
International Journal of Applied Sciences (IJAS)
Singaporean Journal of Scientific Research(SJSR)
Vol.7.No.1 2015 Pp.359-370
available at:www.iaaet.org/sjsr
Paper Received :26-12-2014
Paper Accepted:18-01-2015
Paper Reviewed by: 1Prof. L Selvakumar 2. Chai Cheng Yue
Editor : Dr. Chu Lio

# MULTIDIMENSIONAL ANALYSIS OF DISTRIBUTED XML DATA

*Sambit Pradhan, Business Intelligence Consultant, Avid Technology, Inc.*
*315 Alexandra Road #03-01 SimeDarby Business Centre, Singapore -15994*
*sambit_pradhan@outlook.com*

***Abstract* — The expeditious proliferation of the internet to ubiquity, the infrangible dependence of global enterprises on Web services, the universal adoption of SOA, cloud computing, social media and online publishing has made XML the lingua franca of the digital age and has generated a plethora of data in XML. The immense popularity of NoSQL and document-oriented data stores have also added tremendously to this trend. At the same time the need for cost effective, low maintenance, simple, customizable and highly scalable analytical systems for small and medium size businesses, information and measurement companies, academic and research institutions.**

**This paper presents a novel approach for dimensional analysis of distributed, disparate, heterogeneous, voluminous XML data, the Multidimensional Analysis of Distributed XML data – MAX – a scalable, high-performance, open source, schema-free, document-oriented database. The primary objective of the paper is to propose an architecture for a document-oriented database, including details of its foundation data structures and querying mechanism; based on existing standard technologies for multidimensional analysis of large set of XML data. The Motivations for this approach include simplicity of design, generality, cost-effectiveness, usability, horizontal scaling, storage efficiency, minimal use of memory and resources. The method has virtually no memory limitation or data set size limits and performs relatively well in terms of data latency and resource consumption. The paper details an implementation of this method along with sample performance benchmarks.**

***Index Terms*— Data structures, Dimensional Analysis, OLAP, XML.**

## I. INTRODUCTION

A large amount of heterogeneous and distributed information abounds, today, as document centric and data centric XML.

The former has only few, interspersed mark-ups, the latter is solely created and interpreted by application logic. XML is extremely powerful in representing both structured and semi-structured data. Semi-structured data as XML is widely used for as data exchange format, message format like SOAP [GHM+07], logging and as storage format. It succinctly describes complex information and there are no limitation on the type, content and structure of data that can be stored as XML. Thus it is used in a wide spectrum of application domains: document publication, to computational chemistry, health care and life sciences, multimedia encoding, geology, and e-commerce. Increasing

use of web-based business processes, document-oriented database for big data and real-time web services has led to further acceptance of XML. This entails the analysis of XML data as indispensable.

Most of the existing solutions for the analysis of XML data – XML database, XML data warehousing, distributed querying etc. – have some caveat. Query processing over distributed and fragmented XML data is complex, challenging and resource consuming. Analysis of heterogeneous data only compounds the difficulties and in some cases is impossible to perform. Centralized solutions like MOLAP products are not only expensive and complicated to develop and maintain, but also can only handle limited sized data sets.

MAX – Multidimensional Analysis of Distributed XML Data is an integrated suite of software facilities for the collation of distributed XML data, processing of the data into multidimensional data sets and analysis of the data thereof. The platform embodies the application of novel and pioneering methodologies and data structures on well-established open source platforms using standard and popular programming paradigms. MAX is a highly scalable, high-performance, open source, schema-free, document-oriented bare bones database that facilitates the analysis of distributed, heterogeneous XML data and XML based repositories.

Although this platform, in its current state, is not intended to compete with or be a substitute for the industry leading solutions; it has tremendous potential to grow into a viable low cost commercial product with small to medium organizations as the target users.

## II. Related Work

The individual indexing, storage, XML processing and data structures used and described in this methodology are popular industry standards and are widely used in database and web applications. However, the core concept of this paper and the methodology "segregated index architecture" – The splitting of a relational data set into separate data structures that maintain partial relation information, keeping the dimensional and factual elements segregated, at the same time providing for highly effective indexing and data storage capabilities, applied to extremely large data sets, to the best of my knowledge and belief is a singular, original and novel approach that has not been proposed earlier.

## III. Multidimensional Analysis of Distributed XML Data – An Overview

Provided here, is a detailed description of the MAX platform, its architecture, components, functioning, technical specifications and brief illustration of the conceptual mechanism of query execution that form the conceptual basis of the platform in order to provide a proper perspective when reading the remainder of this document.

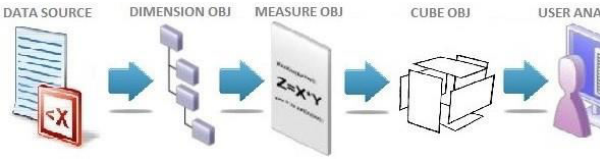



Fig. 1. Complete life cycle of an Analytic Project. The project starts with the identification of the XML data sources, followed by the creation of the dimension and measure object and finally the cube object. After the creation of the Cube object a user can query the

One of the major tenets of the platform was minimal and efficient use of memory. Memory accesses are a common bottleneck in applications and memory is rapidly becoming a precious resource in data processing environments. Memory is also a major limiting factor in the size of dataset that can be handled in the application. Improving on the efficient use of memory and reducing memory dependent operations can drastically reduce the costs of execution and remove limitations on the platform. The GroupBy-Aggregate followed by sorting requires the most amount of memory in a sql operation. Indexing is another process where often, for large datasets, it is not possible to perform index construction efficiently on a single machine.

The max platform uses a "segregated index architecture" as an effective solution to the issues related to memory usage with high volume of data analysis. Rather than maintaining the dimension data, fact data and indexing in a single object the components are stored as different objects. The query in turn access each object in parallel and exclusively and then combines the result set from each of the operations to generate the resultant dataset.

The cardinal component of the MAX platform is the 'Cube object'. A user queries the cube object using a simplified application to perform analysis on datasets of interest. The aforementioned "segregated index architecture" is implemented as the framework of the cube object. The cube object is a combination of three files. Each file stores a specific combination of data and index. A query targeted at a cube, access and processes each of the objects independently to generate intermediate result datasets; and finally combines the intermediate datasets to generate the resultant dataset. Each cube object consists of three objects: Cube Data object, Cube Bitmap object and the Cube Index object. A Cube object is created from XML data sources in conjunction with two metadata objects: The

dimension object and the measure object. The dimension and measure objects provide the metadata framework / model for the creation of the cube object. They provide a layer of logical abstraction from the underlying XML data sources for the cube object.

A user works in the purview of an 'Analytic Project'. An analytic project consists of multiple cubes and the dimension objects related to a subject area under analysis. There are four steps in the process of creation of an analytic project from the source XML data: (i) Creation of the Dimension object (ii) Creation of the Measure object (iii) Creation of the Cube object. After the creation of the Cube object as part of an Analytic project a user can query the cube object.

The query used to retrieve information from a cube object is called the "Analytic Query". The Analytic Query is made of three clauses: The SELECT clause, the WHERE clause and the GROUP BY clause. During the execution of the query the query is first broken down into its constituent clauses and the final result set is achieved by processing each clause separately and combining the outputs from each process.

The concepts and methodologies expounded in the MAX platform can be implemented on several technology stacks such as LAMP, WAMP, MAMP, SAMP etc. using a variety of languages like C, C++, Java, Pearl, PHP and others. As a proof of the concept and to test the validity of the thesis a simple but effective platform was developed. The complete platform has been developed in Python 3.4.0 because of its rapid prototyping, quick turnaround, rich feature set, platform independence and dynamic semantics. The platform uses well established methodologies, programming paradigms and standard protocols like XML shredding, bitmap indexing, inverted indexing and standard data structures like text and YAML. A complete GUI driven user interface has been implemented for all interaction with the platform to illustrate the simplicity and ease of use that can be achieved.

## IV. The Dimension Object

A dimension is a collection of reference information about a measurable events or "facts". Dimensions categorize and describe data warehouse facts and measures in ways that support meaningful answers to business questions. Data (facts) can be filtered and grouped ("sliced and diced") by various combinations of dimension attributes. They form the very core of dimensional modeling. More specifically 'Conformed dimensions' allow facts and measures from disparate sources and different areas of the business to be integrated, categorized and qualified in the same way across multiple facts and/or data marts. The mainstay of a dimensional model is the conformed dimension.

The foundation block of the MAX platform is the 'Dimension Object'. A dimension object is a file that houses the dimension elements and the relationship between the elements (level and hierarchy). The creation of the dimension object is the first step towards the conception of a multidimensional model. A dimension object is created for every dimension that is to be used in the analysis. The dimension object file is It has a file extension of ***.dim*** that identifies it as a dimension object file. E.g. A product dimension can be names as *productline.dim*. The application identifies each dimension by the file name and the extension.

A dimension object is intrinsically a YAML associative array file that maintains the entire dimension element set, with the dimensional hierarchy maintained by outline indentation. The YAML format was preferred due to excellent human readability, easy maintenance, compactness, feature, flexibility in data presentation, multiple documents within a single stream and popular support by platform and languages.

The dimension objects strictly observe the YAML structure: Hierarchy is maintained by outline indentation, all elements are left justified, the child elements are indented one step further to their parent, all sibling elements have the same indentation. Sequence items are denoted by a dash and key value pairs within a map are separated by a colon. There is a single root element "ALL". All elements are children of the root element. Every dimension entry is a key-value pair. The colon maps the key to the value. The left hand side of the colon is the key and the right hand side the value. The value can be enclosed in single quotes incase of special characters.

```
--- # Dimension ■ Geography

ALL: ALL
geoentity : AMERICAS
  region : AMER North
    territory : Minnesota
    territory : Wisconsin
  region : AMER East
    territory : California
    territory : Washington
      salesterritory : LATAM Pro Video

geoentity : EMEA
  region : Southern EMEA
    territory : Africa
    territory : France & Italy
    territory : BELUX
  region : Central EMEA
    territory : DACH
    territory : Turkey
    territory : Middle East
```

Fig. 2. Example of a dimension object file: sample "Geography"

An entire dimension object file is read as a 'nested list' in the MAX application. The data structure follows a standard list/array item with nested elements. This provides for

extremely fast "nested for loops" iterations that facilitate data roll up during query execution.

```
['ALL', ['AMERICAS', ['AMER North', ['Minnesota','
Wisconsin'],'AMER East', ['California', 'Washington',
['LATAM Pro Video']]], 'EMEA', ['Southern EMEA',
['Africa',' France & Italy',' BELUX'], 'Central
EMEA', ['DACH', 'Turkey',' Middle East']]]]
```

Fig. 3. The dimension object file as a standard Python nested array in the MAX application. Every child element is a nested array in the parent array.

### A. *The Dimension Creation Wizard*

The 'Dimension Creation Wizard' is a GUI based utility developed in Python 3.4.0 that creates Dimension Object Files directly from XML data sources. Since the XML data source themselves contain the dimension elements as nodes, they are an excellent source for creating the dimension. The application generates hierarchical YAML files with dimension elements by directly referencing the XML data nodes. The process is completely automated and implicit. Element level and relationship are managed and created automatically by the wizard. The primary process consists of shredding the XML file / data at a particular node to extract the element / node value. The looks up all child elements of the node and writes them as child dimension elements in the dimension file while maintaining the outline indentation in the YAML file.

| **Algorithm1: Dimension_Object_Create(XML Src,Node, Xpath)** |
|---|
| **Input :XML Source, XML Node, XPath expressionof the NODE**<br>**Output : Dimension Object file** |
| 1: CREATE <Dimension>.dim // Create the empty Dimension Object file<br>2: WHILE ( NOT END OF XML source stream)<br>3: // READ Single XML Document from source stream to temp object<br>4: Temp_doc = READ(XML Src.document)<br>5: // Search for the target node using the Node and Xpath Expression<br>6: IF ( search('Node' , 'Xpath') = TRUE)<br>7: Write 'Node : <Node Value>' to <Dimension>.dim file<br>8: // Search for all Children<br>9: IF ( search('ChildNode') = TRUE)<br>10: Write 'ChildNode : <Node Value>' to <Dimension>.dim file<br>11: END |

Fig. 4. Algorithm for the creation of the Dimension Object.

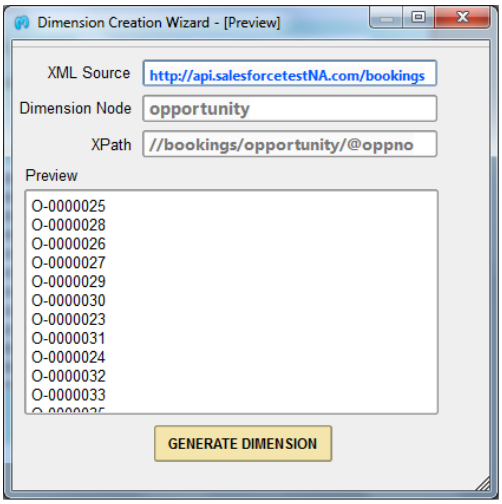



Fig. 5. Dimension Creation Wizard.

## V. THE MEASURE OBJECT

The Measure is a special type of dimension that holds the measurement attributes. Measures are numeric values (e.g., units of sales or total sale amount) associated with the business data. Data analysis usually involves dimensional reduction of the in-put data using various aggregation functions. Like the Dimension object the Measure object is a YAML file that maintains all the measure elements. The difference between the two is that the dimension object stores the elements in an indented structure that represents the dimension hierarchy; in contrast, the elements of the measure dimension are maintained in a single root level, with no hierarchy.

The measure dimension contains two types of elements: standard measure element and derived measure element. The standard measure elements are direct references of the factual fields in the source data. The derived measure elements are, as the name suggests, derived from the combination of standard measure elements, arithmetic operator and numbers. A standard measure element is a simple entry in the dimension. A derived measure element has two parts: a key and an expression. The parts are mapped by a colon, placed between the two. The left has side of the map is the key of the element. The right hand side is an expression. The expression consists of reference to other elements in the measure dimension and arithmetic operator and/or numbers. Between two elements, or a combination of an element and a number an arithmetic operator is placed in parenthesis. The order of operation of the expression strictly follows the BEDMAS rule. Complex expressions can be written by nesting expressions within square brackets. Such an expression is evaluated starting from the inner most nested expression to the outer most expression.

[A (+) B (*) [ D (–) E ]]

By maintaining a separate measure dimension we can specify complex calculations for creating advanced measures that are not available in the data source. These advanced measures are processed and stored within the cube in the process of creation of a cube; these complex calculations do not need to be performed during query execution; thus they drastically reduce query operations and query overhead during query execution to fetch data from a cube.

```
--- # Measure Dimension

AMOUNT
AMOUNT_USD : AMOUNT(*)CURR_RATE
UNIT_COST
QUANTITY
CURR_RATE
TOTAL_COST : UNIT_COST(*)QUANTITY
BOOKING
BOOKING_AFX :
BOOKING_CFX : BOOKING_AFX(*)CURR_RATE
REVENUE
MARGIN : [[REVENUE(-)TOTAL_COST](/)TOTAL_COST]
FORECAST
VARIANCE : FORECAST(-)REVENUE
```

Fig. 6. Example of a measure object file. This is a sample "Finance measure" dimension. There is no hierarchy in the Measure Dimension.

The measure object also facilities the aggregation, combining and calculation of factual data from different data sources. E.g.'Cost' from an inventory dataset can be combined with 'Orders' dataset and 'Payment' data set, each from different sources, to calculate the margin for a purchase transaction. A measure object can only be created manually.

## VI. THE CUBE OBJECT

The Cube object contains the actual data that is used by the analytic query to compute the final result set. The Cube Object is made of three files: the Cube Data object file, the Cube Bitmap object file and the Cube Index object file. The three files complement each other and consecutively take part in successive stages of the query processing. All the files are created simultaneously, during the creation of the cube object in the cube designer. The Cube Data object file has an extension ***.cdo***, the Cube Bitmap object file has an extension ***.cbo*** and the Cube Index object file has an extension ***.cio.***

For each cube the name of the cube is suffixed to the three object file. E.g. For cube Revenue the object files would be named as *Revenue.cdo*, *Revenue.cbo* and *Revenue.cio*. The name of the cube has to be unique within the. The application recognizes the cube objects by their name.

The Cube Data object file only stores the factual data at the leaf or root level or the most granular level available. The relation between the dimensions and the fact data are stored in the Bitmap and the Index object file. The Bitmap and Index object file store partial and complementary aspects of the relationship and only the combination of the both provide the.

Central to the creation of the Cube object is the Virtual Data Set. Although all the three Cube objects are directly created from the data source, it is convenient to visualize a logical data structure that provides a context for the creation of the cube object – Virtual Data Set(VDS). The VDS is discussed in detail in the next section, as the process of creation of the VDS is nothing but the logical depiction of the actual process of creation of the Cube object.

### A. *The Virtual Data Set*

The Virtual Data Set(VDS) is a conceptual two-dimensional relational representation of data that is intrinsically generated in the process of cube object creation from data sources. The VDS is similar to a data base relational table in structure, in that, it consists of tuples that represent a set of ordered and related data and fields that provide the ordered structure for the tuples. The Virtual data set is neither a process nor a physical structure. It is not a temporary file, has no persistence and does not physically manifest anywhere in the platform. The VDS provides the logical view of the and for the visualization of the internal data structure that servers as the source for the creation of the cube object. The functionality of the virtual data set is similar to a data base view. A single tuple of the VSD may contain fields from a single or more data source depending on the number used to generate the cube. The only addition to the incoming data from the data sources is the allocation of a row-id to every single incoming virtual record. This row-id is the unique identifier of the tuple and is the equivalent of a surrogate key. The indexing of the tuples across all the objects is based on this row-id.

A virtual data set is created from the combination of XML data source, dimension object and the measure object. Together they form a logical dimensional model – The data sources as fact tables joined together by

dimension objects and measure object as dimension table. Data from multiple sources are joined together across nodes belonging to a single dimension using the dimension object and the measure object.

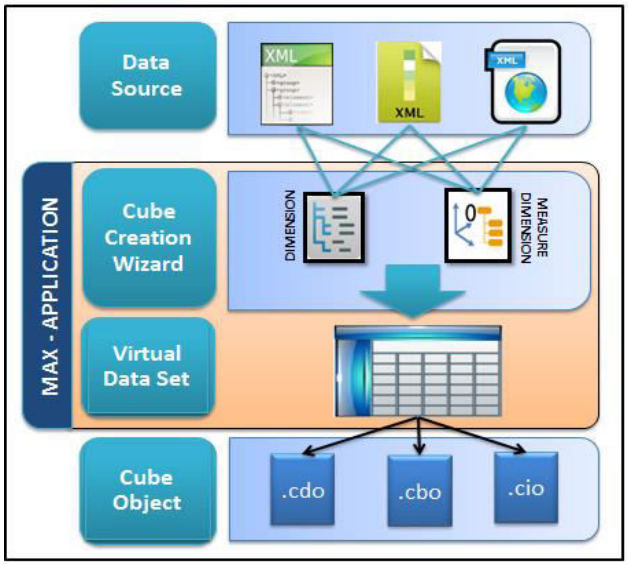



Fig. 7. Overall architecture of the MAX application platform.

### B. *The Cube Bitmap Object File*

The Cube Bitmap Object is a specialized two dimensional array that has a single column for every row of data in the virtual table and row for every unique dimension element in the virtual table. The Bitmap Object represents membership of a dimension element in a VDS tuple. A row is created for each dimension element for all of the dimensions in the VDS. Each column represents a distinct row within the VDS. The Bitmap Index stores partial relationship of a VDS tuple. A node or cell is formed at the cross junction of a VDS row (column) and a VDS dimension element (row).

If a dimension element is present as a key value in a row then the particular node is populated with 1, representing the availability of the dimension element in the row, else the node is defaulted to 0. By populating the nodes formed by the combination of all the rows and all the dimension lements with 1 and 0 a relationship mapping is created. he advantage of using this particular data structure and nethodology is that complex set operations like ntersection, union and set-theoretic differences, conditions pecified in the WHERE clause (or nline query) of a query, can be computed extremely fast nd with minimal use of resources using bitwise logical peration like AND, OR, XOR and NOT to resolve omplex conditions.

**VIRTUAL DATA SET**

| ROWID | OPORTUNITY | REGION | ACCOUNT | TYPE | AMOUNT | FCST_Q1 |
|---|---|---|---|---|---|---|
| 1 | General Dynamics | AMER | Defense | Complex | 350,000 | 100,000 |
| 2 | Boeing | AMER | General | Complex | 1,500,000 | 750,000 |
| 3 | Lockheed Martin | AMER | Defense | Standard | 760,000 | 300,000 |
| 4 | Thyssen Henschel | GERMANY | Defense | Complex | 140,000 | 80,000 |
| 5 | BAE Systems | UK | General | Standard | 1,200,000 | 340,000 |
| 6 | CAP Scientific | UK | Defense | Standard | 300,000 | 110,000 |

| VECTOR KEY ↓ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| AMER | 1 | 1 | 1 | 0 | 0 | 0 |
| GERMANY | 0 | 0 | 0 | 1 | 0 | 0 |
| UK | 0 | 0 | 0 | 0 | 1 | 1 |
| Defense | 1 | 0 | 1 | 1 | 0 | 1 |
| General | 0 | 1 | 0 | 0 | 1 | 0 |
| Complex | 1 | 1 | 0 | 1 | 0 | 0 |
| Standard | 0 | 0 | 1 | 0 | 1 | 1 |

Fig. 8. Illustration of creation of a Bitmap object from a virtual data set. Each dimension element in the VDS is transposed into a row and each row in the VDS into a column in the Bitmap object.

| Algorithm 1: Cube_Bitmap_Object_Create(VDSRow[]) |
|---|
| Input : Virtual Data Set Row Array<br>Output : Cube Bitmap Object file |
| 1: CREATE <Dimension>.cbo // Create the empty Bitmap Object file<br>2: WHILE ( NOT END OF VDS )<br>3: READ VDSRow[] into TEMP_ARR[]<br>// Assign VDSRowID to temp variable<br>4: v_ROWID = TEMP_ARR[0]<br>5: FOR (i=1; NOT END OF TEM P_ARR[]; 1++ )<br>// Assign Dimension Element to temp variable<br>6: v_TEMP_ELEMENT = TEMP_ARR[i]<br>// Search for Element in the Index Object File<br>7: IF(Element Exists)<br>THEN APPEND v_ROWID TO ELEMENT LIST<br>//Else Create a new element entry and a corresponding empty list<br>ELSE CREATE ELEMENT KEY, CREATE EMPTY LIST<br>8: END |

Fig. 9. Algorithm for the creation of the Cube Bitmap Object.

The Cube Bitmap Index object file is a human readable text file that stores the index bits in a nested array structure. There is a single parent array with child arrays as its element. Each child array represents a single row in the Cube Bitmap object file and structurally forms a bit vector. The first element of each bit vector is the dimension element; this is the key of the bit vector. The key is used to select the relevant bit vectors from the complete data set, bitwise operation are performed on the selected bit vectors to compute the resultant data set. The Bitmap Index Object has an extremely small footprint on disk.

The cube Bitmap Index object is used in the 'WHERE' clause of the analytic query. After relevant bit vectors are selected using the vector key, bitwise operation are performed on the bit vectors to provide a resultant list of row-ids of valid rows that satisfy the conditions stated in the WHERE clause.

Space complexity of the Bitmap Object: Let V be a VDS. Let |V| be the cardinality of VDS V i.e. the number of distinct tuples in V. Let D represent a dimension of V and |D| the cardinality D i.e. the number of dimensions in V. Let A represent an attribute of dimension D and |A| the cardinality of A i.e. the number of distinct attributes in dimension D. Thus, the space complexity in terms of bytes for building a bitmap object on an attribute A of dimension D is given as $|V_D|$:

$$size_{bitmap} = \frac{|A||D||V|}{8}$$

The time complexity of the bitmap object (worst case) is:

$$time_{bitmap} = O(|A||D||V|)$$

### C. *The Cube Index Object File*

The file maintains spatial information of VDS rows that have similar dimension element attributes. The Cube Index Object File is a record level inverted index. The file stores the mapping of each dimension element to its location in the data file i.e. its occurrence in the rows. However, instead of indexing the dimension element per row, the inverted index data structure indexes the rows for each dimension element. There are two constituent in each entry of the index file. The first is the dimension element which is the key of the entry and the second is a list of row-ids of the records that have the element as a dimension value. Similar to the creation of the Bitmap Index Object the Index Object file is created sequentially and incrementally, record by record, with the addition of index values for each incoming record. Each new incoming virtual data set record is processed from the left to right.

The Cube Index Object is the exclusively used in GroupBy-Aggregate process of a query. The advantage of using this particular data structure and methodology is that there is absolutely no requirement of sorting, ordering and grouping of records in the process during group by operations since the records are already grouped by dimension elements in single lists. The Cube Index Object is used in conjunction to produce the final resultant row set that is used to fetch the actual data from the Cube Data Object. The Cube Index object file is a human readable text file that stores inverted index as nested array. There is a single parent array with child arrays as its element. Each child array represents a single row in the Cube Index object file and structurally forms vector – element vector. The first element of each vector is the name of the dimension that the element belongs to. The second element of the vector is the dimension element itself. The combinations of these two elements form the key of the vector. This key is used to select the relevant element vectors from the complete data set.

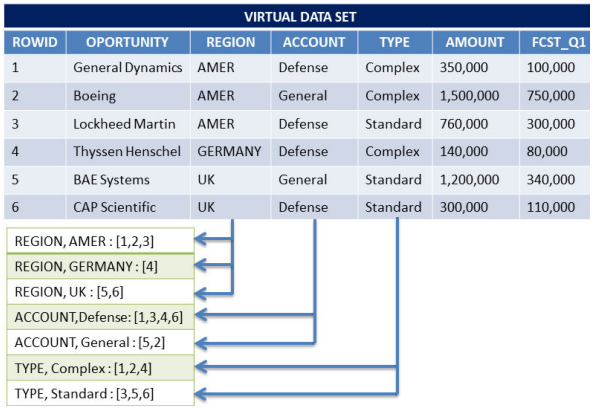



Fig. 10. Example of a Cube Index Object file, created from a sample "Opportunity" dataset. For the "Region" dimension element "AMER" the row-id list consist of rows 1, 2, 3 as AMER occurs in each of the rows. Similarly for each dimension element a corresponding row-id list is mapped.

```
[['Region','AMER',1,2,3],['Region','GERMANY',4],[
'Region','UK',5,6],['Account','Defence',1,3,4,6],[
'Account','General',5,2],['Type','Complex',1,2,4],
['Type','Standard',3,5,6]]
```

Fig. 11. Example of an Index object read as a standard Python nested list/array in the application. Every child element is a nested array in the parent array.

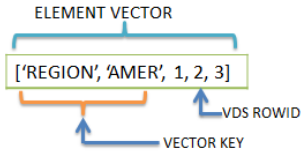



Fig. 12. Each element vector of the Cube Index object consists of two components: A vector key – that identifies the element vector and a list of row ids. In this example the rowid list is made of rows 1,2,3. The rows 1,2,3 have the Region's dimension attribute value as 'AMER'.

### *D. The Cube Data Object File*

The Cube Data object stores the measurable data against the row-id of the VDS tuples. It contains no dimension element or relationship between the fact and the dimension. The cube data object file is a pipe delimited human readable plane text file. The first row of the file contains the header for the columns: Row-id, Measure1, Measure2 etc. The application sequentially writes each of the measure values and corresponding row-id into the Cube Data Object file. Each single row is read as a single array in the application. Array operations are performed on the data arrays in conjunction with result set from the bitmap and index objects processes to provide the final result.

```
ROWID|AMOUNT|FCST_Q1|SALES|ORDERS
8792|45600|50000|45500|32
8793|60000|50000|59000|45
```

Fig. 13. Physical structure of the Cube Data object file. The first column is the ROWID followed by the measure values. Each column is separated by a delimiting special character - "Pipe".

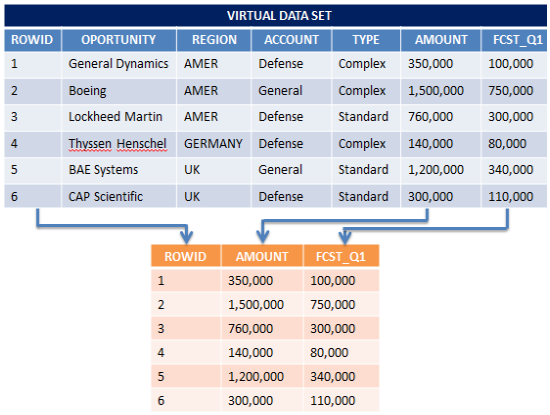



Fig. 14. Example of a Cube Data Object file, created from a sample "Opportunity" dataset. Only the ROWID and the measures(facts) are retrived from the VDS and stored as the Cube Data object. There are no reference of the dimensions in the Cube Data object.

## VII. ANALYTIC QUERY EXECUTION

A user executes an "Analytic Query" on a Cube Object to perform analysis on the data. Unlike traditional relational data bases the analytic query is procedural: it has a fixed set of steps to follow-through in order to compute the result. Since we are dealing with a single cube in each query there is no requirement for query optimization operations and execution plans. The analytic query is designed and executed from, a GUI based application, the "Analytic Query Wizard.

### *A. Analytic Query Wizard*

The Analytic Query Wizard is a GUI application designed in Python 3.4.0 to query the Cube Object for analysis. The application provides for designing all the clauses in a standard SQL statement: SELECT clause, WHERE clause, FROM clause, GROUP BY clause and the ORDER BY clause. The application generates a procedural query, accesses relevant objects, performs the necessary operations on the objects, combines intermediate results and finally displays the output of the query in a new window.

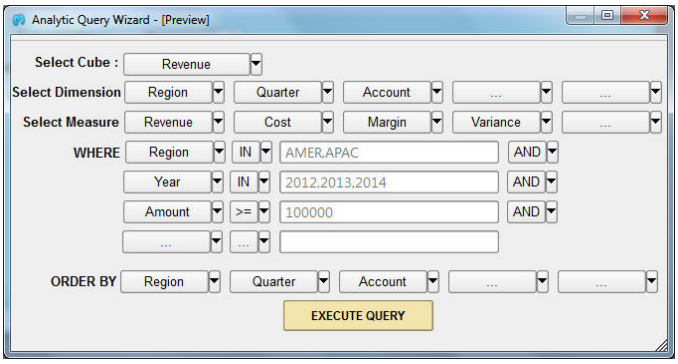



Fig. 15. Example of a dimension read a standard Python nested list/array in the application. Every child element is a nested array in the parent array.

*B. Steps in the query execution*

The Analytic Query statement is first broken into its constituent clauses and each clause is processed individually, in parallel and in sequence, and finally the result sets are combined to provide the query result set. The query maximizes parallel processing facilities of the platform and performs index processing operations simultaneously.

The steps in query execution process are illustrated with the aid of a sample VDS and a sample analytic query:

| SAMPLE - VIRTUAL DATA SET | | | | | | |
|---|---|---|---|---|---|---|
| ROWID | OPORTUNITY | REGION | ACCOUNT | TYPE | AMOUNT | FCST_Q1 |
| 1 | General Dynamics | AMER | Defense | Complex | 350,000 | 100,000 |
| 2 | Boeing | AMER | General | Complex | 1,500,000 | 750,000 |
| 3 | Lockheed Martin | AMER | Defense | Standard | 760,000 | 300,000 |
| 4 | Thyssen Henschel | GERMANY | Defense | Complex | 140,000 | 80,000 |
| 5 | BAE Systems | UK | General | Standard | 1,200,000 | 340,000 |
| 6 | CAP Scientific | UK | Defense | Standard | 300,000 | 110,000 |

# Sample Analytic Query

SELECT
REGION, TYPE ,AMOUNT, FCST_Q1
FROM REVENUE_CUBE
WHERE
REGION ⊆ {'AMER'} AND ACCOUNT ⊆ {'Defense'}
AND TYPE ⊆ {'Complex','Standard'}
GROUP BY REGION, TYPE

**STEP1:** Identify and access the appropriate Cube Bitmap object and Index Object file by using the cube name, **Revenue.*cbo*** and **Revenue.*cio***

**STEP2**: Begin simultaneous process threads in parallel, one for the Bitmap object and one for the Index object.
**STEP2(A):** Processing the Bitmap Object File for computing the conditions specified in the WHERE clause.

The requisite bit vectors from the Revenue.cbo file are read as arrays in the application. Bit wise operations are performed on the array element to compute if a particular row meets all the conditions stated in the WHERE clause of the file dimension rows that are present in the where clause of the query. A ROWID vector is generated for the rows whose result bit is true. For the sample VDS and the sample analytic query under consideration the operation is as illustrated:

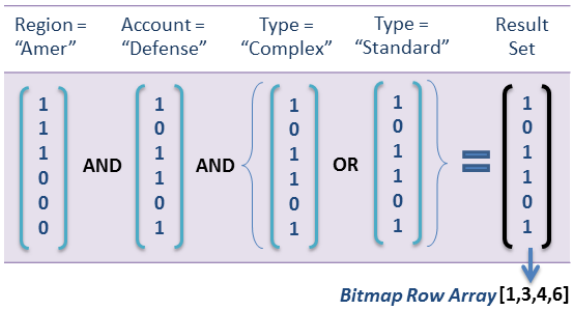



Fig. 16. Bitwise logical operation on Bitmap bit vectors to generate a result set that represents rows that satisfy the WHERE conditions. The Result set is then mapped to the actual ROWID's to generate Bitmap Row Array.

**STEP2(A):** Processing the Cube Index Object File for computing the conditions specified in the GROUP BY clause.
The Cube Index Object file is used in grouping the dimension elements. The file maintains spatial information of VDS rows that have similar dimension element attributes. The application selects the element vectors from the file that are specified in the GROUP BY clause of the analytic query. For the sample VDS and sample query: Element vectors with vector keys – AMER, GERMANY, UK, Complex & Standard – corresponding to the group by clause of grouping data by Region and Type.

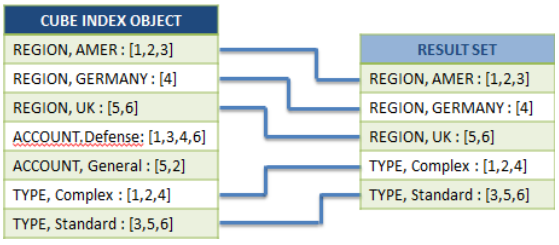



Fig. 17. A subset of element vectors, which have their keys matched to the dimensions specified in the group by clause are selected from the Cube Index object

**STEP3:** Processing the resultant datasets from the Cube Bitmap process and the Cube Index process (STEP2(A) and STEP2(B)) to generate intermediate arrays consisting of row-ids that are common to both the datasets.

Each Element array from the Cube Index object process is intersected with the single ROWID vector from the Cube Bitmap object process to select the row-ids common to both the result sets – generating a set of arrays that satisfy both the WHERE and the GROUP BY clause of the analytic query. These arrays, called the Data Row Arrays, are temporary intermediate data sets that are further processed in the 'Aggregate Data Row Array' process before being finally used with the Cube Data object.

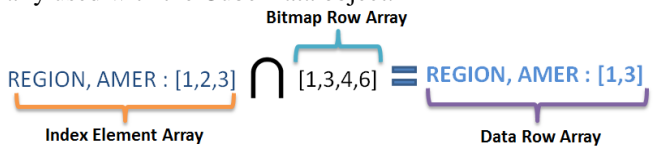



Fig. 18. Illustration of the Join operation of the AMER Index Element Array with the Bitmap Row Array. The rowids 1,2,3 are common to both set and thus are the only ones available in the resultant Data Row Array

| **Algorithm 1: Process Bitmap and Index** (IdxElmtARR[],BitRowARR[]) |
|---|
| **Input : Element Index Arrays, Bitmap Row Array**<br>**Output : Resultant Element Group Arrays: DataRowArr[]** |
| 1: WHILE (NOT END OF IdxElmtARR[])<br>// READ a single Element Index Array into temporary array<br>2: TEARR[] = IdxElmtARR[]<br>// Compare the rowids in the Element Array and the Bitmap Rowid array and allocate the resultant rowids to a new output array : DatRowGrpARR<br>3: DatRowArr[] = INTERSECT(TEARR[], BitRowArr[])<br>4: RETURN(DatRowArr[])<br>5: END |

Fig. 19. Algorithm for the creation of the Cube Bitmap Object.

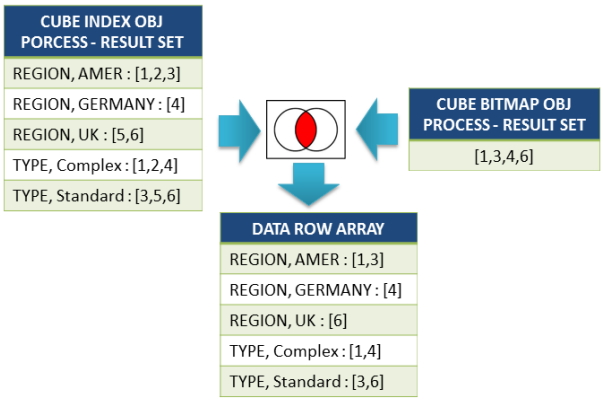



Fig. 20. Resultsets from the output of the Cube Index object and the Cube Bitmap object processes are joined on the rowids' to produce Data Row Array.

**STEP4:** Grouping and organizing the Data Row Arrays into Grouped Data Row Arrays.

The element arrays from one of the dimension is intersected with each element array from other dimensions to generate dimensionally grouped arrays that contain a list of row-ids that are dimensionally grouped.

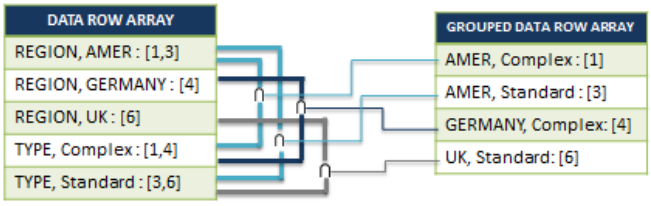



Fig. 21. Data Row Array elements of the "Region" dimension are joined with elements of the "Type" dimension on the rowids' to generate dimension combinations of element that meet the conditions stated in both the GROUP BY and WHERE clause.

**STEP5:** Processing the Cube Data object to retrieve measure data, combine the measure data with appropriate dimensional groups and present the resultset to the user.

In this final step of query processing. Each Grouped Data Row Array is queried against the Cube Data Object to retrieve the rows. The measure values from the Cube Data Object is stitched with the respective dimension groupings of the Grouped Data Row Arrays to finally generate the appropriate Analytic Query output.

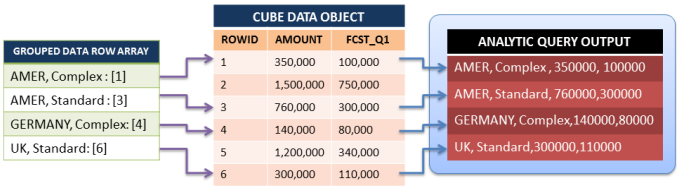



Fig. 22. Grouped Data Row Arrays are joined with the Cube Data object on the ROWIDs' resulting in the combination of the dimension elements in the grouped data row array with the facts in the Cube Data object.

## VIII. PERFORMANCE: COMPARATIVE LOAD TESTING

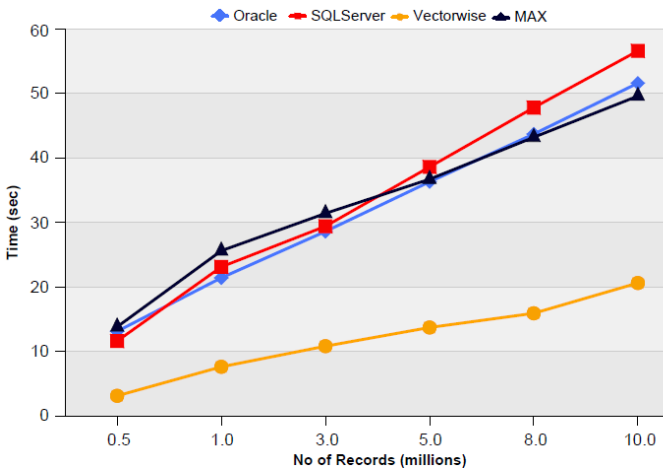
To properly understand the performance tradeoffs between MAX and traditional databases, relational and columnar, a comparative benchmark of the platforms under exact hardware specification at different workloads for query latency and resource consumption was performed. The following interesting aspects: (1) Query latency and resource consumption for varying data loads; (2) Query latency and resource consumption for varied number of dimensions ;(3) Query latency and resource consumption for varying dimension cardinality; Finally we look at the space efficiency of the platform in comparison with others.

A Windows Server 2012 R2 on Intel Xeon E5-2687WV2 (8 cores, 3.40 GHz) with 32 GB (DDR3-1333) memory was chosen as the suitable platform to perform the comparative performance analysis. Two of the industry

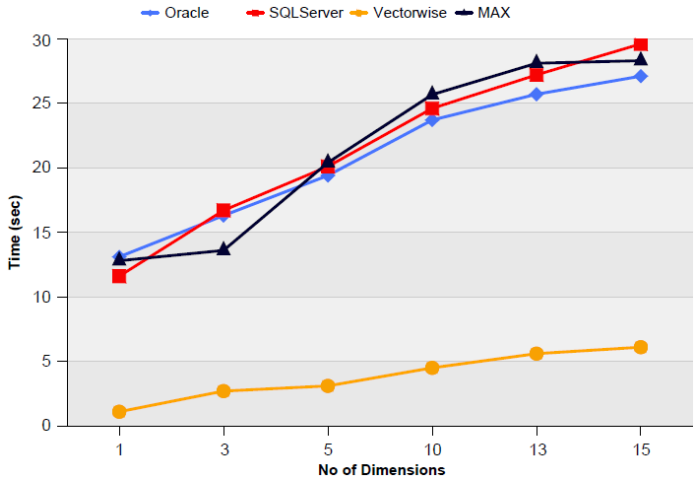
leading relational databases – Oracle 11g, SQL Server 2012 Enterprise Edition and a columnar database – Vectorwise 3.0 were chosen as suitable RDBMS for the purpose of comparative testing. DELL Benchmark factory was used to evaluate the databases with Ingres ODBC driver for Vectorwise 3.0. The python psutil was used to retrieve performance statistics and resource utilization for the MAX platform. Inspiration for the benchmark was drawn from the industry standard TPC-H benchmark framework. While such tests can never take the place of proof of concepts done using the exact use cases and infrastructure that a new platform is targeting, they are useful indicators of the real-world viability of the application.

The synthetic data set use for the benchmarking simulates data from a real-world sales opportunity generating cloud based CRM system.

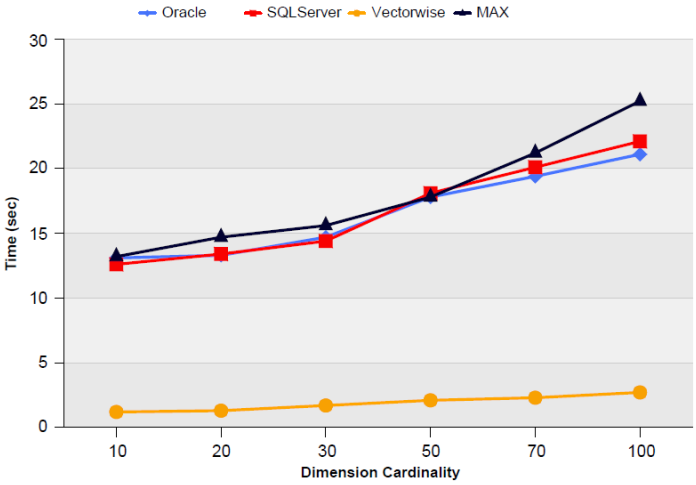
CASE I: Mean Query latency for varying number of records – This test examines the time taken to execute a query for difference number of records. The query latency is measured in seconds and the numbers of records are in millions.



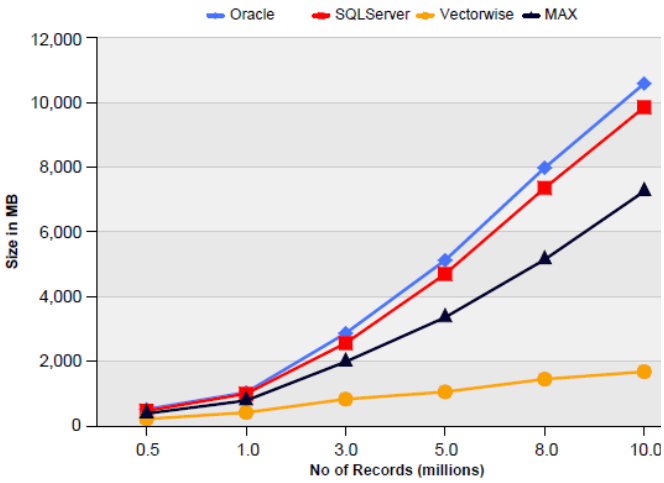

CASE II: Mean Query latency for varying number of dimensions, for constant number of records. This test examines the time taken to execute a query for different number of dimensions for a fixed number of 5 million records .The query latency is measured in seconds.




CASE III: Mean Query latency for varying number of dimension cardinality, for fixed number of dimensions, and fixed number of records. Number of records – constant at 5 million, number of dimension fixed at 10.




CASE IV: Data base size comparison for varying number of records. Data is consistent for all platforms. The total database size under consideration is the sum of the space occupied by the database and the index on disk. The size on physical disk is measured in Megabytes(MB).




## IX. CONCLUSION

The novel approach illustrated in this paper provides insight into the innovative ways of storing and retrieving data. The platform has tremendous potential for further improvement, development and introduction of additional features. After the successful development of the proof of concept on the python platform, as illustrated in this paper, currently, an extended version of this platform is under development using C++ on the LAMP stack. Simultaneously new desirable features: better web integration, user data access security, enhanced SQL features like order by, IN etc. and temporary cache storage are being researched upon to be integrated on the MAX platform.

## REFERENCES

[1] M. T. ¨Ozsu and P. Valduriez, “Distributed Database Design”, ”Design Issues”, “Decomposition & Data Localization”, “Query Processing”, in *Principles of Distributed Database Systems, 3rd ed.* Springer, New York, 2011, pp. 46-138

[2] Jan Palach, “Parallel Algorithms”, “Threads and concurrent features”, ”Multiprocessing and process pool”, in *Parallel Programming with Python,* Packt Publishing, Birmingham, UK, 2014, pp. 11-93

[3] Priscilla Walmsley, “Selecting and joining in FLWORs”, “Advanced Queries”, in *XQUERY – Search Across a variety of XML data*, O'Reilly Media, Sebastopol, California, 2007.

[4] Mark Summerfiled, “File Handeling”, “Processes and Threading”, “Networking”,” GUI Programming”, in *Programming in Python 3 , 2nd ed.* Addison-Wesley, Boston, USA, 2010, pp. 289-476

[5] Brian Benz, John R. Durant, “XML Parsing With DOM”, “XSL Transformations”, “SOAP”,”WSDL”, in *XML Programming Bible,* Wiley Publishing, Inc., New York, 2009, pp. 3-291

[6] Python Software Foundation | Python.org In https://docs.python.org/3/

[7] XML path language (XPath) 2.0. In http://www.w3.org/TR/xpath20.

[8] XQuery 1.0: An XML query language. In http://www.w3.org/TR/xquery.

[9] XQuery 1.0 and XPath 2.0 formal semantics. In http://www.w3.org/TR/query-semantics.

[10] XQuery 1.0 and XPath2.0 data model. In http://www.w3.org/TR/query-datamodel.

[11] XQuery implementation. In http://www.w3.org/XML/Query#implementations.

[12] Stefano Seri, Christos Faloutsos, Richard T. Snodgrass, V.S. Subrahmanian, Roberto Zicari,”Temporal Databases”,”Complex Queries and REasoning”,”Indexing Methods”,”Multimedia Indexing”, in *Advanced Database Systems,* Morgan Kaufmann Publishers, Inc., California, USA, 1997, pp. 97-265, 295-314

[13] D Florescu and D. Kossmann, “Storing and Querying XML Data using and RDBMS” *IEEE Data Engineering Bulletin, 22(3)*, 1999, pp. 27–34