applications with higher read ratio
- Among the algorithm studied here, LZJB is the safest to implement for OLTP and seems to strike the right balance between compression ratio and CPU overhead
- Applications with more complex query like Calling Circle scenario will yield more performance using light weight compression rather than higher compression ratio

We hope that the results of this study can help to increase the adoption of file systems compression in general and ZFS file systems particularly to help to save storage cost and improve OLTP application performance.

This study is limited by time and available equipment and there are still a lot of topics that can be pursued further for future work related to

performance improvement using file systems compression such as:

- OLTP performance improvement using other file systems besides ZFS
- OLTP performance improvement using compression on storage based on ZFS file-system
- OLTP performance improvement using file system compression on flash storage

References

[1] H. Plattner. 2009. A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database. *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09*. Providence, Rhode Island, USA.

[2] M. A. Bassiouni. 1985. Data Compression in Scientific and Statistical Databases. *IEEE Transactions on Software Engineering.* SE-11(10): 1047-1058.

[3] C. Diaconu, C. Freedman, E. Ismert, P.-Å. Larson, P. Mittal, R. Stonecipher, N. Verma and M. Zwilling. 2013. Hekaton: SQL Server's Memory -Optimized OLTP Engine. SIGMOD'13, New York.

[4] A. Cuzzocrea and S. Chakravarthy. 2010. Event-based Lossy Compression for Effective and Efficient OLAP over. *Data & Knowledge Engineering*. 69(1): 678-708.

[5] N. Mukherjee, S. Chavan, M. Colgan, D. Das, M. Gleeson, S. Hase, A. Holloway, H. Jin, . J. Kamp, K. Kulkarni, T. Lahiri, J. Loaiza, N. Macnaughton, V. Marwah, A. Mullick, A. Witkowski, J. Yan and M. Zait. 2015. Distributed Architecture of Oracle Database in-memory. *International Conference on Very Large Data Bases.* Kohala Coast, Hawaii.

[6] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan and M. Zhang. 2015. In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering.* 27(7): 1920-1948.

[7] G. Ray, J. Haritsa and S. Seshadri. 1995. Database Compression: A Performance Enhancement Tool. *International Conference on Management of Data.*

[8] G. Graefe and L. D. Shapiro. 1991. Data Compression and Database Performance. *ACM/IEEE-CS Symposium on Applied Computing, Kansas City.*

[9] T. Westman, D. Kossmann, S. Helmer and G. Moerkotte. 2000. The Implemmentation and Performance of Compressed Databases. *ACM SIGMOD Record*. 29(3): 55-67.

[10] T. Ravichandran. 2013. Real Time Database Compression Optimization Using Iterative. *Computer Science & Information Technology (CS & IT)*. 3(1): 99-105.

[11] A. Habib, A. S. M. L. Hoque and M. S. Rahman. 2012. High Performance Query Operations on Compressed Database. *International Journal of Database Theory and Application.* 5(3): 1-14.

[12] C. Lin, J. Wang and Y. Papakonstantinou. 2016. Data Compression for Analytics over Large-scale In-memory Column Databases. arXiv:1606.09315v2: 1-4,

[13] A. Wenas and Suharjito. 2016. Improving Data Warehouse Performance Using Filesystem Technology with GZIP, LZJB and ZLE Compression. *Jurnal Informatika dan Sistem Informasi*. 2(2): 40-51.

[14] A. Tamrakar and V. Nanda. 2012. A Compression Algorithm for Optimization of Storage Consumption of Non Oracle Database. *International Journal of Advanced Research in Computer Science and Electronics Engineering*. 5(1): 39-43.

[15] M. Sharma and S. Dora. 2012. Efficient Approach for Compression in Data Warehouse. *International Journal of Computer Applications.* 53(9): 9-11.

[16] D. Giles. 2015. Swingbench [Online]. Available: http://www.dominicgiles.com/Swingbench.pdf. [Accessed 2015].

[17] IBM. 2014. Oracle Database 11g and 12c on IBM Power Systems built with IBM Power8 processor technology and IBM FlashSystem 840. IBM Oracle International Competency Center.

[18] VMware. 2010. Oracle Databases on vSphere Workload Characterization Study. VMware, Palo Alto.

[19] F. N. Almari, P. Zavarsky, R. Ruhl, D. Lindskog and A. Aljaedi. 2012. Performance Analysis of Oracle Database in Virtual Environments. *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on, Fukuoka.*

[20] I. E. Tope, P. Zavarsky, R. Ruhl and D. Lindskog. 2011. Performance Evaluation of Oracle VM Server Virtualization Software 64 Bit Linux Environment. Security Measurements and Metrics (Metrisec*). 2011 Third International Workshop on, Banff, AB.*

[21] D. Ye, A. Pavuluri, C. Waldspurger, B. Tsang, B. Rychlik and S. Woo. 2008. Prototyping a Hybrid Main Memory Using a Virtual Machine Monitor. Computer Design, 2008. *ICCD IEEE International Conference on, Lake Tahoe, CA.*

[22] T. L. Saaty. 2008. Decision Making with Analytic Hierarchy Process. *Int. J. Services Sciences*. 1(1): 83-98.

[23] T. L. Saaty and M. Ozdemir. 2003. Negative Priorities in the Analytic Hierarchy Process. *Mathematical and Computer Modelling*. 37(9): 1063-1075.