ISSN 2278-3091
Volume 9, No.1.1, 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse1991.12020.pdf
https://doi.org/10.30534/ijatcse/2020/1991.12020

Query Optimization: Fund Data Generation Applying NonClustered Indexing and MapReduced Data Cube Numerosity Reduction Method

Mercy Burawis[1], **Rosmina Joy Cabauatan**[2]
[1]Technological Institute of the Philippines, Philippines, msb815@yahoo.com
[2]Technological Institute of the Philippines, Philippines, rjmcabauatan@yahoo.com

ABSTRACT

Nowadays, data aggregation is becoming the concern of the financial institution because this is very useful to answer the series of data over the years and at present. Big data is described as a large set of data at a different level that becomes a concern to which it is difficult to manage. It is a problem for most businesses to achieve better query performance while generating and analyzing large data by way of, data aggregation and such an exponential increase of data size makes a query to take a large amount of time and space. The data cube is a widely used tool to provide an efficient way to compute the data into a small data set. In this paper, the query optimization technique is to address the prolonged execution of the query by applying one of the data reduction strategies called numerosity reduction methods; slice and dice data cube operation is to reduce and efficiently aggregate yet maintains the accuracy of the data. The nonclustered index is to quickly retrieve the data without scanning the whole fact table and very useful for some repeated values. MapReduce based approach is for handling large scale data, in which it is of great help to enhance the data cube computation and achieve optimal time over large data set. The technique improves the response time by an average of 94%, and the availability of the memory space becomes 91%. With this, a timely increase in query performance could mean better use of data in operation and timely decision making for management.

Key words : NonClustered Index and MR Data Cube Computation, Numerosity Reduction Method, Query Optimization, MapReduce based approach.

1. INTRODUCTION

Nowadays, data aggregation is a predominant operation used in making business decisions [2]. The financial institution finds the importance of analyzing fund data as it will provide different views that help managements to gain valuable insight and used for decision making in their business. Data becomes “bigger” and the prediction about the size in the next year becomes more difficult, and means to the query for long response time and space. The uncontrollable size of data has been realized as it is now more difficult to produce because of the massive changes in data size over the years. Analyzing large data becomes one of the challenges. The company needs to ensure that the integrity of the data will not compromise and be able to produce correct and reliable insight reports for decision making in a real-time manner, but as it grows, eventually not easy to handle and manage.

Data Reduction is a technique used to reduce the data. Reducing the number of data set into small sets, yet the quality of the original data is maintained [1]. One of the data reduction strategies used is called the Numerosity Reduction method by applying the data cube aggregation model. In a typical database warehouse design, On-Line Analytical Processing (OLAP) is used to provide a summary report at different data levels and attributes [2]. Data Cube is one of the important technology and widely used for data analysis in support of decision making [3]. Aggregation is a common operation used in decision support database systems [2]. The Standard Group-By is significantly scanned the data set to be used for aggregation while in Data Cube, it reduces data for aggregation. Standard Group-by is not a simple operation but also has a significant pattern for the abstraction of data aggregation [4].

In Data Cube, Slice and Dice is one of the types of OLAP operation, embraces the multidimensionality paradigm to provide fast access to data sets when analyzing it from different views [5]. To make reasonable views, the slice-dice operation in a data cube can return a specific subcube to get a certain dimension. Slice and Dice operation can be useful to get only a certain portion of the cube of data for analysis. However, in outsized data, data cube alone is sometimes not enough to obtain significant query performance. A large number of data can be supported by applying an index in which can help to speed up the retrieval of data.

The index is commonly used in a Database management system that organizes the data records on disk and helps to improve the forms of the data retrieval operation [6].

NonClustered uses the keys which are not assigned as a primary key and to help to retrieve the row quickly from the table. A nonclustered index is very useful for attributes that have some repeated values [7]. This type of index is useful when searching in large data sets. Exploiting of index structure means a successful way to speed up the query processing [8]. And by implementing indexes means to improve the performance of the system [9].

To find the appropriate match for the query from a database with millions of records, and reduce into small tasks to compute, MapReduce (MR) is a popular model used for handling large datasets [10]. The MapReduce model is simple, scalable, and fault tolerance; many companies have adopted it for their business analytics applications [10]. MapReduce model can apply to handle a large number of data with many calculations, and the interesting properties of this model are load balancing and better managing of data [11]. MapReduce (MR) is a software framework build for parallel processing for large datasets [12]. MapReduce based approach is one of the big help in Big Data to improve the performance, there are several studies, and prove that have been work in query optimization by using MapReduce. MapReduce is an easy-to-understand model in which becomes the most recommended alternative in designing scalable and distributed algorithms [13].

In the next section, the combine MapReduce and Data Cube Numerosity Reduction method (MRDCNRD) will demonstrate the efficient way to handle data cubes for large datasets rapidly and cost-effectively. The nonclustered index (NCI) role is to speed-up the searching data. Optimizing the existing query to efficiently handle the processes and generation of fund value data within a tolerable time with the low-cost process, no additional resources needed, no delays in terms of operations, and can produce insights to be used for decision making in a real-time manner.

2. METHODOLOGY

2.1 Fund Value Data

The Fund value data used in this paper is conducted from one of the local financial institutions, covering the period from 2005 to 2019 data. The company provides the participant, and member's data, however, the names were not disclosed in this paper to fulfill the data privacy. The Statement of Member's Data is the fact table that comes from different dimensions such as participant, member, fund value, and claim. The fiscal year is a period table contains the return of investment percentage and will use in Statement of Member's Data to sort the data for cube aggregation.

2.2 Concept of the Study

The concept of the study is divided into three major phases, as shown in Figure 1. The first phase is the data collation where the source tables are identified-Fund Data, Participant Information (PI Info), Participant's Member Information (Member Info), Claim, and Return of Investment Percentage (ROIP) that will use to generate a Fact table called a statement of member's data.

The second phase is the optimization query techniques that will perform the following steps; (1) identified the missing index key by generating the estimated query execution cost (2) Implement the nonkey attributes, and (3) Map the input value and search it from sliced and diced data cube, and reduce it by applying the appropriate operation and aggregate the data sets with the help of the nonclustered index for fast data retrieval.
The final phase is the performance evaluation of the query in terms of time and space complexity.

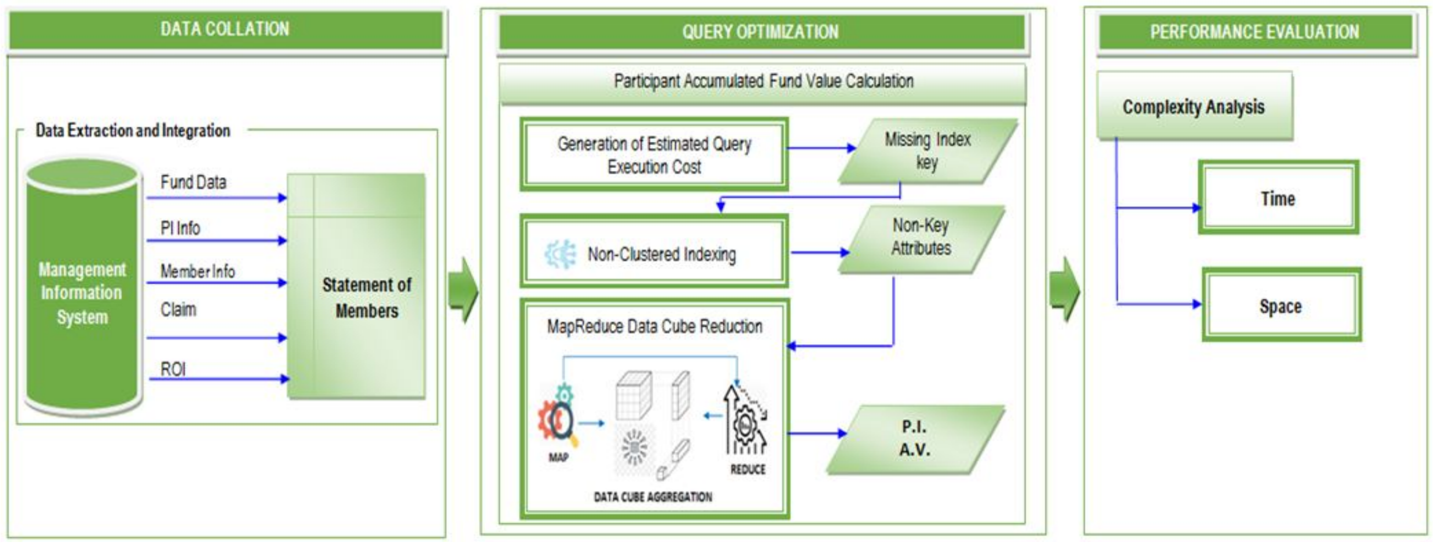


Figure 1: Conceptual Framework

2.3 Procedure

A. Data Collation

The First Phase is to extract and integrate the rows from the data source. It will improve the quantity of the data and choose the appropriate rows to support the mapping phase and eliminate the redundant values; in fact table, there are some of the data which is not necessary for data aggregation. The following are several steps to eliminate duplicate records.

Step 1: Identify the Fiscal Year based on the input period and find the match month *m* and year *y* from the Fiscal Year table.

Algorithm 1: Identify the Fiscal Year

```
Input   : Period
Output  : Fiscal Year From, Month to, Month from , Year from and Year to

1 Declare variables for [Year To], [Year From], [Month To] and [Month From ]
2 Get [Year To] from the input period
3 Get [Month To] from the input period
4 Set [Month From]= $(\rho_{month} \leftarrow month(Period\ Start)\ \pi_{month(Period\ Start)}\ \sigma_{cast([Month\ To]\ as\ varchar)\ +\ '01'\ +\ cast([Year\ to]\ as\ varchar)\ between\ [Period\ start]\ and\ [Period\ End]}(Fiscal\ Year))$
5 Set [Year From]= $(\rho_{year} \leftarrow year(Period\ Start)\ \pi_{year(Period\ Start)}\ \sigma_{cast([Month\ To]\ as\ varchar)\ +\ '01'\ +\ cast([Year\ to]\ as\ varchar)\ BETWEEN\ [Period\ start]\ AND\ [Credit\ End]}(Fiscal\ Year))$
6 Declare variables for Fiscal Year Period End
7 Set [Fiscal Year Period End] = $(\rho_{[period\ end]} \leftarrow [period\ end]\ \pi_{[period\ end]}\ \sigma_{cast([Month\ To]\ as\ varchar)\ +\ '01'\ +\ cast([Year\ to]\ as\ varchar)\ BETWEEN\ [period\ start]\ AND\ [period\ End]}(Fiscal\ Year))$
```

Step 2: To have an effective distribution of data this step will provide the list of the claimed member in the data group, which is not needed for aggregation.

Algorithm 2: Exclusion of the Member's Claim in fact table

```
Input   : Member's Claim Data and period
Output  : List of Claimed Member

1 Let $c$ as Period
2 Let $T$ be the temporary table
3 Select the value from
4 $\pi_{id}\ (\sigma_{(date \geq S\ and\ date \leq e) and\ Status='Approved'}(Claim)\ \cup\ \pi_{id}\ (\sigma_{(left(right(id,6,2)\ ='\ 01'\ )}(Participant)$
5 Insert the List into $T$ ;
```

Step 3: Get the unique identity of all active participants and eliminate the redundant key value in a data group.

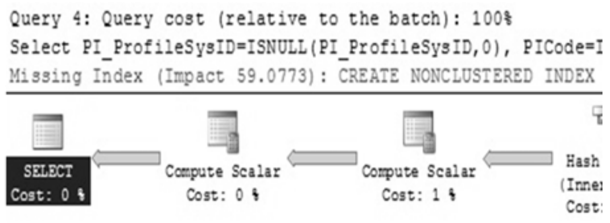
Algorithm 3: Unique identity key for fact table

```
Input   : Parameter for Participant ID
Output  : List of active participant $p$

1 Let $T$ be temporary table for active participant list
2 Select $p = (\pi_{P}^{(Participant)})$
3 Insert $p$ into $T$ cursor table
```

B. Query Optimization

Step 1: Generation of Estimated Query Execution Cost. This section is to perform the execution query cost by using the MS SQL Server Management Studio to identify the missing index key in fact table that affects the performance of data retrieval. In figure 2, it shows, by implementing the nonclustered index, the query will improve by 59.0773%.



```
Query 4: Query cost (relative to the batch): 100%
Select PI_ProfileSysID=ISNULL(PI_ProfileSysID,0), PICode=I
Missing Index (Impact 59.0773): CREATE NONCLUSTERED INDEX
```

Figure 2: Estimated query execution cost

Indexing should be done on large databases where retrieval of data is performed very frequently[14]. This will provide detailed information such as the estimated cost of operation, CPU, and I/O usage that is relative to the rest of the query's operations. The execution plan is used to refined and suggest how to reduce the processing query cost [15]. This tool will suggest what keys need to be part of the nonclustered index. Figure 3 is the SQL syntax to create a nonclustered index.

```
USE [<Database Name>]
GO
CREATE NONCLUSTERED INDEX [<Name of Missing
Index, sysname,>]
ON [schema name].[<table Name>] (<index key>)
INCLUDE (<column name>
GO
```

Figure 3: SQL Syntax to create nonclustered index

Step 2: NonClustered Indexing. The identified missing index, as shown in figure 2, is needed to be first resolved by implementing the recommended index type called nonclustered index for specific keys. The nonclustered index is a balanced tree structure from a root node and includes the intermediate nodes and leaf nodes [9]. The nonclustered index helps to improve aggregate query processing and will be efficient in a single cube. The index is used for storing aggregate data for a specified nonkey attribute, which is not necessarily needed to be part of the primary key attribute, as shown in figure 4, and only implies that it dynamically depends which selected view we focus on and becomes suited in this design.

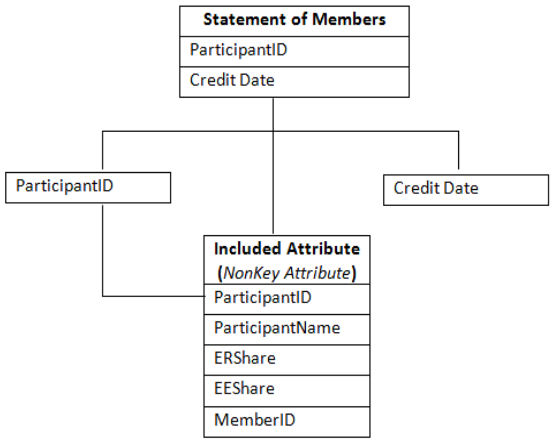


Figure 4: NonClustered index in fact table

Step 3: MapReduce Data Cube Numerosity Reduction. It is possible to implement large scale cube materialization of similar interesting in data cube groups [3],[16]. As illustrated, in figure 5, the fact table can be used to construct a slice and dice cube into smaller called subcube, to efficiency distributing the task and gives different data views and level. Data cube can slice by period and participant and will give different numbers of records as listed in table 2. By selecting a subcube, dice cube can easily now analyze and aggregate the funds by a participant. This particular cube has four attributes – period, participant, and member and fund.

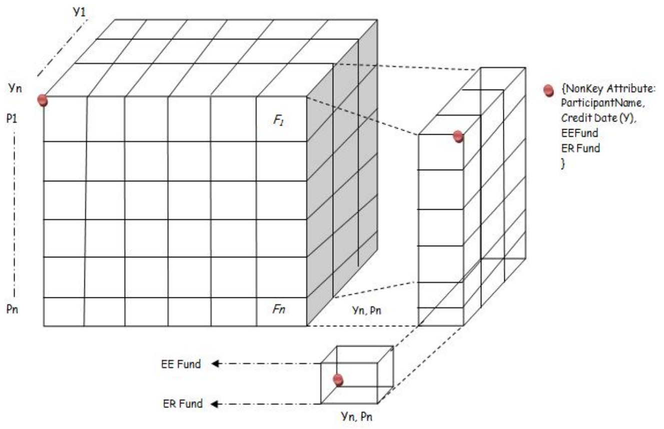


Figure 5: Fact table showing the sliced and diced data cube

The slice and dice cube will be used to relate the different processes to understand, for example, how much the Participant and Member Fund in period 1, like in table 1.

Table 1: N- Dimensional Cube

Period	Participant	Member	Fund
1	P1	M1	1,000,000
1	P1	M2	780,00
1	P1	M3	2,500,00
...	...	...	....
ALL	ALL	ALL	

Constructing data cube needs to generate the power set of the aggregation attributes, the CUBE is a relational operator, and this can be conveniently specified by the SQL GROUP BY [17] as shown below:

SELECT participant, period, sum (member's fund), and sum (participant's fund)
FROM fact table
WHERE period = period
GROUP BY CUBE (participant, period)

A typical data for analysis can involve the different numbers of records like in table 2. Numerous analysis tasks are then concerned as it needs to specify the actual values for a subset of the attributes and aggregating over the other attributes [18].

Table 2: Number of records per period

Period	No. of records
1	566,949
2	528,445
3	422,983
4	526,087
5	2,255,059

In MapReduce data cube they are several algorithms have been developed [19]. In this approach, SQL will be used to manipulate and manage the data cube. Users have to write two functions, the Map and Reduce. MapReduce based approach supports holistic measures as the best option for data analysis [20].

Algorithm 4: MapReduce Data Cube

Input: List of active participant *p*, period *c*, Month *m* and Year *y*
Output: SUM for all Participant based on the period *c* and Participant *p*
Additional Input: Exclusion of Claimed Members

1 WHILE minimum *p* is less than or equal to maximum *p*
2 BEGIN
3 Remove the existing batch group from the Final table
4 *Map for Phase*: Get the minimum (key) *p* and find the pair value *p* in data cube (*stored in the batch group).*

$$\pi \begin{pmatrix} (a_1\ldots an) \\ sum\ (a1), \\ sum\ (a2), \\ count(a3) \end{pmatrix} \sigma(p\geq 0)\gamma \begin{matrix} (a_1\ldots a_n), \\ sum(a_1), \\ sum(a_2), \\ count(a_3) \end{matrix} \begin{pmatrix} c\geq cast(cast(isnull(m,'') \\ as\ varchar(2) + \text{'01'}+ \\ cast(y\ as\ varchar(4)\ as \\ date)\ and\ c\leq c\ and\ p= \\ (p_1\ldots p_n)\ and\ m\ not\ in \\ (m_1\ldots m_n) \end{pmatrix} (f)$$

Slice and Dice Cube Phase in partial materialization based on the period and matched pairs value from map phase
Reduce for Phase: $p_1\ldots.p_n$ in partial materialization based on $p_1\ldots.p_n$, period, month, year and excluding the claimed members
5 Output Phase for Phase $p_1\ldots.p_n$: all $a_1\ldots$an aggregate in batch $b_1\ldots b_n$ with sum and count
6 Set new $p_{minimum}$ $=p_{minimum}$ + 1
7 Repeat until $p_{maximum}$ is reached
8 END

In MapReduce based approach, it consist the following steps:

(1) map phase, the input [key,value] and will locally map in the subcube and match the pair [key,value] in a batch group; and

(2) reduce phase by means of evaluating the measure in the subcube and will be loaded into result table for further exploration [15].

Once the selection and execution of the user query are done, the appropriate cubes in the database are now taking place. The algorithm takes an input where it processes all participants by a batch group. Each participant will get the pair attribute and the summation of the specific attribute.

C. Performance Evaluation

The last phase is where the result is evaluated in terms of time and space complexity

Table 3: Complexity Analysis

Time	**Space**
O(log(N))	O(N)

Where: N is the size of result table

In this section, the approach will evaluate if the execution of the optimized query reaches its optimal time. The space will evaluate in terms of memory consumption from different input period. And time complexity is to get the total minutes of execution of the query. Complexity analysis is used to show the outcomes of the study and evaluate the improvement of the query compared to the previous state.

3. RESULT AND DISCUSSION

In this section, the comparative method is to present the query response time [21], space and evaluate the improvements in query performance implemented in Microsoft SQL on a desktop with 8GB memory, Intel Core i5-7400 CPU @3.00Ghz and 1 TB HDD under Windows Server 2008 Standard Edition. The query performance is analyzed using the varying numbers of records based on the period presented in table 2.

The effect of the data size on memory space and response time is also observed. The experiment verifies the different results: (1) How responsive is the query if the approach were only used the nonclustered index against having no nonclustered index; (2) and by using two combine approaches – NonClustered Index (NCI) and the MapReduce Data Cube Numerosity Reduction Method (MRDCNR) against the normal data aggregation.

Table 4: Data aggregation result using nonclustered index and without having nonclustered index

	Response Time (seconds)						**Available Memory (MB)**					
Description	Period					Improve average	Period					Improve average
	1	**2**	**3**	**4**	**5**		**1**	**2**	**3**	**4**	**5**	
Non NCI	4769	3867	3871	4998	4650	**23.33%**	124	245	216	10	10	**11.32%**
NCI	2242	1106	1221	1326	847	**76.67%**	1698	931	985	796	329	**88.68%**
	Total number of time execution				28897		Total number of space memory				5344	
	*Time Complexity = O(log(N))						*Space Complexity = O(N)					

1.) NonClustered Index versus Without NonClustered Index

Table 4 shows the time execution and availability of the memory by using NCI in fact table. The response time measure where N can be seen as the total number of execution time per period. For period 1, the total number of run time is N*=2242 seconds*, and the time complexity will be O(log(N)) because it considers how many N loops run inside the query. And as for memory space, N is the total memory space available during the execution of the query per period, regardless of the data size.

The space complexity will be O(N) because it will count how many memory available, and in the experiment for period 1

N=1698 megabytes available. Figure 6 presented the graphical representation of the query response time by using a nonclustered index, which improves by an average of 76.67%, compared to without having a nonclustered index, which required approximately 77% more time to search the data based on the 23% average response time as shown in table 4.

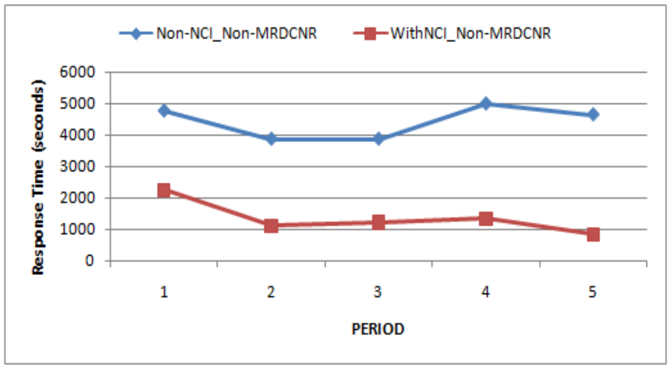


Figure 6: Nonclustered index and without having nonclustered index in terms of response time in seconds

And as for data that was exponentially increased based on the period, it became more critical at some point and requires more time to execute in data cube and was significantly slower than having a nonclustered index. The fact table generally contains a big volume of data, and it costs in time and storage space [22].

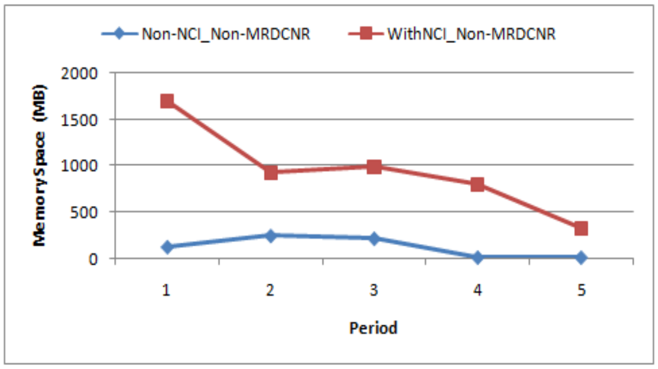


Figure 7: Nonclustered index and without having nonclustered index in terms of memory space available in MB size

Figure 7 shows that the availability of the memory is decreased based on 11.32% average result, as shown in table 4. And without using nonclustered index (NCI) resulting in high memory usage and consumption. Implementing a nonclustered index helps to speed-up the data retrieval in the data cube and increase the availability of memory by 89%.

Table 5: Combined MapReduce Data Cube Numerosity Reduction and NonClustered Index (MDRCNR_NCI) versus Standard Group-by (Non_MDRCNR_NCI)

Description	Response Time (seconds)						Available Memory (MB)					
	Period					Improve average	Period					Improve average
	1	2	3	4	5		1	2	3	4	5	
Non_MDRCNR_NCI	4768.8	3867.6	3870.6	4998.6	4651	**5.92%**	1037	745	713	271	263	**8.75%**
MDRCNR_NCI	301.8	283.2	274.2	282.6	251.4	**94.08%**	6399	6390	6332	6289	6167	**91.25%**
	Total number of time execution				23549		Total number of space memory				34606	
	*Time Complexity = O(log(*N*))						*Space Complexity = O(*N*)					

2.) MapReduce Data Cube Numerosity Reduction Method and NonClustered index

The combined approach shows the significant result in data cube as it was radically improved even at the higher number of records presented in table 2. The time measure where *N* is to quantify the amount of time in seconds taken by the query per period; and for memory space *N* is to quantify the amount of space available in megabytes (MB) per period which was not taken by the query.

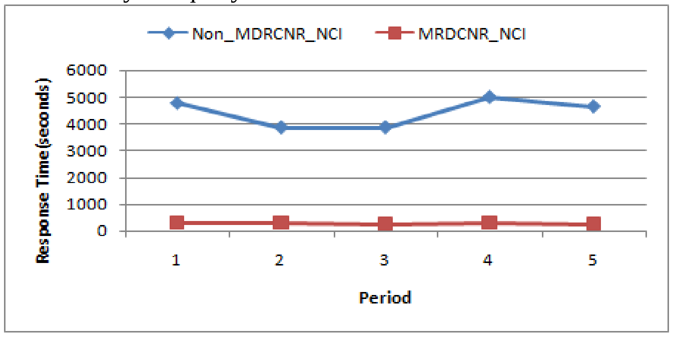


Figure 8: Comparison result between MRDCNR_NCI and NonMRDCNR_NCI in terms of response time in seconds

Table 5 presented the result of the two combined approaches. MDRCNR_NCI has improved the response time by an average of 94%. The query response time for period 1 is 301.8 seconds that is equivalent to 5.030 minutes compare to 4768.8 seconds equivalent to 79.48 minutes to get the data aggregation result. The graphical result can be seen in figure 8. The queries with thousands or millions of records mostly consume the memory to search the data based on 8.75% average availability of memory for NonMDRCNR_NCI.

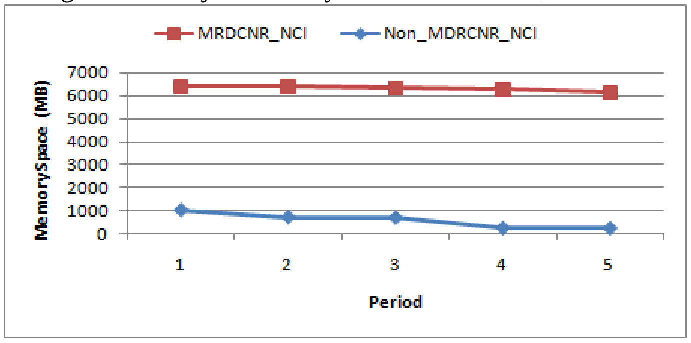


Figure 9: Comparison result between MRDCNR_NCI and NonMRDCNR_NCI in terms of memory space availability in MB size

In the experiment, the memory consumption is only used 9%, resulting in higher memory space available by an average of 91%. Figure 9 illustrates how much the memory space is available during the execution of the query. The MRDCNR and NCI overcome the higher usage in computing data cube and not even reached the critical threshold which means, the improvement in response time remains stable, and the average execution of the query can now be completed within 4 to 5 minutes considering the volume of the data.

3. CONCLUSION

The proposed approach is effective as it does dramatically improve the performance of the query. The intensive experiment validates by applying NonClustered Indexing; speed up the performance of data retrieval. As we cover the index, the query performance is now improved because all the data needed is within the index itself [23].

MapReduce based approach in data cube computation is to load and balance the data by distributing it into a small task and efficiently aggregates the data. And based on the experiment, the combine approaches are considered strong and effective solutions to resolve the prolonged execution of the query and able to complete the task in a reasonable amount of time, reduce memory usage, and the memory space becomes balanced and not over in the stated condition.

ACKNOWLEDGEMENT

The researchers appreciate the PERAA Fund for the support to make this a good one and useful to other institutions.

REFERENCES

[1] J. Han, M. Kamber, and J. Pei, Data Mining Concepts And Techniques, 3rd edition, Elsevier, Waltham, MA 02451, USA, pp. 135-164,2000.

[2] A. Ivanova and B. Rachev, "Multidimensional models - Constructing Data Cube," International Conference on Computer Systems and Technologies, pp. 1–7, 2004. https://doi.org/10.1145/1050330.1050444

[3] D. Puspa, S. Ghazali, R. Latip, M. Hussin, M. Helmy, and A. Wahab, "A review data cube analysis method in Big data environment," ARPN Journal of Engineering and Applied Sciences, volume 10, issue 19, pp. 8525–8532, 2015.

[4] A. Savinov, "From Group-by to Accumulation : Data Aggregation Revisited," Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS), pp. 370–379, 2017.

[5] T. Mamaliga, "Realizing a Process Cube Allowing for the Comparison of Event Data Master Thesis," Master Thesis, Technische Universiteit Eindhoven, University of Technoogy., August, 2013.

[6] M. Patel, B. Parmar, Y. Patel, and H. Joshi, "Efficient Dynamic Index Structure for Natural Number Intensive Application," International Journal of Computer Applications (0975 – 8887), volume 109, issue. 4, pp. 13–20, 2015. https://doi.org/10.5120/19176-0650

[7] Q. Optimization, S. M. Mahajan, and P. Vaishali, "Analysis of Execution Plans in Query Optimization," International Journal of Scientific & Engineering Research, volume. 3, issue 2, pp. 1–4, 2012.

[8] G. Moerkotte, "Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing," Proceedings of the 24th VLDB Conference New York, USA, 1998.

[9] C. Cioloca and M. Georgescu, "Increasing Database Performance using Indexes," Database Systems Journal, volume II, issue 2, pp. 13–22, 2011.

[10] H. G. Kim, "SQL-to-MapReduce Translation for Efficient OLAP Query Processing with MapReduce," International Journal of Database Theory and Application, volume 10, issue 6, pp. 61–70, 2017. https://doi.org/10.14257/ijdta.2017.10.6.05

[11] A. D. Arasteh, D. Mohammadpur, and M. Meghdadi, "MapReduce Based Implementation of Aggregate Functions on Cassandra," International Journal of Electronics Communication and Computer Technology (IJECCT), volume 4, issue 3, May 2014, 2014.

[12] S. Lee and J. Kim, "Performance Evaluation of MRDataCube for Data Cube Computation Algorithm Using MapReduce," 2016 International Conference on Big Data and Smart Computing (BigComp), pp. 325–328, 2016.

[13] B. Wang, H. Gui, M. Roantree, and M. F. O. Connor, "Data Cube Computational Model with Hadoop MapReduce," Proceedings of the 10th International Conference on Web Information Systems and Technologies, Barcelona, Spain, 3-5 Apr 2014.

[14] M. K. Gupta, "Comparative study of indexing techniques in dbms," https://www.researchgate.net/publication/333844844 , no. June, 2019.

[15] S. Wu, F. Li, S. Mehrotra, and C. Ooi, "Query Optimization for Massively Parallel Data Processing," Proceedings of the 2nd ACM Symposium on Cloud Computing, October 2011. https://doi.org/10.1145/2038916.2038928

[16] P. P. Patil, P. Kotian, A. Gaonkar, S. Wani, and P. Gaikwad, "Map-Reduce for Cube Computation," International Journal of Scientific Research Engineering & Technology (IJSRET), ISSN 2278 – 0882, volume 4, issue. 4, pp. 299–303, 2015.

[17] J. Gray, A. Bosworth, F. Pellow, and H. Pirahesh, "Data Cube : A Relational Aggregation Operator Generalizing Group-By , Cross-Tab , and Sub-Totals," Data Mining Knowledge Discovery, volume 1, Issue 1, 1997.

[18] M. Sundararajan and Q. Yan, "A Simple and Efficient MapReduce Algorithm for Data Cube Materialization," https://arxiv.org/abs/1709.10072v1, 2017.

[19] T. Milo and E. Altshuler, "An Efficient MapReduce Cube Algorithm for Varied Data Distributions," Proceedings of the 2016 International Conference on Management of Data, pp. 1151–1165, 2016.

[20] N. R. Bhosale and H. K. Chavan, "Map Reduce Approach for computing interesting," International Journal Advance Computing Engineering Network, volume 3, issue 12, pp. 106–110, 2015.

[21] H. Zhao, S. Yang, Z. Chen, S. Jin, H. Yin, and L. Li, "MapReduce model-based optimization of range queries," 2012 9th International Conference Fuzzy Sysem Knowledge Discovery, pp. 2487–2492, 2012. https://doi.org/10.1109/FSKD.2012.6234050

[22] V. Phan-luong, "A Simple Data Cube Representation for Efficient Computing and Updating," International Journal on Advances in Intelligent Systems, volume 9, issue 3, pp. 255–264, 2016.

[23] S. Mukherjee, "Indexes in Microsoft SQL Server Indexes in Microsoft SQL Server," PhD student University Cumberlands Chicago, United States, pp. 1–16.