Compression Schemes for High Dimensional Data based on Extendible Multidimensional Arrays

By

Md. Rakibul Islam

Roll No: 1007503

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science & Engineering

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

March, 2015

Declaration

This is to certify that the thesis work entitled "Compression Schemes for High Dimensional Data based on Extendible Multidimensional Arrays" has been carried out by Md. Rakibul Islam in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.

Signature of Supervisor

Signature of Candidate

Approval

This is to certify that the thesis work submitted by Md. Rakibul Islam entitled "Compression Schemes for High Dimensional Data based on Extendible Multidimensional Arrays" has been approved by the board of examiners for the partial fulfillment of the requirements for the degree of Master of Science in Computer Science & Engineering in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh in March, 2015.

BOARD OF EXAMINERS

1. 15.03.15
Dr. K. M. Azharul Hasan — Chairman (Supervisor)
Professor, Dept. of CSE
Khulna University of Engineering & Technology, Khulna

2. 15.03.15
Head of the Department — Member
Department of Computer Science and Engineering
Khulna University of Engineering & Technology, Khulna

3. Sadi 15-03-15
Dr. Muhammad Sheikh Sadi — Member
Professor, Dept. of CSE
Khulna University of Engineering & Technology, Khulna

4. Aminul 15/3/15
Dr. Md. Aminul Haque — Member
Professor, Dept. of CSE
Khulna University of Engineering & Technology, Khulna

5. 15.3.15
Dr. Md. Anisur Rahman — Member (External)
Professor, CSE Discipline
Khulna University, Khulna

Acknowledgment

First of all, obeisance to the almighty, omnipresent Allah for giving me the strength and capability for writing the thesis. This thesis would not have been achievable without the instructions, unconstrained support, guidance and the help of numerous individuals. First and foremost, my utmost gratitude goes to my supervisor, Dr. K. M. Azharul Hasan, Professor, Department of Computer Science and Engineering, for his continuous supervision, constructive criticism, valuable advice, instructions and encouragement at all stages of this thesis. I would like to show my heartiest gratitude to Professor Dr. Rokibul Alam, Head of the Department of Computer Science and Engineering, and Dr. Muhammad Sheikh Sadi, Professor, Department of Computer Science and Engineering for their encouragement and numerous varieties of supports. I also like to remember the inspiration, supports and encouragement of my family.

Author

Abstract

Traditional Multidimensional Array (TMA) is an important data structure for handling large scale multidimensional dataset, but they are not extendible during run time. Another problem for representing the real life data by multidimensional arrays is that it creates high degree of sparsity. Due to this sparsity problem and increasing size of the data structures, it becomes necessity to develop a suitable scheme to compress the multidimensional array in an efficient way so that it takes comparatively low memory storage. To minimize both of these sparsity and reorganization problem novel schemes are proposed to compress high dimensional data based on dynamically extendible array. In this research work we propose compression schemes based on Extendible multidimensional array. The proposed compression schemes are Extendible array based Compressed Row Storage (*EaCRS*) scheme, Linearized Extendible array based Compressed Row Storage (*LEaCRS*) scheme and Extendible array based Chunk Offset Compression Scheme (*EaChOff*). The main idea of both the *EaCRS* and *LEaCRS* scheme is to compress the subarrays independently found from the existing extendible array. *LEaCRS* scheme differs from *EaCRS* scheme only in the way that the *LEaCRS* scheme needs to linearize each subarray first and then compresses the subarray independently. *EaChOff* scheme linearizes each subarray independently and breaks a large multi dimensional extendible array into chunks for compressing. In this scheme, a maximum size of each chunk is considered and chunks are formed by one or more subarrays. We evaluated our proposed schemes by comparing compression ratio, data retrieval time and extension cost with *CRS* on TMA and *Chunk-Offset* Compression on TMA. Both analytical analysis and experimental tests were conducted. The analytical analysis and experimental results show that the proposed schemes have better range of usability and compression ratio for practical applications than traditional schemes. Furthermore, we found that the retrieval time of the proposed compression schemes are independent of different dimensions. The increment operation will be efficient in the proposed compression schemes than the existing traditional compression schemes because it increments without reorganizing the previous data.

Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

1.1 Introduction

The process of reducing the size of data in order to save space or transmission time is termed as data compression. Data compression is widely used in data management to save storage space and network bandwidth [1]. The main benefit of data compression is that of increasing the capacity of the storage medium since data compression reduces the storage requirement for the databases. Compressed information can be transferred from one place to another in a higher effective transfer rate. This is because compressed data are encoded using a smaller number of bytes and hence results less time for information transfer. Since data compression reduces the loading of I/O channels, it becomes feasible to process more I/O requests per second and hence achieve higher effective channel utilization. Most importantly, however, is the application of data compression in reducing the cost of data communication in distributed networks. In some applications, data compression can reduce the average search cost and thus leads to improvement in system performance. For example, in some index structures it is possible through compression to pack more keys into each index block. When the database is searched for a given key value, the key is first compressed and the search is performed against the compressed keys in the index blocks [2] which results fewer blocks retrieval. Compression is of two types: data compression and database compression [3]. In data compression, in order to use compressed data, it is necessary to restore the information to its uncompressed format. Data compression techniques (e.g. Arithmetic Coding, Lempel-ZIV, Huffman Coding etc. [4,5,6]) achieve large compression rates that are very useful for archiving. The compressed data sets are not directly queriable without prior decompression. But it is desirable to develop compression techniques so that the data can be accessed in their compressed form and operations can be performed directly on the compressed data. Such techniques are called database compression techniques and usually provide two mapping [7]. One is forward mapping. It computes the location in the compressed data set given a position in the original data set. The other one is backward mapping. It computes the position in the original data set given a location in the compressed data set. A compression method is mapping-complete if it provides both forward mapping and backward mapping. In this research work we are going to propose database compression schemes for handling multidimensional data sets having the facility of dynamic extendibility during runtime. The idea is based on multidimensional extendible arrays.

Arrays are among the best-understood and most widely used data structures. Few classes of data structures are as well understood or as widely used as arrays. Large multidimensional arrays are quite often used as the basic data structure in scientific, statistical and engineering applications for modeling and analyzing scientific phenomena [8,9] such as climate modeling [10], molecular dynamics [11], finite-element methods [12] etc. Different statistical computations can be performed professionally on multidimensional arrays due to its fast random accessing capability [6,13,14]. But this capability depends on the fact that the size of each dimension should be fixed so that a simple addressing function can be used to access an arbitrary element of the array. However, in real Multidimensional Online Analytical Processing (MOLAP) [15,16] applications data size grows incrementally. When a new data value is added, size extension along the corresponding dimension is necessary. Except the extension along last dimension this drawback implies reorganization of the entire array. This extendibility problem of conventional array system can be solved using extendible array model. An extendible array can be extended in any dimension without any repositioning of previously stored data [17,18]. Such advantage makes it possible for an extendible array to be applied into wide application area where required array size cannot be predicted before and / or can vary dynamically during operating time of the system.

1.2 Problem Statement

Traditional Multidimensional Array (TMA) [19,20,21] is a good storage for storing multidimensional 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 array is necessary. The idea of extendible array solves the problem of extendibility. Extendible arrays, in fact, are combination of subarrays. If the array is n dimensional then the subarrays are $n\text{-}1$ dimensional.

Multidimensional arrays are good to store dense data, but most datasets are sparse which wastes huge memory because a large number of array cells are empty and thus are very hard to use in actual implementation [22]. In particular, the sparsity problem increases when the number of dimensions increases. This is because the number of all possible combinations of dimension values exponentially increases, whereas the number of actual data values would not increase at such a rate. For Example in an international trade data set there are several dimensions such as importing country, exporting country, date-time, items, measure amount of items etc. But generally a small number of items are exported from any given country to other countries. Many of the compression schemes based on TMA such as Compressed Row/Column Storage (CRS/CCS) [14,23] or Chunk-offset Compression [22,24] already exist. CRS is commonly used due to its simplicity and purity with a weak dependence relationship between array elements in a sparse array. But this scheme is based on the TMA. Chunk-Offset compression scheme is also well studied in the literature for multidimensional data analysis. But once again it is based on TMA. One main problem of TMA based compressions schemes are that it is static in nature. This is because, if there is any extension in each dimension in TMA based compression schemes, we need to restore compressed data to its original format and perform the desired extension for the new added data sets. Then the reorganized TMA is compressed by using some compression schemes. So, efficient compression schemes are required to store such sparse data for multidimensional data sets [13,25,26] without any reorganization and relocation. In this thesis, we are going to propose and evaluate a new and efficient compression schemes based on extendible multidimensional array (EMA) [27,28,29] to manage the problem of extendibility without reorganization of data and apply a suitable compression scheme on the EMA to have good compression ratio.

1.3 Objectives

Various scientific applications use multidimensional array as a basic data structure to represent high dimensional data. This is because multidimensional array has an inherent facility to compute aggregation operation [30]. Extendibility is an important requirement of those applications since data grows over time. Hence, an array model or realization scheme which can be extended over time is strong requirement of current era. Again because of sparsity most datasets are very hard to use in actual implementation.

Therefore main objective of this research topic can be summarized as follows

- To develop compression schemes for High Dimensional Data based on EMA, which will impose less space and the maximum range of usable data density, will be advanced for practical applications.
- To analyze the increment operation (which is known as extension operation) along with the basic operations on proposed compression schemes, with respect to the existing traditional compression schemes.
- To devise both forward mapping and backward mapping techniques for the proposed scheme i.e. perform efficient and random searching in compressed array for a given logical position of the original array; and also provide an efficient mapping from arbitrary positions in the compressed data back to the corresponding logical position in the original array.
- To analyze the performance and usability of the proposed compression schemes on sparse array.

1.4 Scope of the Thesis

This thesis deals with array system and compression schemes and proposes new and efficient database compression scheme for high dimensional data based on EMA. Other important scopes under this thesis are:

- Compresses the EMA by applying compression scheme on each subarray of the extendible array independently.
- Compares the new schemes with the existing schemes in terms of space requirement/compression ratio (η), range of usability, extension cost and retrieval cost.
- Store the elements in the secondary storage to set the actual η.
- Range key query are evaluated for the retrieval cost analysis.

1.5 Thesis Organization

- **Chapter I** describes the problems of TMA as well as of existing compression schemes. Objectives and scopes of the thesis are also outlined in this chapter.

- **Chapter II** presents an overview of array systems and different types of compression schemes.
- **Chapter III** provides the detailed discussion about the compression schemes for high dimensional data based on extendible array. Forward mapping and backward mapping techniques of the proposed schemes are explained with examples in this chapter. This chapter also describes theoretical analysis along with the cost models for existing schemes as well as proposed schemes.
- **Chapter IV** shows the experimental setup, experimental results and detail analysis of the result. Hence we validate the cost models of the proposed schemes.
- **Chapter V** outlines the concluding remarks and direction of future research work.

CHAPTER II

Literature Review

2.1 Introduction

Large multidimensional arrays are widely used as the basic data structure in scientific, statistical and engineering applications. Multidimensional databases such as MOLAP databases [31,32] frequently make use of multidimensional array for handling large scale multidimensional data. In MOLAP applications, compression is important because database performance of MOLAP database strongly depends on the amount of available memory [13,22]. The solid demand of those applications leads novel researches on organization or implementation schemes for multidimensional arrays on secondary storage and different compression schemes for this multidimensional array. Multidimensional arrays are becoming the most popular data structure because of an inherent facility of random accessing. But capability demands the length, and number of dimension to be fixed – which leads problem of dynamic extension. There are many data structures already exist to represent multidimensional data. Some of them are static in nature and some are dynamic – i.e. resizable without reorganizing the already allocated data. Some of the well-known and prominent data structures are discussed in this section.

2.2 The Multidimensional Array Systems

An Array $A[d_1,d_2,\dots,d_n]$ is an association between n-tuples of integer indices $\langle l_1, l_2, \dots, l_n\rangle$ and the elements of a set of E such that, to each n-tuples given by the ranges $0 \le l_1 < d_1$, $0 \le l_2 < d_2,\dots$, $0 \le l_n < d_n$ there corresponds an element of E. The domain from which the elements are chosen is immaterial and we make the assumption that only one memory location need to be assigned to each n-tuples. Each array may be visualized as the lattice points in a rectangular region of n-space. The set of continuous memory locations into which the array maps is denoted by $A[0{:}D]$ where $D = (\prod_{i=1}^{n} d_i) - 1$. Let $A(d_1, d_2, \dots, d_{n-1}, d_n)$ be an n dimensional array with length of each dimension $d_1, d_2, \dots, d_n$.

2.2.1 Traditional Multidimensional Array (TMA)

Traditional Multidimensional Array (TMA) [16,22,33] is a representation scheme for multidimensional data which represent n dimensional data by n dimensional array. The TMA represent n dimensional data by an array cell in an n dimensional array. The key to the structure of arrays resides in the familiar coordinate system, which pictures an n-dimensional array as being imbedded in the positive orthant of n-dimensional space, with array positions lay on the lattice points.

The fast random accessing capability that is characteristic to multidimensional arrays enables various statistical computations including aggregation to be performed efficiently on stored fact data. This capability is owing to that the size of each dimension of a multidimensional array is fixed so a simple addressing function can be used to address an arbitrary element of the array. An element $(i_n, i_{n-1}, \ldots, i_1)$ in an n dimensional TMA of size $[d_n, d_{n-1}, \ldots, d_1]$ is allocated on memory using an addressing function like equation 2.1 (see section 2.3.1). Although Storage by linearization allows extension without any movement of existing elements only in one of the dimensions, TMA suffers from the reorganization problem; when a new data value is added only in third dimension of a TMA(3), we can readily extend the 3D TMA in third dimension but array size extension along other dimensions necessitates reorganization of the entire array elements.

2.2.2 Extendible Multidimensional Array (EMA)

The idea of extendible multidimensional array is described in [18,32,34]. An n dimensional extendible array A can be extended in any dimension only by the cost of three kinds of auxiliary tables namely *history table* H_i, *address table* L_i, and *coefficient table* C_i for each extendible dimension i $(i=1,\ldots,n)$. See Figure 2.1. History tables and address tables are one dimensional array. History tables memorize extension history. An n dimensional extendible array A is the combination of $n-1$ dimensional subarrays. If the size of A is $[d_1, d_2, \ldots, d_{n-1}, d_n]$ and the extended dimension is i, for an extension of A along dimension i, contiguous memory area that forms an $n-1$ dimensional subarray S of size $[d_1, d_2, \ldots, d_{i-1}, d_{i+1}, \ldots, d_{n-1}, d_n]$ is dynamically allocated and added to A in dimension i. Then the history value counter h is incremented by one and the value is memorized in the history table H_i, also the first address of S is held on the address table L_i. Note that S is a usual fixed size array, and the actual data is stored in these subarrays.

As is well known, an element $<i_1,i_2,...,i_n>$ in an n dimensional conventional fixed size array of size $[d_1,d_2,..., d_n]$ is allocated on memory using an addressing function like equation 2.1 (see section 2.3.1) and *coefficient vector* (defined in section 2.3.1) $<d_2d_3...d_n, d_3d_4...d_n, ..., d_n>$ is held in a coefficient table. For example, let A be a four dimensional extendible array whose current sizes are $[d_1, d_2, d_3, d_4]$. If A is extended by one along the dimension two, a three dimensional fixed array S of sizes $[d_1, d_3, d_4]$ is allocated. The elements of S's are arranged according to the well known *column wise* or *row wise* order. The addressing function to determine the address of the element $<i_1, i_2, i_3>$ is as: $d_1d_3i_1 + d_3i_2 + i_3$

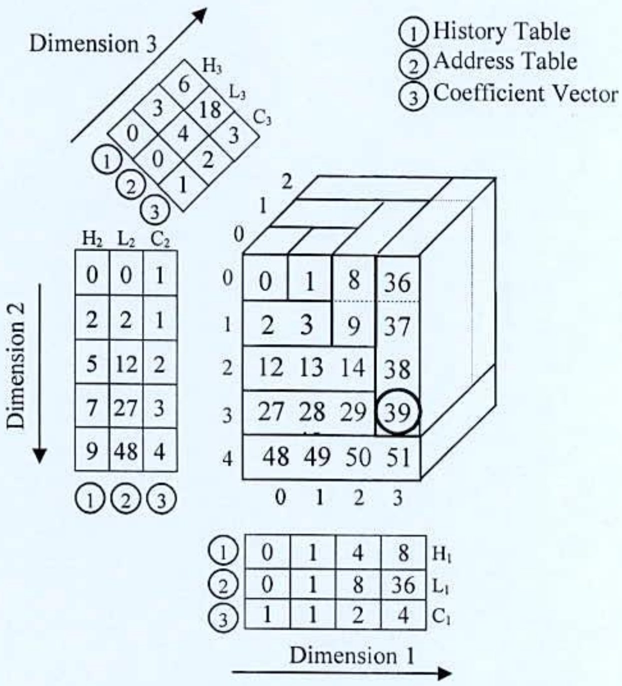


Figure 2.1: A Three dimensional Extendible Multidimensional Array.

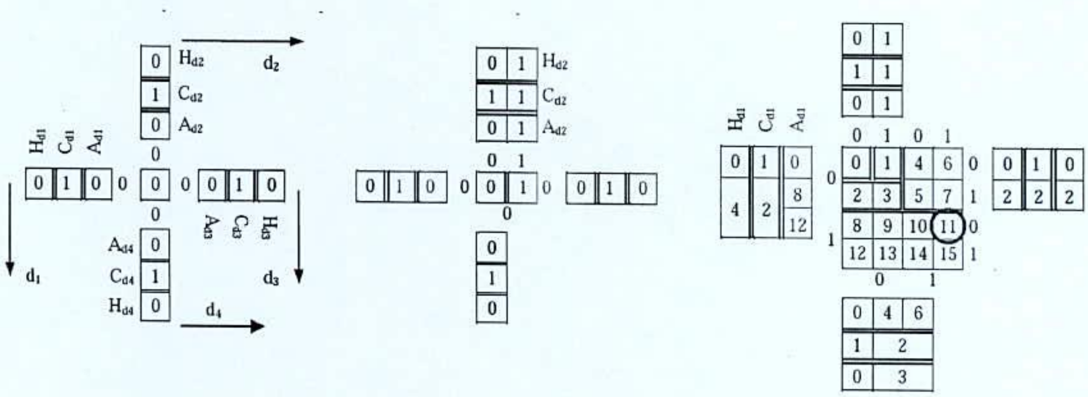
Here $< d_1d_3, d_3 >$ is called a *coefficient vector*. At every extension of A, the corresponding subarray's coefficient vector is computed and memorized in *coefficient table* of the extended dimension. In general, if A is an n dimensional extendible array where n is greater than two, an n-2 dimensional coefficient vectors are required for each extendible dimension.

Using these three kinds of auxiliary tables, the address of an array element can be computed as follows. Consider the element <3,3,0> in Figure 2.4. Compare $H_1[3] = 8$, $H_2[3] = 7$ and $H_3[0] = 0$. Since $H_1[3] > H_2[3]$, $H_1[3] > H_3[0]$, it can be proved that the element <3,3,0> is involved in the extended subarray S having history value 8 and beginning address of the corresponding subarray is 36 which is stored in $L_1[3]$. From the coefficient vector of $C_1[3] = < 4 >$, the offset of element <3,3,0> from the first address of S is computed by $4 \times 0 + 3 = 3$, the address of the element is determined as 39 (See Figure 2.1).

From the above element accessing procedure it can be seen that, the cost to compare n history values is necessary to know the maximum history value therefore to know the extended dimension of the element containing subarray. After knowing the maximum, the offset computation is performed using the addressing function of the corresponding n-1 dimensional fixed size subarray. But, the number of multiplication and addition operations to be performed is less than that of an n dimensional fixed size array [35]. The superiority of the extendible arrays in element accessing speed and memory utilization is shown in [18].

2.2.3 Extendible Karnaugh Array (EKA)

The idea of EKA [35,36,37] is based on Karnaugh Map (K-map) [38,49]. A Karnaugh representation of Extendible Array (EKA) has a history counter and three auxiliary tables, history table, address table and coefficient table. The history table stores the extension history and the address table stores the first address of the extended subarray. The EKA can be extended along any dimension dynamically during runtime only by the cost of these three auxiliary tables. Figure 2.2 shows the details of the EKA scheme for a 4-dimensional array of size $A[s_1,s_2, s_3, s_4]$. It also displays how the different auxiliary tables are maintained during the extension along a particular dimension. Figure 2.2(a) shows the initial setup with history counter 0 stored in history tables, address tables point to the first address of the physical array, and coefficients tables entry is 1, since length of each dimension is 1. During extension along d_1 or d_3 the segment size is $s_2 \times s_4$, so s_2 is chosen as coefficient vector. Similarly, s_3 is used as coefficient vector for extension along d_3 or d_4. Figure 2.2(b) shows the extension along d_2 dimension, the incremented history value 1 is stored in history table of dimension 2. Since s_3 is 1, C_2 stores this value and address table points to the first address which is 1. Figure 2.2(c) shows the extension of d_1 dimension considering that Figure 2.2(b) is already extended once in d_3, and d_4 dimension. As it is already extended in d_3, and d_4 dimension, the history value reaches to 3, now for extending in d_1 the value becomes 4 which is stored in H_1. Coefficient table entry is 2 because of the s_2 is 2. If the length of dimension and number of dimension of a multidimensional array is large then the address space for the TMA and EMA overflows quickly. EKA has the property of dynamic extension during run time and significantly delays the occurrence of address space overflow.



(a) Initial setup (b) Extension along d_2 dimension (c) Extension along d_1 dimension

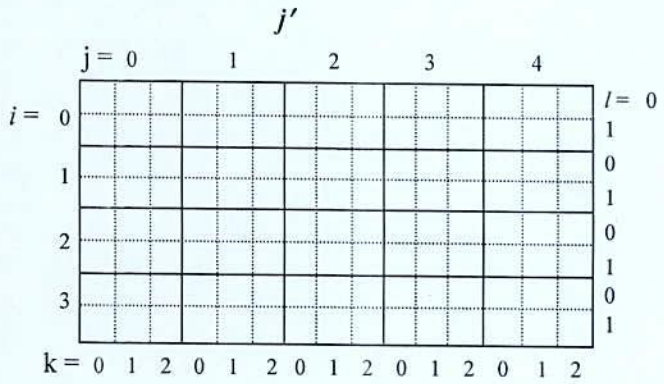
Figure 2.2: Extension realization of EKA (4).

2.2.4 Extended Karnaugh Map Representation (EKMR)

A basic array representation scheme named Extended Karnaugh Map Representation (EKMR) is proposed in [9,40,41]. In this scheme, an n-dimensional array is represented by a set of 2 dimensional arrays. The idea of the EKMR scheme is based on the Karnaugh map (K-map). For n= 1 and 2, the TMA and EKMR Schemes are same. Let A[l][k][i][j] denote a TMA for n=4 with a size of 2×3×4×5. The corresponding EKMR system i.e, EKMR(4) of array A[2][3][4][5] is shown in Figure 2.3(b). Consider a 4 input K-map and its corresponding EKMR(4) in Figure 2.3. The analogy between the EKMR(3) and the 3-input Karnaugh map is that the index variables i, j, k and l correspond to the variables W, X, Y, and Z, respectively. The EKMR(4) is represented by a two-dimensional array with the size of (2 × 4)×(3 × 5). In the EKMR(4), index variable i' is used to indicate the row direction and the index variable j' is used to indicate the column direction. The index i' is a combination of the index variables l and i, whereas the index j' is a combination of the index variables j and k. Placement of elements along the direction indexed by k and l makes the fundamental difference between TMA(4) and EKMR(4).

WX/YZ	00	01	11	10
00	0	1	1	0
01	1	0	0	1
10	1	0	0	1
11	1	1	0	0

(a)



(b)

Figure 2.3: An Example of EKMR(4).

The EKMR(n) can be obtained in the similar way. Based on the EKMR(4), the EKMR(n) for n dimensional array is represented by $d_n \times d_{n-1} \times \ldots \times d_{n-5}$ EKMR(4) and a one-dimensional array X that links all the EKMR(4) where d_i ($5 \le i \le n$) is the length of the corresponding dimension.

2.3 Compression schemes for multidimensional arrays

Multidimensional array are the basic data structure used in many applications such as MOLAP. But in many cases, they are found to be sparse in nature – i.e. many of the array cells contain null values and consume unnecessary space. Some common compression methods are reviewed here.

2.3.1 Offset Compression for TMA

The *n*-dimensional TMA can be mapped into a single *linearized array* by an array linearization function. The *array linearization function* for the multidimensional array, A is

$$F(p_1, p_2, \ldots, p_n) = d_1 d_2 \ldots d_{n-1} p_n + d_1 d_2 d_3 \ldots d_{n-2} p_{n-1} + \ldots \ldots + d_1 p_2 + p_1 \quad \ldots\ldots\ldots (2.1)$$

The logical position (i.e. offset value) is calculated for the records using the above forward mapping function F and stored on a data structure along with the measure value (if exists). The coefficients of the addressing function namely ($d_1 d_2 \ldots d_{n-1}, d_1 d_2 \ldots d_{n-2}, \ldots, d_1$) is referred to as coefficient vector and stored during the construction time. Hence the addressing function can be computed very fast at the element access time. The *reverse array linearization function* of the multidimensional array of $A(d_1, d_2, \ldots, d_{n-1}, d_n)$ for backward mapping is defined as follows:

$$R - F(Y) = (q_1, q_2, \ldots, q_n) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.2)$$

Where $q_n = Y \bmod d_n$

$$q_i = [\ldots[Y / d_n]\ldots] / d_{i+1}] \bmod d_i \qquad for\ 2 \leq i \leq n-1$$

$$q_1 = [[\ldots[[Y / d_n] / d_{n-1}]\ldots] / d_3] / d_2]$$

The backward mapping algorithm *R-F* is used to determine the coordinates of the corresponding multidimensional array.

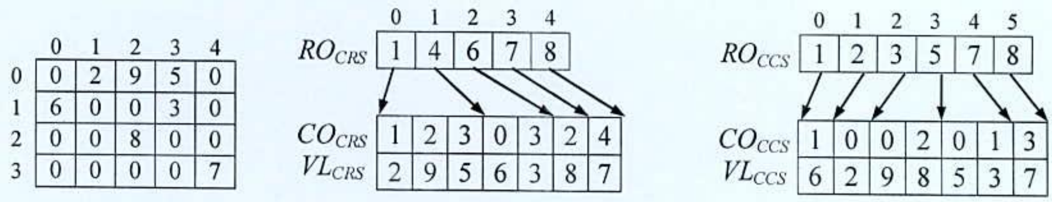
2.3.2 Chunk-offset compression for TMA

In Chunk-offset compression scheme [22,24] the large multidimensional arrays are broken into chunks for storage and processing. Consider an n-dimensional array A, whose dimensionality is $d_1 \times d_2 \times \ldots \times d_n$. The chunks can be formed by breaking each d_i into several ranges. Within A, two positions are in the same chunk if and only if, in every dimension, they fall within the same range. In memory or disk, values within a chunk are stored consecutively. Elements in a chunk are arranged according to the pre-specified order of dimensions.

In this compression scheme, the pairs of (*OffsetInChunk, dataValue*) is physically stored in secondary storage only for nonempty elements in a chunk. This set of pairs is sorted in the order of the offset values. Note that the chunks which have no nonempty elements are not physically allocated in the secondary storage. The offset inside the chunk (*OffsetInChunk*) can be computed using the multidimensional array linearization function described in section 2.3.1. The reverse array linearization function (see equation 2.2) is used for backward mapping to get the original coordinates of the array.

2.3.3 CRS/ CCS scheme for Multidimensional Arrays

The CRS/CCS schemes [14,23,42] compress all the nonzero elements along the rows/columns of the multidimensional sparse array by using one one-dimensional floating point array VL and two one-dimensional integer arrays RO and CO. The base of these arrays is 0. Array VL stores the values of nonzero array elements. Array RO stores information of nonzero array elements of each row (columns for CCS). If the number of rows is k for the array then RO contains k+1 elements. RO[0] contains 1, RO[1] contains the summation of the number non zero elements in row 0 of the array and R[0]. In general, RO[i] contains the number of nonzero elements in (i-1)th row [(j-1)th column for CCS] of the array plus the contents of RO[i-1]. The number of non zero array elements in the ith row (jth column for CCS) can be obtained by subtracting the value of RO[i] from RO[i+1]. Array CO stores the column (rows for CCS) indices of nonzero array elements of each row (columns for CCS). Figure 2.4 shows an example of the CRS and CCS schemes for a two dimensional array.



(a) A sparse array (b) The CRS Scheme (c) The CCS Scheme

Figure 2.4: The CRS/CCS schemes for a two-dimensional sparse TMA.

Figure 2.4(a) shows a 4×5 two-dimensional sparse array. Figure 2.4(b) and Figure 2.4(c) show the corresponding CRS and CCS schemes, respectively. In Figure 2.4(b), the number of nonzero elements of row 1 can be found by $RO_{CRS}[2]\text{-}RO_{CRS}[1] = 2$. The column indices of the nonzero array elements of row 1 are stored in $CO_{CRS}[RO_{CRS}[1]\text{-}1]$ and $CO_{CRS}[RO_{CRS}[1]]$ i.e $CO_{CRS}[3]$ and $CO_{CRS}[4]$, since there are 2 nonzero array elements exist in row 1. Finally the values of the nonzero array elements of row 1 can be found in $VL_{CRS}[3]$, and $VL_{CRS}[4]$. For n-dimensional sparse array based on TMA, (n-1) numbers one dimensional integer arrays *CO* are needed.

2.3.4 EKA Based Compression (SCEKA)

A compression technique is proposed based on the EKA in [35,36,37] namely Segment based Compression scheme for Extended Karnaugh Array (SCEKA). The main idea of the scheme is to compress each of the segments of the EKA using the position information only. To compress the EKA, the SCEKA stores only the position information of the each segment of the array i.e. the construction history, the segment number and the offset inside the array. The data stored in the SCEKA scheme can be accessed in compressed form and at the same time it can grow and shrink in length or number of dimensions at run time. SCEKA stores the tuple ⟨*history value*, *segment number*, *offset*⟩ for array cell mapping and the data is stored as well. The history value is unique and can uniquely determine the subarray. The segment number inside the subarray is also unique and can also be determined uniquely. The offset value inside the segment is also unique and can be determined by the addressing function. Hence the tuple ⟨*history value*, *segment number*, *offset*⟩ can uniquely map an array cell of the EKA.

2.3.5 EKMR Based Compression (ECRS or ECCS)

The scheme is similar to CRS/ CCS scheme for Multidimensional Arrays [14,23,42] but the structure used is EKMR. The ECRS (or ECCS) scheme compresses all the nonzero array elements along rows (columns for ECCS). Array V stores the values of nonzero array elements. Array R stores information of nonzero array elements of each row. R[i] contains the number of nonzero elements in (i-1)th row of the array plus the contents of RO[i-1] and the contents of R[0] is 1. The number of non zero array elements in the ith row can be obtained by subtracting the value of R[i] from R[i+1]. Array CK stores the column (rows for ECCS) indices of nonzero array elements of each row (columns for ECCS).

Some other important compression schemes that can be applied to higher dimensional data are summerized as follows:

The header compression method [43,44] is used to suppress sequences of missing data codes, called *constants*, in linearized arrays by counts. This method makes use of a *header* that is a vector of counts. The odd-positioned counts are for the unsuppressed sequences, and the even positioned counts are for suppressed sequences. Each count contains the cumulative number of values of one type at the point at which a series of that type switches to a series of the other. The counts reflect accumulation from the beginning of the linearized array to the switch points. In addition to the header file, the output of the compression method consists of a file of compressed data items, called the *physical file*. The original linearized array, which is not stored, is called the *logical file*.

In the following example, L represents the uncompressed form of a database, where 0's are the constant to be suppressed and the V's are the unsuppressed values. H represents the header database/file which contains the number of data or constants where odd position represents the data and even position represents constants.

The BAP compression [43,45] method consists of three parts: Bit Vector(BV), Address Vector(AV), Physical Vector(PV) and therefore called BAP compression method.

Let DB={$x_1,x_2,...,x_n$) be a logical database and c be the constants. The physical vector PV is the vector of non-constants in DB, that is, PV=($y_1,y_2,...,y_n$) where y_i are in DB and $y_i \neq c$.

The y_i are arranged according to their logical order in DB. No compression algorithm is applied on PV because it stores only non-constants values. The Bit Vector BV indicates the locations of constants and non-constants in the database. The bit vector is BV=($b_1,b_2,\ldots,b_n$) where $b_i=1$ if $x_i \neq c$ and $b_i=0$ if $x_i=c$ for $1\leq i \leq N$. where BV consists of N bits. The Address Vector AV is typically small and is used as an index for searching the database. It is stored in main memory rather than secondary storage. In addition to efficient compression fast forward and backward mapping between logical and physical databases is also important. To do this, BV is divided into subvectors of D bits each. The subvectors are compressed independently. This division of BV into subvectors makes the Address Vector AV sufficiently small to store it in main memory. BV can be compressed by run-length encoding method (also discussed in this chapter). The division of BV into subvectors imposes a division of the database DB into $d=\lceil N/D \rceil$ sections, each consisting of D elements. The address vector is defined as: AV=($a_1,a_2,a_3,\ldots a_d$); Where $a_1=0$ and for $i\geq 2$, ai is the relative position in PV of the last non-constant element in the (i-1)th section of DB if such a non-constant exists, otherwise we set $a_i=a_i-1$.

A bitmap compression [43,45] scheme consists of a bitmap and a physical database which stores the non-constant values of a linearized array. The bitmap is employed to indicate the presence or absence of non-constant data. The access time for both forward and backward mapping for the bitmap scheme is O(N), where N is the number of bits in the bitmap, or equivalently the number of elements in the database.

The history offset compression [17,46] scheme is based on extendible array. In this technique, an element is specified using the pair of history value and offset value of the extendible array. Since a history value is unique in extendible array and has one to one correspondence with the corresponding subarray, the subarray including the specified element of an extendible array can be referred to uniquely by its corresponding history value h. Moreover, the offset value (i.e., logical location) of the element in the subarray can be computed by using the addressing function and this is also unique in the subarray. Therefore, each element of an n-dimensional extendible array can be referenced by specifying the pair (history value, offset value). Like Chunk-offset compression, the extended sparse subarray elements are stored in memory in sorted fashion.

2.4 Discussion

All the array systems described in this chapter have both merits and limitations. Since TMA and EKMR have pre-specified length and dimension, they are good for random accessing. But they suffer in case of dynamic extension; when a new data value is added, array size extension along the corresponding dimension is necessary and this implies reorganization of the entire array elements. EMA and Flexible resizable array [47] are good for dynamic extension. EMA provides extension only from the surrounding of the array where as Flexible array allows even in the middle of the array. Classical compression schemes have some limitations in compressing data. Like Bitmap and Header compression provide good performance in terms of removing long runs of constants, but they have a poor forward and backward mapping capability. Also, these methods can't be used on dynamic database environment where additions and deletions may be required. The scheme Compressed Row Storage (*CRS*) or Chunk Offset compression are effective for compressing large sparse arrays. But still they cannot be applied on extendible databases. So, it is important to design a compression technique that will be better than these classical compression techniques. The scheme should be efficient enough so that operation can be done over the compressed data. Though, there are a lot of research has been done on compression techniques, but only a few researches have been made on dynamic array organization. Hence we propose new compression techniques based on dynamic array model which will outperform over TMA. The details of the proposed schemes are presented in the next chapter.

CHAPTER III

Compression Schemes for High Dimensional Data based on Extendible Multidimensional Array

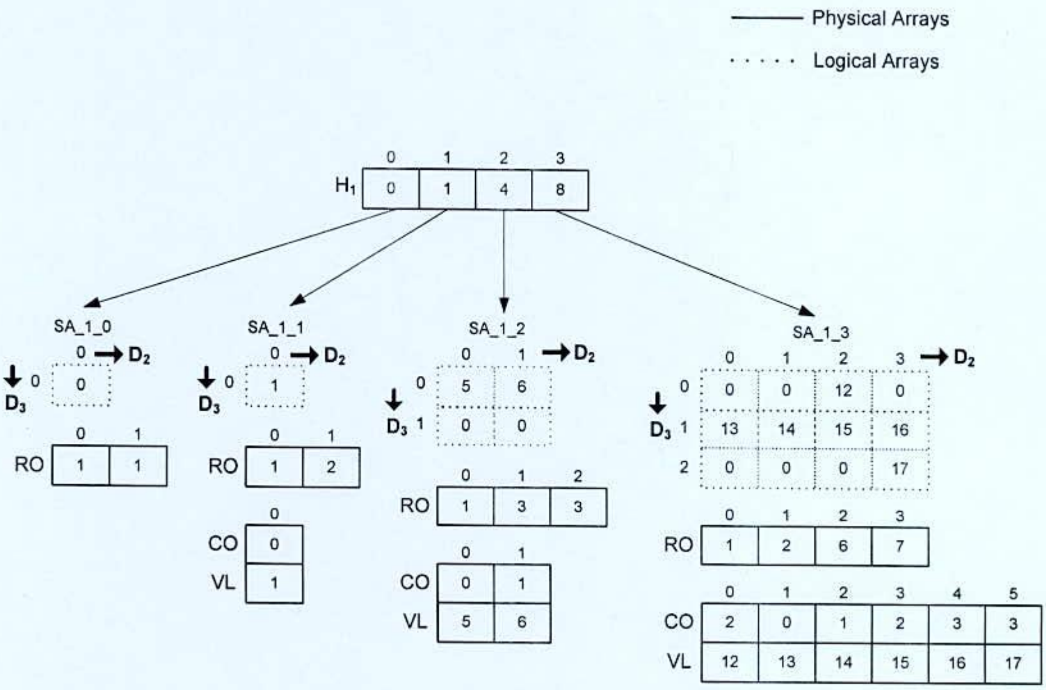
3.1 Introduction

In this chapter, novel methodologies have been proposed to compress high dimensional data based on EMA. In these methods, the basic idea is to apply compression scheme on each subarray of the extendible array independently. Analytical analysis of the proposed schemes is also presented in this chapter. The details of the approaches are discussed in the following sections.

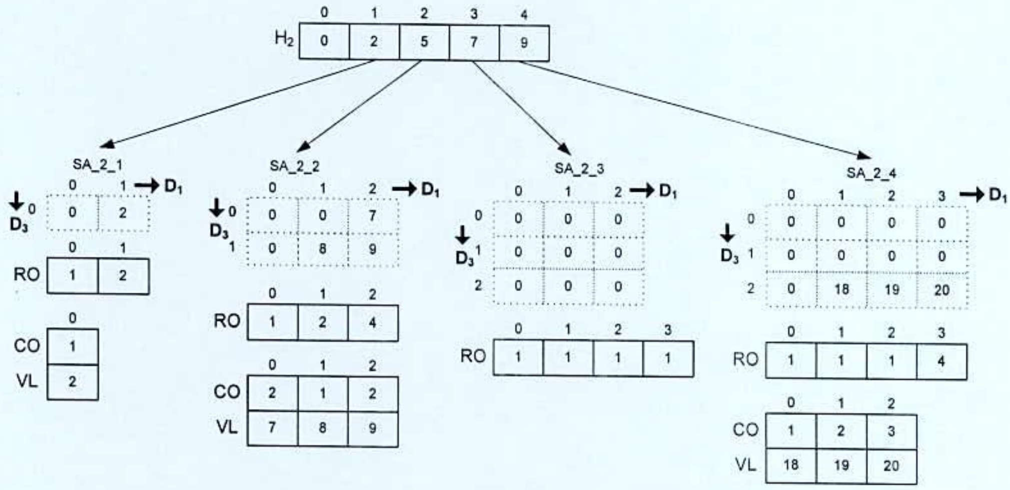
3.2 Extendible Aarray Based Compressed Row Storage Scheme (*EaCRS*)

Given a three dimensional EMA. The Extendible Array Based Compressed Row Storage (*EaCRS*) scheme compresses each subarray independently. This scheme use one one-dimensional floating point array *VL* and two one dimensional integer array *RO* and *CO* for each subarray of the extendible array as the subarrays are two dimensional (since for an n dimensional EMA, subarrays are *n-1* dimensional as described in section 2.2.2) for the three dimensional EMA. This scheme compresses all of the nonzero array elements along the rows of the multidimensional subarays. Array *RO* stores information of nonzero array elements of each row. The dimension with the current minimum length (except the dimension being extended) at the time of extension is considered as the row dimension. If the number of rows is *k* in a subarray then *RO* contains *k+1* elements. *RO[0]* contains 1, *RO[1]* contains the summation of the number non zero elements in row 0 of the subarray and *RO[0]*. In general, *RO[i]* contains the number of nonzero elements in *(i-1)*th row of the array plus the contents of *RO[i-1]*. The number of non zero array elements in the *i*th row can be obtained by subtracting the value of *RO[i]* from *RO[i+1]*. Array *CO* stores the column indices of nonzero array elements of each row. Array *VL* stores the values of nonzero array elements. For each subarray, the base of these three arrays is 0.

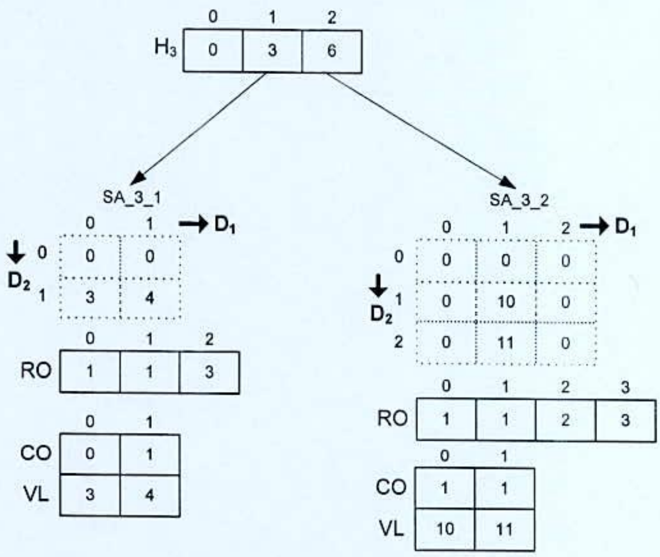
In the *EaCRS* scheme, for an n dimensional EMA, among the three kinds of auxiliary tables (*history table, address table, coefficient table*) only the *history table* H_i is required to store for each dimension. History tables are used to compute the extension dimension of the subarray and the length of other dimension to compute the row dimension and number of row of that subarray. An example of the *EaCRS* scheme for a three dimensional EMA of Figure 2.3 is shown in Figure 3.1. For convenience here we name each subarray as *SA_i_j*, where i indicates the extended dimension that the subarray belongs to and j indicates the length of that dimension. For example, *SA_1_0, SA_1_1, ..., SA_1_L*$_1$ are the subarrays of dimension 1, *SA_2_1, SA_2_2, ..., SA_2_L*$_2$ are the subarrays of dimension 2 and so on.



(a) Subarrays of dimension 1 using *EaCRS* scheme.



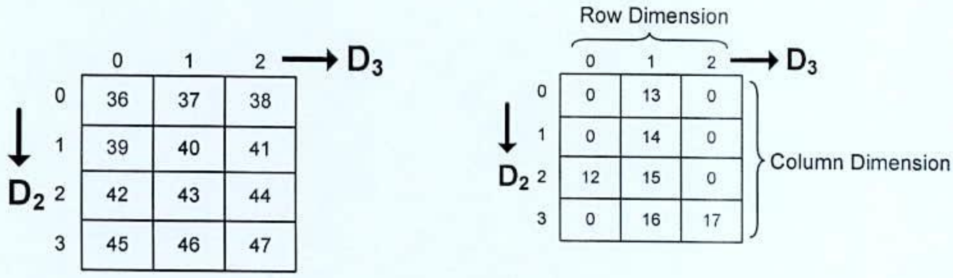
(b) Subarrays of dimension 2 using *EaCRS* scheme.



(c) Subarrays of dimension 3 using *EaCRS* scheme.

Figure 3.1: *EaCRS* scheme for a three dimensional EMA.

Consider a subarray *SA_1_3* of Figure 2.1. This subarray is extended along dimension 1 and the subarray is shown in Figure 3.2(a). Here 36, 37, 38, …, 47 indicates the logical position of each of the subarray elements in the given three dimensional EMA. For explaining the sparseness here we assign each subarray elements to some zero and nonzero values (e.g. logical position 36 is assigned to 0, 37 is assigned to 13, 38 is assigned to 0 and so on.). Since *SA_1_3* is extended along dimension 1(see Figure 2.1), the other two dimensions (dimension 2 and dimension 3) are considered as the row dimension and column dimension. For *SA_1_3,* the length of dimension 3 is less than that of the dimension 2. This is because dimension 3 is considered as the row dimension and dimension 2 is considered as the column dimension in this *EaCRS* scheme (see Figure 3.2(b)).



(a) Subarray *SA_1_3* showing the logical position of each of the subarray elements in a given three dimensional EMA.

(b) Subarray *SA_1_3* showing the sparseness and the considered row dimension and column dimension for the *EaCRS* scheme.

Figure 3.2: A subarray *(SA_1_3)* of the given 3-dimensional EMA at Figure 2.1.

In the subarray *SA_1_3,* there are 3 rows and row 0 contains one nonzero value, 12 (see Figure 3.1(b)). This is because *RO[1]* contains 2 (see Figure 3.1(a)) i.e. *RO[1] = RO[0] + total no. of nonzero array elements in row 0*. Similarly, *RO[2] = 6* (row 1 contains four nonzero values), *RO[3] = 7* (row 2 contains one nonzero value) and so on. *VL* array stores all the nonzero array elements (12, 13, 14, 15, 16) of this subarray and *CO* stores the corresponding column indices of these nonzero array elements.

Logical database and physical database refer to the uncompressed and compressed database respectively. Forward mapping and backward mapping techniques for the *EaCRS* scheme are described as follows:

3.2.1 Forward Mapping for *EaCRS* scheme

Consider the element *<3,3,1>* of the EMA. Compare $H_1[3] = 8$, $H_2[3] = 7$ and $H_3[1] = 3$. Since $H_1[3] > H_2[3]$ and $H_1[3] > H_3[1]$, extended dimension is 1 and the element is involved in the subarray *SA_1_3*. The dimension with the minimum length at the time of subarray *SA_1_3*'s extension is considered as the row dimension for the subarray *SA_1_3*. Since $H_2[3] < H_1[3] < H_2[4]$ and $H_1[3] > H_3[2]$, it can be said that the subarray's

(SA_1_3) size is 4×3, dimension 3 is the row dimension and the number of row is 3. Since subarrays are two dimensional, in this case dimension 2 is the only column of the subarray *SA_1_3*. In Figure 3.1(a), the number of nonzero elements of row 1 can be found by $RO[2] - RO[1] = 6 - 2 = 4$. The column indices of the nonzero array elements of row 1 are stored in *CO[RO[1] - 1]*, *CO[RO[1]]*, *CO[RO[1] + 1]* and *CO[RO[1] + 2]* i.e. *CO[1]*, *CO[2]*, *CO[3]* and *CO[4]*, since there are 4 nonzero array elements exist in row 1. Finally the values of the nonzero array elements of row 1 can be found in *VL[1]*, *VL[2]*, *VL[3]* and *VL[4]*.

3.2.2 Backward Mapping for *EaCRS* scheme

Consider the physical position *<9,4,3>* of the physical database; where *<9>* is the history value, *<4>* is the value that *RO* stores and *<3>* is the column index of a nonzero array element i.e. *<3>* is the value that *CO* stores. We perform the binary search on the history tables to find the given history value *<9>*. Since *<9>* is stored in $H_2[4]$ (see Figure 3.1(b)), we need to access only the *CO* and *RO* arrays that are stored for the subarray *SA_2_4* (i.e. subarray extended at dimension 2 at length 4). Therefore the second co-ordinate value of the desired logical position is *<4>* in logical database and the other two dimensions (dimension 1 and 3) are considered as the row dimension and column dimension. As we described above the dimension with the minimum length at the time of subarray *(SA_2_4)'s* extension is considered as the row dimension for the subarray *SA_2_4*. Since $H_2[4] > H_1[3]$ and $H_2[4] > H_3[2]$, subarray's *(SA_2_4)* size is 4×3.

Dimension 3 is the row dimension because $H_3[2] < H_1[3]$ and the number of row is 3. Since subarrays are two dimensional, in this case dimension 1 is the only column dimension of the subarray *SA_2_4* and the first co-ordinate value of the desired logical position is *<3>* in logical database. As there are 3 rows in the subarray and *<4>* is stored in *RO[3]* (see Figure 3.1(b)), it can be said that column index *<3>* is stored for the nonzero elements of 3rd row of *SA_2_4* i.e. the third co-ordinate value of the desired logical position is *<2>* in logical database. Hence the physical position *<9,4,3>* of physical database is mapped to a logical position *<3,4,2>* in logical database.

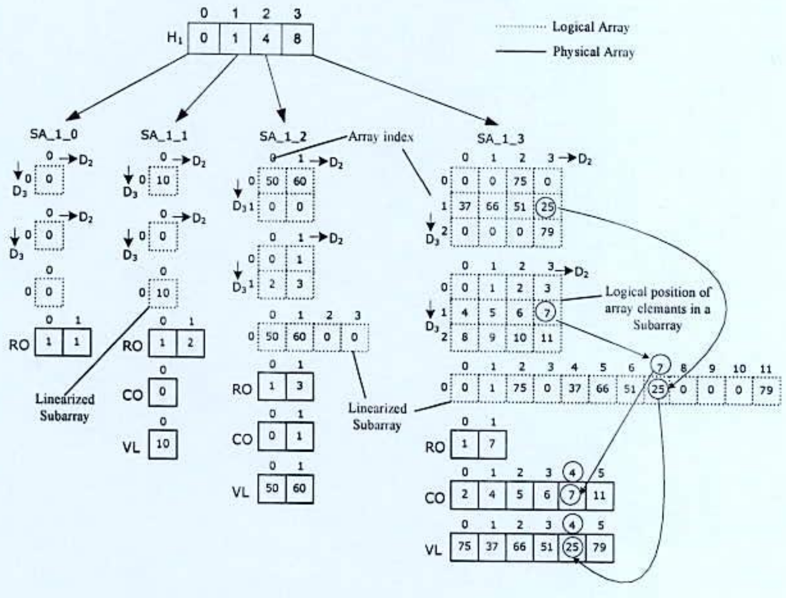
As described above *EaCRS* scheme has the ability to perform both forward mapping and backward mapping and so *EaCRS* scheme is mapping complete.

Based on the *EaCRS* scheme, an extendible multidimensional array of dimension four can be compressed by adding one more one-dimensional integer array *KO*. In the *EaCRS* scheme array *KO* stores the third dimension indices of nonzero array elements of each row. For higher dimensions more one-dimensional integer arrays are needed.

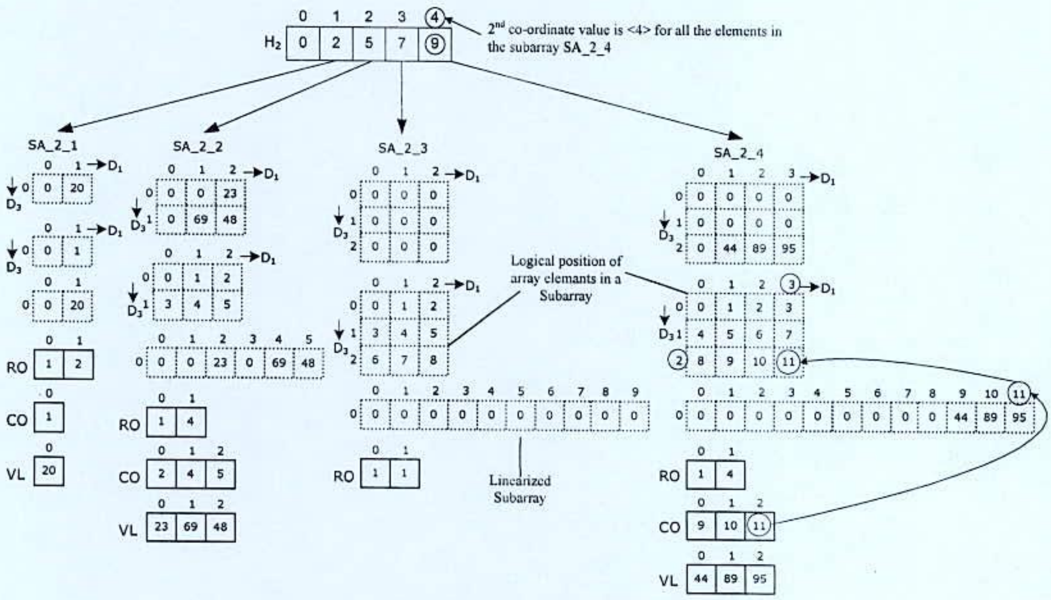
3.3 Linearized Extendible Array Based Compressed Row Storage Scheme (*LEaCRS*)

Given a 3-dimensional EMA. The Linearized Extendible Array Based Compressed Row Storage (*LEaCRS*) scheme compress each subarray independently. This scheme use one one-dimensional floating point array *VL* and two one dimensional integer array *RO* and *CO* for each subarray of the extendible array. This scheme linearize (see section 2.3.1) each subarray independently and then compresses all the nonzero array elements along the only row of each subaray. Array *RO* stores information of nonzero array elements of each subarray. After linearization, as the number of row is 1 in a subarray, then *RO* contains 2 elements. *RO[0]* contains 1, *RO[1]* contains the summation of the number non zero elements in the subarray and *RO[0]*. The number of non zero array elements in each subarray can be obtained by subtracting the value of *RO[0]* from *RO[1]*. Array *CO* stores the column indices of nonzero array elements of each subarray. Array *VL* stores the values of nonzero array elements. For each subarray, the base of these three arrays is 0.

In the *LEaCRS* scheme, for an n dimensional EMA, among the three kinds of auxiliary tables (*history table, address table, coefficient table*) only the *history table* H_i is required to store for each dimension. History tables are used to compute the extension dimension of the subarray and the length of other dimension to carry out the linearization computation for that subarray. An example of the *LEaCRS* scheme for a three dimensional EMA of Figure 2.1 is shown in Figure 3.3. For convenience here we name each subarray as *SA_i_j*, where i indicates the extended dimension that the subarray belongs to and j indicates the length of that dimension. For example, *SA_1_2* is the subarray of dimension 1 at length 2. Similarly *SA_2_1, SA_2_2, ..., SA_2_*L_2 are the subarrays of dimension 2 and so on.



(a) Subarrays of dimension 1 using *LEaCRS* scheme.



(b) Subarrays of dimension 2 using *LEaCRS* scheme.

Figure 3.3: *LEaCRS* scheme for a three dimensional EMA.

Consider a subarray *SA_1_3* of Figure 2.1. This subarray is extended along dimension 1 at dimension length 3 and the subarray is shown in Figure 3.3(a). Here 0, 1, 2, 3, …, 11 indicates the logical position of each of the subarray elements for a linearized subarray. For explaining the sparseness here we assign each subarray elements to some zero and nonzero values (e.g. logical position 1 is assigned to 0, 2 is assigned to 75, 3 is assigned to 0, 3 is assigned to 37 and so on.). Since *SA_1_3* is extended along dimension 1(see Figure 2.1), the other two dimensions (dimension 2 and dimension 3) are considered as the column dimension and row dimension respectively.

In the subarray *SA_1_3,* there are 6 nonzero values. This is because *RO[1]* contains 7 (see Figure 3.3(a)) i.e. *RO[1] = RO[0] + total no. of nonzero array elements in the subarray. VL* array stores all the nonzero array elements (75, 37, 66, 51, 25, 79) of this subarray and *CO* stores the corresponding column indices of the linearized subarray of these nonzero array element.

Forward mapping and backward mapping techniques for the *LEaCRS* scheme are described as follows:

3.3.1 Forward Mapping for *LEaCRS* scheme

Consider the element *<3,3,1>* of the EMA. Compare $H_1[3] = 8$, $H_2[3] = 7$ and $H_3[1] = 3$. Since $H_1[3] > H_2[3]$ and $H_1[3] > H_3[1]$, it can be said that the extended dimension is 1 and the element is involved in the subarray *SA_1_3*. The dimension that is last in the order is considered as the row dimension and other dimension(s) are considered as the column dimension for each subarray. Since $H_2[3] < H_1[3] < H_2[4]$ and $H_1[3] > H_3[2]$, subarray *SA_1_3*'s size is 4×3. Dimension 3 is the row dimension. Since subarrays are two dimensional, in this case dimension 2 is the only column of the subarray *SA_1_3*. In Figure 3.3(a), the number of nonzero elements of the subarray *SA_1_3* can be found by $RO[2] - RO[1] = 7 - 1 = 6$. The linearized column indices of these 6 nonzero array elements are stored in *CO* array. For computing the logical position of the array element *<3,3,1>*; we consider dimension 2 as $d_1 = 4$, dimension 3 as $d_2 = 3$, second co-ordinate value of the given array element as $p_1 = 3$, third co-ordinate value of the given array element as $p_2 = 1$ and the desired logical position of the given array element can be computed as follows using the *array linearization* function (described in section 2.3.1):

$$d_1 p_2 + p_1 = 4 \times 1 + 3 = 7 \text{ [See Figure 3.3(a)]}$$

Binary search is performed on the *CO* array to find logical position 7 and it can be found that *CO[4]* stores the logical position 7 (since *<3,3,1>* array element is a nonzero array element). Finally the values of the nonzero array element can be found in *VL[4]*.

3.3.2 Backward Mapping for *LEaCRS* scheme

Consider the physical position *<9,11>* of the physical database; where *<9>* is the history value and *<11>* is the column index of a nonzero array element in the linearized subarray i.e. *<11>* is the value that *CO* stores. We perform the binary search on the history tables to find the given history value *<9>*. Since *<9>* is stored in $H_2[4]$ see Figure 3.3(b)), we need to access only the *CO* and *RO* arrays that are stored for the subarray *SA_2_4* (i.e. subarray extended at dimension 2 at length 4). Therefore the second co-ordinate value of the desired logical array indices is *<4>* in logical database and the other two dimensions (dimension 1 and 3) are considered as the row dimension and column dimension. As we described above the dimension that is last in the order is considered as the row dimension. Since $H_2[4] > H_1[3]$ and $H_2[4] > H_3[2]$, subarray's *(SA_2_4)* size is 4×3. Dimension 3 is the row dimension and the number of row is 3. Since subarrays are two dimensional, in this case dimension 1 is the only column dimension of the subarray *SA_2_4*. For computing the first co-ordinate and third co-ordinate value of the desired logical array indices in the logical database from the given physical position *<9,11>*; we consider dimension 1 as $d_1 = 4$, dimension 3 as $d_2 = 3$, first co-ordinate value of the desired logical array indices as q_1, third co-ordinate value of the desired logical array indices as q_2, linearized column index *<11>* as Y and the desired logical array indices can be computed as follows using the *reverse array linearization* function (described in section 2.3.1):

$$q_2 = Y \; mod \; d_2 = 11 \; mod \; 3 = 2$$

$$q_1 = Y/d_2 = 11/3 = 3$$

Hence the physical position *<9,11>* of physical database is mapped to a logical position *<3,4,2>* in logical database.

LEaCRS compression scheme is mapping complete because it provides forward mapping and backward mapping (As described above).

3.4 Extendible Array Based Chunk Offset Compression Scheme (*EaChOff*)

Given a three dimensional EMA. The Extendible Array Based Chunk Offset Compression (*EaChOff*) scheme linearize each subarray independently and break a large multi dimensional extendible array into chunks for storage and processing. In this scheme, a maximum size of each chunk is considered and chunks can be formed by single or several subarrays. This scheme use one one-dimensional auxiliary table namely $ChunkNo_i$ for each dimension i and one one-dimensional integer array *NR*. The chunk number assigned to a subarray is held on the *ChunkNo* table. Array *NR* stores information of nonzero array elements of each subarray. If the number of subarrays is k in a EMA then *NR* contains $k+1$ elements. $NR[0]$ contains 1, $NR[1]$ contains the summation of the number of nonzero elements in 0^{th} subarray and $NR[0]$. In general, $NR[i]$ contains the number of nonzero elements in $(i-1)$th subarray of the EMA plus the contents of $NR[i-1]$. The number of non zero array elements in the ith subarray can be obtained by subtracting the value of $NR[i]$ from $NR[i+1]$. This scheme also uses one one-dimensional floating point array *data* and one dimensional integer array *OffsetInChunk* for each chunk of the EMA. Array *data* stores the values of nonzero array elements of each chunk. Array *OffsetInChunk* stores the offset in a chunk of nonzero array elements of each chunk. For each chunk, the base of these two arrays is 0.

In the *EaChOff* scheme, for an n dimensional EMA, among the three kinds of auxiliary tables (*history table, address table, coefficient table*) the *history table* H_i and *address table* L_i are required to store for each dimension. History tables are used to compute the extension dimension of the subarray and the length of other dimension to carry out the linearization computation for that subarray. Address tables are used to point the starting address of each chunk as well as the starting address of each subarray in a chunk. An example of the *EaChOff* scheme for a three dimensional EMA of Figure 2.1 is shown in Figure 3.4. For convenience here we name each subarray as *SA_i_j*, where i indicates the extended dimension that the subarray belongs to and j indicates the length of that dimension. For example, *SA_1_2* is the subarray of dimension 1 at length 2. Similarly *SA_2_1, SA_2_2, …., SA_2_*L_2 are the subarrays of dimension 2 and so on.

Consider a chunk *Chunk1* of Figure 3.4. In this example the maximum chunk size considered is 16. *Chunk1* comprise of subarrays *SA_1_0, SA_1_1, SA_2_1, SA_3_1* and *SA_1_2* in sequence because these subarrays are extended in 1^{st}, 2^{nd}, 3^{rd}, 4^{th} and 5^{th} position in order. The length of this chunk is 12 because the 6^{th} subarray i.e. *SA_2_2*'s length is 6 and 12 plus 6 is 18 which is greater than 16. Alike the length of *Chunk2* is 15 and length of *Chunk3* is 9 and so on.

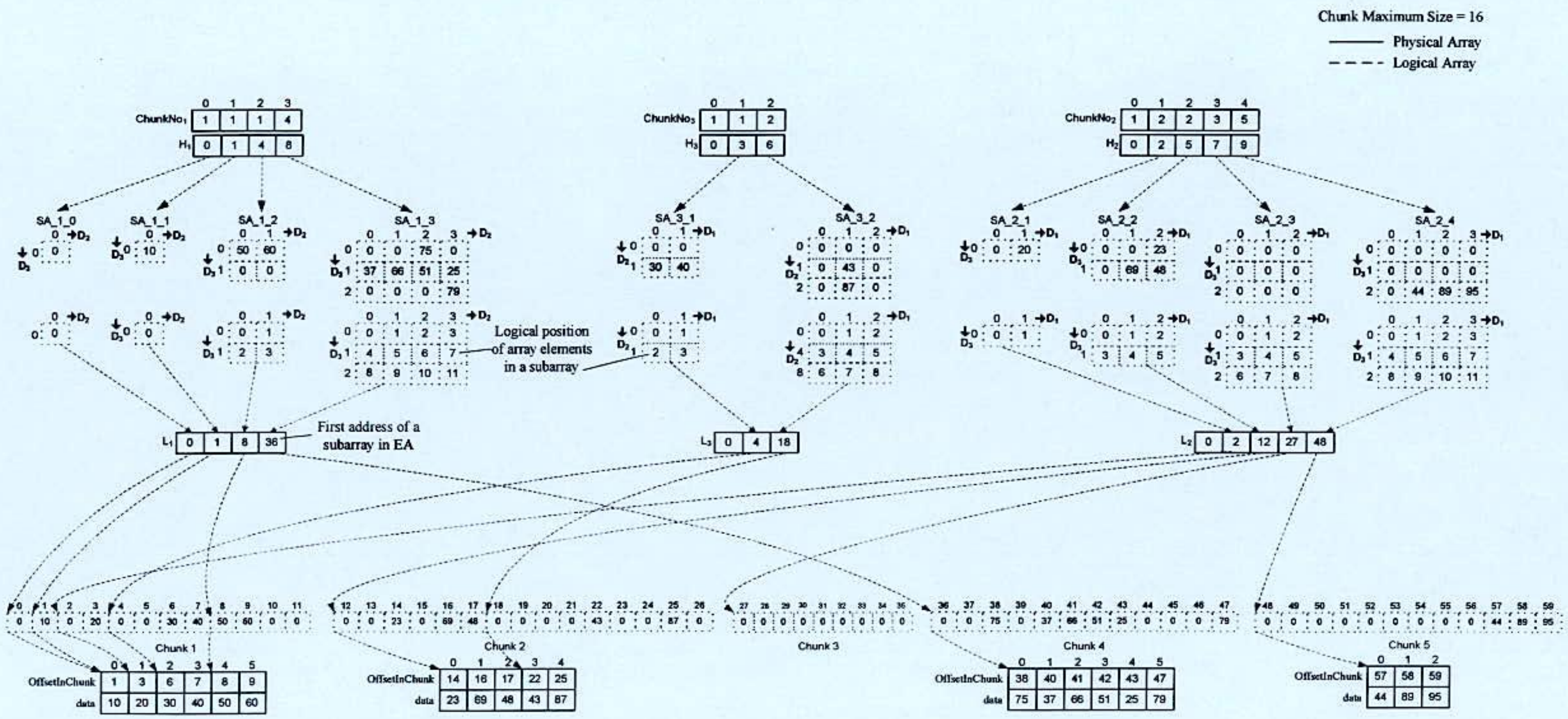


Figure 3.4: *EaChOff* scheme for a three dimensional EMA.

Since *SA_1_0* subarray is assigned to *Chunk1*, $chunkNo_1[0]$ stores 1; likewise $chunkNo_1[1]$ stores 1 for the subarray *SA_1_1*, $chunkNo_2[1]$ stores 1 for the subarray *SA_2_1* and so on. *Chunk3*, *Chunk4* and *Chunk5* consist of a single subarray *SA_2_3*, *SA_1_3* and *SA_2_4* respectively. If the EMA is extended along any dimension then a new chunk namely *Chunk6* will be comprised of this new subarray.

Forward mapping and backward mapping techniques for the *EaChOff* scheme are described as follows:

3.4.1 Forward Mapping for *EaChOff* scheme

Consider the element *<3,3,1>* of the EMA. Compare $H_1[3] = 8$, $H_2[3] = 7$ and $H_3[1] = 3$. Since $H_1[3] > H_2[3]$ and $H_1[3] > H_3[1]$, extended dimension is 1 and the element is involved in the subarray *SA_1_3*. $ChunkNo_1[3] = 4$ indicates that we need to access only *chunk4* for the given element. In Figure 3.5, the number of nonzero elements of the 9th subarray *SA_1_3* can be found by $NR[9] - NR[8] = 7 - 1 = 6$. The chunk offset of these 6 nonzero array elements are stored in *OffsetInChunk* array. For computing the logical position of the array element *<3,3,1>*; we consider dimension 2 as $d_1 = 4$, dimension 3 as $d_2 = 3$, second co-ordinate value of the given array element as $p_1 = 3$, third co-ordinate value of the given array element as $p_2 = 1$ and the desired logical position of the given array element can be computed as follows using the *array linearization* function (described in section 2.3.1):

$$d_1 p_2 + p_1 = 4 \times 1 + 3 = 7 \text{ [See Figure 3.5]}$$

Addition of $L_1[3] = 36$ and logical position 7 give the desired chunk offset value 43 for the given array element. Binary search is performed on the *OffsetInChunk* array to find logical position 43 and it can be found that *OffsetInChunk[4]* stores the logical position 43 (since *<3,3,1>* array element is a nonzero array element). Finally the values of the nonzero array element can be found in *data[4]*.

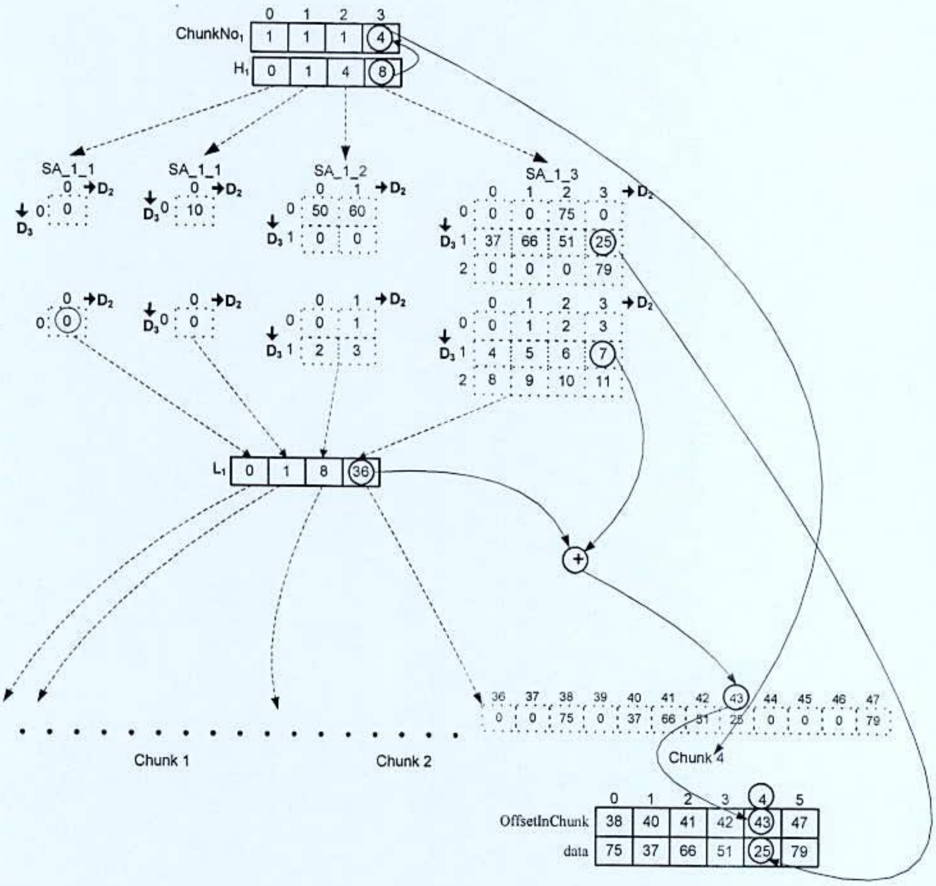


Figure 3.5: An Example of forward mapping for *EaChOff* scheme.

3.4.2 Backward Mapping for *EaChOff* scheme

Consider the physical position *<9,59>* of the physical database; where *<9>* is the history value and *<59>* is the logical index of a nonzero array element in a chunk i.e. *<59>* is the value that *OffsetInChunk* stores. We perform the binary search on the history tables to find the given history value *<9>*. Since *<9>* is stored in $H_2[4]$ (see Figure 3.6), we need to access only the *OffsetInChunk* array that is stored for the subarray *SA_2_4* (i.e. subarray extended at dimension 2 at length 4). Therefore the second co-ordinate value of the desired logical array indices is *<4>* in logical database. The linearized column index of the subarray *SA_2_4* can be computed by subtracting the first address ($L_2[4] = 48$) of the subarray from the given logical chunk index i.e linearized column index = 59 – 48 = 11. For computing the first co-ordinate and third co-ordinate value of the desired logical array indices in the logical database from the given physical position *<9,59>*; we consider dimension 1 as $d_1 = 4$, dimension 3 as $d_2 = 3$, first co-ordinate value of the desired logical array indices as q_1, third co-ordinate value of the desired logical array indices as q_2, linearized column index *<11>* as *Y* and the desired logical array indices can be computed as follows using the *reverse array linearization* function (described in section 2.3.1):

$$q_2 = Y\ mod\ d_2 = 11\ mod\ 3 = 2$$

$$q_1 = Y/d_2 = 11/3 = 3$$

Hence the physical position *<9, 59>* of physical database is mapped to a logical position *<3,4,2>* in logical database.

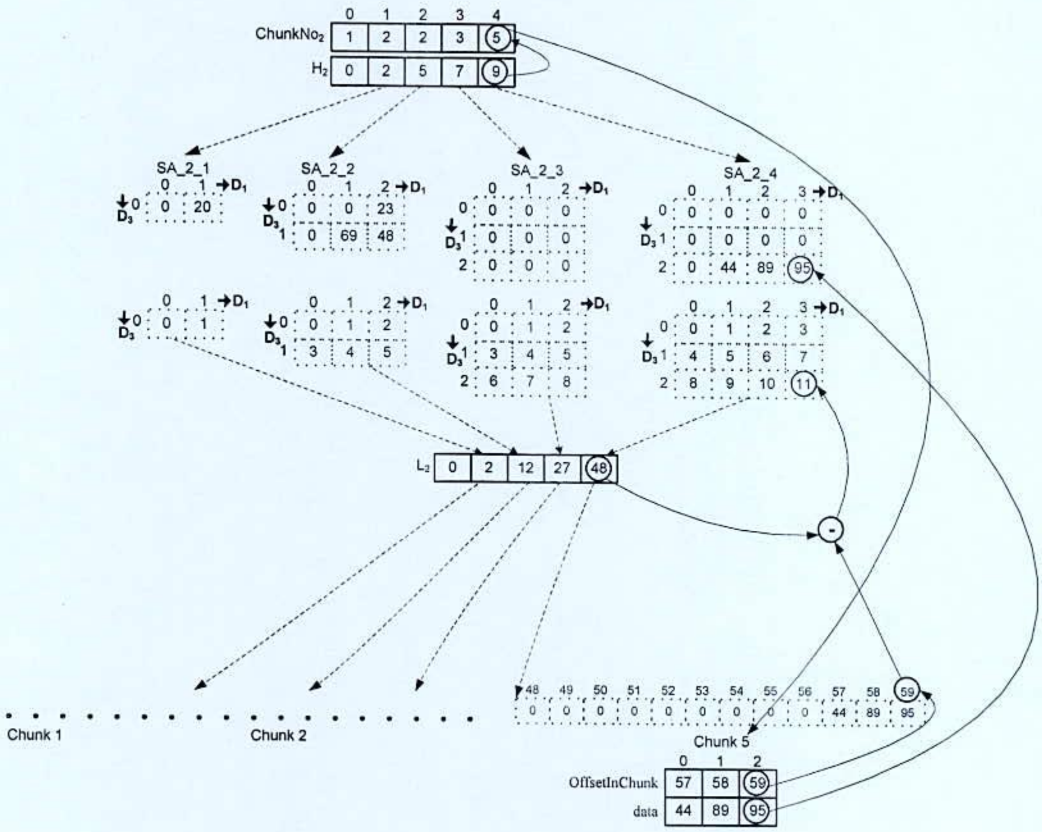


Figure 3.6: An Example of backward mapping for *EaChOff* scheme.

EaChOff compression scheme is also mapping complete because it provides forward mapping and backward mapping (As described above).

3.5 Theoretical Analysis

In this section the cost model for the compression schemes is developed. The analytical analysis is compared with the experimental implementation in chapter IV. Before starting the theoretical analysis the following definitions are important.

Definition 3.1 (Density of Array, ρ). Array density is a parameter to measure the sparsity of an array. It is the ratio of non-empty array cells with total number of cells. Maximum value the density can be one. Formally we can write,

$$\rho = \frac{\text{Total number of cell having non null values}}{\text{Total number of array cells}}$$

Definition 3.2 (Compression Ratio, η): it is defined as the proportionate size of the compressed array with that of uncompressed one, formally

$$\text{Compression ratio, } \eta = \frac{\text{Compressed size of Array}}{\text{Uncompressed size of Array}}$$

The value of η is preferable to be less than one.

Definition 3.3 (Range of usability). Range of usability of a compression scheme is defined as the maximum range of data density up to which the compression ratio is less than 1.

In this section, we model the space requirement and hence the compression ratio for the proposed EMA based schemes that is for *EaCRS*, *LEaCRS* and *EaChOff* schemes. We analyse their range of usability for practical applications as well as their extension cost. We also compare this model with the TMA based schemes i.e. for *CRS* and *Chunk-Offset* (*ChOff*) schemes.

3.5.1 Assumptions

To simplify the model we make the following assumptions.

(i) The length of dimensions extends in round robin manner for both Traditional multidimensional array (TMA) and Extendible multidimensional array (EMA).

(ii) The length of each dimension is equal and when extension occurs each of the dimensions are extended by equal length.

(iii) The records are uniformly distributed in the corresponding TMA or EMA.

3.5.2 Parameters

The parameters are grouped as shown in Table 3.1. Some of these parameters are provided as input, while others are derived from the input parameters. All lengths or sizes are in bytes.

Table 3.1: Parameters Considered for theoretical analysis.

Parameters	**Description**
UC_{TMA}	The uncompressed size of the Traditional Multidimensional Array(TMA)
UC_{EMA}	The uncompressed size of the Extendible Multidimensional Array (EMA)
hc	Total number of subarrays in EMA (i.e. history counter)
n	Number of dimension of both TMA and EMA
L_i	Length of each dimension i ($0 \leq i \leq n$) for both the TMA and EMA
l	Length of Chunk for the TMA
δ	Length of extension
SE_i	Size of extension along dimension i
ρ	Density of records both for TMA and EMA
α	Size of subscripts for TMA and EMA
β	Size of a cell of the TMA and EMA
$sz_i(k)$	Size of subarray k along dimension i
$row\_no_i(k)$	Number of rows in a subarray k along dimension i

SC_{CRS}	Compressed size of TMA using the *CRS* scheme
SC_{ChOff}	Compressed size of TMA using the *Chunk-Offset* Compression scheme
SC_{EaCRS}	Compressed size of EMA using the *EaCRS* scheme
SC_{LEaCRS}	Compressed size of EMA using the *LEaCRS* scheme
η_{CRS}	Compression ratio for the *CRS* scheme for TMA; $\eta_{CRS} = \frac{SC_{CRS}}{UC_{TMA}}$
η_{ChOff}	Compression ratio for the *Chunk-Offset* Compression scheme for TMA; $\eta_{ChOff} = \frac{SC_{ChOff}}{UC_{TMA}}$
η_{EaCRS}	Compression ratio for the *EaCRS* scheme for EMA; $\eta_{EaCRS} = \frac{SC_{EaCRS}}{UC_{EMA}}$
η_{LEaCRS}	Compression ratio for the *LEaCRS* scheme for EMA; $\eta_{LEaCRS} = \frac{SC_{LEaCRS}}{UC_{EMA}}$
$\eta_{EaChOff}$	Compression ratio for the *Chunk-Offset* Compression scheme for EMA; $\eta_{EaChOff} = \frac{SC_{EaChOff}}{UC_{EMA}}$

3.5.3 Cost Model for Compression Ratio

In this section we will derive cost model for compression ratio of TMA based compression schemes i.e. for *CRS* and *ChOff* schemes as well as for EMA based compression schemes i.e. for *EaCRS, LEaCRS* and *EaChOff* schemes.

(a) Cost Model for TMA based schemes

If the length of different dimension L_i $(0 \leq i \leq n)$ is known then storage requirement can be calculated as

$$UC_{TMA} = (\prod_{i=1}^{n} L_i) \times \beta = L^n \times \beta \text{ (assumption (ii), } L_1 = L_2 = \ldots = L_n = L)$$

The number of nonzero array elements of sparse array A is $\rho \times L^n$.

Cost Model for CRS scheme

In the *CRS* scheme, for sparse array *A:*

The size of array RO is: $RO_{CRS} = (L + 1) \times \alpha$

The size of VL array is: $VL_{CRS} = (\rho \times L^n) \times \beta$

The size of each of the CO array is: $CO_{CRS} = (\rho \times L^n) \times \alpha$. There are *n-1* such CO_{CRS} exists. Hence the compressed size of the array *A* i.e. the space requirement of the *CRS* scheme (SC_{CRS}) is,

$$SC_{CRS} = (n-1) \times CO_{CRS} + RO_{CRS} + VL_{CRS}$$

$$= (n-1)\rho L^n \times \alpha + (L+1)\alpha + \rho L^n \beta$$

$$= \big((n-1)\rho L^n + L + 1\big)\alpha + \rho L^n \beta \quad \text{......} \textit{(3.1)}$$

Compression ratio for the *CRS* scheme (η_{CRS}) can be revealed as

$$\eta_{CRS} = \frac{SC_{CRS}}{UC_{TMA}}$$

$$= \frac{\big((n-1)\rho L^n + L + 1\big)\alpha + \rho L^n \beta}{L^n \times \beta} \quad \text{......} \textit{(3.2)}$$

Cost Model for Chunk-Offset Compression scheme

In the *Chunk-Offset* scheme, for sparse array *A:*

No of Chunk in the TMA is:

$$no\_of\_chunk_{Choff} = \frac{L^n}{l^n} \quad \text{(assumption (ii), } L_1 = L_2 = \ldots = L_n = L\text{)}$$

Space required for storing the pointers of all the chunks is:

$$chunkPointers_{Choff} = \frac{L^n}{l^n} \times \alpha$$

Space required for storing the nonzero element counter information for each chunk is:

$$chunkNonzero_{Choff} = \frac{L^n}{l^n} \times \alpha$$

The size of $data$ array is: $data_{Choff} = (\rho \times L^n) \times \beta$

The size of the $OffsetInChunk$ array is: $OffsetInChunk_{Choff} = (\rho \times L^n) \times \alpha$.

The compressed size of the array *A* i.e. the space requirement of the *Chunk Offset* scheme (SC_{Choff}) is,

$$SC_{ChOff} = chunkPointers_{ChOff} + chunkNonzero_{ChOff} + OffsetInChunk_{ChOff} + data_{ChOff}$$

$$= \frac{L^n}{l^n} \times \alpha + \frac{L^n}{l^n} \times \alpha + \rho L^n \times \alpha + \rho L^n \beta$$

$$= 2 \times \frac{L^n}{l^n} \times \alpha + \rho L^n \times \alpha + \rho L^n \beta \qquad \text{.................................... (3.3)}$$

Compression ratio for the *Chunk Offset* scheme (η_{ChOff}) can be revealed as

$$\eta_{ChOff} = \frac{SC_{ChOff}}{UC_{TMA}}$$

$$= \frac{2 \times \frac{L^n}{l^n} \times \alpha + \rho L^n \times \alpha + \rho L^n \beta}{L^n \times \beta} \qquad \text{.. (3.4)}$$

From equation (3.1) and (3.3) we find that space required for storing the VL_{CRS} and CO_{CRS} is equal to that of $data_{ChOff}$ and $OffsetInChunk_{ChOff}$ respectively. For convenience we ignore the space required for the RO_{CRS}, $chunkPointers_{ChOff}$ and $chunkNonzero_{ChOff}$ arrays, since the size of most of sparse arrays in practical application is large and space required for these arrays is negligible with respect to that of VL_{CRS}, CO_{CRS}, $data_{ChOff}$ and $OffsetInChunk_{ChOff}$ arrays for very large sparse arrays. Therefore $SC_{CRS} > SC_{ChOff}$ i.e. space requirement for the Chunk-Offset scheme is less than that of the *CRS* scheme. This is because, for *n*-dimensional TMA *(n-1)* nos. CO_{CRS} is required for the *CRS* scheme (equation 3.1), but only one $OffsetInChunk_{ChOff}$ array is required for the *ChOff* scheme (equation 3.3).

(b) Cost Model for EMA Based schemes

Let sparse extendible array, A' be the corresponding sparse array based on the EMA. As the length of dimension is equal for all the schemes, the uncompressed size of the array A' will be identical to the uncompressed size of A i.e. $UC_{EMA} = L^n \times \beta$.

If the length of *i*th dimension of A' is L_i, the total number of subarray is:

$$hc = \sum_{i=1}^{n}(L_i - 1) + 1$$

$$= (L - 1) \times n + 1 \quad \text{(assumption (ii), } L_1 = L_2 = \ldots = L_n = L\text{)}$$

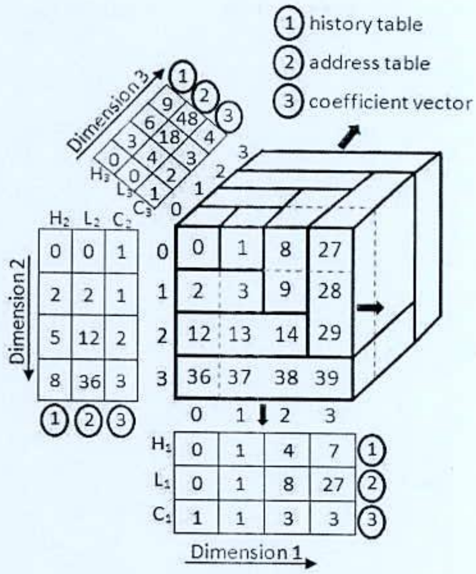


Figure 3.7: A three dimensional extendible array in which each dimension extends in round robin manner and L is 4.

For example, consider a three dimensional extendible array as shown in Figure 3.7; in which the length of each dimension is extended in round robin manner and the length of each dimension is equal (L= 4).

Therefore the total no. of subarray will be

$$hc = (4-1) \times 3 + 1 = 10.$$

The size of the ith subarray for extension along any arbitrary dimension $k\,(0 \leq k \leq n)$ can be calculated as:

$$sz_i(k) = \prod_{j=1}^{n} L_j \;\; [j \neq k]$$

The number of nonzero array elements of ith subarray along extension-dimension k of *EMA* is $sz_i(k) \times \rho$ and the size of the ith VL_i array is $sz_i(k) \times \rho \times \beta$.

The total numbers of nonzero array elements of A' can be obtained by the summation of all of the subarray's nonzero elements. Hence the size of the total VL array and *data* array for EMA based schemes becomes:

$$VL_{EA} = data_{EA} = \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta \quad [1 \leq k \leq n] \qquad \text{... (3.5)}$$

and the size of the total CO array and *OffsetInChunk* array for EMA based schemes will be:

$$CO_{EA} = OffsetInChunk_{EA} = \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \alpha \quad \text{.......... (3.6)}$$

Cost Model for EaCRS scheme

The *EaCRS* scheme does not linearize the subarray. Hence it requires more auxiliary arrays. For the *EaCRS* scheme row dimension of the *i*th subarray for extension along dimension k is the dimension with the minimum length at the time of *i*th subarray's extension among the n dimensions (other than k) and the number of row will be:

$$row\_no_i(k) = \min(d_j) \quad [1 \le j \le n \text{ and } j \ne k]$$

No. of elements in the *i*th *RO* array for *i*th subarray = $row\_no_i(k) + 1$

Since *RO[0]* stores 1 in each *RO* array, we do not require to store *RO[0]* for each *RO* array.

Therefore the size of the total *RO* array for *EaCRS* scheme is:

$$RO_{EaCRS} = \left(\sum_{i=1}^{hc} row\_no_i(k)\right) \times \alpha \quad \text{.......... (3.7)}$$

Compressed size of the array A' using *EaCRS* scheme *i.e.* (SC_{EaCRS}) is,

$$SC_{EaCRS} = (n-2) \times CO_{EA} + RO_{EaCRS} + VL_{EA}$$

$$= [(n-2)\left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) + \sum_{i=1}^{hc} row\_no_i(k)] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta \quad \text{..... (3.8)}$$

Compression ratio for the *EaCRS* scheme (η_{EaCRS}) can be revealed as

$$\eta_{EaCRS} = \frac{SC_{EaCRS}}{UC_{EMA}}$$

$$= \frac{[(n-2)\left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) + \sum_{i=1}^{hc}(row\_no_i(k))] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta}{L^n \times \beta} \quad \text{.......... (3.9)}$$

Cost Model for LEaCRS scheme

In the *LEaCRS* scheme, $row\_no_i(k)$ =1 because there is only one row for each subarray after linearization.

Number of elements in the *i*th *RO* array for *i*th subarray = $row\_no_i(k) + 1 = 2$.

Since *RO[0]* stores 1 in each *RO* array, we do not require to store *RO[0]* for each *RO* array.

Therefore the size of the total *RO* array for *LEaCRS* scheme is:

$$RO_{LEaCRS} = \left(\sum_{i=1}^{hc} 1\right) \times \alpha$$

Compressed size of the array A' using *LEaCRS* scheme *i.e.* (SC_{LEaCRS}) is,

$$SC_{LEaCRS} = CO_{EA} + RO_{LEaCRS} + VL_{EA}$$

$$= \left[\left(\sum_{i=1}^{hc} sz_i\,(k) \times \rho\right) + \sum_{i=1}^{hc} 1\right] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta \quad \text{.................. (3.10)}$$

Compression ratio for the *LEaCRS* scheme (η_{LEaCRS}) can be revealed as

$$\eta_{LEaCRS} = \frac{SC_{LEaCRS}}{UC_{EMA}}$$

$$= \frac{\left[\left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) + \sum_{i=1}^{hc}(1)\right] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta}{L^n \times \beta} \quad \text{.. (3.11)}$$

Cost Model for EaChOff scheme

The *EaChOff* scheme stores pointers and nonzero element information for each subarray. Therefore the size of the total *chunkpointers* and *chunknonzero* array for *EaChOff* scheme is:

$$chunkPointers_{EaChoff} = chunkNonzero_{EaChoff} = hc \times \alpha$$

$$= [(L-1) \times n + 1] \times \alpha \quad \text{....... (3.12)}$$

Compressed size of the array A' using *EaChOff* scheme *i.e.* $(SC_{EaChoff})$ is,

$$SC_{EaChoff} = chunkPointers_{EaChoff} + chunkNonzero_{EaChoff} + data_{EA} + OffsetInChunk_{EA}$$

$$= [2 \times \{(L-1) \times n + 1\} + \left(\sum_{i=1}^{hc} sz_i\,(k) \times \rho\right)] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta \quad \text{..... (3.13)}$$

Compression ratio for the *EaChOff* scheme (η_{EaCRS}) can be revealed as

$$\eta_{EaChoff} = \frac{SC_{EaChoff}}{UC_{EMA}}$$

$$= \frac{[2 \times \{(L-1) \times n + 1\} + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right)] \times \alpha + \left(\sum_{i=1}^{hc} sz_i(k) \times \rho\right) \times \beta}{L^n \times \beta} \quad \text{.. (3.14)}$$

Table 3.2 shows the total size of the *VL, data, CO, RO* and $OffsetInChunk$ arrays for *EaCRS*, *LEaCRS* and *EaChOff* schemes for 3-dimensional, 4-dimensional and n-dimensional EMA based on the above discussions.

Table 3.2: Total size of the *VL, data, CO, RO* and $OffsetInChunk$ arrays for *EaCRS, LEaCRS* and *EaChOff* schemes.

Arrays / Dimensions	$VL_{EA}/$ $data_{EA}$	$CO_{EA}/$ $OffsetInChunk_{EA}$	RO_{EaCRS}	RO_{LEaCRS}
3-D	$\rho L^3\beta$	$\rho L^3\alpha$	$\frac{L(3L-1)}{2}\times\alpha$	$(3L-2)\times\alpha$
4-D	$\rho L^4\beta$	$\rho L^4\alpha$	$\frac{L(4L-2)}{2}\times\alpha$	$(4L-3)\times\alpha$
n-D	$\rho L^n\beta$	$\rho L^n\alpha$	$\frac{L(nL-(n-2))}{2}\times\alpha$	$(nL-(n-1))\times\alpha$

From equation (3.8), (3.10) and (3.13) we find that $SC_{EaCRS} > SC_{LEaCRS}$ and $SC_{EaCRS} > SC_{EaChoff}$ and $SC_{LEaCRS} \approx SC_{EaChoff}$.This is because for n-dimensional EMA,$VL_{EA} = data_{EA} = \rho L^n\beta$ (equation 3.5 and Table 3.2) and $CO_{EA} = OffsetInChunk_{EA} = \rho L^n\alpha$ (equation 3.6 and Table 3.2). *EaCRS* scheme requires storage for *(n-2)* nos. CO_{EA} arrays (equation 3.8) but *LEaCRS* and *EaChOff* schemes require storage for only one CO_{EA} array (equation 3.10) and only one $OffsetInChunk_{EA}$ array (equation 3.13) respectively. For convenience we ignore the space required for the RO_{EaCRS}, RO_{LEaCRS}, $chunkPointers_{EA}$ and $chunkNonzero_{EA}$ arrays, since space required for these arrays is negligible with respect to that of VL_{EA}, CO_{EA}, $data_{EA}$ and $OffsetInChunk_{EA}$ arrays.

So, $\eta_{CRS} > \eta_{EaCRS}$ because *EaCRS* scheme requires one less *CO* auxiliary array for each subarray than the *CRS* scheme since subarrays are *n-1* dimensional for *n*-dimensional EMA. Similarly $\eta_{EaCRS} > \eta_{LEaCRS}$, because *LEaCRS* scheme requires only one CO_{EA} auxiliary array for each subarray. We also find that, $\eta_{Choff} \approx \eta_{EaChoff} \approx \eta_{LEaCRS}$ because *ChOff* scheme requires only one $OffsetInChunk_{ChOff}$ auxiliary array for the TMA and *EaChOff* scheme requires only one $OffsetInChunk_{EA}$ auxiliary array for the EMA. Since *OffsetInChunk* array stores offset information for non zero values only; $OffsetInChunk_{ChOff} = OffsetInChunk_{EA} = CO_{EA}$ (equation 3.6).

3.5.4 Range of usability Analysis

(a) Range of usability analysis for TMA based schemes

Now we derive the range of usability for a three dimensional traditional multidimensional array for the *CRS* and *Chunk Offset* schemes.

CRS scheme

One of the goals to use the data compression scheme is to reduce the memory space required for sparse array. From equation (3.2) we can derive the range of usability of the *CRS* scheme.

For example if we consider $n = 3$, from equation (3.1) we get,

$$SC_{CRS} = \left((3-1)\rho L^3 + L + 1\right)\alpha + \rho L^3 \beta$$

$$= (2\rho L^3 + L + 1)\alpha + \rho L^3 \beta$$

For deriving the range of usability for the *CRS* scheme we consider $\eta_{CRS} = 1$and n=3 in equation (3.2) and we get,

$$\frac{(2\rho L^3+L+1)\alpha+\rho L^3\beta}{L^3\beta} = 1$$

$$or,\ (2\rho L^3 + L + 1)\alpha + \rho L^3\beta = L^3\beta$$

$$or,\ \rho L^3(2\alpha + \beta) = L^3\beta - (L+1)\alpha$$

$$or,\ \rho = \frac{\beta}{2\alpha+\beta} - \left(\frac{L+1}{L^3} \times \frac{\alpha}{2\alpha+\beta}\right)$$

$$or,\ \rho < \frac{\beta}{2\alpha+\beta}$$

Chunk-Offset scheme

From equation (3.4) we can derive the range of usability of the *Chunk-Offset* scheme.

For example if we consider $n = 3$, from equation (3.3) we get,

$$SC_{Choff} = 2 \times \frac{L^3}{l^3} \times \alpha + \rho L^3 \times \alpha + \rho L^3 \beta$$

For deriving the range of usability for the *Chunk-Offset* scheme we consider $\eta_{Choff} =$ 1 and n=3 in equation (3.2) and we get,

$$\frac{2 \times \frac{L^3}{l^3} \times \alpha + \rho L^3 \times \alpha + \rho L^3\beta}{L^3 \times \beta} = 1$$

$or,\ 2 \times \frac{L^3}{l^3}\,\alpha + \rho L^3\alpha + \rho L^3\beta = L^3\beta$

$or,\ \rho L^3(\alpha + \beta) = L^3\beta - 2 \times \frac{L^3}{l^3}\,\alpha$

$or,\ \rho = \frac{\beta}{\alpha+\beta} - (\frac{2}{l^3} \times \frac{\alpha}{\alpha+\beta})$

$or,\ \rho < \frac{\beta}{\alpha+\beta}$

Table 3.3 shows the range of usability of the *CRS* scheme (derived from equation (3.2)) and the *ChOff* scheme (derived from equation (3.4)) for 3-dimensional, 4-dimensional and n-dimensional TMA.

Table 3.3: The range of usability of the TMA based (*CRS* and *ChOff*) schemes

Schemes / **Dimensions**	***CRS***	***Chunk-Offset***
3-D	$\rho < \frac{\beta}{2\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$
4-D	$\rho < \frac{\beta}{3\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$
n-D	$\rho < \frac{\beta}{(n-1)\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$

(b) Range of usability analysis for EMA based schemes

Now we derive the range of usability for a three dimensional extendible array (See Figure 3.7) for the *EaCRS*, *LEaCRS* and *EaChOff* schemes.

If we consider the length of each dimension is L, the value of hc for such an array is:

$$hc = (L-1) \times 3 + 1 = 3L - 2.$$

From equation (3.5) we get,

$VL_{EA} = data_{EA} = (\sum_{i=1}^{3L-2} sz_i(k) \times \rho) \times \beta \quad [1 \le k \le n]$

Where, k is the extension dimension of ith subarray. Since the length of each dimension is extended in round robin manner and length of each dimension is equal (Assumption (i) and (ii))

therefore, $\sum_{i=1}^{3L-2} sz_i(k) = L^3 \quad [1 \le k \le n]$ and $VL_{EA} = data_{EA} = \rho L^3\,\beta$

Similarly from equation (3.6) we get, $CO_{EA} = OffsetInChunk_{EA} = \rho L^3\,\alpha$

EaCRS scheme

For the *EaCRS* scheme, the size of the total RO_{EaCRS} array will be like this (Using assumption (i) and (ii))):

$$RO_{EaCRS} = [1 + 1 + 1 + 2 + 2 + 2 + \cdots + (L-1) + (L-1) + (L-1) + L]\alpha$$

[See Figure 3]

$$= [3 \times \{1 + 2 + \cdots + (L-1)\} + L]\alpha$$

$$= [3 \times \frac{(L-1)(L-1+1)}{2} + L]\alpha$$

$$= \frac{L(3L-1)}{2}\alpha$$

From equation (3.8) we get,

$$SC_{EaCRS} = (3-2) \times \rho L^3\,\alpha\, + \frac{L(3L-1)}{2}\alpha + \rho L^3\,\beta\,.$$

$$= \rho L^3\,\alpha\, + \frac{L(3L-1)}{2}\alpha + \rho L^3\,\beta\,.$$

For deriving the range of usability for the *EaCRS* scheme we consider $\eta_{EaCRS} = 1$ in equation (3.9) and we get,

$$\frac{\rho L^3\,\alpha + \frac{L(3L-1)}{2}\alpha + \rho L^3\,\beta}{L^3\,\beta} = 1$$

$$or,\ \rho L^3\,\alpha\, + \frac{L(3L-1)}{2}\alpha + \rho L^3\,\beta = L^3\,\beta$$

$$or,\ \rho L^3\,(\alpha\, + \beta) = L^3\,\beta - \frac{L(3L-1)}{2}\alpha$$

$$or,\ \rho = \frac{\beta}{\alpha + \beta} - \left(\frac{L(3L-1)}{2L^3} \times \frac{\alpha}{\alpha + \beta}\right)$$

$$or,\ \rho < \frac{\beta}{\alpha + \beta}$$

LEaCRS scheme

For the *LEaCRS* scheme, the size of the total RO_{LEaCRS} array will be like this:

$$RO_{LEaCRS} = \left(\sum_{i=1}^{3L-2}(1)\right) \times \alpha$$

$$= (3L-2) \times \alpha$$

From equation (3.10) we get,

$SC_{LEaCRS} = \rho L^3\, \alpha\, + (3L-2)\alpha + \rho L^3\, \beta$.

For deriving the range of usability for the *LEaCRS* scheme we consider $\eta_{LEaCRS} = 1$ in equation (3.11) and we get,

$$\frac{\rho L^3\, \alpha\, + (3L-2)\alpha + \rho L^3\, \beta}{L^3\, \beta} = 1$$

$$or,\ \rho L^3\, \alpha\, + (3L-2)\alpha + \rho L^3\, \beta = L^3\, \beta$$

$$or,\ \rho L^3\,(\alpha\, + \beta) = L^3\, \beta - (3L-2)\alpha$$

$$or,\ \rho = \frac{\beta}{\alpha+\beta} - \left(\frac{(3L-2)}{L^3} \times \frac{\alpha}{\alpha+\beta}\right)$$

$$or,\ \rho < \frac{\beta}{\alpha+\beta}$$

EaChOff scheme

For the *EaChOff* scheme, the size of the total $chunkPointers_{EaChoff}$ and $chunkNonzero_{EaChoff}$ array will be like this:

$$chunkPointers_{EaChoff} = chunkNonzero_{EaChoff} = (3L-2) \times \alpha$$

From equation (3.13) we get,

$SC_{EaChoff} = 2 \times (3L-2)\alpha + \rho L^3\, \alpha + \rho L^3\, \beta$.

For deriving the range of usability for the *EaChOff* scheme we consider $\eta_{EaChoff} = 1$ in equation (3.14) and we get,

$$\frac{2 \times (3L-2)\alpha + \rho L^3\, \alpha + \rho L^3\, \beta}{L^3\, \beta} = 1$$

$$or,\ 2 \times (3L-2)\alpha + \rho L^3\, \alpha\, + \rho L^3\, \beta = L^3\, \beta$$

$$or,\ \rho L^3\,(\alpha\, + \beta) = L^3\, \beta - 2 \times (3L-2)\alpha$$

$$or,\ \rho = \frac{\beta}{\alpha+\beta} - \left(\frac{2 \times (3L-2)}{L^3} \times \frac{\alpha}{\alpha+\beta}\right)$$

$$or,\ \rho < \frac{\beta}{\alpha+\beta}$$

Table 3.4 shows the range of usability the *EaCRS* scheme (derived from equation (3.7) using Table 3.2), the *LEaCRS* scheme (derived from equation (3.10) using Table 3.2) and

EaChOff scheme (derived from equation (3.14) using Table 3.2) for 3-dimensional, 4-dimensional and n-dimensional EMA.

Table 3.4: The range of usability of the EMA based (*EaCRS, LEaCRS* and *EaChOff*) schemes

Dimensions \ Schemes	***EaCRS***	***LEaCRS***	***EaChOff***
3-D	$\rho < \frac{\beta}{\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$
4-D	$\rho < \frac{\beta}{2\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$
n-D	$\rho < \frac{\beta}{(n-2)\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$	$\rho < \frac{\beta}{\alpha + \beta}$

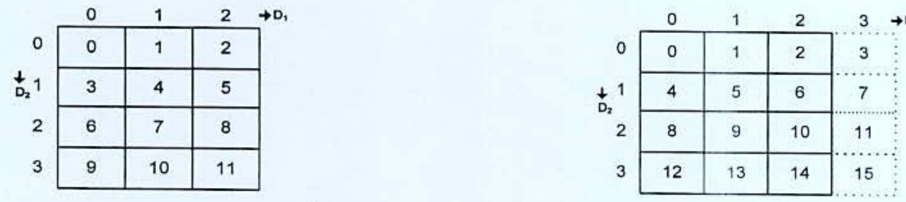
In Table 3.3 and Table 3.4, we can see that the range of usability of the *ChOff, LEaCRS* and *EaChOff* schemes are almost equal and wider than that of both the *CRS* and *EaCRS* schemes. Range of usability of the *ChOff, LEaCRS* and *EaChOff* schemes are same for any dimensional EMA whereas the range of usability of the *CRS* and *EaCRS* schemes decrease with the increase of dimensionality.

3.5.4 Extension Cost Analysis

Since the volume of *RO* array is much smaller with respect to the volume of *VL* and *CO* arrays in all the cases of the *CRS* based compression schemes and *chunkPointers* and *chunkNonzero* arrays are much smaller than *data* and *OffsetInChunk* arrays in all the cases of Chunk Offset based compression schemes, we ignore the extension cost for the *RO, chunkPointers* and *chunkNonzero* arrays for the convenience of calculation.

(a) Extension Cost for TMA based schemes

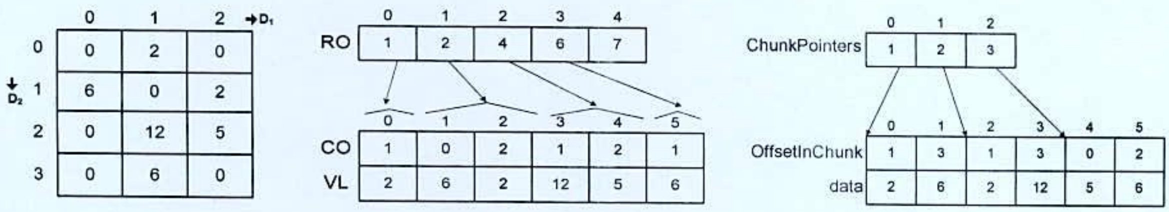
Figure 3.9(b), 3.9(e) and Figure 3.9(c), 3.9(f) pairs show the before and after view of extension of *CRS* and *Chunk Offset* respectively for a 2 dimensional TMA. *CRS* and *Chunk Offset* arrays has to be reorganized to extend because the offset values are changed when the TMA is extended in dimension 1 (shown in Figure 3.8(a) and 3.8(b)). Since the offset values are subject to change; to get the correct value of a cell we have to fetch the previously allocated data and then reorganize the arrays.



(a) Before extension

(b) After extension

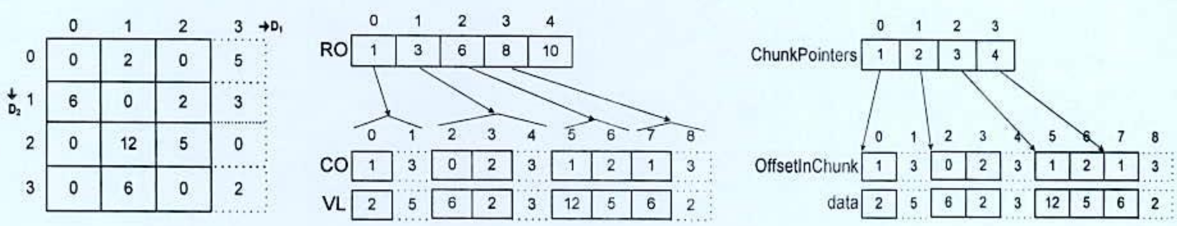
Figure 3.8: Extension of a 2 dimensional TMA.



(a) Sparse TMA (before extension)

(b) CRS on sparse array (before extension)

(c) Chunk Offset on sparse array (before extension)



(d) Sparse TMA (after extension)

(e) CRS on sparse array (after extension)

(f) Chunk Offset on sparse array (after extension)

Figure 3.9: Extension cost analysis for TMA based scheme.

Cost for CRS scheme

Let us consider a TMA(n), with initial volume $V = L^n$ for each dimension length $L_i = L$ before compression.

Initial volume of the VL array is: $V_{VL}^{CRS} = (\rho \times L^n)$

Initial volume of the CO array is: $V_{CO}^{CRS} = (\rho \times L^n)$

Therefore initial volume of CRS is:

$V_{CRS} = V_{VL}^{CRS} + (n-1) \times V_{CO}^{CRS}$ [Since (n-1) nos. *CO* array exist for *CRS* scheme for n-dimensional TMA]

$$= \rho L^n + (n-1)\rho L^n$$

$$= n\rho L^n$$

For extending TMA, it requires to reorganize the array and rewrite both existing and new data elements. The existing elements of the initial array need to be fetched and recalculate the new offsets due to the extension for TMA.

Hence the cost of fetching (FC) the existing array elements of *CRS* becomes

$$FC_{CRS} = V_{CRS} = n\rho L^n$$

If a TMA is extended by δ then a new TMA of length $L + \delta$ is to be reallocated, hence reallocation cost of *CRS* is:

$$RC_{CRS} = RC_{VL}^{CRS} + RC_{CO}^{CRS}$$

$$= \rho(L+\delta)^n + \rho(n-1)(L+\delta)^n$$

$$= n\rho(L+\delta)^n$$

So, total extension cost for *CRS* is: $EC_{\delta}^{CRS} = FC_{CRS} + RC_{CRS}$

$$= n\rho L^n + n\rho(L+\delta)^n$$

$$= n\rho L^n + n\rho\left(\sum_{i=0}^{n} {}^nC_i\, L^{n-i}\delta^i\right)$$

$$= n\rho L^n + n\rho\left({}^nC_0 L^n + \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i\right)$$

$$= 2n\rho L^n + n\rho \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i$$

Cost for Chunk-Offset scheme

Consider a TMA(n), with initial volume $V = L^n$ for each dimension length $L_i = L$

Initial volume of $data_{ChOff}$ array is: $V_{data} = (\rho \times L^n)$

Initial volume of the $OffsetInChunk_{ChOff}$ array is: $V_{OffsetInChunk} = (\rho \times L^n)$

Therefore initial volume of *Chunk Offset* is: $V_{ChOff} = V_{data} + V_{OffsetInChunk}$

$$= \rho L^n + \rho L^n$$

$$= 2\rho L^n$$

For extending TMA, it requires to reorganize the array and rewrite both existing and new data elements. The existing elements of the initial array need to be fetched and recalculate the new offsets due to the extension for TMA.

Hence the cost of fetching (FC) the existing array elements of *Chunk Offset* becomes

$$FC_{ChOff} = V_{ChOff} = 2\rho L^n$$

If a TMA is extended by δ then a new TMA of length $L + \delta$ is to be reallocated, hence reallocation cost of *Chunk Offset* is: $RC_{ChOff} = RC_{data} + RC_{OffsetInChunk}$

$$= \rho(L+\delta)^n + \rho(L+\delta)^n$$

$$= 2\rho(L+\delta)^n$$

So, total extension cost for *Chunk Offset* is: $EC_\delta^{ChOff} = FC_{ChOff} + RC_{ChOff}$

$$= 2\rho L^n + 2\rho(L+\delta)^n$$

$$= 2\rho L^n + 2\rho\left(\textstyle\sum_{i=0}^{n} {}^nC_i\, L^{n-i}\delta^i\right)$$

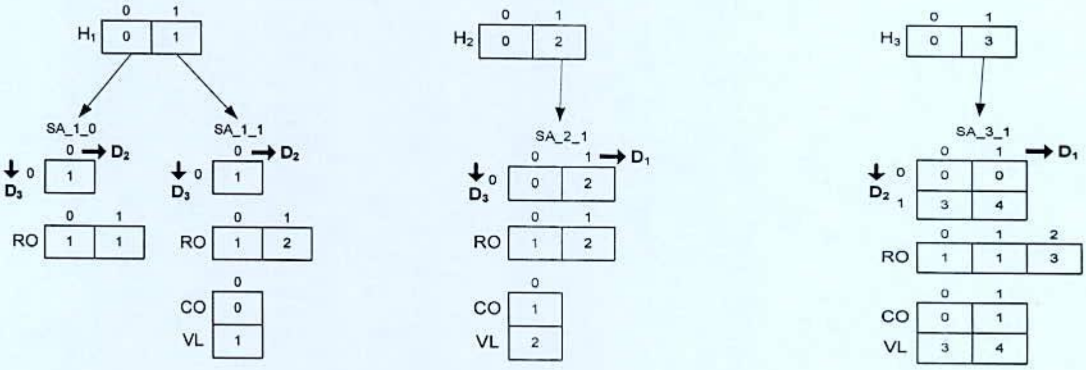
$$= 2\rho L^n + 2\rho\left({}^nC_0 L^n + \textstyle\sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i\right)$$

$$= 4\rho L^n + 2\rho \textstyle\sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i$$

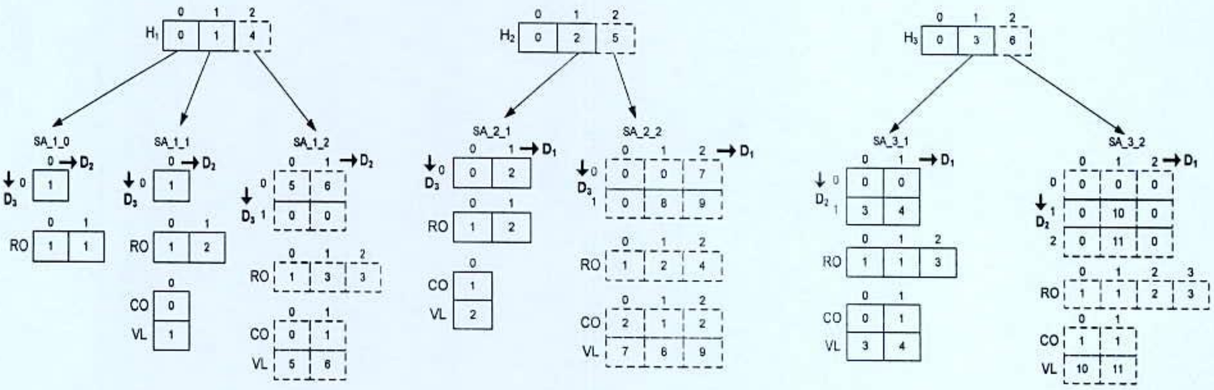
(b) Extension Cost for EMA based schemes

Figure 3.9 shows the pictorial view of δ unit extension of *EaCRS*(3). By δ unit extension we mean that all dimensions of the EMA are extended a value δ. From Figure 3.10(a) and

3.10(b), we see that for extension of *EaCRS* we need to apply *CRS* only on the newly extended subarray. Similarly for *LEaCRS* and *EaChOff* extension, we do not require to process the previously allocated subarray; we need to apply compression scheme only on the newly extended subarray.



(a) Before extension *EaCRS*(3)



(b) After extension *EaCRS*(3)

Figure 3.10: Extension cost analysis for EMA based scheme.

Cost for EaCRS scheme

Let us consider EMA(n), with initial volume of the array before compression V = L^n (considering length of each dimension $L_i = L$)

Initial volume of the VL array is: $V_{VL}^{EA} = (\rho \times L^n)$

Initial volume of the CO array is: $V_{CO}^{EA} = (\rho \times L^n)$

Therefore initial volume of *EaCRS* is:

$$V_{EaCRS} = V_{VL}^{EA} + (n-2) \times V_{CO}^{EA}$$ [Since (n-2) nos. *CO* array exist for *EaCRS* scheme for n dimensional EMA]

$$= \rho L^n + (n-2)\rho L^n$$

$$= (n-1)\rho L^n$$

Now consider EMA(5), with initial volume of the array $V = L^5$ before compression (considering length of each dimension $L_i = L$)

Extending a δ unit along dimension i, the size of extension SE_i^{VL} for VL array is

$SE_1^{VL} = \rho \times \delta \times L_2 \times L_3 \times L_4 \times L_5 = \rho\delta L^4$, and due to extension $L_1 = L + \delta$

$SE_2^{VL} = \rho \times \delta \times L_1 \times L_3 \times L_4 \times L_5 = \rho\delta\,(L+\delta)L^3$, and due to extension $L_2 = L + \delta$

$SE_3^{VL} = \rho \times \delta \times L_1 \times L_2 \times L_4 \times L_5 = \rho\delta\,(L+\delta)^2L^2$, and due to extension $L_3 = L + \delta$

$SE_4^{VL} = \rho \times \delta \times L_1 \times L_2 \times L_3 \times L_5 = \rho\delta(L+\delta)^3L$, and due to extension $L_4 = L + \delta$

$SE_5^{VL} = \rho \times \delta \times L_1 \times L_2 \times L_3 \times L_4 = \rho\delta(L+\delta)^4$, and due to extension $L_5 = L + \delta$

Total Extension Cost for VL array, δ unit extension in each dimension, becomes

$$EC_\delta^{VL} = SE_1 + SE_2 + SE_3 + SE_4 + SE_5$$

$$= \rho\delta \sum_{i=0}^{k} L^{k-i}(L+\delta)^i, \text{ where } k = 4$$

Similarly for EMA(n), total extension cost for VL array, for δ unit extension in each dimension, can be written as

$$EC_\delta^{VL} = SE_1 + SE_2 + SE_3 + \ldots + SE_{n-1} + SE_n$$

$$= \rho\delta \sum_{i=0}^{k} L^{k-i}(L+\delta)^i, \text{ where } k = n-1 \qquad \text{......} (3.15)$$

Expanding the summation, $\sum_{i=0}^{k} L^{k-i}(L+\delta)^i$, we get

$$\sum_{i=0}^{k} L^{k-i}(L+\delta)^i$$

$$= L^k(L+\delta)^0 + L^{k-1}(L+\delta)^1 + L^{k-2}(L+\delta)^2 + \cdots + L^1(L+\delta)^{k-1} + L^0(L+\delta)^k$$

$$
\begin{aligned}
= {} & L^k + \\
& L^{k-1}({}^1C_0 L + {}^1C_1\delta) + \\
& L^{k-2}({}^2C_0 L^2 + {}^2C_1\delta L + {}^2C_2\delta^2) + \\
& L^{k-3}({}^3C_0 L^3 + {}^3C_1\delta L^2 + {}^3C_2\delta^2 L + {}^3C_3\delta^3) + \\
& L^{k-4}({}^4C_0 L^4 + {}^4C_1\delta L^3 + {}^4C_2\delta^2 L^2 + {}^4C_3\delta^3 L + {}^4C_4\delta^4) + \\
& \vdots \\
& + \\
& L^0({}^kC_0 L^k + {}^kC_1\delta L^{k-1} + {}^kC_2\delta^2 L^{k-2} + \ldots\ldots + {}^kC_{k-1}\delta^{k-1} L + {}^kC_k\delta^k)
\end{aligned}
$$

After multiplying and collecting the coefficients of L^p, p = 0, 1, …, k, we get

$$
\sum_{i=0}^{k} L^{k-i}(L+\delta)^i = L^k \sum_{i=0}^{k} {}^iC_0 + L^{k-1}\delta \sum_{i=1}^{k} {}^iC_1 + L^{k-2}\delta^2 \sum_{i=2}^{k} {}^iC_2 + \ldots + L\delta^{k-1} \sum_{i=k-1}^{k} {}^iC_{k-1} + \delta^k \sum_{i=k}^{k} {}^iC_k
$$

$$
= {}^{k+1}C_1 L^k + {}^{k+1}C_2 L^{k-1}\delta + {}^{k+1}C_3 L^{k-2}\delta^2 + \ldots + {}^{k+1}C_k L\delta^{k-1} + {}^{k+1}C_{k+1}\delta^k
$$

$$
\left[\text{Since } \sum_{j=0}^{p} {}^jC_r = {}^{p+1}C_{r+1} \right]
$$

$$
= \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^{i-1}, \ \textit{where } n = k+1
$$

Putting the above value in equation (3.15), we get

$$EC_\delta^{VL} = \rho\delta \sum_{i=0}^{k} L^{k-i}(L+\delta)^i, \text{ where } k = n-1$$

$$= \rho\delta \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^{i-1}$$

$$= \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^{i} \qquad \ldots\ldots\ldots\ldots \textit{(3.16)}$$

Similarly for EMA(n), total extension cost for *CO* array, for δ unit extension in each dimension, can be written as

$EC_\delta^{CO} = (n-2)\,\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$ [Since (*n-2*) nos. CO array exist for each subarray]

So, total extension cost for *EaCRS* is: $EC_\delta^{EaCRS} = EC_\delta^{VL} + EC_\delta^{CO}$

$$= \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i + (n-2)\,\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

$$= (n-1)\,\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

Extension Gain of EaCRS over CRS scheme

The difference of extension cost between the *CRS* and *EaCRS* schemes is referred to as *Extension Gain* ($EG_{n,\delta}^{EaCRS}$) of *EaCRS* over *CRS* scheme

$$EG_{n,\delta}^{EaCRS} = EC_{\delta}^{CRS} - EC_{\delta}^{EaCRS}$$

$$= 2n\rho L^n + n\rho \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i - (n-1)\,\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

$$= 2n\rho L^n + \rho \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i$$

$$= 2V_{CRS} + \text{Extension cost of a single } CO \text{ array of } EaCRS$$

So, $EG_{n,\delta}^{EaCRS}$ is equal to the twice of the initial volume of *CRS* and extension cost for a single *CO* array of *EaCRS* (since *EaCRS* scheme requires one less *CO* auxiliary array for each subarray than the *CRS* scheme). That is the extension gain is constant (more than twice of the initial volume) for any values of δ with a fixed initial volume.

Cost for LEaCRS scheme

Initial volume of *LEaCRS* is:

$$V_{LEaCRS} = V_{VL}^{EA} + V_{CO}^{EA}$$ [Since (*n-2*) nos. *CO* array exist for *EaCRS* scheme for n-dimensional EMA]

$$= \rho L^n + \rho L^n$$

$$= 2\rho L^n$$

In the *LEaCRS* scheme, total extension cost for the *VL* array is same as equation 3.15. In this scheme total extension cost for the *CO* array is: $\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$, since there is only one *CO* array for each subarray.

Therefore, total extension cost for *LEaCRS* is:

$$EC_{\delta}^{LEaCRS} = \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i + \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i = 2\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

Extension Gain of LEaCRS over CRS scheme

The difference of extension cost between the *CRS* and *EaCRS* schemes is referred to as *Extension Gain* ($EG_{n,\delta}^{LEaCRS}$) of *EaCRS* over *CRS* scheme

$$EG_{n,\delta}^{LEaCRS} = EC_{\delta}^{CRS} - EC_{\delta}^{LEaCRS}$$

$$= 2n\rho L^n + n\rho \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i - 2\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

$$= 2n\rho L^n + (n-2)\rho \sum_{i=1}^{n} {}^nC_i\, L^{n-i}\delta^i$$

$= 2V_{CRS}$ + Extension cost of *(n - 2)* nos. *CO* array of *LEaCRS*

So, $EG_{n,\delta}^{LEaCRS}$ is equal to the twice of the initial volume of *CRS* and extension cost for *(n - 2)* nos. *CO* array of *LEaCRS* (since *LEaCRS* scheme requires *(n - 2)* nos. less *CO* auxiliary array for each subarray than the *CRS* scheme). That is the extension gain is constant (more than twice of the initial volume) for any values of δ with a fixed initial volume.

Cost for EaChOff scheme

Consider a EMA(n), with initial volume V = L^n before compression for each dimension length $L_i = L$

Initial volume of $data_{EA}$ array is: $V_{data}^{EA} = V_{VL}^{EA} = (\rho \times L^n)$ [from eqn. *(3.5)*]

Initial volume of the $OffsetInChunk_{EA}$ array is: $V_{OffsetInChunk}^{EA} = V_{CO}^{EA} = (\rho \times L^n)$

[from eqn. *(3.6)*]

Therefore initial volume of *EaChOff* is: $V_{EaChOff} = V_{data}^{EA} + V_{OffsetInChunk}^{EA}$

$$= \rho L^n + \rho L^n$$

$$= 2\rho L^n = V_{ChOff}$$

In the *EaChOff* scheme, total extension cost for the *data* array and *OffsetInChunk* array are same as to the extension cost of *VL* array and *CO* array of the *LEaCRS* scheme respectively.

Therefore, total extension cost for *EaChOff* is:

$$EC_{\delta}^{EaChOff} = \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i + \rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i = 2\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

Extension Gain of EaChOff over Chunk Offset scheme

The difference of extension cost between the *Chunk Offset* and *EaChOff* schemes is referred to as *Extension Gain* ($EG_{n,\delta}^{EaChOff}$) of *EaChOff* over *Chunk Offset* scheme

$$EG_{n,\delta}^{EaChOff} = EC_{\delta}^{ChOff} - EC_{\delta}^{EaChOff}$$

$$= 4\rho L^n + 2\rho \sum_{i=1}^{n} {}^nC_i \, L^{n-i}\delta^i - 2\rho \sum_{i=1}^{n} {}^nC_i L^{n-i}\delta^i$$

$$= 2 \times 2\rho L^n$$

$$= 2V_{ChOff}$$

$$= 2V_{EaChOff}$$

That is the extension gain ($EG_{n,\delta}^{EaChOff}$) is constant (twice of the initial volume) for any values of δ with a fixed initial volume.

3.6 Conclusion

In this chapter we present our proposed schemes in details that are how the multidimensional array can be compressed with the facility of dynamic extension but excluding the already stored data reorganization. We also describe the forward mapping and backward mapping techniques for all the proposed schemes. The analytical analysis of the proposed compression schemes including theoretical analysis of the traditional *CRS* and *Chunk-Offset* schemes are also presented in this chapter. Analytical analysis shows that Extension gain of the proposed *EaCRS* and *LEaCRS* scheme over *CRS* scheme is more than twice of the initial volume of *CRS* and extension gain of *EaChOff* scheme over *ChOff* scheme is exactly twice of the initial volume of *Chunk Offset* for any values of δ with a fixed initial volume. But it is worth mentioning that this gain is in theoretical aspect. Practically, EG would be little less, because there will some cost increase due to populating those auxiliary tables we have used. *ChOff, EaChOff* and *LEaCRS* schemes outperform *CRS* and *EaCRS* schemes in terms of range of usability as well as compression ratio. As *ChOff scheme* is based on TMA it suffers from extendibility problem. Therefore *LEaCRS* and *EaChOff* schemes are more suitable for practical applications with higher values of ρ than the *CRS, ChOff* and *EaCRS* schemes. In the next chapter we will show the details experimental results that confirm the theoretical analysis presented here.

CHAPTER IV

Experimental Analysis

4.1 Experimental Setup

In this chapter, the experimental results for storage and retrieval cost as well as range of usability of both the TMA based schemes (*CRS* and *ChOff*) and EMA based schemes (*EaCRS, LEaCRS, EaChOff*) are analyzed. We simulate the retrieval cost for range key query and extension cost for all the TMA and EMA based schemes. To evaluate the efficiency of the proposed schemes, the schemes were experimented on multidimensional array systems. All lengths or sizes of storage areas are in bytes. For experimental work, all systems are implemented in C++ language (Microsoft Visual Studio 6.0) and are run on a machine (Intel Pentium dual core processor) of 2.7 GHz, 1GB RAM, 4GB virtual memory and as an operating system Windows 7 Ultimate are used. Since execution time of the program is dependent on several system specification parameters like processor speed, size of the primary memory and the number of thread running on the system; so extension cost and data access time may different at different machine.

Table 4.1: The values of the parameters considered for experimental analysis.

L	δ	ρ	α	β	n
$4 \sim 40$	5	$0.10 \sim 0.70$	4	4, 8	3, 4, 5, 6

4.2 Experimental parameters

CRS, ChOff, EaCRS, LEaCRS and *EaChOff* schemes are implemented by placing all the arrays in secondary storage. Among the three auxiliary tables of extendible array, coefficient vector and address table are void for the *EaCRS, LEaCRS* and *EaChOff* schemes and only the history table is required for these schemes. History table acts as an index for locating the subarrays. Thus history tables are stored in main memory for fast access since the sizes of the auxiliary tables are negligible comparing to the main arrays.

Table 4.1 shows the parameter values used for experimental analysis (See Table 3.1 for definitions of the parameters).

4.3 Experimental Results

4.3.1 Comparison of Compression Ratio

Figure 4.1 shows the compression ratio (η) found by experimental results of the TMA based (*CRS* and *ChOff*) schemes and EMA based (*EaCRS, LEaCRS* and *EaChOff*) schemes. It is an important metric to determine the range of usability (see definition 3.1) of the compression schemes. Reorganization of the equations 3.2 and 3.4 give the followings respectively:

$$\eta_{CRS} = \frac{(n-1)\rho\alpha}{\beta} + \frac{(L+1)\alpha}{L^n\beta} + \rho \quad \ldots\ldots \quad (4.1)$$

$$\eta_{ChOff} = \frac{2\alpha}{L^n\beta} + \frac{\rho\alpha}{\beta} + \rho \quad \ldots\ldots \quad (4.2)$$

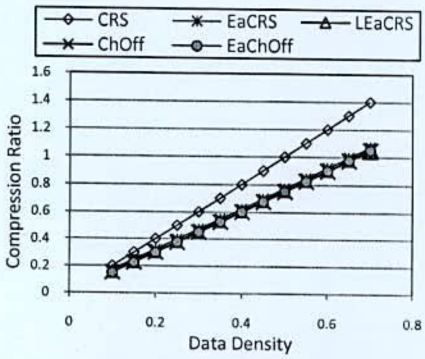
By reorganizing the equations 3.9, 3.11 and 3.14 and using Table 3.2, we have the followings respectively:

$$\eta_{EaCRS} = \frac{(n-2)\rho\alpha}{\beta} + \frac{(nL-n+2)\alpha}{2L^{n-1}\beta} + \rho \quad \ldots\ldots \quad (4.3)$$

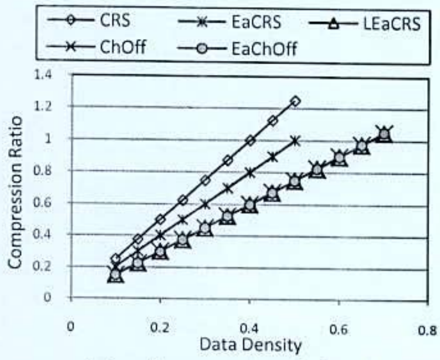
$$\eta_{LEaCRS} = \frac{\rho\alpha}{\beta} + \frac{(nL-n+1)\alpha}{L^n\beta} + \rho \quad \ldots\ldots \quad (4.4)$$

$$\eta_{EaChOff} = \frac{\rho\alpha}{\beta} + \frac{(2n(L-1)+1)\alpha}{L^n\beta} + \rho \quad \ldots\ldots \quad (4.5)$$

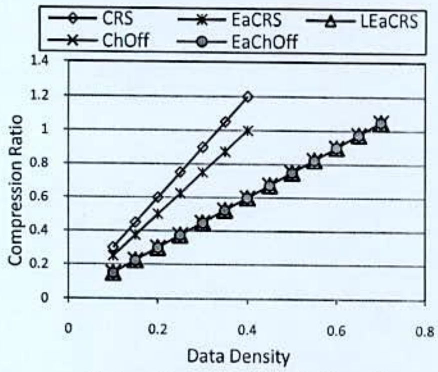
Figure 4.1(a), (b) and (c) shows the experimental results for $\beta = 8$ and varying ρ and $n = 3$, $n = 4$ and $n = 5$ respectively. It is found that η increases with the increase of ρ. This is because; from the above cost analysis (see equation 4.1, 4.2, 4.3, 4.4, 4.5), we found that η is directly proportional to the value of ρ for a constant value of n, L, α and β. In Figure 4.1(a), for $n = 3$; η_{CRS} crosses the value 1 at an approximate $\rho = 0.50$ but η_{ChOff}, η_{EaCRS}, η_{LEaCRS}, and $\eta_{EaChOff}$ cross the value 1 at an approximate $\rho = 0.66$. In Figure 4.1(b), for $n = 4$; η_{CRS} and η_{EaCRS} crosses the value 1 at an approximate $\rho = 0.40$ and $\rho = 0.50$ respectively but η_{ChOff}, η_{LEaRS}, and $\eta_{EaChOff}$ cross the value 1 at an approximate $\rho = 0.66$. In Figure 4.1(c), for $n = 5$; η_{CRS} and η_{EaCRS} crosses the value 1 at an approximate $\rho = 0.33$ and $\rho = 0.40$ respectively but η_{ChOff}, η_{LEaCRS}, and $\eta_{EaChOff}$ cross the value 1 at an approximate $\rho = 0.66$.



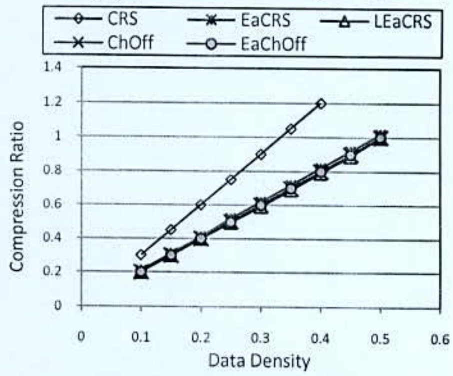
(a) η for $n = 3$ and $\beta = 8$.



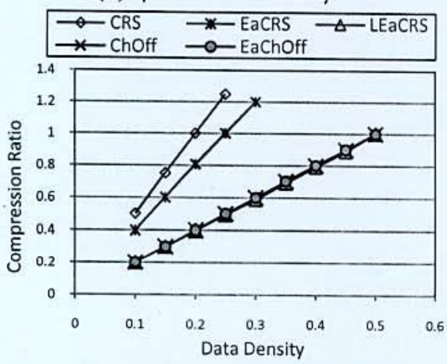
(b) η for $n = 4$ and $\beta = 8$.



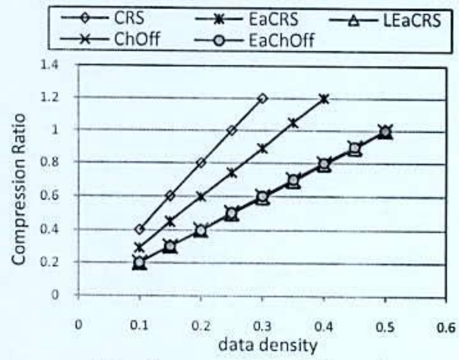
(c) η for $n = 5$ and $\beta = 8$.



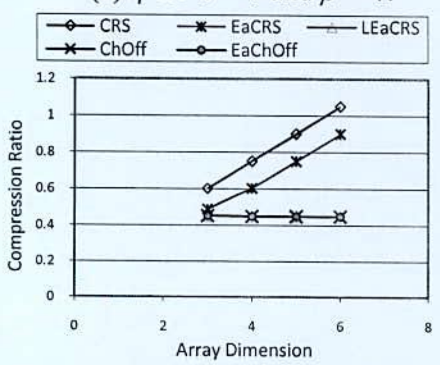
(d) η for $n = 3$ and $\beta = 4$.



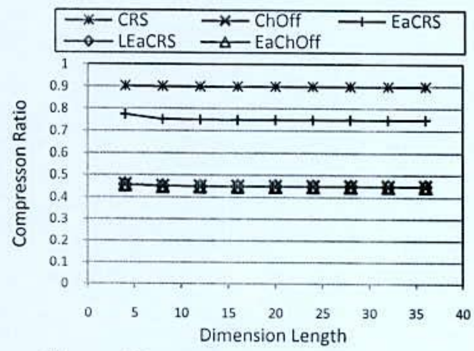
(e) η for $n = 4$ and $\beta = 4$.



(f) η for $n = 5$ and $\beta = 4$.



(g) η with constant ρ and varying n.



(h) η with constant ρ and varying L.

Figure 4.1: Comparison of compression ratio for *CRS, EaCRS, LEaCRS, ChOff* and *EaChOff* schemes.

In all the cases, *ChOff, LEaCRS* and *EaChOff* outperform *CRS* and *EaCRS* schemes for compression ratio as well as range of usability. This is because *CRS* scheme requires *n* auxiliary arrays for *n* dimensional sparse array and *EaCRS* scheme requires *n-1* auxiliary arrays for the same sparse array but *ChOff, LEaCRS* and *EaChOff* scheme requires only 2 auxiliary arrays for any dimensional sparse array.

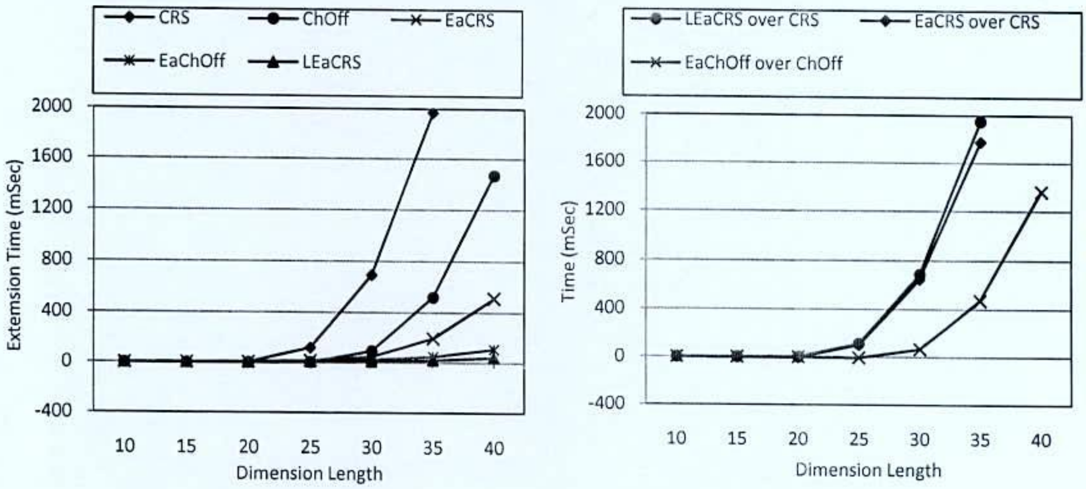
Figure 4.1(d), 4.1(e) and 4.1(f) shows the experimental results for $\beta = 4$ and varying ρ and $n = 3$, $n = 4$ and $n = 5$ respectively. It is found that, in all the cases η_{CRS}, η_{ChOff}, η_{EaCRS}, η_{LEaCRS}, and $\eta_{EaChOff}$ crosses the value 1 for lower value of ρ with respect to the value of ρ in figure 4.1(a), 4.1(b) and 4.1(c). This is because; from the above cost analysis (see equation 4.1, 4.2, 4.3, 4.4, 4.5), we found that η is inversely proportional to the value of β for a constant value of n, L, α and ρ.

Figure 4.1(g) shows the range of usability comparison among *CRS, ChOff, EaCRS, LEaCRS* and *EaChOff* schemes for $\rho = 0.30$. The tests were conducted for various values of n (3 ~ 6) and $\beta = 8$. η_{CRS} and η_{EaCRS} increases with the increase of n, but η_{ChOff}, η_{LEaCRS}, and $\eta_{EaChOff}$ remains approximately same for all the cases. This is because; from equation 4.1 and 4.3 we found that, $\eta_{CRS} \approx (n-1)$ and $\eta_{EaCRS} \approx (n-2)$; considering values of n, L, α and β constant and we can ignore the second term (see equation 4.1 and 4.3) of both the equation for large values of L. On the other hand n has no effect on η_{ChOff} (see equation 4.2) and n has very small effect on η_{LEaCRS}, and $\eta_{EaChOff}$ (see equation 4.4 and 4.5) for large values of L; since we can ignore the second term of the equation 4.4 and 4.5 for large values of L. Hence range of usability of *CRS* and *EaCRS* schemes decreases with the increase of n, but remains almost constant for *ChOff, LEaCRS and EaChOff* schemes for any dimensional sparse array as explained in Chapter III.

Figure 4.1(h) shows the test results of the space requirement of the *CRS, ChOff, EaCRS, LEaCRS* and *EaChOff* schemes for varying L. The tests were conducted for $n = 5$, $\beta = 8$ and $\rho = 0.3$. From Figure 4.1(h) we can see that L has no effect on η for all the schemes, which validate the above cost analysis (see equation 4.1, 4.2, 4.3, 4.4 and 4.5).

4.3.2 Extension Cost

Figure 4.2(a) shows the extension cost for *CRS, ChOff, EaCRS, LEaCRS* and *EaChOff* schemes. The TMA based schemes (both *CRS* and *ChOff*) reorganizes the array whenever there is an extension to it. The TMA based schemes need to fetch the existing elements then reorganize for the extension. On the other hand the EMA based schemes namely *EaCRS*, *LEaCRS* and *EaChOff* schemes extend the initial array with segment of subarrays containing the new data as described in chapter III. Hence the EMA based schemes can reduce the cost of array extensions significantly. In figure 4.2(a), the extension times are shown with $n = 5$, $\rho = 0.3$, $\beta = 8$ and $\delta = 5$, where we find the extension times for TMA based compression schemes are much higher than the EMA based compression schemes. Figure 4.2(b) shows the extension gain i.e. the extension time difference between the *EaCRS* and *CRS*, *LEaCRS* and *CRS* and *EaChOff* and *ChOff* schemes.



(a) Extension cost

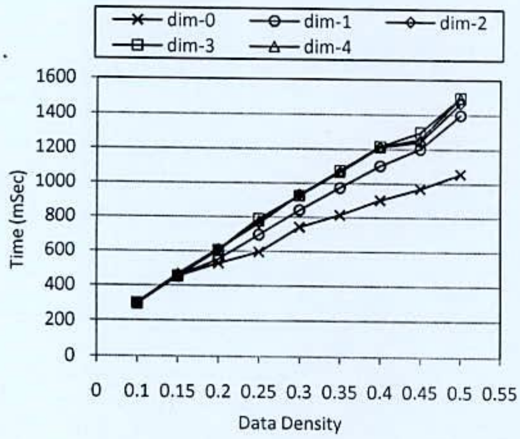
(b) Extension gain

Figure 4.2: Extension cost and Extension gain comparison of *CRS*, *EaCRS*, *LEaCRS*, *ChOff* and *EaChOff* schemes for $n = 5$, $\rho = 0.3$, $\beta = 8, \delta = 5$ for varying L.

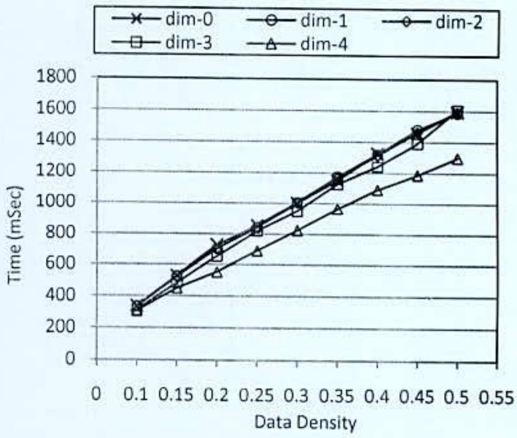
The extension cost as well as extension gain depends on the initial volume of the array i.e. the values of n and L before the array is extended. Hence, if n and L increase, then EMA based schemes need less data to store than TMA based schemes without any reorganization of data. So TMA based schemes need higher times than EMA based schemes and thus gain increases. We can conclude that if the initial volume is large then the extension cost for TMA based schemes are higher.

4.3.3 Retrieval Cost

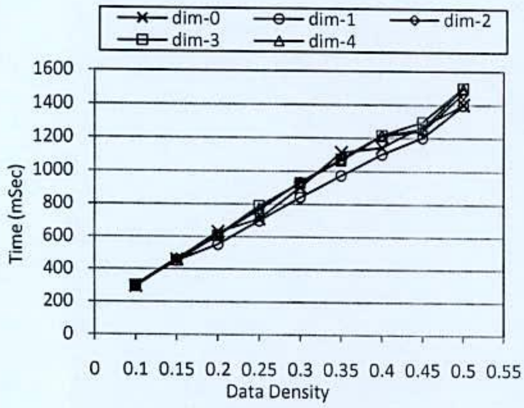
Figure 4.3 shows the retrieval performance for range key query of TMA and EMA based compression schemes for $n=5$, $L=30$ with different density and the query ranges from dimension length 7 to dimension length 21 of the array for the tests.



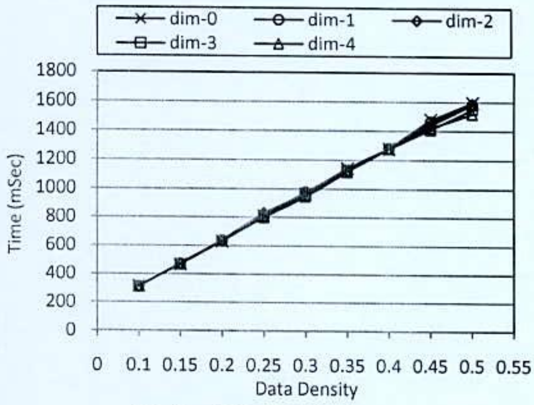
(a) for *CRS* scheme.



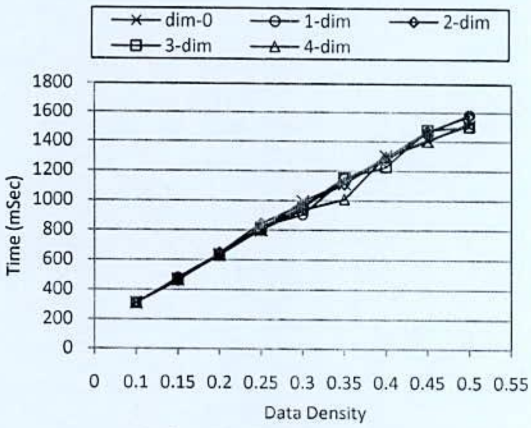
(b) for *EaCRS* scheme.



(c) for *ChOff* scheme.



(d) for *LEaCRS* scheme.



(e) for *EaChOff* scheme.

Figure 4.3: Retrieval cost analysis for *CRS*, *EaCRS*, *LEaCRS*, *ChOff* and *EaChOff* schemes for different known dimensions.

In Figure 4.3(a) the retrieval performance for *CRS* scheme for different known dimension is shown. It shows that, the retrieval time is lower for dimension-0. This is because the element inside the TMA can be organized as row major order or column major order. If the elements are organized in one order (say row major) and it is searched in the same dimension; the target elements for the query are consecutively organized. This is not true for all other dimensions and therefore that dimensions take longer times. Similarly Figure 4.3(b) shows that the retrieval time is lower for dimension-4 for *EaCRS* scheme. This is because the subarrays of EMA(n) are *n-1* dimensional; the elements inside the subarrays again can be organized as row major order or column major order. Hence for *EaCRS* scheme, the same situation occurs i.e. for one known dimension *EaCRS* takes lower time than others as shown in Figure 4.3(b). Figure 4.3(c), 4.3(d) and 4.3(e) show the retrieval performance for *ChOff*, *LEaCRS* and *EaChOff* schemes respectively for different known dimensions. In all the cases, retrieval time is almost same for different known dimensions. This is because, in these compression schemes; the array is linearized in a single data stream using the addressing function; therefore all the offset values of the array elements are considered as a single row. Hence the range of candidate offset values for a query can be determined uniquely.

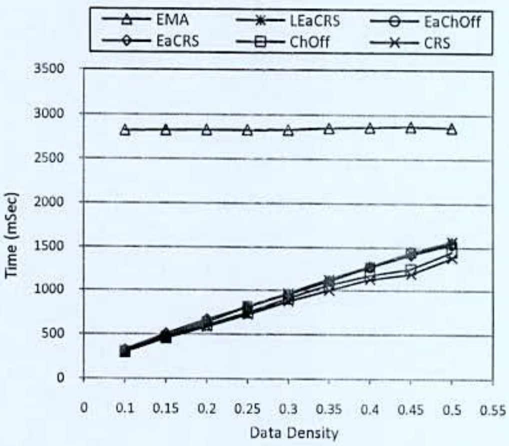


Figure 4.4: Comparison of Average retrieval time for *CRS*, *EaCRS*, *LEaCRS*, *ChOff* and *EaChOff* schemes for different dimension.

Figure 4.4 shows the average range key retrieval time of both compression schemes and uncompressed EMA with different density for $n = 5$, $L = 30$ and $\beta = 8$. Retrieval is made for the dimension length 7 – 21, considering each dimension as known dimension and then averaged. From Figure 4.3, we find that retrieval time increases linearly with the increase of data density for all the compression schemes (*CRS*, *ChOff*, *EaCRS*, *LEaCRS* and *EaChOff*). This is because for an n-dimensional array with a particular length L and density ρ the number of non empty cell is ρL^n. So if ρ changes the total number changes linearly and hence the retrieval time. However there is no effect of data density on the retrieval time of uncompressed EMA. The reason is, in uncompressed EMA whatever the density, the sizes of subarrays remain same, and hence retrieval time is constant.

4.4 Discussion

In this chapter we present the experimental outcomes of the proposed scheme. We compare space requirement and range of usability of the *EaCRS*, *LEaCRS* and *EaChOff* schemes with that of *CRS and ChOff* schemes on TMA. Retrieval time of the *CRS*, *ChOff*, *EaChOff*, *EaCRS* and *LEaCRS* schemes are examined and compared with the retrieval time of the EMA. In each case we found relevancy with the theoretical analysis what we made in Chapter III. Furthermore we find that, proposed compression schemes outperform TMA based compression schemes for extension operation.

CHAPTER V

Conclusion

5.1 Concluding Remarks

The amount of information stored and analyzed in modern data sciences are very large. Since they can be very large; must be stored and retrieved from disk in costly I/O operations. So, many scientific applications extensively use multidimensional array to represent their data for efficient processing. However in many cases the total number of data or dimension cannot be predicted beforehand. Besides this, representing the real world data in multidimensional array creates a very sparse array. Compressing the data has important advantages. The most obvious advantages are the consequences of the smaller space usage. In this research work, we managed both sparsity and the dynamic extension problem by presenting database compression schemes based on EMA. We propose three new compression schemes namely *EaCRS*, *LEaCRS* and *EaChOff* for multidimensional array representation. Since *EaCRS*, *LEaCRS* and *EaChOff* schemes are based on an extendible multidimensional array system and compression scheme is applied for each subarray independently, such an array can extend its size dynamically along an arbitrary dimension without any relocation of existing data. We evaluated the proposed compression schemes both analytically and experimentally. In all the cases experimental results confirm the theoretical model. Hence the analytical model is validated. Again we compared the proposed schemes with TMA based compression schemes namely *CRS* and *ChOff* and found better results for the proposed schemes.

5.2 Future Recommendations

The future applications and recommendations can be summarized as follows

- The proposed schemes can easily be implemented in parallel platform. Because the subarrays of the extendible array are independent to each other, the subarrays can be distributed among the processors [48] and hence *EaCRS*, *LEaCRS* and *EaChOff* schemes can be applied over the subarrays in parallel. Hence it will be very efficient to apply these schemes in parallel and multiprocessor environment.

- The schemes can be applied to implement the compressed form of MOLAP server and data warehouses. As the extension occurs incrementally for EMA and the proposed schemes are based on EMA. *EaCRS*, *LEaCRS* and *EaChOff* schemes can efficiently be applied for incremental aggregation i.e is form of velocity for big data analysis. Hence it is applicable for big data analytics.
- The scheme can be applied to multidimensional database implementations using usual RDBMS for multidimensional data analysis.

REFERENCES

1. Pedro Furtado and Henrique Madeira, 2000, "Data Cube Compression with QuantiCubes", DaWaK 2000, LNCS 1874, pp. 162–167.

2. D. Chatziantonian and K. Ross, 1996, "Querying Multiple Features in Relational Databases", Proc. of 22nd International Conf. Very Large Databases, pp. 295-306.

3. M. A. Roth and S.J. Van Horn, 1993, "Database Compression", SIGMOD Record, vol. 22, no. 3, pp. 19-29.

4. J. Ziv, Lempel, 1977, "A Universal algorithm for sequential data compression", IEEE Transactions on Information Theory, Volume 23, N° 3, pp. 337-343.

5. Welsh, Terry, June 1984, "A Technique for High-Performance Data Compression" IEEE Computer, Volume 17, N° 6, pp. 8-19.

6. M. Nelson, J-L Gaily, "The Data Compression Book", 2nd edition, 1996 - M&T Books, ISBN 1-55851-434-1.

7. M.A. Bassiouni, 1985, "Data Compression in Scientific and Statistical Databases", IEEE Trans. Software Eng., vol. 11, no. 10, pp. 1047-1058.

8. Sarawagi, S. and Stonebraker, M.,1994, "Efficient Organization of Large multidimensional Arrays", Proc. of 10th International Conference on Data Engineering, pp. 328–336, Houston, TX , USA.

9. 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 Parallel and Distributed Systems, 14(7), pp. 625-639.

10. Manuel Ujaldon, Emilio L. Zapata, Shamik D. Sharma, and Joel Saltz, 1996, "Parallelization Techniques for Sparse Matrix Applications," Journal of parallel and distribution computing.

11. J.K. Cullum and R.A. Willoughby, 1985, "Algorithms for Large Symmetric Eigen value Computations," vol. 1.

12. G.H. Golub and C.F. Van Loan, 1989, Matrix Computations, 2nd ed. (Johns Hopkins Univ.Press, Baltimore).

13. Li, J. and Srivastava, J., 2002, "Efficient Aggregation Algorithms for Compressed Data Warehouses", IEEE Transaction on Knowledge and Data Engineering, Vol. 14, No. 3, pp. 515–529.

14. White J. B. and Sadayappan P., 1997, "On Improving the Performance of Sparse Matrixvector Multiplication", Proc. of International Conference on High Performance Computing, pp. 711–725.

15. H. Kang and C. Chung, 2002, "Exploiting versions for On-line data warehouse maintenance in MOLAP servers", Proc. of VLDB, pp.742-753.

16. Acker, R., Pieringer, R. and Bayer, R., 2005, "Towards Truly Extensible Database Systems", Proc. of DEXA, LNCS, Vol. 3588, pp. 596–605.

17. Hasan, K.M.A., Azuma, M.N., Tsuji, T., and Higuchi, K., 2005, "An Extendible Array Based Implementation of Relational Tables for Multidimensional Databases", Proc. of DaWak, LNCS, Vol. 3580, pp. 233–242.

18. Otoo, E. J. and Merrett, T.H., 1983, "A Storage Scheme for Extendible Arrays", Computing, Vol. 31, pp. 1–9.

19. K. M. Azharul Hasan, T. Tsuji, and K. Higuchi, 2007, "An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation", Proc. of DASFAA , LNCS 4443, pp. 288 – 299.

20. G. Colliat, 1996, "OLAP, Relational and Multidimensional Databases Systems", SIGMOD Record, vol. 25, no. 3.

21. Kumakiri, M., Bei, L., Tsuji, T. and Higuchi, K., 2006, "Flexibly Resizable Multidimensional Arrays", Proc. of 22nd International Conference on Data Engineering Workshops, pp. 83–88.

22. Zhao, Y., Deshpande, P.M. and Naughton, J. F., 1997, "An Array Based Algorithm for Simultaneous Multidimensional Aggregates", ACM SIGMOD, pp. 159–170.

23. Barret R., Berry M., Chan T.F., Dongara J., Eljkhhout V., Pozo R., Romine C. and Van H., 1994, "Templates for the Solution of Linear Systems: Building Blocks for the Iterative Methods", SIAM, 2nd. ed.

24. Tsuji, T., Hara, A. and Higuchi, K., 2006, "An Extendible Multidimensional Array System for MOLAP", SAC'06 April pp. 23–27.

25. Shimada, T., Fang, T., Tsuji, T. and Higuchi, K., 2006, "Containerization Algorithms for Multidimensional Arrays", Asia Simulation Conference, pp. 228–232.

26. Tsuji, T., Jin, D. and Higuchi, K., 2008, "Data Compression for Incremental Data Cube Maintenance", proc. of DASFAA, LNCS, Vol. 4947, pp. 682–685.

27. T.Tsuji, G.Mizuno, T.Hochin, K.Higuchi, 2003, "A Deferred Allocation Scheme of Extendible Arrays", Transaction of IEICE, Vol.J86-D-I, pp. 351-356.

28. Rosenberg, A.L., 1974, "Allocating Storage for Extendible Arrays". Journal of the ACM (JACM), Vol. 21, pp. 652–670.

29. Rosenberg, L. and Stockmeyer, L. J., 1977, "Hashing Schemes for Extendible Arrays", JACM, Vol. 24, pp.199–221.

30. P. Vassiliadis, 1998, "Modeling multidimensional databases, Cubes and Cube Operations", Proc. of SSDBM, pp. 53-62.

31. Pedersen, T. B. and Jensen, C. S., 2001, "Multidimensional Database Technology", IEEE Computer, Vol. 34, No.12, pp. 40–46.

32. Rotem, D. and Zhao, J.L., 1996, "Extendible Arrays for Statistical Databases and OLAP Applications", Proc. of 8th International Conference on SSDBM, pp. 108–117, Stockholm, Sweden.

33. K. E. Seamons and M. Winslett, 1994, "Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications", Proc. of 7th International Conference on Scientific and Statistical Database Management (SSDBM), pp. 218–227, IEEE CS, Washington, DC, USA.

34. T. Tsuji, M. Kuroda, and K. Higuchi, 2008, "History offset implementation scheme for large scale multidimensional data sets," Proc. of ACM Symposium on Applied Computing, pp. 1021-1028.

35. Sk. Md. Masudul Ahsan, "An Efficient Implementation Scheme for Multidimensional Index Array Operations and Its Evaluation", A Thesis - submitted to Computer Science and Engineering Department, Khulna University of Engineering and Technology, CSER-M-12-01, January, 2012.

36. Sk. Md. Masudul Ahsan and K. M. A. Hasan, 2013, "Extendible Multidimensional Array Based Storage Scheme for Efficient Management of High Dimensional Data," International Journal of Next-Generation Computing, Vol 4, No 1, pp. 88-105.

37. Sk. Md. Masudul Ahsan and K. M. Azharul Hasan, 2013"An Efficient Encoding Scheme to Handle the Address Space Overflow for Large Multidimensional Arrays", Journal of Computers, Vol 8, No 5, pp. 1136–1144.

38. Halder, A.K., 2005, "Karnaugh map extended to six or more variables", Electronics Letters, Vol. 18, No. 20, pp. 868–870.

39. Holder, M.E., 2005, "A modified Karnaugh map technique ", IEEE Transactions on Education, Vol. 48, No. 1, pp. 206–207.

40. Chun-Yuan Lin, Yeh-Ching Chung, Jen-Shiuh Liu, December 2003, "Efficient Data Compression Methods for Multidimensional Sparse Array Operations Based on the EKMR Scheme," IEEE Transactions on Computers, Vol. 52, No. 12, pp.1640-1646.

41. Chun, Y. L., Jen, S.L. and Yeh, C.C., 2002, "Efficient Representation Scheme for Multidimensional Array Operations," IEEE Transactions on Computers, Vol. 51, No. 3, pp. 327–354.

42. J.B. White and P. Sadayappan, 1997, "On improving the performance of sparse matrix vector multiplication", Proc. of Int'l Conf. High Performance Computing, pp. 711-725.

43. J. Li, D. Rottem and H.K. Wong, 1987, "A New Compression Method with Fast Searching on Large Databases", Proc. of 13th int'l conference on Very large databases, pp. 311-318.

44. S. Eggers and A. Shoshani, 1980, "Efficient Access of Compressed Data", Proc. of sixth int'l conference on Very large Databases, pp. 205-211.

45. K. M. Azharul Hasan, 2009, "Compression Schemes of High Dimensional Data for MOLAP", In the Edited Book, "Evolving Application Domains of Data Warehousing and Mining: Trends and Solutions" Chapter IV, Information Science Reference, pp. 64-81.

46. Sk. Md. Masudul Ahsan, K. M. Azharul Hasan, 2013 "An Implementation Scheme for Multidimensional Extendable Array Operations and Its Evaluation", Proc. of ICIEIS 2011, pp. 136–150, Springer-Verlag.

47. Li, B., Tsuji, T. and Higuchi, K., 2007, "Sharing Flexibly Resizable Multidimensional Arrays in Client/Server Environment" Proc. of the International Workshop on Databases for Next Generation Researchers, pp. 19–24, Istanbul.

48. T.Tsuji, H.Kawahar, K.Higuchi, T.Hochin, 2001, "Sharing Extendible Arrays in a Distributed Environment", Proc. of IICS, LNCS, 2060, pp. 41-53.