Improving Data Warehouse Performance Using Filesystem Technology with GZIP, LZJB and ZLE Compression

Andre Wenas[1], Suharjito[2]

Abstract— **Data warehouse application is commonly used by many corporations as an analytical platform to develop marketing strategy and give advantage edge to their business. Many times, data warehouse platform need to manage huge amount of data as it needs to extract, load and transform enterprise data from many sources. To improve performance of data warehouse system, compression has been used in various ways, both at database layer or at filesystem layer. Compression brings an other benefit as it reduce storage capacity therefore reduce cost. However compression also add data processing that may impact to overall application response time. In this research, three different compression algorithms which are zle, lzjb and gzip were tested on data warehouse filesystem to understand the performance impact and the capacity saving benefit. Using swingbench as the benchmark tool and oracle database 12c, it shows that zle is the best compression algoritm with performance improvement of 92%, follows by lzjb with 78% performance improvement and gzip with 53% improvement. In terms of compression ratio, gzip can deliver the highest with 3.31 compression ratio, follows by lzjb with 2.17 compression ratio and zle with 1.55 compression ratio. AW.**

Keywords: data warehouse; performance; compression; zle; lzjb; gzip

I. INTRODUCTION

The complexity of the modern business world contributes to the complexity of today's IT infrastructure. Growth, mergers, acquisitions and long-term IT investment has created the technology landscape is characterized by silos of information held on different systems in different departments, subsidiaries and geographies. More data captured and stored by the business now than ever before. The explosion of data and data processing applications continue to be a challenge. For example, Infosys, a multinational company in India, the data warehouse system grown by 45% with transaction volume growing by 110% annually [1].

The amount of data that must be processed increases exponentially, so it takes a certain method that can store data more efficiently over a long period. One method for reducing the amount of data storage is to use a compression (Roth and Van Horn, 1993). Compression technology is widely used to store individual files and is used to store data in a relational database in corporations.

As data warehouse is one of the important applications for commercial business, therefore improving performance and reducing storage capacity will bring significant benefits for the business. For example, with a compression ratio of 2:1, businesses will spend half of their IT budget to meet the same requirements. It will help directly to the profitability of the company. At the same time, improving the performance will give better customer satisfaction [2].

In this study, we want to see whether data compression can increase performance of data warehouse using the latest hardware technology. We need to observe how much increase in performance, impact on CPU overhead and saving compression ratio.

There are few option where to implement compression, compression can be done at the database layer or at the filesystem layer. At filesystem layer that can perform compression transparently, filesystem will perform compression processing in memory cache that can reduce the overhead of compression and uncompress processes, and accelerate the writing of the data that has been compressed into a disk [3].

In this paper, we are experimenting to use filesystem compression with zle, lzjb and gzip compression to improve data warehouse performance. A simulated data warehouse benchmark was conducted to measure the performance improvement, compression ratio, response time and CPU overhead against filesystem without compression and oracle database ACO compression.

Compression refers to the process of reducing the amount of data of the original information. Database compression is widely used in data management to performance improvement and to save storage space. Data compression is the art of finding a short description for the long string. each compression algorithm can be decomposed into zero or more transformations, model, and program [4]. The goal of data compression is to reduce the redundancy of data stored or communicated, thereby increasing the effective data density. Data compression has important applications in the areas of file storage and distributed systems [2].

[1]*Information Technology Graduate Program, Bina Nusantara University Information Technology Faculty,e-mail: andre.wenas@gmail.com*

[2]*Information Technology Graduate Program, Bina Nusantara University Information Technology Faculty,e-mail: suharjito@binus.edu*

Data compression is often referred to as coding, in which the coding is a very general term that covers any specific data representations that meet specific needs. Information theory is defined as the study of efficient coding and its consequences, such as the transmission speed and probability of error. Data compression can be regarded as a branch of information theory in which the main objective is to minimize the amount of data transferred. A simple characterization data compression is that it involves changing a string of characters in multiple representations (such as ASCII) into a new string (bits, for example) which contains the same information but whose length is as small as possible [5]. Lossless compression algorithm for text can be divided into 2 groups: compression algorithm based on statistical entropy encoding for example Shannon-Fano coding, Huffman coding, Adaptive coding and compression algorithms based encoding directory such as Lempel-Ziv, LZ77 and LZ78 [6] [7].

While data compression have significant benefits for businesses, also inherited a problem: the overhead of compression / decompression [3]. Overhead issued by the process to compress data and expansion process to recover the original data is one of the most serious weaknesses of the data compression. For some applications, this overhead can be large enough to prevent consideration for the use of data compression. On the other hand, the use of compression will reduce the amount of data that must be read or written. Reduction of amount of data that must be read or written will reduce the number of transactions to storage IO significantly. Because IO transactions are much slower than data processing, the use of compression will improve performance of the database significantly.

In four decades we have witnessed an exponential growth in computing power. Chip microprocessor technology used billions of transistors, including multiple processor cores, running at speeds measured in gigahertz, and provides more than 4 million times the performance of the original [8]. On the other hand, the basic architecture of the disk drive has not changed much since it was introduced. The growth curve relative performance (IOPS) of about 25 percent, less than the rate of increase disk capacity [9]. Very likely that with the latest technology micro-processor, the CPU is much faster to process compression / decompression that will help to improve overall performance.

The effectiveness of the technique of compression algorithms is determined by three things: the compression ratio to see the savings capacity, performance compression and de-compression performance. Comparison of different lossless compression algorithms is listed on table 1.

Table 1 – comparison of different compression algorithms

Literature	Compression algorithm	Comparison
[6]	Entropy encoding: Huffman dan Shannon-Fano. Directory encoding: LZW from Lempel-ziv family	Compression ratio: LZW provides best ratio, follows by Huffman and Shannon-Fano. Huffman and Shannon-Fano compression is faster than LZW, whereby for decompression LZW is faster than Shannon-Fano.
[7]	Entropy/statistic encoding: Shannon-Fano Coding, Huffman coding, Adaptive coding, Run Length encoding (RLE), Arithmetic encoding. Directory encoding: Lempel-Ziv family, LZ77 dan LZ78	Compression ratio: Directory encoding Lempel-ziv has better compression ratio compare to entropy encoding family.

This research will use three types of algorithms are gzip, lzjb and zle. Here is a description of the intended compression:

- ZLE (Zero Length Encoding) algorithm is a compression algorithm that eliminates or compress the data with zero value (zero). With zero data compression saves storage and accelerate data transmission. This algorithm is derived from the algorithm RLE (Run Length Encoding) that belong to the entropy encoding although some are classified RLE encoding as a separate group. RLE algorithm is very easy to implement, therefore it has a very fast performance [10].
- LZJB algorithm (Lempel Ziv Jeff Bonwick) is a compression algorithm invented by Jeff Bonwick to perform data compression on ZFS filesystem. This algorithm is based on the method LZRW1 algorithm that also included family Lempel-Ziv compression algorithm by encoding the directory. LZRW algorithms designed to get the performance as soon as possible so it is suitable for compression filesystem [10].
- GZIP algorithm is a compression algorithm invented by Jean Loup gaily and Mark Adler in 1992 and is widely used in the open-source software, especially Linux. This algorithm is based on the DEFLATE algorithm is a combination of LZ77 (including family Lempel-Ziv algorithms – directory encoding) and Huffman coding (entropy encoding) [11].

Oracle Advanced Compression Option (ACO) provides compression capabilities at the database layer to help boost performance while reducing storage costs. This

allows database administrators to significantly reduce database storage for all types of data including relational data (tables), unstructured (file), an index, and data backup. Although the cost savings in storage and server optimization (production, development, QA, Test, Backup) that can be seen as the most tangible benefit, in addition to the compression ACO is designed to improve performance for all infrastructure components, including memory, network bandwidth and storage [12].

Although research on database compression has been around almost as long as there is the research database, the compression method is not widely used in the database until the 1990s. This may be because in the early research compression only focused on reducing storage space and not much see the effects on the performance of the database. On some of this research, it appears that although the compression techniques can reduce the number of IO, but the CPU overhead of compression process should be reduced as little as possible. Because it is necessary to find a technique that light compression algorithm so that the overhead of the compression process is much less of a reduction in the number of IO [13].
Some research has been done to see the effect of compression on the data warehouse as listed in table 2.

Table 2 – Literature study on database compression

Literature	Compression method and database	Relevance	Performance evaluation
[14]	Few compression method on column oriented database such as RLE, Bit-Vector Encoding, Dictionary, FOR, Patching Technique on MonetDB, VectorWise dan C-Store database	Using experimental database instead of commercial database	No performance test
[15]	H-HIBASE compression (enhancement of HIBASE compression).	Shows better performance compare to Oracle 10g. Still experimental stage.	Storage performance is 25-40% better than Oracle database 10g and query performance is 10-25% better than Oracle database 10g
[4]	ILC (Iterative Length Compression) on Microsoft SQL Server 2008	Showing high compress ratio and better performance. Need to analyze database and create a model before doing compression.	Compression ratio of ILC is 2.76.
[16]	gzip, lzo dan snappy compression on HBase – Hadoop database column oriented data-stores	This study shows compression can reduce storage requirement and improve performance on hadoop database.	Compression ratio: gzip – 13,4%, lzo – 20.5%, snappy – 22.2%. Encoding performance: gzip- 21 MB/s, lzo-135MB/s, snappy – 172 MB/s, Decoding: gzip – 118MB/s, lzo – 410MB/s, snappy – 409MB/s
[13]	RLE, Bit-vector, Dictionary single-value dan dictionary multi-value compression on C-vector database	This study compares response time of 4 different compressions on small and big size of data.	dictionary single and multiple value compression showing better performance. RLE compression showing worse performance on small data and C vector showing worse performance on big data.

Research in the field of data compression is usually aimed at the individual files. The use of compression on individual files already widely used and many benefits in everyday use such as data storage or sending files by email. However, the use of compression of individual files is not practical for database transactions since the database is usually stored in a file record is great and always changing. Some particular database also has the ability to perform compression in the database, such as Oracle databases. But requires additional license fees are not cheap. Because the researchers wanted to use compression technology in the filesystem. So from the database do not need no additional license fee, also do not need to compress individual files manually.

In this connection, will give rise to questions such as:

1. Technology filesystem what can be used?
2. How much overhead cpu to do the compression?
3. Is the compression technology will improve performance? Or otherwise slow down the performance of the database?
4. How much storage capacity savings?

In this research we use oracle database, Solaris 11 operating system and ZFS filesystem. ZFS filesystem has features that can make transparent the data compression. This means that compression is not necessary in individual files, but enough was set on a particular file system, automatically all the files in the filesystem will compress the appropriate algorithm is selected.

II. METHODS

Research in the field of data compression is usually directed at the individual files. The use of compression on the individual files are already widely used, and many benefits in everyday use such as data storage or transmission of files via email. However, the use of compression of individual files to a database transaction is not practical because it is usually stored in a database record files are huge and ever-changing. In this research the authors use the Solaris 11 operating system and ZFS filesystem. ZFS file system has features that can make transparent the data compression. This means that compression is not necessary in individual files, but simply set on a particular file system, automatically all the files in the file system will compress with appropriate algorithm is selected.

The steps of this study are: literature studies, instrument installation, implementation of compression for data warehouse, data loading, test for 3 compression algorithms, data collection, analysis of the results and conclusions and suggestions. In the initial phase of research begins by determining the background and purpose of the study as well as defining the scope. The literature study is done to deepen the understanding of the compression technology for data warehouse. Moreover, the study of literature is also conducted to find out the results of compression technology ever done for the data warehouse. The second phase of this study is the installation of instrumentation testing. Tool to be used is swingbench. In the third phase, carried out the implementation of compression technology for data warehouse using Oracle Database 12c with zfs filesystem. After it is done loading the data for the type of data warehouse. Furthermore, the tests performance for three different compression algorithm that is gzip, lzjb and zle. It will also be measured against the same dataset without compression (baseline) and compression using compression techniques ACO from oracle database. In the sixth stage of data collection that is the compression ratio and improved performance. In the seventh stage to analyze the data and then the last stage is to draw conclusions and make suggestions.

Tool measurement that will be used is swingbench benchmark tool. Swingbench is a tool to benchmark and generate load which are designed to test the performance of Oracle database version 10g, 11g and 12c. Swingbench have 4 types of tests for different workload characteristics, namely order entry, sales history, calling circle and stress test. We will use type test for data warehouse category was sales history, therefore in this research will be used test saleshistory. Sales History based schema "sh" which is included in Oracle Database 11g and 12c and is designed to test the performance of complex queries against very large tables. This database is read-only and can be expanded from 1GB to 1TB [17].

Tool swingbench chosen as a measurement tool because this tool is widely used as a measure of the performance of companies that provide hardware for data warehouse such as:

- IBM: in one of his Redbook, IBM uses swingbench to demonstrate the performance of data warehouse using IBM Power series server [18].
- EMC: use swingbench to demonstrate the performance of the data warehouse using the EMC VMAX storage [19].
- Oracle: use swingbench to show performance data warehouse appliance using a database [20].
- VMware: use swingbench to demonstrate the performance of data warehouse using flash technology read cache [21].
- Cisco: use swingbench to demonstrate the performance of the data warehouse using UCS and EMC products clarion storage [22].

Sales History database used to wear a star schema. Is a simple star schema data warehouse schema. It is called a star schema because the diagram resembles a star schema star, with points radiating from a center. Star center consists of one or more tables facts and points of the star are the dimension tables as in figure-1.

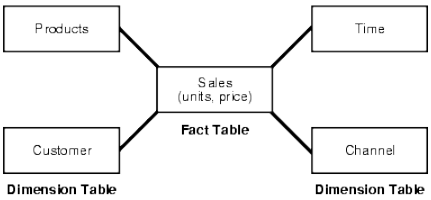

Figure-1 Star schema

In the measurement, carried out three different types of transactions to represent the data warehouse load. The three types of transaction are listed in table 3.

Table 3 – Testing three types of transactions

Transaction type	SQL statement
Sales rollup by month and channel	SELECT channels.channel_desc, calendar_month_desc, countries.country_iso_code, TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$ FROM sales, customers, times, channels, countries WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND customers.country_id = countries.country_id AND sales.channel_id = channels.channel_id AND channels.channel_desc IN ('Internet','Partners') AND times.calendar_month_desc IN ('2005-06','2005-07','2005-08','2005-09','2005-10') AND countries.country_iso_code IN ('NL','NZ') GROUP BY ROLLUP(channels.channel_desc, calendar_month_desc, countries.country_iso_code)
Sales within quarter by country	SELECT SUM(amount_sold), t.calendar_month_desc, t.calendar_week_number, c.country_name FROM sales s, times t, countries c, customers cu WHERE s.time_id = t.time_id AND t.calendar_month_desc = '2011-01' AND cu.country_id = c.country_id AND s.cust_id = cu.cust_id AND c.country_iso_code = 'AU' group by t.calendar_month_desc, t.calendar_week_number, c.country_name

Top Sales by quarter	SELECT * FROM (SELECT times.calendar_quarter_desc, customers.cust_first_name, customers.cust_last_name, customers.cust_id, SUM(sales.amount_sold), rank() over(PARTITION BY times.calendar_quarter_desc ORDER BY SUM(amount_sold) DESC) AS rank_within_quarter FROM sales, customers, times WHERE sales.cust_id = customers.cust_id AND times.calendar_quarter_desc = '2006-1' AND times.time_id = sales.time_id GROUP BY customers.cust_id, customers.cust_first_name, customers.cust_last_name, customers.cust_id, times.calendar_quarter_desc) WHERE rank_within_quarter < 16

Testing was done using the following machine configuration:

- Number of CPU cores: 4
- Memory size: 16 GB
- Storage: internal storage 1x 300GB for OS and 4x 300GB with RAID 0.
- Operating system: Solaris 11.2

After OS installation, the zfs filesystem was created using the following command:

```
# zpool create backend \
c0t5000CCA0162A4D60d0 \
c0t5000CCA0162A4D60d0 \
c0t5000CCA01632AE1Cd0 \
c0t5000CCA0162C5CBCd0
```

Creation of filesystem with compression to store the data using the following command:

```
# zfs create backend/baseline
# zfs create backend/aco
# zfs create backend/zle
# zfs create backend/lzjb
# zfs create backend/gzip
```

Next step was setting up the compression using the following command:

```
# zfs set compression=zle backend/zle
# zfs set compression=lzjb backend/lzjb
# zfs set compression=gzip backend/gzip
```

After this setup, all data that stored in the filesystem will be compressed with specific compression algorithm. To find out the compression ratio of each filesystem:

```
# zfs get compressratio backend/zle
```

For Oracle ACO compression, the compression setup was done during data loading.

The next stage is to install Oracle enterprise database 12c in all filesystems that has been created which are backend / baseline, backend / aco, backend / zle, backend / lzjb and backend / gzip. After oracle database installation was done swingbench tools that will be used to create data and conduct trials. Loading data using scripts facilities already provided by swingbench. This script is called by shwizard. Again the data loading process was conducted in all five filesystems repeatedly. While loading data ACO filesystem backend / aco, advanced compression option must be selected as shown in figure 2.

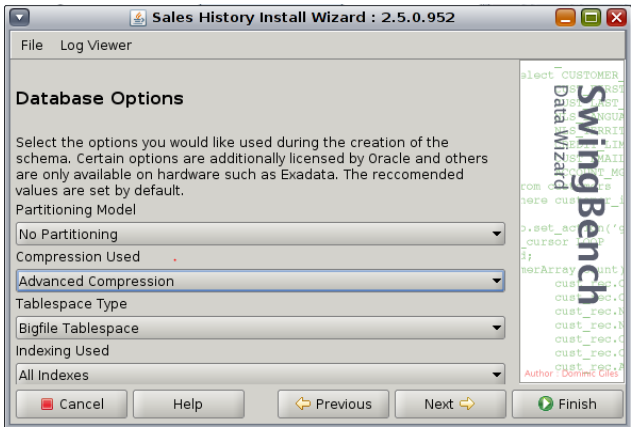

Figure 2 – Choosing advanced compression during data loading to test ACO

Execution of the test was done using the following command:

```
# /charbench -c shconfig.xml -r hasil/result.xml -
stats full -dumptx -dumptxdir hasil.
```

III. RESULT

Measurements were made against a database with the same amount of data and the same number of users for each 20 minutes. The results of these measurements to get the number of transactions, as shown in Table 4.

Table 4 – comparison of number of transactions

Database	Number of transaction in 20 minutes	Average transaction per second	Performance improvement
Without compression - baseline	156	7.8	-
With ACO compression	198	9.9	26%
With ZLE compression	300	15	92%
With LZJB compression	279	13.95	78%
With GZIP compression	239	11.95	53%

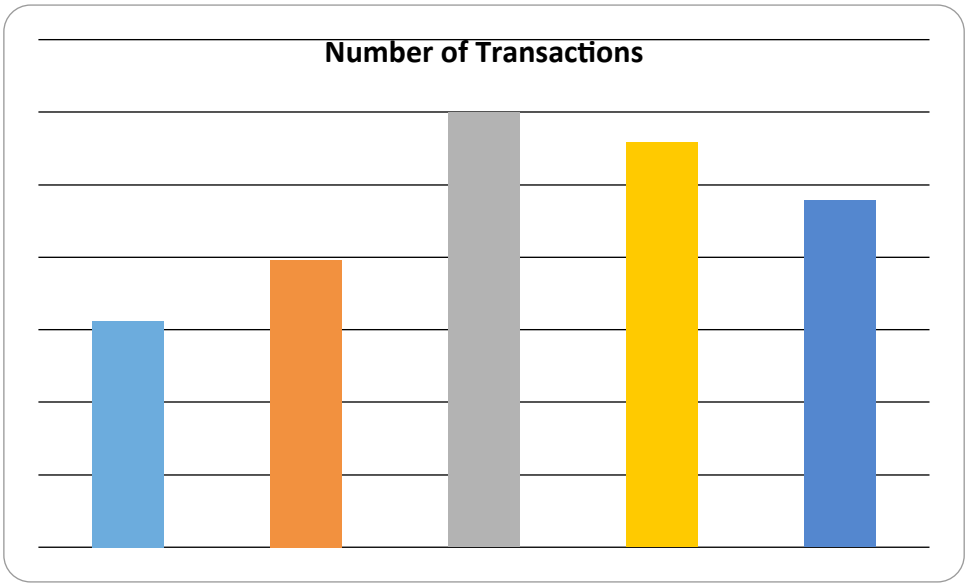

Figure 3 – Throughput comparison

With filesystem compression the number of transactions was improved by 30% to 100%. By using ACO compression, performance was increased by 26%, lower when compared to the filesystem compression.

Compression ratio shows how much storage can be saved from the use of compression. The higher the compression ratio, the more-efficient use of storage and cost savings. Comparison of the compression ratio obtained from experiments performed is shown in Table 5.

Table 5 – comparison of compression ratio

Compression type	Compression ratio
Baseline – without compression	1
ACO	1.3
ZLE	1.55
LZJB	2.17
GZIP	3.31

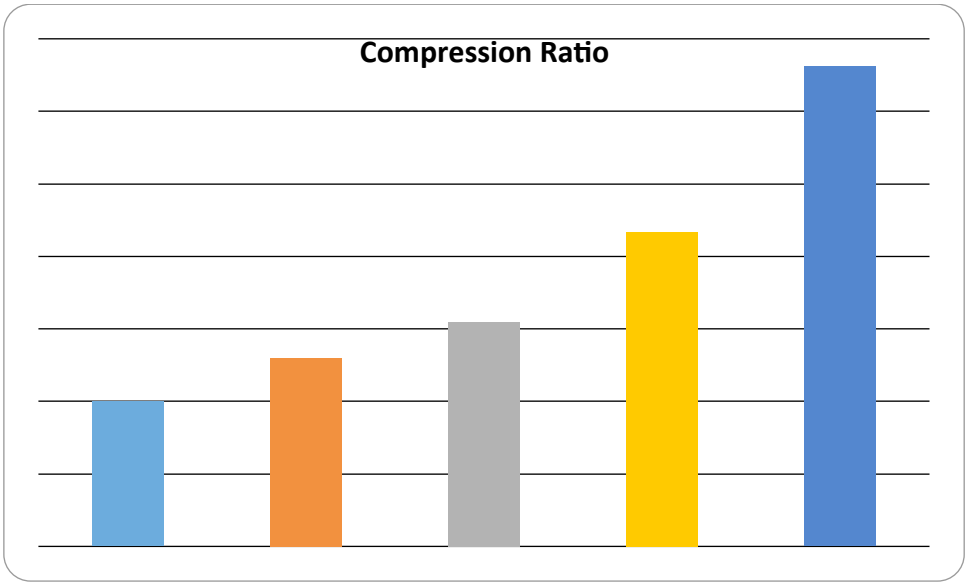

Figure 4 – compression ratio

Figure 4 shows gzip compression helps save storage capacity the most followed by lzjb, zle and aco. This is consistent with other literature that shows the encoding compression group directory including the group Lempel-Ziv provide better compression ratio than RLE compression group [7].

Average utilization while running the test on different compression type is shown in table 5 and figure 4.

Table 6 – CPU utilization comparison

Compression type	Average % user CPU	Average %system CPU	Overhead %system CPU to process compression
Baseline	7	1	-
ACO	14	2	1
ZLE	22	2	1
LZJB	19	5	4
GZIP	14	11	10

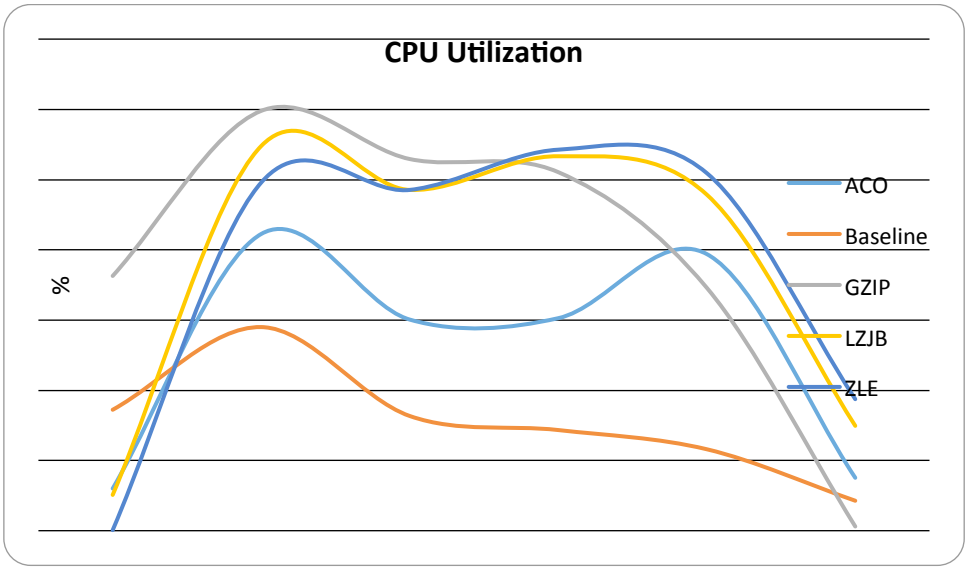

Figure 5 – CPU utilization

Table 5 shows CPU usage to run user mode looks to increase with compression. This is because the database can do more transactions than without compression. While the use of the CPU for increased system visible mode to perform the compression and decompression of data. This is consistent with other literature review that zle algorithm is the simplest therefore it almost has no processing overhead [10].While gzip based on the DEFLATE algorithm is a combination of LZ77 (including family Lempel-Ziv algorithms - encoding group directory) and Huffman coding (entropy encoding group) [11], because it uses two compression methods once so has the longest compression process and result in high CPU overhead.

Response time comparison during the test is shown in table 6 and figure 5.

Table 7 – Response time comparison

Transaction type	Compression	Minimum (ms)	Average (ms)	Maximum (ms)
Rollup by Month and Channel	Baseline	49,136	205,388	347,436
	ACO	70,159	146,543	277,050
	ZLE	31,327	102,107	179,215
	LZJB	35,679	107,040	184,698
	GZIP	46,236	126,223	230,842
Top Sales by Quarter	Baseline	77,234	252,924	397,200
	ACO	54,521	180,828	285,186
	ZLE	44,856	116,660	206,228
	LZJB	44,138	126,154	224,033
	GZIP	43,161	149,613	241,453
Sales within Quarter by Country	Baseline	48,162	199,312	350,949
	ACO	54,582	143,375	235,511
	ZLE	35,363	104,787	179,202
	LZJB	46,793	106,359	181,566
	GZIP	46,082	124,903	216,585

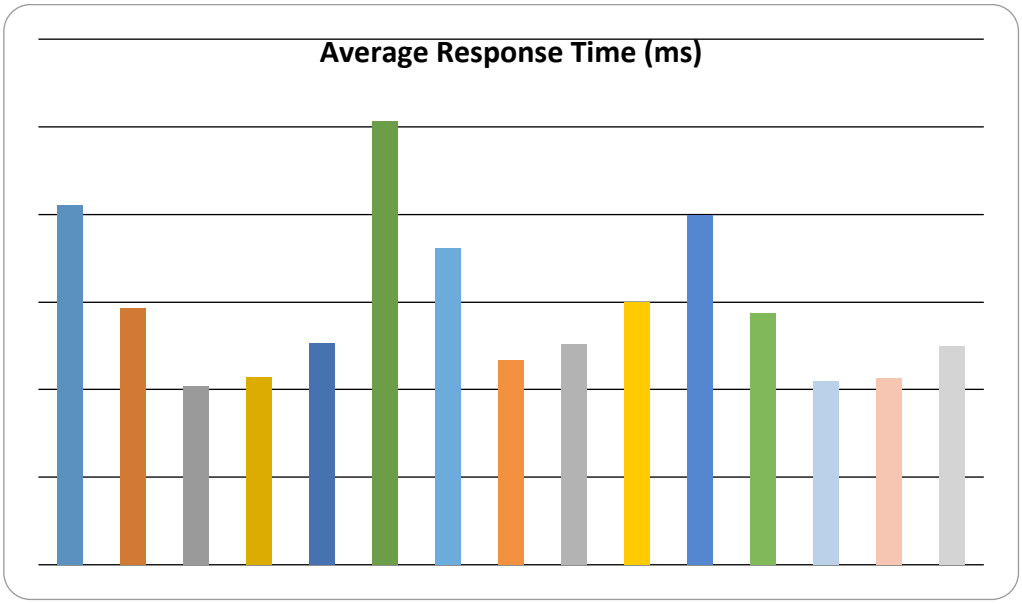

Figure 6 – Average response time

Table 7 and figure 6 shows filesystem compression can reduce the response time. However, the higher the compression ratio, the smaller the reduction of response time. This is because the more dense compression is done, the higher overhead of compression and decompression process.

Based on some of the observations above, the overall performance of the database with the most excellent ZLE compression, but judging from the GZIP compression ratio has the best ratio GZIP but in this study does not give a good performance. This is because the gzip compression algorithm requires more intensive compression process indicated by the high average CPU utilization system for compression by 10%. The gzip compression process into high overhead causes the average response time higher than zle or lzjb algorithms and result in lower throughput performance gzip algorithm.

IV. CONCLUSION

From the test results and measurements made for the performance of the data warehouse with different types of compression can be summed up as follows:

- The use of compression at the level of compression in the filesystem or database level can improve data warehouse performance compared with no compression.
- Improved performance using compressed filesystem better than the use of advanced compression techniques oracle option or ACO.
- Use of compression will increase CPU utilization. The higher the compression ratio, the higher the CPU is required. The addition of the CPU load is approximately 1% to 10% depending on the type of compression used.
- Filesystem compression saves storage capacity utilization significantly from 155% to 331% while the ACO compression techniques can save storage at 130%.
- The use of filesystem compression will increase the response time by 30% to 50%.
- Improved performance of the largest when using this type of compression zle, but compression zle has the lowest compression ratio.
- Increasing the highest compression ratio when using gzip compression types, but gzip compression requires additional CPU maximum of 10%.

In keeping with the focus of this study was to determine the increase in performance datawarehouse it can be concluded that the filesystem compression algorithms zle can improve performance by 92% with the best compression ratio of 1.5x or 150%. If desired compression ratio greater then lzjb algorithm can be selected with a compression ratio of 2.17x or 217% despite the increase in performance only by 53%.

Compared with ACO compression, filesystem compression does not depend on of database being used. So that could be used for other databases such as MySQL, Postgress, Sybase and so on. Although the increase in performance for other types of databases can be varied and needs further research. While the benefits of using compression techniques oracle ACO, can be done in a variety of operating systems and file systems supported by Oracle database.

V. REFERENCES

[1] S. Sharma and R. Jain, "Enhancing Business Intelligence using Data Warehousing : A Multi Case Analysis," *Int. J. Adv. Res. Comput. Sci. Manag. Stud.*, vol. 1, no. 7, pp. 160–167, 2013.

[2] M. A. Roth and S. J. Van Horn, "Database compression," *Sigmod Record*, vol. 22, no. 3, pp. 31–39, 1993.

[3] M.A. Bassiouni, "Data Compression in Scientific and Statistical Databases," *IEEE Transactions on Software Engineering,* vol. SE-11, no. 10, pp.1047-1058, 1985.

[4] M. Murugesan and T. Ravichandran, "Evaluate Database Compression Performance and Parallel Backup," *International Journal of Database Management System (IJDMS),* vol. 5, no. 4, 2013.

[5] D.A. Lelewer and D. S. Hirschberg, "Data Compression," *ACM Comput. Surv.,* vol 19, no. 3, pp. 261-296, 1987.

[6] S. Shanmugasundaram and R. Lourdusamy, "A Comparative Study of Text Compression Algoritms", *International Journal of Wisdom Based Computing,* vol. 1, no.3, pp. 68, 2011.

[7] A.K. Bhattacharjee, T. Bej, and S. Agarwal, "Comparison Study of Lossless Data Compression Algoritms for Text Data", *IOSR Journal of Computer Engineering (IOSR-JCE)*, pp.15-19, 2013.

[8] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "CPU DB: Recording Microprocessor History", 2012. [Online]. Available: http://queue.acm.org/detail.cfm?id=2181798. [Accessed: 14- Jun- 2015].

[9] E. Grochowski and R.D. Halem, "Technological impact of magnetic hard disk drives on storage systems", *IBM Systems Journal, ProQuest Science Journals*, 338, 2003.

[10] Y. Rathore, M.K. Ahirwar and R. Pandey, "A Brief Study of Data Compression Algorithms", *International Journal of Computer Science and Information Security (IJCSIS),*vol. 11, no. 10, 2013.

[11] W. Chang, B. Fang, X. Yun and S. Wang, "The Block Lossless Data Compression Algorithm", *International Journal of Computer Science and Network Security (IJCSNS)*, vol.9, no.10, pp. 116, 2009.

[12] Oracle, "Oracle Advanced Compression with Oracle", 2015. [Online]. Available: http://www.oracle.com/technetwork/database/options/compression/advanced-compression-wp-12c-1896128.pdf. [Accessed: 14- Jun- 2015].

[13] D. Abadi, S. Madden and M. Ferreira, "Integrating compression and execution in column-oriented database systems", *Proceedings of the 2006 ACM SIGMOD international conference on Management of data,* pp. 671-682, 2006.

[14] D. Abadi, P. Boncz, S. Hrizopoulos, S. Idreos, and S. Madden, "The Design and Implementation of Modern Column-Oriented Database Systems", *Foundations and Trends in Databases,* vol. 5, no. 3, pp. 197-280, 2012.

[15] A. Habib, A. S. Hoque and M .S. Rahman, "High Performance Query operations on Compressed Database", *International Journal of Database Theory and Application,* vol. 5, no. 3, 2012.

[16] P. Raichand and R.R. Aggarwal, A Short Survey of Data Compression Techniques for Column Oriented Databases. *Journal of Global Research in Computer Science*, vol. 4, no. 7, 2013.

[17] D. Giles, *Swingbench*, 2015. [Online]. Available: http://www.dominicgiles.com/Swingbench.pdf [Accessed: 14- Jun- 2015].

[18] D. Quintero, "IBM Power Systems Performance Guide Implementing and Optimizing", 2013. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg248080.pdf. [Accessed: 14- Jun- 2015].

[19] EMC, "EMC Tiered Storage for Oracle Database 11g - Data Warehouse", 2010. [Online]. Available: http://estonia.emc.com/collateral/solutions/white-papers/h7068-tiered-storage-oracle-vmax-fast-ionix-wp.pdf. [Accessed: 14- Jun- 2015].

[20] Ramachandran, "Evaluating and Comparing Oracle Database Appliance", 2014. [Online]. Available: http://www.oracle.com/technetwork/database/ database-appliance/documentation/oda-eval-comparing-performance-1895230.pdf. [Accessed: 14- Jun- 2015].

[21] S. Sivathanu, "Performance of vSphere Flash Read Cache in VMware Vsphere 5.5.", [Online]. Available: http://www.vmware.com/files/pdf/ techpaper/vfrc-perf-vsphere55.pdf. [Accessed: 14- Jun- 2015].

[22] Cisco, "Deploying Oracle Real Application Clusters on the Cisco Unified Computing System with EMC Clariion Storage", 2012. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-b-series-blade-servers/white_paper_c11-562881.pdf. [Accessed: 14- Jun- 2015].