ISSN (Online): 2277-5420
www.IJCSN.org
Impact Factor: 1.5

Incremental Aggregation on MOLAP Cube Based on n-Dimensional Extendible Karnaugh Arrays

[1] Jakaria Rabbi; [2] Md. Abdul Awal; [3] K M Azharul Hasan

[1, 2, 3] Department of Computer Science and Engineering,
Khulna University of Engineering & Technology (KUET), Khulna 9203, Bangladesh.

Abstract - Data is increasing so rapidly that new data warehousing approaches are required to process and analyze data. Aggregation of data incrementally is needed to fast access of data and compute aggregation functions. Multidimensional arrays are generally used for this purpose. But some disadvantages such as address space requirement is large and processing time is comparatively slow in case of aggregation. For this purpose we use Extendible Karnaugh Array (EKA). EKA is an efficient scheme which has better performance than other data structures that we have tested in our research. In this research work we use EKA as basic structure for implementing incremental aggregation of data and evaluate its performance over other approaches. We use Multidimensional Online Analytical Processing (MOLAP) which stores data in optimized multi-dimensional array storage, rather than in a relational database. We create 4 and 6 dimensional MOLAP data cube using Traditional Multidimensional Array (TMA) and EKA scheme and compare incremental aggregation with Relational Online Analytical Processing (ROLAP). The effective outcome of EKA structure for incremental aggregation on 4 and 6 dimensional MOLAP structure is shown by some experimental results and efficiency is proved for n higher dimensions.

Keywords—*Multidimensional Array, Extendible Array, Karnaugh Map, Dynamic Extension, Data Cube, ROLAP, OLAP, MOLAP, EKA, TMA.*

1. Introduction

Aggregation of data is widely used in data warehousing applications for rapid calculation and easy access. Methods of aggregation has been studied and commercialized with great success [1]. Data warehouse and aggregates can be recomputed with the addition of new data, but it is quite expensive to recompute the whole process. Since the degree of modification to base data cube is normally small, incremental aggregation method can be applied for dynamically increasing database datasets [2]. However, recently, the emergence of the data stream processing presents new challenges to compute incremental aggregates over ever-changing data streams [1]. Various data structures are used to compute incremental aggregation efficiently with less time and less storage. Here we implement Extendible Karnaugh Array (EKA) scheme to aggregate data incrementally. Performance of incremental aggregation using EKA is quite enhanced in comparison to traditional multidimensional array and relational database. In EKA address space overflow occurs later in comparison to traditional extendible array. So, better performance is achieved.

Multidimensional Online Analytical Processing (MOLAP) is an alternative to the Relational Online Analytical Processing (ROLAP) technology. Both ROLAP and MOLAP are designed to allow analysis of data through the use of a multidimensional data model, but MOLAP differs significantly in the way that it requires the pre-computation and storage of information in the data cube. Most MOLAP solutions store these data in optimized multidimensional array storage, rather than in a relational database (i.e. in ROLAP). There are many existing array systems to represent multidimensional data for MOLAP and incremental aggregation such as Traditional Multidimensional Array (TMA) [3][4], Extended Karnaugh Map Representation (EKMR) [5][6] and Karnaugh Representation of Extendible Array (KEA) [7]. TMA is one of the storage structures for MOLAP and incremental aggregation of data but one serious drawback is that they are not dynamically extendible. To insert a new column value in the TMA the total reorganization of the data in array is necessary. The idea of extendible array solves the problem of extendibility. So, an efficient extendible array is necessary for efficient incremental aggregation.

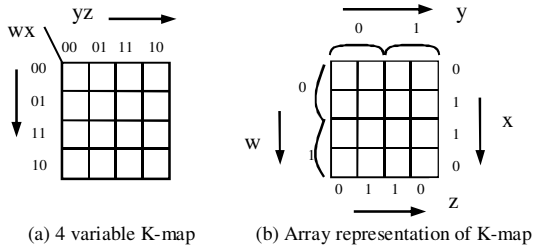
Extendible Karnaugh Array (EKA) proposed in [8][9] is an efficient scheme which has better performance than other data structures. We evaluated the performance of this data structure when implemented for MOLAP. 4 and 6 dimensional MOLAP data cube is used here. The remainder of this paper is organized as follows. In section 2, we discuss about EKA briefly. Then in section 3, we

focus on incremental aggregation using this scheme. Experimental result is discussed in section 4. Works that are related to this work are mentioned in section 5. The paper concludes with section 6, which presents a summary of our work and application.

2. Extendible Karnaugh Array

2.1 The Four Dimensional EKA Scheme

The idea of EKA is based on Karnaugh Map (K-map) which is used for minimizing Boolean expressions. Details of EKA structure is proposed in [8][9]. In this section we have little insight of EKA. Figure 1 (a) shows a 4 variable K-map to represent possible 2^4 combinations of a Boolean function. The variables (w, x) represent the row and the variables (y, z) represent the column that indicates the possible combinations in a two dimensional array for the four Boolean variables. We can have array representation of the K-map for 4 variable Boolean functions which is shown in Figure 1(b).



(a) 4 variable K-map (b) Array representation of K-map

Figure 1. Boolean function using K-map.

DEFINITION 1: (Adjacent Dimension). The dimensions (or index variables) that are placed together in the Boolean function representation of K-map are termed as adjacent dimensions (written as adj(i) = j). The dimensions (w, x) are the adjacent dimensions in Fig. 1(a) and (b) i.e. adj(w) = x or adj(x) = w.

Figure 2 shows the dynamic extension of the array of Figure 1(b). EKA is a system of array which is the combination of sub arrays. To maintain dynamic extension and the subarrays, three types of auxiliary tables are needed. They are *history table*, *coefficient table*, and *address table*. These tables are maintained for each dimension. Construction history of the subarrays is stored in the *history table*. The *address table* stores the first address of a segment and is used to compute the correct position of an element. Any element in the *n* dimensional array is determined by an addressing function as follows

$$f(x_n, x_{n-1}, x_{n-2}, \ldots, x_2, x_1) = l_1 l_2 \ldots l_{n-1} x_n + l_1 l_2 \ldots l_{n-2} x_{n-1} + \ldots + l_1 x_2 + x_1$$

The coefficients of the addressing function namely $\langle l_1 l_2 \ldots l_{n-1}, l_1 l_2 \ldots l_{n-2}, \ldots, l_1 \rangle$ are referred as coefficient vector which are stored in *coefficient table*. The subarrays are divided into equal size segments (See Figure 2) that can be stored contiguously on the disk. Consider a 4-dimensional array of size A[l_1, l_2, l_3, l_4] where l_i (i = 1, 2, 3, 4) is the length of each dimension d_i that varies from 0 to $l_i - 1$. The dimension (d_1, d_3) and (d_2, d_4) are grouped as adjacent dimensions respectively. The length of the extended subarray which is allocated dynamically for the extension along dimension d_i is determined by $\prod_{j=1}^{4} d_j \ (j \neq i)$

The number of segments in any subarray (belongs to dimension d_i) is determined by the length of the adjacent dimension of d_i. The number of segments determines the number of entries in the address table and is equal to from the length of adjacent dimension. After extending along any dimension d_i, the length of the corresponding dimension is incremented by 1. For each extension the auxiliary tables namely *history table*, *address table* and *coefficient tables* are maintained. Figure 4 shows the extension realization of a 4 dimensional EKA, where H_{d1}, C_{d1}

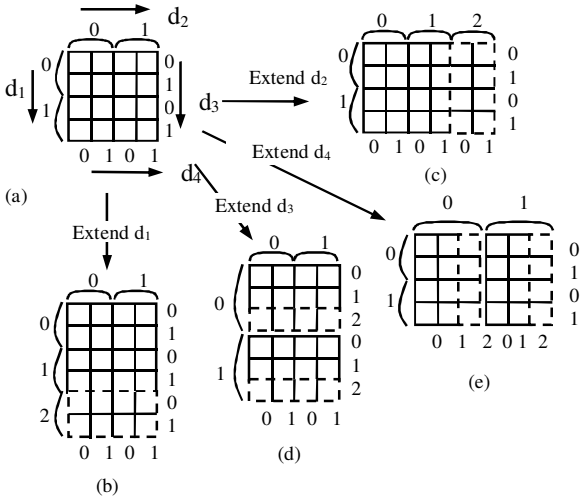


Figure 2. Logical extension of 4-dimensional EKA.

and A_{d1} are *history table*, *coefficient table* and *address tables* of dimension 1. In this figure extension is occurred by extending d_2, then d_3,d_4 and d_1 The EKA scheme can be generalized to *n* dimensions using a set of EKA(4)s. We write EKA(n) to denote an n dimensional EKA. Figure 3 shows an EKA(6) represented by a set of EKA(4)s in two levels containing 5th and 6th dimensions each of lengths 3 and 2 respectively. Each higher dimensions (d_5 and d_6) are represented as one dimensional array of pointers that points to the next lower dimension and each cell of d_5 points to each of the EKA(4). So each EKA(4) can be accessed simply by using the subscripts of higher dimensions. In the case of EKA(n), the similar hierarchical structure will be needed to locate the appropriate EKA(4). Hence the EKA(n) is a set of EKA(4)s and a set of pointer arrays. The segments are always 2 dimensional for an

EKA (n). Hence in this model the coefficient vector becomes single dimensional such as $\langle l_1 \rangle$ only. The EKA can be extended along any dimension dynamically during runtime only by the cost of these auxiliary tables.

Let the value stored in the subscript (x_1, x_2, x_3, x_4) is to be retrieved. The maximum history value among the subscripts h_{max} = max ($H_{d1}[x_1]$, $H_{d2}[x_2]$, $H_{d3}[x_3]$, $H_{d4}[x_4]$) and its corresponding dimension (say d_1) that corresponds the h_{max} is determined. h_{max} is the subarray that contains our desired element. The first address and offset from the first address is found out using the auxiliary tables. The adjacent dimension $adj(d_1)$ (say d_3) and its subscript x_3 is found. The first address is found from $H_{d1}[x_1].A_{d1}[x_3]$. The *offset* from the first address is computed using the addressing function; coefficient vectors are stored in C_{d1}. Then adding the *offset* with the first address, the desired array cell (x_1, x_2, x_3, x_4) is found.

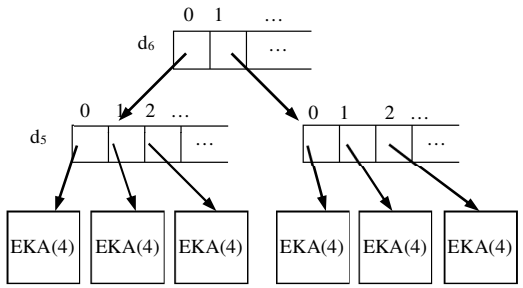


Figure 3. The realization of EKA(n).

2.2 Illustrative Example of Four Dimensional EKA

We illustrate an example in this section. Here, from the initial set up EKA(4) is extended through d_2, d_3, d_4, d_1 dimensions sequentially. The values of the subscripts (1, 0, 1, 1) are determined as follows (See Fig. 4). Here h_{max} = max($H_{d1}[1] = 4$, $H_{d2}[0] = 0$, $H_{d3}[1] = 2$, $H_{d4}[1] = 3$) =4, and dimension corresponding to h_{max} is d_1 whose subscript is x_1=1 and $adj(d_1) = d_3$ and $x_3 = 1$. So the first address is in $H_{d1}[1]$. $A_{d1}[1] = 12$, and offset is calculated using the coefficient vector stored in coefficient table $C_{d1}[1] = 2$. Here offset = $C_{d1}[1] * x_4 + x_2 = 2*1+0 = 2$. Finally adding the first address with the offset the desired location 12 + 2 = 14 is found (encircled in Figure 4)

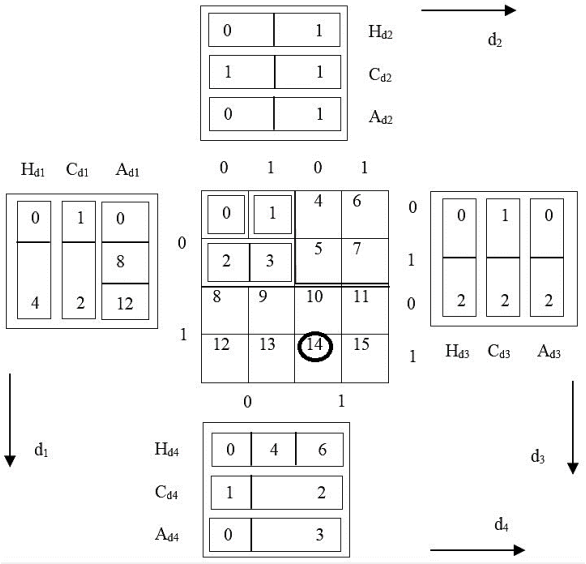


Figure 4. Illustrative example of 4-dimensional EKA.

3. Incremental Aggregation of data using EKA

3.1 Traditional ways for Incremental Aggregation

a) TMA: Traditional multidimensional arrays are used for incremental aggregation. Their main disadvantage is that, when we add length of a dimension then whole array must be reorganized. For adding extra dimension, a new array of new dimension is declared and all data from the array of previous dimension is to be copied to the new array.

b) ROLAP: Relational OLAP is maintained inrelational data table stored in a database management systems. Data cube is created according to aggregation function. When new data is to be inserted the whole data with previous and new data need to be recalculated. The reqiured storage and time to aggregate data is also high compared to the array storage system.

Here in Figure 5 (a) we see a relational base table and Figure 5 (b) there is ROLAP data cube for various combination for SUM aggregate. If we add new data in the Figure 5 (a), then data cube is recomputed. In this way, we can create 6 dimensional EKA and ROLAP base table and data cube.

Shop	Product	Time	City	Price
S0	P0	T0	C0	100
S0	P1	T0	C0	200
S0	P0	T1	C1	100
S1	P1	T1	C1	200

(a) Base Table

Shop	Product	Time	City	Price
S0	P0	T0	C0	100
S0	P1	T0	C0	200
S0	P0	T1	C1	100
S1	P1	T1	C1	200
S0	P0	T0	All	100
.....	.....	.....	.....	.....
S0	P0	All	All	200
.....	.....	.....	.....	.....
All	All	T0	C0	300
.....	.....	.....	.....	.....
S0	All	All	All	400
S1	All	All	All	200
All	All	All	All	600

(b) Data Cube

Figure 5. ROLAP Base table and Data Cube.

3.2 Construction of MOLAP Structure using EKA

Using EKA structure illustrated in section 2 we made data cube. We use data file to write the value of the price and used mapping for each index. Four tuple such as <0,0,0,0> is used to indicate the position of the value in file. If S1, P1, T1 and C1 is to be selected then tuple is created as <1,1,1,1> and this key value is pointed to the position of address of a file. Data cube is computed for all combination and stored incrementally. If new data is inserted than sum is calculated from previous sum and new data. For n higher dimension we use n tuples. Here we use 6 tuples for EKA(6) data cube.

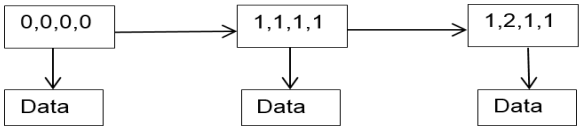


Figure 6. Data mapping from primary memory to secondary storage.

3.3 Incremental Aggregation calculation on MOLAP using EKA Data Structure

In Figure 7 we propose our structure graphically for incremental aggregation. We use EKA structure as the base of our aggregate

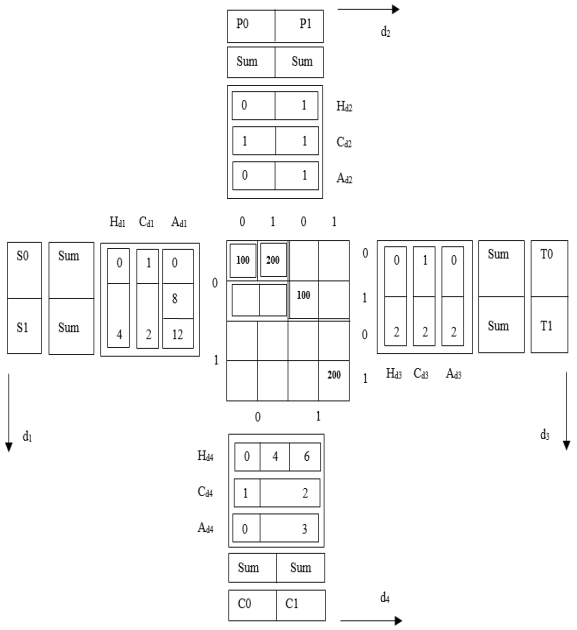


Figure 7. Incremental Aggregation calculation on MOLAP using EKA Data Structure.

structure. We calculate SUM() aggregation function in four dimensional EKA scheme along with additional data structure which hold sum of value such as product price in the extendible array and also we keep track of four properties of value along with four dimensions. From this structure we can derive much more complex system which has numerous prices of various combinations of shop, product, time and city. We have added country and continent by using 6 dimensional EKA. In these way we can get our results for n (n = 4, 5, 6, 7…n) dimensional EKA. With the inclusion of new price value of different combination of these four variables the sum is incrementally updated using the previous sum. So, we need not calculate whole aggregation from the beginning.

4. Experimental Results

In our method we used EKA scheme for both 4 and 6 dimensions. EKA and TMA is stored in secondary storage. An auxiliary table of EKA and mapping function to point position of data is stored in main memory as their size is small. The test results for aggregation function SUM() is analyzed in this section.

We run all our tests on a computer with 4GB RAM having 4 Kbyte of disk page size and processor of 2.5 GHz (4 CPU's).

4.1 Incremental Aggregation Computing Time Comparison

We computed time for incremental aggregation and storage needed in MOLAP using EKA, TMA and ROLAP when length (L) of all dimensions is 20 and 40 for both 4 and 6 dimensional MOLAP.

a) Computation for 4 dimension:

When data density (ρ) is less MOLAP using TMA(4) has bad performance compared to MOLAP using EKA(4) and ROLAP(4). When data density increases ROLAP(4) needs increased time compared to other two methods and performance of MOLAP using EKA(4) is increased. At ρ = 0.7 we observed that ROLAP(4) needs 13 times more time than MOLAP using EKA(4) when L= 20 and 17 times more when L = 40.

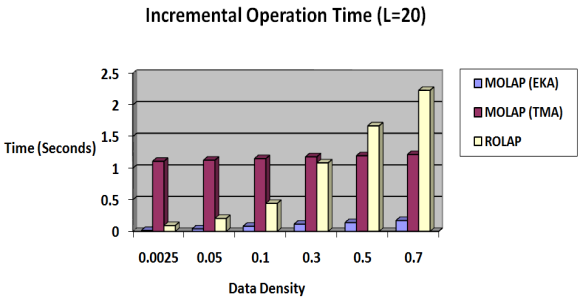


Figure 8. Incremental operation time comparison between MOLAP using EKA(4), TMA(4) and ROLAP(4) (L=20)

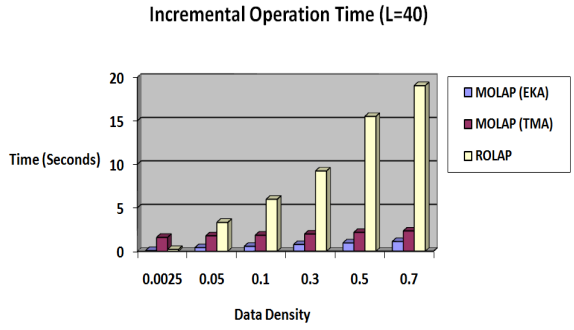


Figure 9. Incremental operation time comparison between MOLAP using EKA(4), TMA(4) and ROLAP(4) (L=40)

b) Computation for 6 dimension:

When data density (ρ) is less MOLAP using TMA(6) has bad performance compared to MOLAP using EKA(6) and ROLAP(6). When data density increases ROLAP(6) needs increased time compared to other two methods and performance of MOLAP using EKA(6) is increased. At ρ = 0.7 we observed that ROLAP(6) needs 11 times more time than MOLAP using EKA(6) when L= 20 and 35 times more when L = 40.

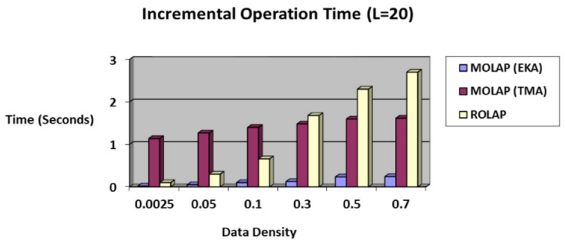


Figure 10. Incremental operation time comparison between MOLAP using EKA(6), TMA(6) and ROLAP(6) (L=20)

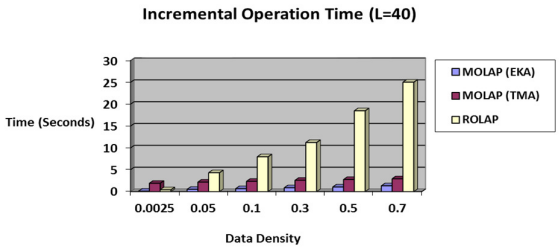


Figure 11. Incremental operation time comparison between MOLAP using EKA(6), TMA(6) and ROLAP(6) (L=40)

4.2 Storage Comparison for Incremental Aggregation

a) Computation for 4 dimension:

In storage comparison we observed that MOLAP using EKA(4) and TMA(4) takes same space. When data density is less ROLAP(4) takes less space. But with the increase of data density MOLAP using EKA(4) and TMA(4) take less space than ROLAP(4). At ρ = 0.7 we observed that ROLAP(4) needs 4 times more space than MOLAP using EKA(4) for both configuration of L = 20 and L= 40.

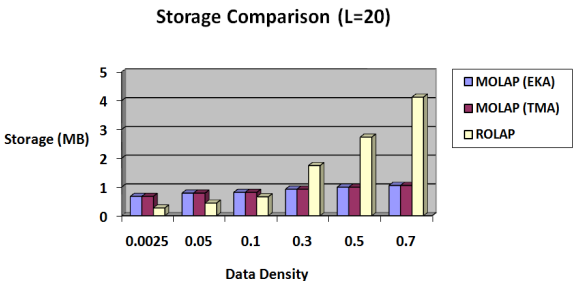


Figure 12. Storage comparison between MOLAP using EKA(4), TMA(4) and ROLAP (L=20)

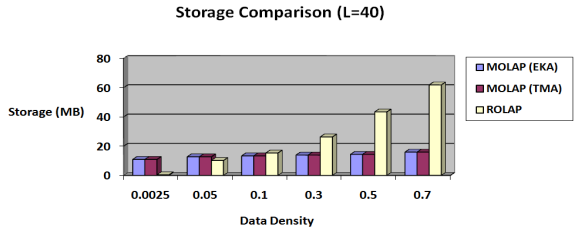


Figure 13. Storage comparison between MOLAP using EKA(4), TMA(4) and ROLAP (L=40)

b) Computation for 6 dimension:

In storage comparison we observed that MOLAP using EKA(6) and TMA(6) takes same space. When data density is less ROLAP(6) takes less space. But with the increase of data density MOLAP using EKA(6) and TMA(6) take less space than ROLAP(6). At $\rho = 0.7$ we observed that ROLAP(6) needs 4 times more space than MOLAP using EKA(6) for both configuration of L = 20 and L= 40.

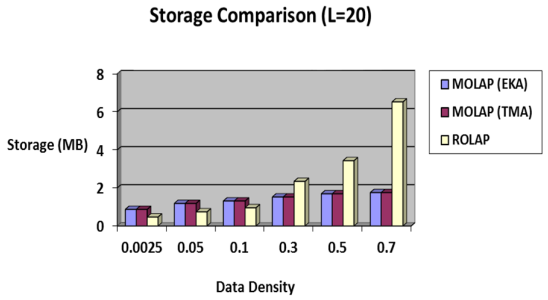


Figure 14. Storage comparison between MOLAP using EKA(6), TMA(6) and ROLAP (L=20)

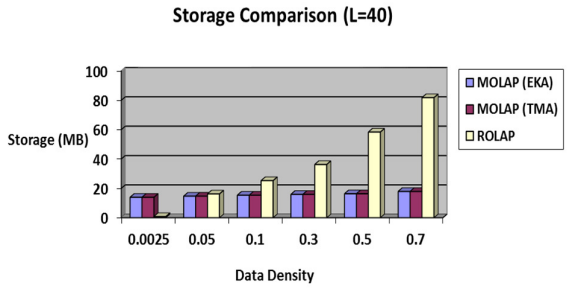


Figure 15. Storage comparison between MOLAP using EKA(6), TMA(6) and ROLAP (L=40)

5. Related Works

There are several works has been done on incremental aggregation on multidimensional array scheme. In [10] aggregation is calculated with relational table and multidimensional array. But extendible arrays are not used here to evaluate performance. Processing in the array based algorithm is done chunk by chunk basis on MOLAP systems in [10][11]. They also uses conventional multidimensional array. Multidimensional extendible array is used in [12] and a data structure is proposed to retrieve, compress and reorganize data for fast access and less memory. So, none of these works evaluate the performance of EKA scheme on MOLAP cube and incremental aggregation. We have shown that EKA structure is quite better to calculate incremental aggregates in MOLAP systems

6. Conclusion

We propose and evaluate the performance of an efficient extendible MOLAP system to aggregate data incrementally with low cost in comparison with other two approaches. This approach saves both time and storage for aggregation. A large amount of space and time is reduced in this approach as data reorganization is not necessary. This technique can be used in data warehousing applications for efficient aggregation and data analysis.

7. Future Scope

This technique can be used in data warehousing applications for efficient aggregation and data analysis.

References

[1] Jin, Chun, and Jaime Carbonell, "Incremental aggregation on multiple continuous queries," Foundations of Intelligent Systems, Springer Berlin Heidelberg, pp. 167-177, 2006.

[2] Nesamoney, Diaz, et al, "Method for incremental aggregation of dynamically increasing database data sets," U.S. Patent No. 5,794,246. 11 Aug. 1998.

[3] K.E. Seamons and M. Winslett (1994),"Physical schemas for large multidimensional arrays in scientific computing applications," Proceedings of SSDBM, pp. 218-227.

[4] S. Sarawagi and M. Stonebraker (1994), " Efficient organization of large multidimensional arrays," Proceedings of ICDE, pp. 328-336.

[5] Y.L. Chun, C.C. Yeh, and S.L. Jen (2002), "Efficient Representation Scheme for Multidimensional Array Operations, "IEEE Transactions on Computers, 51(3), pp. 327-345.

[6] Y.L. Chun, C.C. Yeh, and S.L. Jen (2003), "Efficient Data Parallel Algorithms for Multidimensional Array Operations Based on the EKMR Scheme for Distributed Memory Multicomputer," IEEE Transactions on Parallel and Distributed Systems, 14(7), pp. 625-639.

[7] KMA Hasan, K Islam, M Islam, T Tsuji, "An extendible data structure for handling large multidimensional data sets", Proceedings of 12th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, pp. 669-674, 2009.

[8] Ahsan, Sk Md Masudul, and KM Azharul Hasan, "An implementation scheme for multidimensional extendable array operations and its evaluation," Informatics Engineering and Information Science. Springer Berlin Heidelberg, 2011. 136-150.

[9] M Ahsan, S Md, KM Hasan, "An Efficient Encoding Scheme to Handle the Address Space Overflow for Large Multidimensional Arrays," Journal of Computers 8 (5), pp. 1136-1144, 2013.

[10] Y. Zhao, P. M. Deshpande and J. F. Naughton, "An array-based algorithm for simultaneous multidimensional aggregates", In Proceedings of the ACM SIGMOD Conference on Management of Data, pp.159-170, 1997.

[11] S. Sarawagi and M. Stonebraker, "Efficient organization of large multidimensional arrays", Proc. of ICDE, pp. 328-336, 1994.

[12] Hasan, KM Azharul, Tatsuo Tsuji, and Ken Higuchi. "An efficient implementation for MOLAP basic data structure and its evaluation," Advances in Databases: Concepts, Systems and Applications. Springer Berlin Heidelberg, 2007. 288-299.

Jakaria Rabbi

Department of Computer Science and Engineering, Khulna University of Engineering & Technology (KUET), Khulna 9203, Bangladesh.

Abdul Awal

Department of Computer Science and Engineering, Khulna University of Engineering & Technology (KUET), Khulna 9203, Bangladesh.

K M Azharul Hasan

Department of Computer Science and Engineering, Khulna University of Engineering & Technology (KUET), Khulna 9203, Bangladesh.