

# WOLAP: Wavelet-Based Range Aggregate Query Processing

Mehrdad Jahangiri and Cyrus Shahabi

**Abstract**—The Discrete Wavelet Transform has emerged as an elegant tool for data analysis queries. It was not until the time we proposed a new wavelet technique, ProPolyne, for fast exact, approximate, or progressive polynomial aggregate query processing that data did not have to be compressed, unlike most of the prior studies in this area. In this paper, after reviewing our ProPolyne technique in more depth with more intuitive and practical discussions, we address its inefficiency in dealing with scientific datasets due to the cube sparseness, subsequently, we propose a new cube model, CFM, to enhance ProPolyne’s both space and query efficiency. While ProPolyne assumed storing the data as large data frequency distribution cubes, CFM organizes the data as a collection of smaller fixed measure cubes to reduce the overall query and storage costs. We combine both cube models in an integrated framework, called WOLAP, for efficient polynomial aggregate query processing. We further enhance WOLAP by proposing practical solutions for real-world deployment in scientific applications. In particular, we show how to incorporate data approximation, how to improve wavelet filter selection, and how to work on datacubes with arbitrary domain sizes.

**Index Terms**—Aggregate Query, OLAP, Wavelet Transform, Data Compression, Query Approximation, Progressive Query, Scientific Data Analysis.

## I. INTRODUCTION

WITH the current advancements of sensing technology and high capacity of new storage media, massive datasets are regularly collected and stored in various fields. It is crucial to effectively process data analysis queries on these datasets to expedite extraction of useful information for better decision makings. Thus, range aggregate queries, as the most frequent data analysis queries, need to be processed in an efficient manner.

Discrete Wavelet Transform has emerged as a favorable tool for range aggregate query processing on multidimensional datasets. However, most of the methods using DWT share the disadvantage of providing only approximate answers by compressing the data. The efficacy of these approaches is highly data dependent, that is, it only works when the data have a concise wavelet approximation. In addition, these methods suffer from query-blind compression as these approaches mostly decompress the data first to process the query. The main pitfall here is that saving space by itself does not necessarily reduce the query complexity.

In the past, we proposed a query processing technique, ProPolyne [1], where data did not have to be compressed.

The authors are with the University of Southern California, Department of Computer Science, Information Laboratory (InfoLab), Los Angeles, CA 90089, USA. Emails: {jahangir,shahabi}@usc.edu. Phone/Fax: 1-213-8211462.

Instead, we employed wavelet transform to compact incoming queries rather than the underlying data. This resulted in reducing the query cost from being a function of the range size to a function of the logarithm of the data size without sacrificing the update cost. In addition, ProPolyne provided data-independent query approximations after a small number of I/Os by only using the most significant query wavelet coefficients. This approach naturally resulted in a progressive algorithm.

Unlike former range aggregate techniques with limited support only for typical aggregate queries, ProPolyne enables efficient processing of a variety of polynomial queries. Toward this end, ProPolyne utilizes the wavelet transform of data frequency distribution, known as *DFD*, to form and process such queries. However, during the last five years of deploying this methodology in various real-world scientific applications, we have experienced that preparing data frequency cubes is neither feasible nor efficient with most scientific datasets due to their sparsity.

To address the impracticality of *DFD*, we propose a new cube model, *Collection of Fixed Measures (CFM)*, for polynomial query processing on scientific datasets. While prior work assumed storing the data as one large data frequency distribution cube, the new model organizes the data as a collection of smaller fixed measure cubes to enhance both space and query efficiency of ProPolyne in scientific applications. We analytically and empirically compare the two models in this paper and show that *CFM* significantly outperforms *DFD* on real-world scientific datasets.

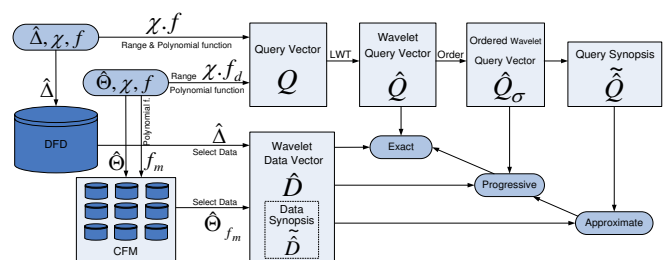


Fig. 1. WOLAP framework

We integrate both models into a single framework, named WOLAP: Wavelet On-Line Analytical Processing, for efficient support of polynomial range-aggregate query processing. Figure 1 illustrates the components of this framework and provides a reference to the terminologies we use throughout the paper. Let us briefly describe the framework here. Given a polynomial range-aggregate query, WOLAP forms a query

vector and efficiently wavelet transform it on-the-fly. Using either of the cube models, WOLAP selects the appropriate cube as the data vector and provides the exact query answer by computing the dot product of the two vectors. WOLAP can also provide an approximate query result by simply using a subset of data and/or query coefficients. By extending and generalizing the query subset selection, we enable progressive query processing at no additional cost. In addition to presenting all these components in this paper, we fully describe how to efficiently transform a multidimensional query on-the-fly. We also show how to incorporate compressed data into WOLAP and still benefit from progressiveness of WOLAP.

Toward realization of WOLAP as a practical framework, we relax two common assumptions of most wavelet techniques. First, we relax the unrealistic assumption that domain of each attribute must be in power of two. We propose a method that allows WOLAP to work with arbitrary domain sizes by padding enough auxiliary coefficients in both data and query vectors. Second, we relax the assumption of using a single filter for multidimensional transformation by proposing multiple 1-dimensional transformations for each dimension with a specific filter. The intuition behind this comes from the fact that we can separate a multidimensional aggregation to multiple 1-dimensional aggregations and associate each aggregation with the corresponding transformation. This allows us to treat each dimension differently and this is very beneficial if attributes are subjected to unequal polynomial degrees.

We have designed and developed a real system [2] under the WOLAP framework and verified the concepts reviewed in this paper. Using this system, we have conducted several experiments with four real-world datasets. The result of our experiments demonstrates the effectiveness of WOLAP in practice.

We begin our discussion with reviewing the related work and the wavelet preliminaries (Sections II and III). Then, we define the polynomial range aggregate query and show how to utilize wavelets to perform this type of queries (see Section IV). In Section V, we propose an efficient algorithm for query transformation and then we analyze the complexity of wavelet query processing. Next in Section VI, we address the inefficiency of the former data model and propose a practical model and analytically compare the two. We extend our query framework by providing approximate and progressive query processing in Section VII. In Section VIII we address and relax two unrealistic assumptions about wavelets to further enhance WOLAP in real-world systems. Finally, we extensively examine our framework with real-world datasets in Section IX and we conclude our discussion in Section X.

## II. RELATED WORK

Extensive research has been done to find efficient ways to evaluate range aggregate queries. The prefix-sum method [3] publicized the fact that careful pre-aggregation can be used to evaluate range aggregate queries in time independent of the range size. This led to a number of new techniques that provide similar benefits with different query/update cost tradeoffs [4]–[7]. Iterative Data Cubes [8] generalize all of these techniques and allow different methods to be used on each dimension.

Approximate query answering can be used to deliver fast query results. Use of synopsis data structures such as histograms has been studied thoroughly [9], [10]. References [11]–[13] create synopses that estimate the data frequency distribution. The flexible measures supported by this approach inspire its use in this paper. Online aggregation, or progressive query answering, has also been proposed [14]–[17]. These methods, whether based on sampling, R-trees, or multiresolution analysis, share a common strategy: answer queries quickly using a low resolution view of the data, and progressively refine the answer while building a sharper view. WOLAP is fundamentally different. It makes optimal progressive estimates of the query, not the data. Furthermore, the progressive evaluation comes for free since WOLAP is an excellent exact algorithm. We have extensively studied progressive query processing in [18].

Wavelet Transform has emerged as a powerful tool for approximate answering of aggregate [17], [19], [20] and relational algebra [21] queries. Streaming algorithms for approximate population of a wavelet database are also available [22]–[24]. Most of the wavelet query evaluation studies have focused on using wavelets to compress the underlying data, reducing the size of the problem. WOLAP not only can use compressed data but also is designed to work as an exact algorithm with uncompressed data. We provide approximate query results by compressing queries, not data.

In [24] we have introduced two wavelet operations for efficient maintenance of wavelet-transformed data. In particular, we study four common data maintenance scenarios, i.e., wavelet transformation of massive datasets, appending data to already wavelet-transformed data, approximation of multidimensional data streams using wavelets and partial data reconstruction from wavelet-transformed data. While the focus of [24] has been on the maintenance of wavelet-transformed data, here we provide the essential techniques to perform polynomial queries on wavelet-transformed data. We believe these two parallel studies are essential toward building an efficient query and storage system.

## III. PRELIMINARIES

In this section, we overview the preliminary concepts of Wavelet Transform that we use throughout this paper.

### A. Discrete Wavelet Transform

Discrete Wavelet transform is defined as a series of decompositions on data by creating “rough” and “smooth” views of the data at different resolutions. The “smooth” view consists of averages or summary coefficients, whereas the “rough” view consists of differences or detail coefficients. At each resolution, termed level of decomposition, the summaries and details are constructed by pairwise averaging and differencing of the summaries of the previous level.

More formally, the summary of the data is produced by a low-pass filter  $H$ , which filters out the rough elements. On the other hand, the detail view of the data is produced by a high-pass filter  $G$ , which filters out the smooth elements. A filter is simply comprised by a set of coefficients that

perform the convolution-decomposition on the input to produce the output. Let  $H = \{h_0, \dots, h_{l-1}\}$  and  $G = \{g_0, \dots, g_{l-1}\}$  be the wavelet filters of length  $l$  and let  $u_{k,j}$  and  $w_{k,j}$  be the  $j$ -th summary and the  $j$ -th detail coefficient for the  $k$ -th level of decomposition. We have:

$$u_{k+1,i} = \sum_{j=0}^{N/2^k-1} h_{j-2i} u_{k,j} \quad w_{k+1,i} = \sum_{j=0}^{N/2^k-1} g_{j-2i} u_{k,j}$$

DWT is performed by chaining these filters on the output of the low-pass filter; doing so recursively leads to the multiresolution view of the data. Figure 2 illustrates the discrete wavelet transform of a vector  $\mathbf{a}$ . It shows that the 3-level wavelet decomposition of  $\mathbf{a}$  consists of the 3-rd level summary coefficients and the detail coefficients across all levels of decomposition. We denote that vector  $\hat{\mathbf{a}}$  is the Discrete Wavelet Transform of  $\mathbf{a}$  by  $\hat{\mathbf{a}} = \text{DWT}(\mathbf{a})$ . Note that the untransformed vector  $\mathbf{a}$  contains the summary coefficients at the 0-th level of decomposition:  $\mathbf{a}[j] = u_{0,j}$ , for  $0 \leq j \leq N$ .

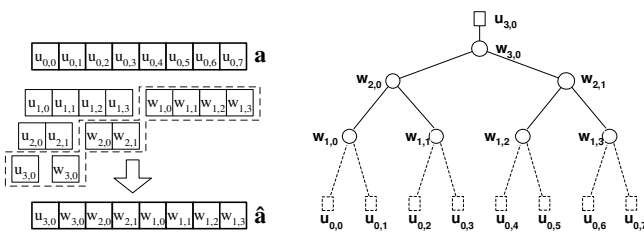


Fig. 2. Discrete Wavelet Transform and Wavelet Tree

In the case of Haar filter as the simplest and the first discovered wavelet filter, the low-pass filter is comprised of the coefficients  $\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}$  and the high-pass filter consists of the coefficients  $\{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\}$ . Let us show how to use this filter to wavelet transform an array by providing an example.

*Example 3.1:* Consider a vector of 4 values  $\{2, 6, 7, 1\}$ . Let us apply DWT on this vector using Haar filter. We first start by taking the pairwise summaries:  $\{\frac{2+6}{\sqrt{2}}, \frac{7+1}{\sqrt{2}}\}$  and the pairwise details  $\{\frac{2-6}{\sqrt{2}}, \frac{7-1}{\sqrt{2}}\}$ . The result is 2 vectors each half the size of original, containing a smoother version of the data, the summaries, and a rougher version, the details; these coefficients form the first level of decomposition,  $\{4\sqrt{2}, 4\sqrt{2}, -2\sqrt{2}, 3\sqrt{2}\}$ . We continue by constructing the summary and detail coefficients from the smooth version of the data:  $\{4\sqrt{2}, 4\sqrt{2}\}$ . The new summary is  $\{8\}$  and the new detail is  $\{0\}$ , forming the second and last level of decomposition. Notice that 8 is the weighted average of the entire vector as it is produced by recursively averaging the summaries. Similarly, 0 represents the weighted difference between the summary of the first half of the vector and the summary of the second half. The final summary and the differences produced at all levels of decomposition form the Haar transform:  $\{8, 0, -2\sqrt{2}, 3\sqrt{2}\}$ . Notice that at each level of decomposition the summaries and details can be used to reconstruct the averages of the previous level.

In order to ensure that  $H$  and  $G$  act as ‘‘summary’’ and ‘‘detail’’ filters,  $h$  and  $g$  coefficients require to have the

following properties:

$$\begin{aligned} g_i &= (-1)^i h_i && \text{Mirror Relationship} \\ \sum_{i=0}^{l-1} h_i &= \sqrt{2} && \text{Normality} \\ \sum_{i=0}^{l-1} h_i h_{i+2j} &= \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{if } j \neq 0 \end{cases} && \text{Orthogonality} \\ \sum_{i=0}^{l-1} i^\nu g_i &= 0 && \nu \text{ Vanishing Moment} \end{aligned}$$

The above conditions imply that these filters are orthonormal bases for transformation and consequently preserve dot product as the following lemma states.

*Lemma 3.2:* If  $\hat{\mathbf{a}}$  is the DWT of a vector  $\mathbf{a}$  and  $\hat{\mathbf{b}}$  is the DWT of a vector  $\mathbf{b}$  then

$$\langle \mathbf{a}, \mathbf{b} \rangle = \langle \hat{\mathbf{a}}, \hat{\mathbf{b}} \rangle \Leftrightarrow \sum_i \mathbf{a}[i] \cdot \mathbf{b}[i] = \sum_j \hat{\mathbf{a}}[j] \cdot \hat{\mathbf{b}}[j]$$

Let us end this section by introducing the notion of *Wavelet Tree*. Wavelet tree exploits the relationships between wavelet coefficients and thus simplifies our presentation throughout the paper. Figure 2 shows a Haar wavelet tree for a vector of size 8; summary coefficients are shown with squares, whereas wavelet coefficients are shown in circles. The original data is drawn with dotted line as children of the leaf nodes of the tree. Haar wavelet tree is a binary tree where each node  $w_{k,j}$  has exactly two children,  $w_{k-1,2j}$  and  $w_{k-1,2j+1}$ . The summary coefficient  $u_{n,0}$  is the root of the tree having only one child  $w_{n,0}$ . This tree structure has been given several names in the wavelet bibliography, such as error tree [19], [20], dependency graph [25], etc. We recommend the reader to see [24] for more information about Wavelet Trees.

## B. Multidimensional DWT

To perform the wavelet transformation of multidimensional datasets we need multidimensional low-pass and high-pass filters. In general, the multidimensional filters are constructed from tensor products of single dimensional low-pass and high-pass filters. The standard form of multidimensional wavelet transform is performed by applying a series of one-dimensional decompositions along each dimension.

For illustration purposes, we focus our discussion on 2-dimensional transformations. The extension to higher dimensionality is straightforward. To decompose a 2-d array of size  $N^2$ , we first completely decompose one dimension and then the other, with the order not being important. This means that we first transform each of the  $N$  rows of the array to construct a new array and then take each of the  $N$  columns of the new array and again perform 1-d DWT on them. The final array is the 2-d standard transform of the original array. We refer the reader to [24], [26] for further information.

## IV. POLYNOMIAL AGGREGATE QUERIES

In this section we introduce the class of polynomial range-sum queries, and show how we recast these queries as vector queries. Then we show how we use the same query model in the wavelet domain.

### A. Defining Polynomial Queries as Vector Dot Products

A polynomial query is a mathematical expression including a sum of one or more attributes each powered to a degree. Most statistical queries can be formed from this class of queries. Therefore, we work on the most general form of polynomial queries to support common and ad-hoc statistical aggregate queries. To start the discussion, let us mathematically define polynomial queries.

*Definition 4.1:* Given a  $t$ -tuple dataset  $T$ , a range  $R$ , and a polynomial function  $f : x \rightarrow \prod_{i=0}^{t-1} \hat{x}_i^{\delta_i}$  for any tuple  $x = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_t)$  in  $T$ , the polynomial aggregate query is written as

$$\mathcal{Q}(T, R, f) = \sum_{x \in T \cap R} f(x)$$

We call this query a polynomial range-sum query of *degree*  $\delta_i$  on attribute  $i$ .

*Example 4.2:* Consider a 2-tuple dataset with entries  $T = \{(age, height) : (15, 140), (15, 160), (15, 180), (20, 140), (20, 160), (20, 180), (25, 160), (25, 200), (30, 140), (30, 200)\}$ . Let the range  $R$  be the set of all ages between 15 and 25.

*COUNT* query: The number of people with ages between 15 and 25 is the number of tuples inside the range  $R$  by choosing  $f(x) \equiv \mathbf{1}(x) = 1$ .

Intersecting the range with the dataset, we have:  $T \cap R = \{(15, 140), (15, 160), (15, 180), (20, 140), (20, 160), (20, 180), (25, 160), (25, 200)\}$ . Count query returns  $|T \cap R| = 8$  as following:

$$\mathcal{Q}(T, R, \mathbf{1}) = \sum_{x \in T \cap R} \mathbf{1}(x) = 8$$

*SUM* query: By choosing  $f(x) \equiv height(x)$ , the query returns the sum of *height* inside the range  $R$ .

$$\mathcal{Q}(T, R, height) = \sum_{x \in T \cap R} height(x) = 1320$$

*AVERAGE* query: An *AVERAGE* query can be composed of the two queries mentioned above.

$$Avg(height) = \frac{\mathcal{Q}(T, R, height)}{\mathcal{Q}(T, R, \mathbf{1})} = \frac{1320}{8} = 165$$

We can generalize this and form other common aggregate queries, e.g. variance or covariance, as polynomial queries. We refer the reader to [1] for more information.

Let us define two useful functions, *characteristic function* and *data frequency distribution*, which we use to simplify the query definition.

*Definition 4.3:* Given a range  $R$ , we define the corresponding characteristic function  $\chi$  as following:

$$\chi(x) = \begin{cases} 1 & \text{if } x \in R; \\ 0 & \text{otherwise.} \end{cases}$$

*Definition 4.4:* The *data frequency distribution* is the function  $\Delta_T$  that maps a point  $x$  to the number of times it occurs in  $T$ . Let us omit the subscript and rewrite this as following:

$$\Delta(x) = \sum_{x \in T} \mathbf{1}(x)$$

In databases,  $\Delta$  is a  $t$ -dimensional datacube that describes the dataset  $T$  in a multidimensional view. Every point of this cube is filled by the frequency of the occurrence of the

represented tuple in the relational data. That is why we name this cube a *data frequency distribution cube* or *DFD* in short.

Now we can rewrite the basic definition of the range-sum as

$$\mathcal{Q}(\Delta, \chi, f) = \sum_x f(x)\chi(x)\Delta(x)$$

This equation can be considered as a vector dot product of two multidimensional vectors:  $f(x)\chi(x)$  and  $\Delta(x)$ . We name these vectors the query vector  $Q$  and the data vector  $D$  respectively.

$$\begin{aligned} \mathcal{Q}(\Delta, \chi, f) &= \langle f\chi, \Delta \rangle \\ &= \langle Q, D \rangle \end{aligned}$$

*Example 4.5:* The figure below shows the multidimensional view of the earlier example. The  $4 \times 4$  cubes represent the input data and the queries in multidimensional views.

The count query is answered by  $\langle Q_1, \Delta \rangle = 8$  in which  $Q_1$  is defined as  $Q_1 = \mathbf{1} \cdot \chi$  (see Figure 3b).

The sum query is recasted as  $Q_2 = height \cdot \chi$ . This means that every item  $x$  inside the range  $R$  has the value of  $height(x)$  (see Figure 3c). We calculate the sum query by the dot product of  $\langle Q_2, \Delta \rangle = 1320$ .

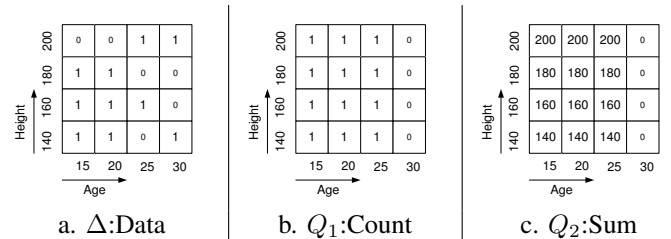


Fig. 3. Multidimensional View of Data and Queries

### B. Wavelet Range-Sum Queries

As shown in Section 3.2, Discrete Wavelet Transform preserves the Euclidean norm. Thus, the generalized Parseval equality applies to DWT, that is, the dot product of two vectors equals to dot product of the wavelet-transformed of the vectors.

$$\langle Q, D \rangle = \langle \hat{Q}, \hat{D} \rangle$$

This property results in the following useful corollary.

*Corollary 4.6:* The answer to a range-sum query  $Q$  over a datacube  $D$  is given by  $\langle \hat{Q}, \hat{D} \rangle$ .

The answer to this dot product is given by the summation of the multiplication of all the query items  $Q$  with the corresponding items of the data  $D$ ,

$$Q = \sum_{\xi \in \hat{D}} \hat{Q}(\xi) \cdot \hat{D}(\xi)$$

Let us rewrite this summation as the following to hint the main advantage of Wavelets; i.e. if  $\hat{Q}$  contains many zeros, then the query is answered very quickly. We will prove why  $\hat{Q}$  contains many zeros in the next section.

$$Q = \sum_{\xi \in \hat{Q}, \text{if } \hat{Q}(\xi) \neq 0} \hat{Q}(\xi) \cdot \hat{D}(\xi)$$

0	1.41	0	0
0	0	0	-1
0.5	0.5	0	0
2.5	0.5	0	0

0	0	0	0
0	0	0	0
0	0	0	0
3	1	0	1.414

a. Wavelet Datacube ( $\hat{\Delta}$ )      b. Wavelet Count Query ( $\hat{Q}_1$ )

Fig. 4. Wavelet-Transformed Data Vector and Query Vector

*Example 4.7:* Let us answer the count query of Example 4.2 in the wavelet domain. We similarly form the  $Q_1$  and then wavelet transform as shown in Figure 4b. The dot product of  $\hat{\Delta}$  and  $\hat{Q}_1$  produces the same query result,  $\langle \hat{\Delta}, \hat{Q}_1 \rangle = 2.5 * 3 + 0.5 * 1 + 0 * 1.414 = 8$ . Using wavelets, we reduced the number of operations from 12 to 3.

Now, we sketch the algorithm of wavelet-based range aggregate query processing as following:

---

**Algorithm 1** Wavelet Query Processing (naive).

---

**Require:** Wavelet Datacube  $\hat{D}$ ,  $R$  {Range},  $f$  {polynomial function}, and  $L$  {Wavelet Filter}

- 1:  $\chi \leftarrow R$ ;
  - 2:  $Q \leftarrow \chi \cdot f$ ;
  - 3:  $\hat{Q} \leftarrow \text{DWT}(Q, L)$ ;
  - 4:  $result \leftarrow 0$ ;
  - 5: **foreach**  $k$  in  $|\hat{Q}|$  **do**
  - 6:   **if**  $\hat{Q}(k) \neq 0$  **then**
  - 7:      $result \leftarrow result + \hat{Q}(k) \cdot \hat{D}(k)$ ;
  - 8:   **end if**
  - 9: **end for**
  - 10: **return**  $result$ ;
- 

Notice that we did not include  $\hat{D} \leftarrow D$  in the algorithm as the datacube is essentially transformed at the time the database is built. To study this offline task, we encourage the reader to explore [24] in which we introduce an efficient algorithm to wavelet transform large data. In Section V, we describe the DWT function and analyze the complexity of this algorithm. Later in Section VII, we explore the lines 5 ~ 9 in more details.

## V. QUERY TRANSFORMATION

Query transformation differs from data transformation in that the transformation is performed on-the-fly using the minimum resources, rather than performing this task offline with enough resources. The complexity for naive wavelet transformation of a datacube is  $O(N^t)$  where  $t$  is the number of cube dimensions and  $N$  is the domain size for each dimension. However, this means that a memory space of the same size as the datacube is needed for query transformation. Clearly, this is neither efficient nor feasible as the datacube size is potentially very large and multiple queries are submitted into the database simultaneously. In this section we propose a fast algorithm for query transformation and compute the complexity of Algorithm 1. We start with the 1-dimensional data and then extend it to the multidimensional case.

### A. One-dimensional Query

In this section we present the advantage of the wavelet transformation in a 1-dimensional range query  $q$ . Here, we show that for a range of size  $R$  on a data of size  $N$  we only need  $O(l \log N)$  coefficients from  $\hat{q}$ , independent of the range size, which is a major benefit specially when  $R \gg l \log N$ . Let us start the discussion with an example.

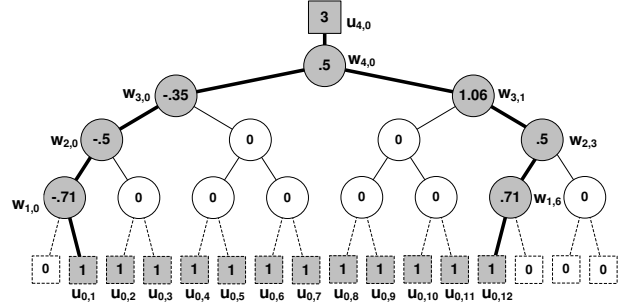


Fig. 5. Lazy Wavelet Transform using Haar filter

*Example 5.1:* Consider the transformation of the range  $R = [1, 12]$  on the 0-degree polynomial query  $q$  of size 16 using Haar filter. The query  $q$  and its wavelet tree is depicted in Figure 5. The only non-zero values of the transformed query  $\hat{q}$  are located on the boundaries of the range to the root of the wavelet tree. Having only two non-zero values per each level of the wavelet tree, we have  $2 \log N$  non-zero values in total.

It turns out that we always observe a similar trend using an arbitrary wavelet filter of length  $l$ . Thus, we operate only on the boundaries and produce at most  $l$  non-zero detail coefficients per step on the boundary sides. Every step can be carried out in constant time, allowing us to perform the entire transform in time and space of  $O(l \log N)$  instead of  $O(N)$ . We call this algorithm the *Lazy Wavelet Transform* (LWT) since it only evaluates wavelet coefficients when they are needed at the next recursive step.

Now we show that with appropriate choices of wavelet filters, we can obtain the same result for general polynomial range queries. Here, we formally define the required condition on choosing wavelet filters.

*Definition 5.2:* A wavelet filter  $\{g_0, \dots, g_{l-1}\}$  is said to satisfy *moment condition* of  $\delta$  if  $\sum_l g_i i^\nu = 0$  for  $0 \leq \nu \leq \delta$ . This condition is met when the wavelet filter has vanishing moment equal or greater than  $\delta$  (see Section III-A).

When a wavelet filter satisfies the moment condition of  $\delta$ , it guarantees that operating the filter on a polynomial query of degree  $\delta' \leq \delta$  produces zero coefficients anywhere except the boundaries. The proof is straightforward. Operating the detail filter with the internal query points of power  $\delta'$  produces only zero coefficients because the vanishing property of the filter,  $\sum_l g_i i^{\delta'} = 0$ , produces only zero values since  $\delta' \leq \delta$ .

Ingrid Daubechies [26] introduced a family of wavelet filters with the maximal number of vanishing moments for the given filter length. Haar filter, named *db1* in this family, has the least vanishing moment of *zero*. Next filter in this family, *db2*, has *1* vanishing moment. In general, longer filters provide higher vanishing moments  $\nu$ . In particular, it is proven that

vanishing moment and filter length are directly proportional for Daubechies filter family as  $\nu = l/2 - 1$  (see [26] for further information). We utilize this property in the following theorem and then we provide a descriptive example afterwards.

*Theorem 5.3:* Using Daubechies filters, a wavelet filter of length  $l = 2\delta + 2$  is required to allow the use of LWT on a polynomial query of degree  $\delta$ .

*Example 5.4:* Let us form three queries with polynomial degrees of 0, 1, and 2 over the domain of integers from 1 to  $N$ . The set of wavelet filters satisfying the moment condition of the corresponding polynomial degree is presented as below:

$$\begin{aligned} q &= \{1, 1, 1, \dots, 1\} & \delta = 0 &\rightarrow l \geq 2 & \text{(i.e. db1, db2, ...)} \\ q &= \{1, 2, 3, \dots, N\} & \delta = 1 &\rightarrow l \geq 4 & \text{(i.e. db2, db3, ...)} \\ q &= \{1, 4, 9, \dots, N^2\} & \delta = 2 &\rightarrow l \geq 6 & \text{(i.e. db3, db4, ...)} \end{aligned}$$

It is clear that the data domain is not a domain of integers for most datasets. However, we show here that if the data domain satisfies the following condition, we can still benefit from LWT for any dataset. We apply this condition as a restriction on the data domain when a datacube is being generated.

*Definition 5.5:* A domain  $X = \{X_0, X_1, \dots, X_{N-1}\}$  is said to satisfy *domain condition* if  $X_i = X_0 + i(X_1 - X_0)$ . This condition ensures that the domain  $X$  is uniformly partitioned, a.k.a. *equi-width* partitioned.

Following this condition, the data domain is basically a linear combination of an integer domain plus a constant. The following useful lemma allows us to apply LWT on such domains.

*Lemma 5.6:* A domain  $X$  satisfying domain condition is applicable for LWT, that is, operating the query with the detail filter produces zeros any where except the boundary points.

*Proof:* This domain is basically an arithmetic series with the initial point of  $X_0$  and the increment of  $X_1 - X_0$ . Let us rewrite  $X_i$  as  $X_i = \alpha + i\beta$  for simplicity and apply the detail filter on  $X$ . We need to prove that  $\sum_i g_i X_i^\nu = 0$ . We have:

$$\begin{aligned} \sum_i g_i X_i^\nu &= \sum_i g_i (\alpha + i\beta)^\nu \\ &= \sum_i g_i \sum_\mu C_\mu^\nu \alpha^{\nu-\mu} \beta^\mu i^\mu \end{aligned}$$

Let us distribute  $g_i$ , swap the order of sigmas, and factor out the rest. We have:

$$\sum_i g_i X_i^\nu = \sum_\mu C_\mu^\nu \alpha^{\nu-\mu} \beta^\mu \sum_i g_i i^\mu$$

Because of the  $\nu$  vanishing moment of the filter and knowing that  $\mu \leq \nu$ , we have  $\sum_i g_i i^\mu = 0$ . Therefore, this proves our lemma  $\sum_i g_i X_i^\nu = 0$ . ■

Let us summarize our discussion so far. We ensure that we can transform any polynomial query in time and space complexity of  $O(l \log N)$  if we meet these conditions (moment condition and domain condition).

Now we briefly discuss the update query for wavelet-transformed data before ending this section. Here, we show that we can recast update queries as vector queries and efficiently perform them in wavelets.

Let  $\lambda$  be the update for the  $(i+1)$ -th value of  $D$ ,  $D[i] \leftarrow D[i] + \lambda$ . Let  $Q$  be a point query on  $(i+1)$ -th element of

$D$  as shown in Figure 6. We have  $D \leftarrow D + \lambda \cdot Q$ . Now we wavelet transform both sides. Note that wavelet transform is a linear transformation, that is, it preserves the vector addition and scalar multiplication. This means we have:  $D + \lambda \cdot Q = \widehat{D} + \lambda \cdot \widehat{Q}$ . Therefore, the following equation is performed when an update query is issued.

$$\widehat{D} \leftarrow \widehat{D} + \lambda \cdot \widehat{Q}$$

*Theorem 5.7:* Let  $\widehat{D}$  be the wavelet transform of vector  $D$  of size  $N$  using filter length  $l$ . Any update value on  $D$  results in  $O(l \log N)$  updates on the wavelet coefficients of  $\widehat{D}$ .

*Proof:* The proof is straightforward.  $Q$  has only  $O(l \log N)$  non-zero values. Thus, this update needs only  $O(l \log N)$  operations. ■

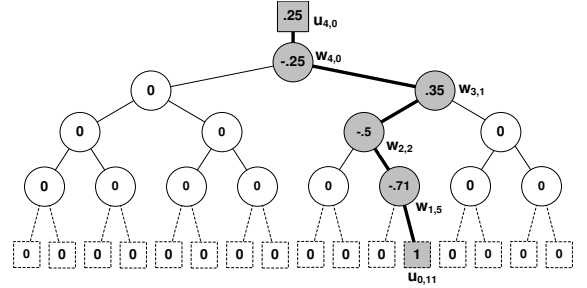


Fig. 6. Wavelet Update using Haar filter

## B. Multi-dimensional Query

A  $t$ -dimensional query  $Q$  is separable to  $t$  1-dimensional queries of  $q_i$  along each dimension,  $q_i(x_i) = f(x_i)\chi(x_i)$ . The tensor product, a.k.a. outer product, of these vectors produces the original query as following:

$$Q = q_1 \otimes q_2 \otimes q_3 \dots \otimes q_t$$

This equation is also held for wavelet domain if we use *standard multidimensional wavelet transform* since standard wavelet transformation is basically tensor product of wavelet subspaces (see Section III-B for more details).

$$\widehat{Q} = \widehat{q}_1 \otimes \widehat{q}_2 \otimes \widehat{q}_3 \dots \otimes \widehat{q}_t$$

This leads us to the following multidimensional extension.

*Lemma 5.8:* A multidimensional query  $Q$  is wavelet transformed by tensor producing the wavelet-transformed aggregate queries along each dimension  $q_i$ .

*Example 5.9:* Figure 7 illustrates the use of tensor product for query transformation. Query vector  $Q$  is the tensor product of  $q_1$  and  $q_2$ . Thus, we transform  $q_1$  and  $q_2$  to  $\widehat{q}_1$  and  $\widehat{q}_2$  and produce the  $\widehat{Q}$  by tensor producing of the two wavelet vectors, i.e.  $\widehat{Q} = \widehat{q}_1 \otimes \widehat{q}_2$ .

We modify the earlier algorithm by incorporating tensor product as Algorithm 2 shows.

The iteration of calculating the tensor product and answering query are combined in lines 7 ~ 12 and consequently no extra memory is needed. Notice that the whole construction of  $\widehat{Q}$  is not performed here. Instead, only  $t$  1-dimensional

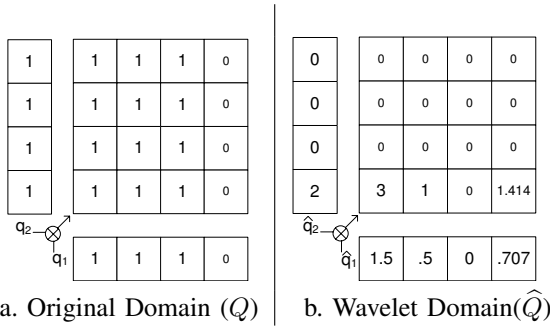


Fig. 7. Multidimensional Query using Tensor Product

**Algorithm 2** Wavelet Query Processing.

---

**Require:**  $\hat{D}$  {Wavelet Datacube},  $R$  {Range},  $f$  {measure function}, and  $L$  {Wavelet Filter}

- 1:  $result \leftarrow 0$ ;
- 2:  $\chi \leftarrow R$ ;
- 3: **for**  $i=1$  to  $d$  **do**
- 4:  $q_i \leftarrow f_i \chi_i$ ;
- 5:  $\hat{q}_i \leftarrow \text{LWT}(q_i, l_i)$ ;
- 6: **end for**
- 7: **foreach**  $k$  in  $|\otimes_i \hat{q}_i|$  **do**
- 8:  $\widehat{Q_{coef}} \leftarrow \prod_i q_i(k)$ ;
- 9: **if**  $\widehat{Q_{coef}} \neq 0$  **then**
- 10:  $result \leftarrow result + \widehat{Q_{coef}} \cdot \hat{D}(k)$ ;
- 11: **end if**
- 12: **end for**
- 13: **return**  $result$ ;

---

arrays of  $q$  are transformed and stored in the memory. This infers that the memory complexity of query transformation is  $O(tl \log N)$  which is a major improvement compared to the naive algorithm with the memory complexity of  $O(N^t)$ . Moreover, the tensor product of the non-zero values of each vector results in  $O(l^t \log^t N)$  number of non-zero coefficients therefore  $O(l^t \log^t N)$  number of data retrieval is required. To the best of our knowledge, this is the best query complexity if we do not sacrifice the update cost.

The following theorem summarizes what we described in this section.

*Theorem 5.10:* Answering a polynomial range-sum query on a  $t$ -dimensional cube with domain size of  $N$  for all dimensions has the memory complexity of  $O(tl \log N)$  and the time and I/O complexity of  $O(l^t \log^t N)$ .

Let us end this section by extending the update query for the multidimensional cube. We perform the multidimensional update query similar to the one-dimensional case, that is, we transform a multidimensional point query and multiply it with the update value  $\lambda$  and add it to the wavelet data  $\hat{D}$ .

$$\hat{D} \leftarrow \hat{D} + \lambda \cdot \hat{Q}$$

*Theorem 5.11:* Let  $\hat{D}$  be the wavelet transform of a  $t$ -dimensional cube with domain size of  $N$  for all dimensions using filter length  $l$ . Any update value on  $D$  results in  $O(l^t \log^t N)$  updates on the wavelet coefficients of  $\hat{D}$ .

*Proof:* A  $t$ -dimensional point query is the tensor product of  $t$  1-dimensional point query:  $\hat{Q} = \otimes_i \hat{q}_i$ . Therefore, we

transform  $Q$  with the complexity of  $O(l^t \log^t N)$  and this is the update complexity. ■

## VI. COLLECTION OF FIXED MEASURES

In this section, we address the inefficiency of the former cube model, DFD, in scientific applications and propose an alternative model for effective storage and query processing.

## A. The CFM model

As discussed earlier, we generate a  $t$ -dimensional cube  $\Delta$  from a  $t$ -attribute data. This means that we treat all the attributes symmetrically, that is, both the dimension attributes and the measure attributes are considered as cube dimensions. Although this model is beneficial to model polynomial queries on any attribute, it suffers from its high sparsity for most real-world datasets in which the dimension attributes uniquely identify each tuple. More specifically in scientific datasets, every event is uniquely identified by the dimensions and the corresponding facts are measured by measure attributes. This results in a very sparse cube as we formally describe here.

Let  $d$  be the number of dimension attributes,  $m = t - d$  be the number of measure attributes, and  $N$  be the domain size for all  $t$  attributes. If the  $d$  dimensions are uniquely identifying the  $t$ -tuple data, there exists only one  $m$ -tuple of measures for each  $d$ -tuple of dimensions. In this case,  $\Delta$  holds only one non-zero value of 1 per  $N^m$  cells in the best case, that is, the highest density ratio is  $\frac{1}{N^m}$ . Yet, when the datacube is wavelet transformed, its density significantly grows by producing a lot of non-zero values. The proof is straightforward. Consider a cube of  $N^m$  items with all zeros. When one of the original values becomes 1 we need to update  $l^m \log^m N$  values to non-zero values (see Theorem 5.11). This means  $\hat{\Delta}$  has  $l^m \log^m N$  times more non-zero values than  $\Delta$  in the worst case. To store this datacube, either we store only the non-zero values,  $O(N^d l^m \log^m N)$ , and use a hash index on top, or we use an array-based storage to store the entire cube,  $O(N^t)$ .

To address this problem, we separate the attribute function  $f(x)$  to a dimension function  $f_d(x_d) = \prod_{i=0}^{d-1} \hat{x}_i^{\delta_i}$  and a measure function  $f_m(x_m) = \prod_{i=d}^{t-1} \hat{x}_i^{\delta_i}$ . We have  $f(x) = f_d(x_d) \cdot f_m(x_m)$ . We similarly separate the characteristic function,  $\chi(x) = \chi_d(x_d) \otimes \chi_m(x_m)$ . Thus we rewrite the general range-sum query equation  $\mathcal{Q}(\Delta, \chi, f)$  as following:

$$\begin{aligned} \mathcal{Q} &= \sum_{x_d, x_m} f_d(x_d) \chi_d(x_d) f_m(x_m) \chi_m(x_m) \Delta(x_d, x_m) \\ &= \sum_{x_d} f_d(x_d) \chi_d(x_d) \sum_{x_m} f_m(x_m) \chi_m(x_m) \Delta(x_d, x_m) \end{aligned}$$

Let us define a new function to simplify the equation above.

*Definition 6.1:* The *fixed measure* is the function  $\Theta_{f_m \chi_m}$  that maps a point  $x_d$  to a value calculated based on  $f_m$ . Let us omit the subscript  $\chi_m$  as  $\chi_m = \mathbf{1}$  for most queries and rewrite the second part as following:

$$\Theta_{f_m}(x_d) = \sum_{x_m} f_m(x_m) \Delta(x_d, x_m)$$

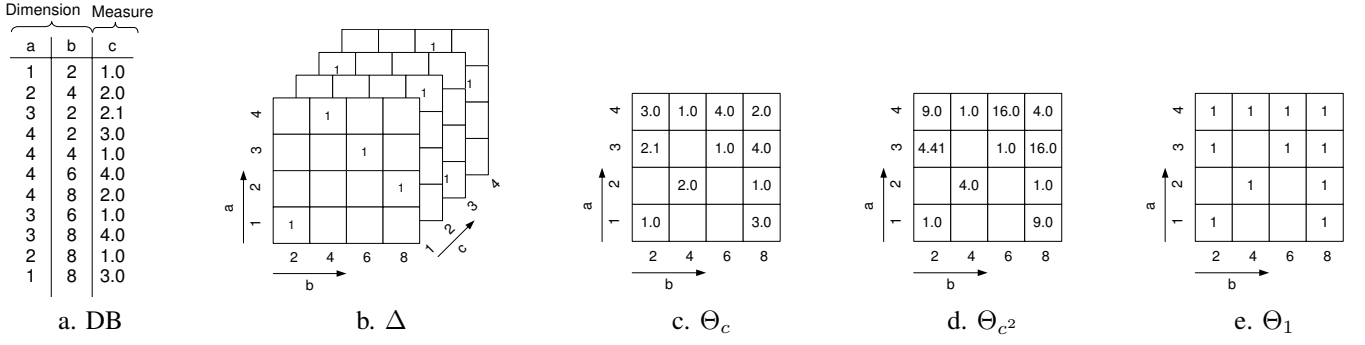


Fig. 8. Multidimensional View of Data

In databases,  $\Theta_{f_m}$  is a  $d$ -dimensional datacube that describes the dataset  $f_m$  in a multidimensional view. Every point  $x_d$  of this cube stores  $f_m(x_d)$  representing a fixed measure of the relational data. In this model, data items are characterized by the dimension attributes and carry fixed fact measure values. That is why we name this cube a *fixed measure* cube or *FM* in short.

Using this model, we have:

$$\begin{aligned} \mathcal{Q}(\Delta, \chi, f) &= \sum_{x_d} f_d(x_d) \chi_d(x_d) \Theta_{f_m}(x_d) \\ &= \langle f_d \chi_d, \Theta_{f_m} \rangle \\ &= \mathcal{Q}(\Theta_{f_m}, \chi_d, f_d) \end{aligned}$$

This implies that we can answer the same query using  $\Theta_{f_m}$  datacube, if  $\Theta_{f_m}$  is known in advance.

*Example 6.2:* Consider a dataset with two dimension attributes of  $a$  and  $b$  and one measure attribute  $c$  as shown in Figure 8. While all three attributes are considered as cube dimensions for the DFD model (see Figure 8a), only the two dimension attributes have become cube dimensions for the FM model (see Figures 8b, 8c, and 8d). Figure 8c shows an FM cube with the measure function of  $f(c) = c$ . For example, the tuple of  $(4, 2, 3)$  is represented as the data point of  $(4, 2)$  in this cube with the value of 3. Similarly, the FM datacubes with the measure functions of  $f(c) = c^2$  and  $f(c) = 1$  are depicted in Figure 8d and Figure 8e, respectively. Using these three datacubes, we can answer polynomial queries up to the degree 2, such as *count*, *sum*, *average*, and *variance*. For example, we compute the average of  $c$  for the range  $\chi$  using this set as following:

$$\text{Avg}(\chi, c) = \frac{\mathcal{Q}(\Theta_c, \chi, 1)}{\mathcal{Q}(\Theta_1, \chi, 1)}$$

This query results in the same answer as if we had formed it on a DFD cube as following:

$$\text{Avg}(\chi, c) = \frac{\mathcal{Q}(\Delta, \chi, c)}{\mathcal{Q}(\Delta, \chi, 1)}$$

To answer arbitrary polynomial queries, it is necessary to pre-compute all the required polynomial functions in advance. In practice, a realistic subset of all the possible combinations of the measures powered up to the degree  $\delta$  suffices. We formally define this set as following:

*Definition 6.3:* A function set  $\mathcal{F}_\delta$  is the set of required fixed measure functions to answer polynomial queries up to the

degree  $\delta$ . For each polynomial degree  $\delta' \leq \delta$ , it is practical enough to compute all attributes and all pairs of attributes powered to  $\delta'$ .

*Example 6.4:* Consider the dataset discussed in Example 6.2. We like to answer polynomial queries up to the degree 2 on the dataset illustrated in Figure 8 with only one measure attribute. This means that the measure functions of  $f(c) = 1$ ,  $f(c) = c$ , and  $f(c) = c^2$  need to be pre-computed in advance. In another work, the function set for this example is  $\mathcal{F}_2 = \{\mathbf{1}, c, c^2\}$ .

*Example 6.5:* Consider a dataset with three measure attributes of  $u$ ,  $v$ , and  $w$ . The function set for  $\delta = 1$  is  $\mathcal{F}_1 = \{\mathbf{1}, u, v, w, uv, uw, vw\}$  having 7 measure functions. The function set for  $\delta = 2$  is  $\mathcal{F}_2 = \{\mathbf{1}, u, v, w, uv, uw, vw, u^2, v^2, w^2, u^2v^2, u^2w^2, v^2w^2\}$  including 13 measure functions.

*Lemma 6.6:* The function set of  $m$  measure attributes supporting polynomial queries up to the degree  $\delta$  is bounded as below.

$$|\mathcal{F}| = 1 + \delta m(m+1)/2$$

*Proof:* Function set has at least the function of  $\mathbf{1}$  for count query. For any degree  $\delta' \leq \delta$  function set includes  $\binom{m}{2}$  pair of measures with power of  $\delta'$ . This set also includes all the  $m$  measures individually powered to  $\delta'$ . Therefore we have:

$$|\mathcal{F}| = 1 + \delta m + \delta \frac{m(m-1)}{2} = 1 + \delta m(m+1)/2$$

We store each function of the function set in a separate FM cube and name the entire set as *Collection of Fixed Measures* or *CFM* in short. As shown in the earlier lemma, the cardinality of CFM is directly proportional to  $m$  and  $\delta$ . This mean that the number of required datacubes increases as the number of measures or polynomial degree grows. However, the number of FM cubes grows more gently than the cube size of DFD because  $F \ll l^m \log^m N$ . We also empirically study this fact in the experimental section.

All these cubes can be hidden from the high level users by defining a general cube framework  $\Theta$  for CFM. As shown in WOLAP framework (see Figure 1), the relevant datacube is selected upon the query submission:

$$\mathcal{Q}(\Theta, \chi, f) \rightarrow \mathcal{Q}(\Theta_{f_m}, \chi, f_d)$$

### B. Analytical Comparison Between DFD and CFM

In this section, we analytically compare the two models and later in Section IX we empirically compare them in more details.

Figure 9 shows the complexities of the two models in terms of storage, update, and query costs. It shows that CFM model outperforms DFD in all three complexities.

	DFD $\hat{\Delta}$	CFM $\hat{\Theta}$
Storage	$\begin{cases} N^t & \text{array-based} \\ N^d l^m \log^m N & \text{hash-based} \end{cases}$	$FN^d$
Update	$l^t \log^t N$	$F l^d \log^d N$
Query	$l^t \log^t N$	$l^d \log^d N$

Fig. 9. Complexity Table,  $l = 2(\delta + 1)$ ,  $F = 1 + \delta m(m + 1)/2$ , and  $t = d + m$

In addition to the complexity difference, these two models differ from one major aspect. Since measure attributes become cube dimensions in the DFD model, it is necessary to quantize them to limit the cube domain sizes while we store the exact measure values in the FM cubes. For example, Figure 8b shows that 2.1 is considered 2.0 in the DFD model to limit the domain size of dimension  $c$  to 4 while the exact value 2.1 is stored in CFM model (see Figure 8c and Figure 8d).

The difference between the actual value and the quantized value is called quantization error  $\epsilon_q$ . To meet the domain condition, each dimension domain must be divided into equal spaces, a.k.a. *uniform quantization* (see Definition 5.5). When we uniformly quantize an attribute  $X$  to  $N$  buckets (domain size of  $N$ ), the median quantization error is calculated as following:

$$\epsilon_q = \frac{X_{N-1} - X_0}{2N}$$

Later in Section IX we empirically show that limiting the quantization error to our desired accuracy significantly increases the domain size  $N$  and consequently increases the storage and query costs.

## VII. APPROXIMATION AND PROGRESSIVENESS

In this section, we show how to incorporate approximate and progressive query processing into our framework. These two bonuses come with the multi-resolution property of Wavelets at no extra cost.

Approximation is essentially needed when either storage space is limited or there is a time constraint on the query execution. When the storage is limited data needs to be approximated and when the time is limited the query needs to be approximated. Here, we study the data approximation first and then we study the query approximation. We further extend the query approximation to progressive query processing. As the reader may notice, the query approximation and progressiveness are essentially the query compression while the data approximation is considered as data compression.

### A. Data Approximation

There are many real-world situations in which either the data is extremely large or the storage capability is limited. For either of these, compression of datacubes is essential in practice. When the storage is limited to  $B$  datapoints, we only store the  $B$  most significant wavelet coefficients. Using these  $B$ -term coefficients we provide approximate results to the queries. Let us state two widely used methods for selecting  $B$ -term wavelet coefficients.

**FIRST-B:** The most significant coefficients are the coefficients with the lowest frequencies. The intuition of this approach is that high frequency coefficients can be seen as noise and thus they should be dropped.

**HIGHEST-B:** The most significant coefficients are those that have the most energy. The energy of a coefficient is defined as power two of the coefficient value. The intuition comes from signal processing: the highest powered coefficients are the most important ones for the reconstruction of the transformed signal which leads to the common non-linear “hard thresholding” compression rule.

*First-B* has the advantage of fast preprocessing and smaller storage space requirement in practice. The reason is that coefficients below a certain frequency are selected in  $O(B)$  without any coefficient sorting. Here, storing indices of the coefficients are not required if an array-based storage is employed. On the other hand, *Highest-B* method introduces the smallest error for total reconstruction of the data as it is widely accepted in signal processing literature.

*Example 7.1:* Consider the wavelet datacube illustrated in Figure 4a. When the data storage is limited to store only 3 coefficients we store the following 3 coefficients either using First-B or Highest-B schema.

$$\hat{\Delta}_{FB} = \{2.5, 0.5, 0.5\} \quad \hat{\Delta}_{HB} = \{2.5, 1.41, -1\}$$

For First-B, we selected the coefficients with the least frequencies whereas for Highest-B, we selected the highest absolute values.

Now we incorporate the compressed data into our framework. In WOLAP, we iterate through the non-zero coefficients of transformed query cube and multiply each coefficient with the corresponding wavelet data. In this process, if the data coefficient is not stored, i.e. dropped previously due to the data compression, we assume this wavelet data coefficient is zero and continue the process. This assumption is basically the implementation of hard thresholding. In other word,  $\tilde{D}(k)$  returns zeros if there is no coefficient for the index  $k$ .

$$\tilde{D}(k) = \begin{cases} \hat{D}(k) & \text{if } k \in B; \\ 0 & \text{if } k \notin B. \end{cases}$$

### B. Query Approximation and Progressiveness

In many applications, the accuracy of the query result can be traded off for a better response time, that is, a fast less accurate result may be preferred to an exact late result. Since the dominant factor for query processing is the database retrieval, we limit the retrievals to a certain number  $B$ . When the query

execution is limited to  $B$  retrievals, we only retrieve the  $B$  most significant wavelet coefficients contributing to the query.

*Definition 7.2:* The *approximate answer* for a range-sum query is computed by performing the dot product for only a subset of the query vector indexed by  $\sigma$ :

$$\tilde{Q}_\sigma = \sum_{i=0}^{B-1} \hat{Q}(\sigma_i) \hat{D}(\sigma_i)$$

The most contributing coefficients are the ones with higher values of the pair of query and data items,  $\hat{Q}(\sigma_i) \hat{D}(\sigma_i)$ . However, at the database population time, one cannot optimally sort the coefficients for all possible queries. Therefore, we advocate selecting the query coefficients at the query time to achieve good approximate results. Toward this end, the  $B$  most significant query coefficients are selected using Highest-B or First-B method. The major observation from our experiments is that this selection performs very well for most queries although we do not hold any assumption on data coefficient distribution. We can further enhance this selection with a hybrid approach in which we store a synopsis of data for a better query coefficient selection. We refer the readers to [18] for more information regarding various techniques used in wavelet query approximation.

By progressively increasing the term  $B$ , we can *order* coefficients based on their significants. We exploit this ordering to answer the query in a progressive manner so that each step produces a more precise evaluation of the actual answer.

*Definition 7.3:* The *approximate answer at iteration  $j$*  for a range-sum query is computed as following at the  $j$ -th progressive step:

$$\tilde{Q}_\sigma(j) = \sum_{i=0}^{j-1} \hat{Q}(\sigma_i) \hat{D}(\sigma_i)$$

If the index visits only the non-zero query coefficients then approximate answer converges to the exact answer as  $j$  reaches  $O(l^d \log^d N)$ . We use the same significance functions used for query approximation to make the approximate answer converge fast to the exact answer (see [18] for more details).

## VIII. TOWARD REALIZATION OF WOLAP

In this section, we address two practical problems to further enhance WOLAP for its use in real-world systems.

### A. Filter Selection

Data attributes are not symmetrically participating in the polynomial queries; that is, some attributes are subjected to higher polynomial degrees. In particular, the dimension attributes are queried with low degree functions, typically 0 or 1, and the measure attributes are queried with higher degree functions, typically 2 or 3. For example, when a dimension attribute is in nominal format, such as *Gender* or *Country*, it is only queried with 0 degree functions. In another word, the dimension is only utilized for defining the boundaries of the range queries.

Unequal polynomial degrees on cube dimensions raise the demand for using different filters for transforming along each dimension. Let us recall that standard multidimensional

wavelet transform is a series of 1-dimensional transform along each dimension. By carefully looking at this transformation, we observe that each 1-dimensional transformation is associated with an aggregation along that dimension. Therefore, we separate the multidimensional transform into multiple 1-dimensional transforms with a specific filter for each one.

Let us describe how we consider this fact in our framework. For a  $d$  dimensional datacube, we define the set of required polynomial degrees  $\{\delta_1, \delta_2, \dots, \delta_d\}$  which states the maximum polynomial degree along all  $d$  cube dimensions. We obtain the corresponding set of filters  $\{l_1, l_2, \dots, l_d\}$  in which  $l_i$  satisfies moment condition of  $\delta_i$  for attribute  $i$  (see Definition 5.2). We transform both data and query vectors with filter  $l_i$  along dimension  $i$ . Note that the same filter chosen for data transformation is required to be used for query transformation in order to utilize the parseval theorem.

*Example 8.1:* Consider the 2-dimensional datacube of  $\{Country, Population\}$ . We would like to transform this cube in such a way that we can perform polynomial queries up to the degree 2 on *Population* attribute. We define the set of polynomial degrees as  $\{0, 2\}$  and consequently transform this data with  $\{Haar, db3\}$ . We first transform the data with *Haar* filter along the *Country* dimension and then we transform it with *db3* along the *Population* dimension.

In practice, we use *Haar* (a.k.a. *db1*) wavelet for dimension attributes and longer filters for measure attributes. This lowers the query complexity to  $O(2^d l^m \log^t N)$  for DFD cubes. We denote this customized-transformed DFD cube by DFD-DM to emphasize that its dimension attributes and its measure attributes are transformed differently. For the CFM cubes, we use *haar* filter for all dimensions, therefore, we have  $O(2^d \log^d N)$  as the query complexity. Later in Section IX, we show how this optimization dramatically decreases the number of I/Os required for range-sum queries.

### B. Auxiliary Coefficients

Generally, it is widely assumed that the data size must be in power of two to allow wavelet transform. However, real-world data are not usually in power of two. Here we show how to wavelet transform data with irregular sizes using the *auxiliary* coefficients.

*Definition 8.2:* To produce summaries and details of  $k$ -th level, we add an arbitrary coefficients if any summary of the lower level  $k - 1$  does not exist. We call the padded values *auxiliary* coefficients.

The auxiliary coefficients can be estimated based on various methods, we suggest four of which as follows. We borrow the names from signal extension on the boundaries in wavelets [27].

- **Zero Padding:** The auxiliary values are zeros in this method. Padding with zero basically rounds the data size to an upper power of two number without a need to store the extra zeros. This simple method has the disadvantage of introducing large detail coefficients by creating large discontinuities. For example in Figure 10a, we pad the array by  $u_{0,5} = 0$  and  $u_{1,3} = 0$  and produce two very large detail coefficients as  $w_{1,2} = 45.25$  and  $w_{2,1} = 32$ .

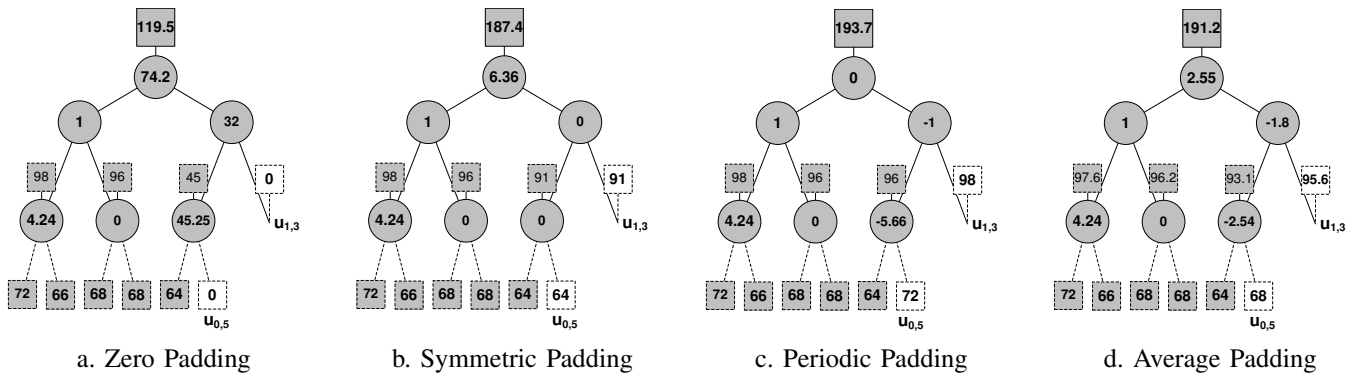


Fig. 10. Auxiliary Coefficients

- **Symmetric Padding:** This method (a.k.a. Mirroring) is replicating the last existing coefficient on the same resolution. Mirroring has the advantage of introducing zero coefficients. For example in Figure 10b, we pad the in array by  $u_{0,5} = u_{0,4} = 64$  and  $u_{1,3} = u_{1,2} = 91$  and produce  $w_{1,2} = 0$  and  $w_{2,1} = 0$
- **Periodic Padding:** This method (a.k.a. Repeating) extends the existing summary with the same order. Repeating is an intuitive method of implementing the extension and may introduce significant noise in some cases. Figure 10c shows an example. If we pad the array with  $u_{0,5} = u_{0,0} = 72$  and  $u_{1,3} = u_{1,0} = 98$ , we calculate the following details:  $w_{1,2} = -5.66$  and  $w_{2,1} = -1$
- **Average Padding:** The auxiliary coefficients are calculated by averaging the existing summaries of the same level. This more advanced method smoothes the signal and introduces the least amount of noise to the transformed data (see Figure 10d).

Except the first method, these methods basically assume that there are some non-zero values in the original array. Therefore, at least one of the query or the data vectors need to be transformed by only using zero-padding. As a rule of thumb, we use Average Padding or Symmetric Padding when it is needed to add the least amount of noise to a vector. In particular, if progressive query or query approximation is the main purpose of using WOLAP, we suggest to use Zero Padding for data transformation and use either of the Average Padding or the Symmetric Padding for query transformation. If data approximation is being used in WOLAP, we recommend using Zero Padding for query transformation and using either of the Averaging Padding or the Symmetric Padding for data transformation.

## IX. EXPERIMENTS

In this section, we empirically examine WOLAP using four real-world scientific datasets. Previously, we showed in [1] that our former model, ProPolyne, outperforms the best known MOLAP techniques and we compared its different progressive methods in [18]. Consequently, our focus here is to show that the new model of WOLAP significantly outperforms from our previous model. We would like to emphasize that our experiments are performed on real datasets using our real system (see [2], [28] for further information).

We start the experiments by describing the datasets employed in our study. Subsequently, we describe how we generate the corresponding WOLAP datacubes. Later, we compare the WOLAP cube models and show that CFM model outperforms DFD model in storage, query, and update performance. Finally, we demonstrate the advantage of WOLAP in progressive and approximate query processing.

### A. Experimental Datasets

We evaluate our framework with four real-world scientific datasets, namely *Precipitation*, *AIRS*, *GPS*, and *LH*.

*Precipitation* [29] is a 4-dimensional dataset that measures the daily precipitation for the Pacific NorthWest for 45 years. It consists of three dimension attributes, latitude, longitude, and time, and one measure attribute, precipitation.

*AIRS*, standing for Atmospheric Infrared Sounder, collects the Earth's atmospheric water vapor and temperature profiles at a very high rate. This 7-dimensional dataset provided by NASA/JPL includes latitude, longitude, pressure level, and time as dimension attributes, and temperature, water vapor mass mixing ratio, and ozone volume mixing ratio as measure attributes. This data is gathered over a year.

*GPS* Occultation Observations Level 2 data set contains profiles of atmospheric temperature, pressure, refractivity, and water vapor pressure with resolution of about a kilometer, derived from radio occultation data. This 8-dimensional dataset is provided by NASA/JPL and includes latitude, longitude, pressure level, and time as dimension attributes, and temperature, refractivity, altitude, and water vapor pressure as measure attributes. We obtained this data for a 9 month period.

*LH* contains monthly production and injection history data for a waterflood oil reservoir. This dataset is an 8-dimensional dataset and includes well ID and time as dimension attributes, and oil production, gas production, water production, water injection, steam injection, and co2 injection as measure attributes. We obtained this data for 57 years.

Figure 11 summarizes the number of attributes for our four datasets.

### B. WOLAP Datacubes

We generated the WOLAP datacubes corresponding to each dataset using our both data models, DFD and CFM.

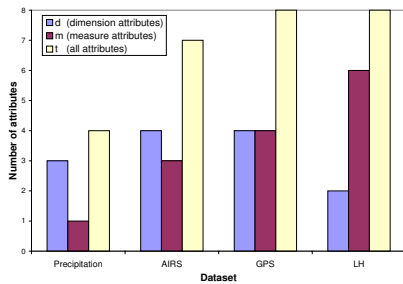


Fig. 11. Number of Attributes in our Experimental Datasets

1) *WOLAP DFD model*: Since all the attributes become cube dimensions in this model, we must uniformly quantize them (see Definition 5.5 for quantization details). This results in a lower query accuracy when we limit the domain sizes to small numbers. To illustrate this, we changed the domain size of the first measure attribute of our four datasets to see how the quantization error varies correspondingly. Figure 12 shows that as we increase the domain size, the quantization error decreases dramatically. Note that the y-axis is in logarithmic scale. We decide on the domain size of each measure attribute by limiting the quantization error to 1% in this experiment.

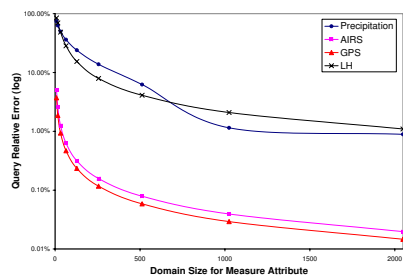


Fig. 12. Query Error introduced by Quantization

Now we need to select the appropriate wavelet filter to transform the datacubes. Here, we study the impact of filter length on the query performance and show that using longer filters increases the query costs.

We transform our DFD cube with 5 different filters, Haar (db1), db2, db3, db4, and db5. Subsequently, we generate 100 random range sum queries (a random range with polynomial degree of 1 for each query) and count the number of required disk I/Os to answer each query. Figure 13 shows the result of our experiment. Note that the y-axis is in logarithmic scale. As expected, use of longer filters dramatically increases the query cost. This observation recommends that we must limit the filter length to the required length  $l = 2\delta + 2$  (see Definition 5.2). In another word, this study shows that the use of longer filters is very costly therefore we must avoid using unnecessary long filters.

For the rest of our experiments we assume  $\delta = 2$  for measure attributes to provide efficient processing of common statistical queries (e.g. count, sum, average, covariance, variance, and correlation). Therefore, we wavelet transform our DFD cubes with db3. In addition, if we limit the dimension attributes for range selection, we can separate wavelet filters

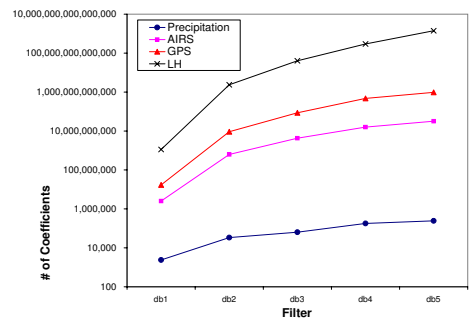


Fig. 13. Query Performance vs. Wavelet Filter

for dimension attributes and measure attributes. Therefore, we also transform the DFD cubes with db1 along dimension attributes and db3 along measure attributes and name this cube DFD-DM. We choose this name to emphasize that this is a DFD cube only with this difference that we wavelet transform its dimension attributes and its measure attributes with different wavelet filters.

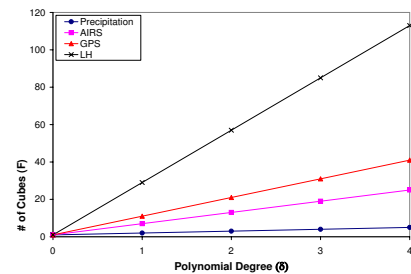


Fig. 14. Cardinality of CFM vs. Polynomial Degree

2) *WOLAP CFM model*: It is essential to predetermine the highest degree of our polynomial queries to prepare the CFM cubes. Let us investigate how polynomial degree can affect the cardinality of CFM.

Figure 14 shows the fact that the size of the function set linearly grows as the polynomial degree increases. To recall, the size of function size represents the number of cubes required in the CFM model. For count query ( $\delta = 0$ ), CFM has only 1 cube member for all datasets. However, CFM grows with the slope of  $O(m^2)$  when  $m$  is the number of measure attributes (see Lemma 6.6). That is why the cardinality of CFM cubes increases more rigorously in LH with 7 measure attributes compared to Precipitation with only 1 measure attribute.

We assume  $\delta = 2$  for the rest of our experiments. Therefore, the cardinality of the corresponding CFMs is 4, 7, 13, and 43 for Precipitation, AIRS, GPS, and LH, respectively. Later we show that CFM model outperforms DFD cube in both storage and query performance although it requires to store more than one datacube.

### C. Storage, Query, and Update performance

In this section we compare the WOLAP cube models in terms of storage, query, and update performance. We generated 100 random range sum queries (a random range with

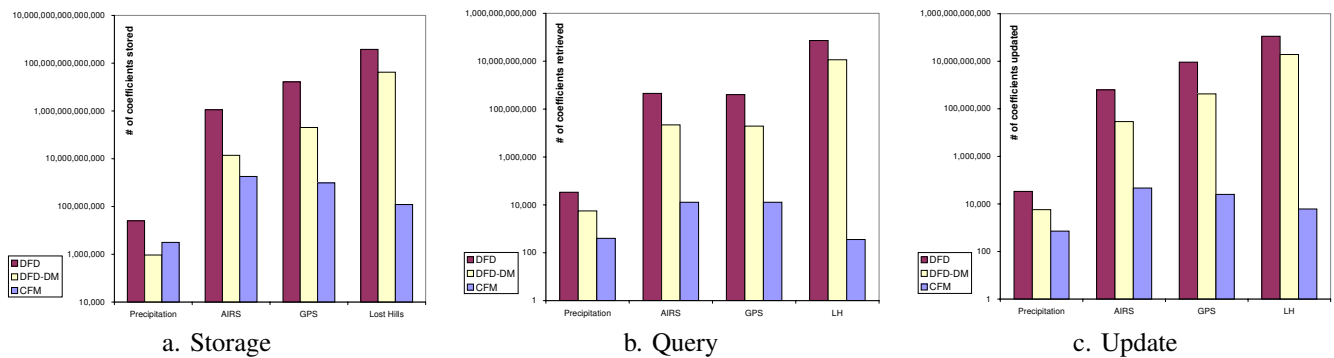


Fig. 15. Storage, Query, and Update performance of WOLAP's cube models

polynomial degree of 1 for each query) and 100 random update queries (an update for a random data point per each query). Here, we report the average performance among all these queries.

Figure 15 shows that CFM model significantly outperforms DFD and DFD-DM in storage, query, and update performance. For DFD and DFD-DM cubes, we observe that the number of coefficients stored or queried inclines as the number of attributes grows whereas the number of dimension attributes is the effective parameter in the CFM model. Following the same trend, DFD-DM cubes outperform DFD cubes in all three figures because of using shorter wavelet filters for dimension attributes.

In particular, Figure 15a shows that the storage requirement for DFD models exponentially increases as the number of attributes  $t = d + m$  grows whereas CFM models are affected mainly by only the number of dimension attributes  $d$ . While it is practically impossible to generate and store a DFD cube of very high dimensional datasets, CFM cubes are constructed using reasonable storage space although we have generated more than one datacube. For example, observe the significant difference between CFM model and DFD model of *LH* dataset in Figure 15a. Although we store 43 FM cubes in the CFM model of *LH*, it requires to store only 121 million coefficients. This is significantly smaller than 378 million million coefficients of the only DFD cube in its DFD model. This figure basically shows that  $F \ll l^m \log^m N$  (see Figure 9 for more details). Also, notice the significant difference between the storage requirement of the DFD-DM model and the DFD model. Figure 15a shows that the DFD-DM model outperforms the DFD model by 90% in the worst case. Note that the y-axis is in logarithmic scale.

Figure 15b depicts the range aggregate query performance. CFM significantly outperforms DFD in all four datasets. In particular, the significant difference between the DFD and the CFM query performance on *LH* data shows the inefficiency of DFD in querying high dimensional datasets. Also notice that the query performance on *AIRS* and *GPS* are exactly similar in the CFM model. This is due the fact that both models have the same size of cubes in the CFM model and that query performance is not dependant on the cardinality of the CFM model. Moreover, Figure 15b shows the significant difference between the query cost of the DFD-DM model and the DFD

model. Here, the DFD-DM model outperforms the DFD model by 85% in the worst case.

Figure 15c illustrates the update performances for our datasets. Similarly, CFM outperforms DFD although multiple cubes need to be updated upon a single update query in the CFM model. In a CFM model, the update cost is generally higher than the query cost because of the same reason, that is, the cardinality of the CFM model  $F$  influences the update cost (see Figure 9). Similar to update comparison, the update cost of the DFD-DM model is significantly better than the DFD model (85% less in the worst case).

#### D. Approximation and Progressiveness

In this section we investigate the WOLAP performance in progressive and approximate query processing. Here, we report our results only on CFM cubes although the general trend remains the same for DFD cubes as well.

First, we monitor the average progression of 100 random range sum queries (a random range with polynomial degree of 1 for each query). In this experiment, we order the query coefficients based on their frequency significance, First-B ordering of query, for simplicity. We can further improve this ordering by deploying Highest-B ordering of query or the hybrid ordering of data and query coefficients. We refer the reader to [18] for more empirical study on progressive query processing.

Figure 16 shows that for all the datasets we achieve 99% query accuracy only by retrieving 30% of data coefficients required for the query. It is important to recall that 100% of query progress is met when we retrieve  $O(l^d \log^d N)$  number of data coefficients.

If space is scarce, we approximate data by keeping a set of significant data coefficients using Highest-B or First-B hard thresholding. For this set of experiments, we employ Highest-B ordering and run the experiments with various synopsis sizes. We often show the synopsis size with the term *compression ratio*:

$$\text{Compression ratio} = \frac{\text{Synopsis size}}{\text{Data size}}$$

For each synopsis size, we performed 100 random sum queries and computed the mean relative error over the exact answer computed with the complete data. Figure 17 shows that for

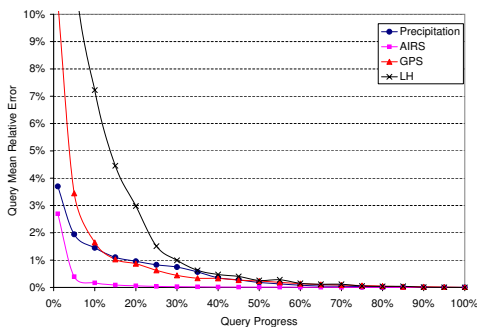


Fig. 16. Progressiveness

all the four datasets we achieve accurate query result, less than 1% relative error, with only keeping less than 1% of the data coefficients. This empirically illustrates that wavelet transform is a favorable tool for data compression and query answering at the same time. This is more noticeable when a dataset is well-compressible like AIRS dataset in this study. Here, storing only 0.001% of AIRS data still results in 99% query accuracy.

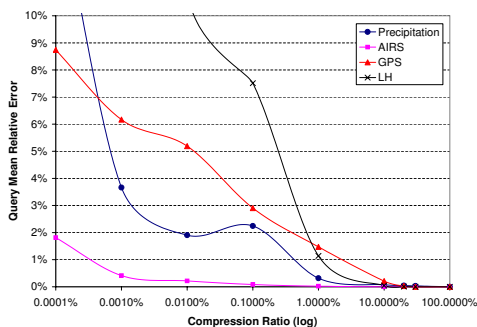


Fig. 17. Data Approximation

## X. CONCLUSION

We have introduced a new framework, named WOLAP, in which we support exact, approximate, and progressive polynomial aggregate query processing. We have extended our former cube model, the DFD model, to be deployed in scientific applications by introducing a less sparse model, the CFM model. We have extensively studied the realization of both models and shown that CFM model is a more practical method for real-world datasets. We have also introduced the filter collection concept to reduce query and storage cost. Last but not least, we relaxed the common assumption on power-two domain of dimensions to an arbitrary number. All these contributions enable WOLAP to be deployed for real-world scientific data analysis applications.

## ACKNOWLEDGMENT

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and IIS-0238560 (PECASE), unrestricted cash gifts from Google and Microsoft, and partly funded by NASA's GENESIS-II REASON project and the Center of Excellence for Research and Academic Training on

Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California - Chevron initiative.

## REFERENCES

- [1] R. Schmidt and C. Shahabi, "Propolyne: A fast wavelet-based technique for progressive evaluation of polynomial range-sum queries," in *EDBT*, 2002.
- [2] M. Jahangiri and C. Shahabi, "ProDA: A Suite of WebServices for Progressive Data Analysis," in *ACM SIGMOD (demonstration)*, 2005.
- [3] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant, "Range queries in OLAP data cubes," in *ACM SIGMOD*, 1997.
- [4] S. Geffner, D. Agrawal, A. E. Abbadi, and T. Smith, "Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes," in *ICDE*, 1999.
- [5] C.-Y. Chan and Y. E. Ioannidis, "Hierarchical cubes for range-sum queries," in *VLDB*, 1999.
- [6] S. Geffner, D. Agrawal, and A. E. Abbadi, "The dynamic data cube," in *EDBT*, 2000.
- [7] M. Riedewald, D. Agrawal, and A. E. Abbadi, "Space-efficient datacubes for dynamic environments," in *Data Warehousing and Knowledge Discovery (DaWaK)*, 2000.
- [8] D. A. Mirek Riedewald and A. E. Abbadi, "Flexible data cubes for online aggregation," in *ICDT*, 2001.
- [9] V. Poosalu and V. Ganti, "Fast approximate answers to aggregate queries on a data cube," in *SSDBM*, 1999.
- [10] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Optimal and approximate computation of summary statistics for range aggregates," in *PODS*, 2001.
- [11] J. Shanmugasundaram, U. Fayyad, and P. Bradley, "Compressed data cubes for OLAP aggregate query approximation on continuous dimensions," in *ACM SIGKDD*, 1999.
- [12] R. R. Schmidt and C. Shahab, "Wavelet based density estimators for modeling multidimensional data sets," in *IAM Workshop on Mining Scientific Data Sets*, 2001.
- [13] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi, "Approximating multi-dimensional aggregate range queries over real attributes," in *ACM SIGMOD*, 2000.
- [14] J. M. Hellerstein, P. J. Haas, and H. Wang, "Online aggregation," in *ACM SIGMOD*, 1997.
- [15] I. Lazaridis and S. Mehrotra, "Progressive approximate aggregate queries with a multi-resolution tree structure," in *ACM SIGMOD*, 2001.
- [16] M. Riedewald, D. Agrawal, and A. E. Abbadi, "pCube: Update-efficient online aggregation with progressive feedback," in *SSDBM*, 2000.
- [17] Y.-L. Wu, D. Agrawal, and A. E. Abbadi, "Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes," in *CIKM*, 2000.
- [18] C. Shahabi, M. Jahangiri, and D. Sacharidis, "Hybrid Query and Data Ordering for Fast and Progressive Range-Aggregate Query Answering," *International Journal of Data Warehousing and Mining*, vol. 1, 2005.
- [19] J. S. Vitter, M. Wang, and B. R. Iyer, "Data cube approximation and histograms via wavelets," in *CIKM*, 1998.
- [20] J. S. Vitter and M. Wang, "Approximate computation of multidimensional aggregates of sparse data using wavelets," in *ACM SIGMOD*, 1999.
- [21] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim, "Approximate query processing using wavelets," in *VLDB*, 2000.
- [22] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing wavelets on streams: One-pass summaries for approximate aggregate queries," in *VLDB*, 2001.
- [23] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "One-pass wavelet decompositions of data streams," in *IEEE TKDE*, 2003.
- [24] M. Jahangiri, D. Sacharidis, and C. Shahabi, "SHIFT-SPLIT: I/O Efficient Maintenance of Wavelet-Transformed Multidimensional Data," in *ACM SIGMOD*, 2005.
- [25] C. Shahabi and R. Schmidt, "Wavelet disk placement for efficient querying of large multidimensional data sets," in *Department of Computer Science Technical Reports*. University Of Southern California, 2004.
- [26] I. Daubechies, *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, 1992.
- [27] G. Strang and T. Nguyen, *Wavelets and filter banks*. Wellesley-Cambridge Press, 1996.
- [28] "ProDA: <http://infolab.usc.edu/projects/proda/>."
- [29] M. Widmann and C. Bretherton, "50 km resolution daily precipitation for the pacific northwest," 1949-94.