DA-IICT

NoSQL

Supervisor: Prof. PM Jat

SciDB: Scale Array Storage, Processing and Analysis

9th May 2020

Aniket Kaduskar(201701005)

Yashvi Shah(201701045)

Bachelor in Technology

Information and Communication Technology

Semester 6

1 Introduction

1.1 Topic

SciDB is an open-source systematic DBMS that is used to solve the data administration needs of scientists who deal with petabytes of data on regular basis. For the fields like astronomy, climate modeling, financial services sector and stock markets, bio-science information management, and the analysis of web data generated by e-commerce website, the amount of data generated is huge and a proper database management system is needed.

1.2 Motivation

Organizations are currently using RDBMS i.e relational database management system for their database management needs but this database model is not useful as the needs are constantly increasing in today's big data world especially in the scientific applications and the analysis of a large data obtained from sensors. NoSQL databases have emerged to help database users solve the problems associated with the relational database model. The scientific data generated by the sensors is different from the commercial data in the following aspects:

- They consist of rectangular arrays of individual sensors and the data collected has a necessary and implicit specific ordering.
- Analysis on scientific data is complex, both mathematically and algorithmically and required advanced data processing methods. Along with this, the sensor data is noisy so it needs to be heavily pre-processed and cleaned before it becomes the input to complex analytical processing.
- With the advancement in technology the manufacturing of sensors have increased greatly and also they can be used multiple times, so the data generated by modern scientific devices is extremely large.

For RDBMS systems we need to specify the schema and store it in the database, while for the array data model such specifications are not required as it is indirectly ordered. Also to deal with the increasing complexity of data processing, the users require an extensible DBMS platform apart from those currently available and SciDB proves to be perfect for the required functionalities. SciDB is one of such first and foremost system for the storage, processing and analysis of such huge scale data.

1.3 Requirements

Scientists across the various domains and the representative from different web platforms have reported the following requirements for the design of a database which will be used for scientific data analysis and processing:

1. Multi-petabyte amounts of data: The scientific applications generate petabyte scale array data from domains such as astronomy, climate modeling, commercial applications like risk management system in the financial services sector and stock markets etc.

2. A preponderance of array data: Geo spatial and temporal data such as oceanographic telescope data, satellite images, and most simulation data including genomics data generated from high throughput sequencing machines, all are naturally modeled as arrays. The sophisticated capabilities are required by the scientist in domains like the satellite imagery where the data could be in various coordinate system and also of different resolutions, and such data needs to be re-oriented in order to perform data correlation from multiple satellites.

3. Complex Analytic: The sophisticated capabilities are required by the scientist in domains like the satellite imagery where the data could be in various coordinate system and also of different resolutions, and such data needs to be re-oriented in order to perform data correlation from multiple satellites. There could be various disturbance in the images like clouds, which should be removed to obtain a cloud free image from multiple instances captured and complex operations are required for such a application.

4. Open Source Code: Often times the community requires to add their own features depending upon the application and fix the bugs if proper service is not provided by the application provider which requires the software to be open source. The closed source software are hated by the community due to the following reasons:

    - A multi decade support is required by the large projects in the scientific community.
    - The whole software stack cannot be recompiled without the permission from the application provider.
    - There are problems with maintaining a closed source software when large collaborations are involved which include hundreds of institution.

    In effect, only open source software is acceptable which can be used for immediate debugging and changing the implementation according to requirements.

    The scientific data is never thrown away, not even the old data. For example, the data about the earth are of no use at present to the scientists but that could change in the future, so throwing away any kind of data is not an option. The data with errors that have been corrected is also kept. The main reason behind this is to re-analyse the data and establish a comparison with the current data, hence the data needs to be present indefinitely.

5. No Overwrite: The scientific data is never thrown away, not even the old data. For example, the data about the earth are of no use at present to the scientists but that could change in the future, so throwing away any kind of data is not an option. The data with errors that have been corrected is also kept. The main reason behind this is to re-analyse the data and establish a comparison with the current data, hence the data needs to be present indefinitely. A dimension of history is added to updatable arrays so as to keep a track of new updates to the previously existing data.

A large amount of scientific data is generated from the sensors observing the physical process. The data(raw data) is then processed which is called cooked data which contains the finished information. This data is cooked by converting it into standard data types, correcting for errors and cloud servers etc. There are two approaches that are used for cooking data:

6. Integration of Cooking Process:A large amount of scientific data is generated from the sensors observing the physical process. The data(raw data) is then processed which is called cooked data which contains the finished information. This data is cooked by converting it into standard data types, correcting for errors and cloud servers etc. There are two approaches that are used for cooking data:

   - The cooking can be performed inside the DBMS by first loading the data into it and then perform the processing.
   - The data can be cooked externally in some hardware and then load to the DBMS.

   SciDB allows the user to perform cooking inside the DBMS. It can be done by loading the data into the DBMS and then using the User Defined Functions, data processing and manipulation operations to operate on data. Hence a strong data manipulation capacity is required by the DBMS.

7. Named Versions: One of the important requirements by the scientific community is the concept of named versions. Using this functionality a change can be made to the small portion of the data while maintaining the rest of the data in arrays. Using named versions the cost of copying and the space required to store that copy is saved. The versions are stored as an offset of their parents in SciDB. They also do not consume space until some updates are made(4).

8. Provenance: The scientific community requires a mechanism where they can trace back through various derivations to determine the previous data is anything wrong is suspected. This way they will be able to detect the fault and subsequent faults raised due to it. The search requirements for an element x can be:

   - Find the processing steps that are generated from the given input data x.
   - Find the data whose values are impacted by change in the value of a given input data element x.

9. Uncertainty: All the scientific data contains imprecision. To store such data the DBMS needs to support data elements that are uncertain. SciDB provides functionality that can store "uncertain x" for any data type x that is available. Two values needs to be specified for such data type i.e value and error, which helps the calculation to be more precise.

10. Version Control: There is no standard regarding the cooking of the unprocessed data to derived data sets. Therefore the scientist require a functionality where they can re-cook the data for their specific study. Thus version control software would be required.

11. In-situ Data: There is a common problem with scientists that it takes a lot of time to load the data. A high cost needs to be paid for loading the data which often exceeds the value received from using the DBMS. SciDB does not require to load the data as it operates on in-situ data.

2 Statement of Purpose

In today's world, people mostly use relational tables for their data management needs. There have been very few people who actually require tables for storage. It is not a natural data model for them. Also very few are satisfied with SQL as their language interface. It turns out that the array data model is a default data model for a large part of the users and more specifically in domain like astronomy, climate modeling, commercial applications like risk management system in the financial services sector and stock markets, bio-science in-formation management, and the analysis of web data generated by e-commerce website. Also a table is a one dimensional array with a primary key. Thus arrays can also satisfy the needs of people who are happy with the tables.

Our term paper describes an array data model as it is really useful and serves a large part of the community for their data management needs. The system also supports different dimension arrays and nested arrays where each cell could be an array itself.

The term paper is organized in the following structure:

- Section 1: Describes the motivation for using SciDB over other databases especially in the field of science where large amount of data is generated and requires mostly linear algebraic operations for processing. Also it describes the different needs of today's scientific community which are solved by SciDB in the later section.
- Section 2: Emphasizes SciDB as the need of the hour and the reason why SciDB should be developed for future needs.
- Section 3: Describes the features and functionalities used in SciDB that sets it apart from other database management system.
- Section 4: Describes the architecture of SciDB along with the storage manager, chunking and partitioning techniques.
- Section 5: This section discusses the insight and some additional information that supports the advancement of SciDB.
- Section 6: Concludes the term paper.
- Section 7: Discusses the future works and the ongoing development in this field.

3 Features and Functionalities

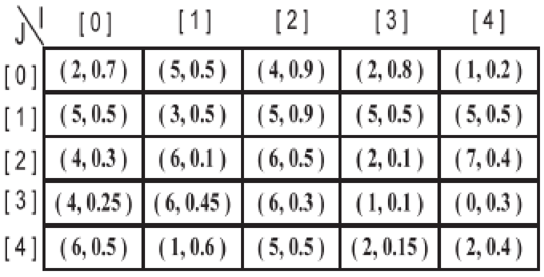
SciDB adopts an array data model and is a **column oriented database management** system. The database is a collection of n-dimensional arrays. The cells of an array contain a tuple of values and each of this value in the tuple have a unique attribute name associated with it.

3.1 Data Model

SciDB uses an array data model instead of a table(2) because of the following three reasons:

1. Arrays are the natural data objects for most of the sciences.
2. They prove to be faster than Relational DBMS for a science workload.
3. Complex analysis done on the the scientific data is mostly linear algebra operations like matrix multiplication. These operation can be done in a simpler way using an array and using a table would require to convert it to array and back again to table which increases the processing time.

The arrays could be of any dimension in SciDB. It could be general integer values with any start and end point. It could also be unbounded in both the directions. Along with this, SciDB also supports the dimension to be of any user defined type.



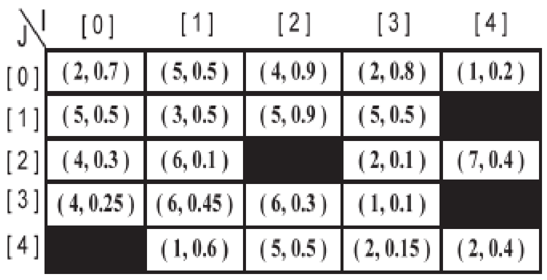
J\I	[0]	[1]	[2]	[3]	[4]
[0]	(2, 0.7)	(5, 0.5)	(4, 0.9)	(2, 0.8)	(1, 0.2)
[1]	(5, 0.5)	(3, 0.5)	(5, 0.9)	(5, 0.5)	(5, 0.5)
[2]	(4, 0.3)	(6, 0.1)	(6, 0.5)	(2, 0.1)	(7, 0.4)
[3]	(4, 0.25)	(6, 0.45)	(6, 0.3)	(1, 0.1)	(0, 0.3)
[4]	(6, 0.5)	(1, 0.6)	(5, 0.5)	(2, 0.15)	(2, 0.4)

Figure 1: Array in SciDB

A combination of dimension defines a cell in the array. It can hold any number of attribute of any user defined type. But all the cells in an array should be of the same type i.e the array should be uniform.

The array is called empty if all of the cells in it are left undefined. Arrays with empty cells are referred to as **sparse arrays** in the SciDB. We can also mark the individual attribute values as missing with several of the possible SciDB null codes. An array could also have uneven edges along with a group of empty cells and the surrounding cells would contain the actual value. Empty cells are usually ignored during computation. Cells in SciDB have the following data types: `characters, strings, float64, float32, int8, int64, int32, int16, uint32, unit64, uint8, uint16, and bool`

SciDB also supports a nested array where a cell in an array could contain another array. This helps in finer



J \ I	[0]	[1]	[2]	[3]	[4]
[0]	(2, 0.7)	(5, 0.5)	(4, 0.9)	(2, 0.8)	(1, 0.2)
[1]	(5, 0.5)	(3, 0.5)	(5, 0.9)	(5, 0.5)	■
[2]	(4, 0.3)	(6, 0.1)	■	(2, 0.1)	(7, 0.4)
[3]	(4, 0.25)	(6, 0.45)	(6, 0.3)	(1, 0.1)	■
[4]	■	(1, 0.6)	(5, 0.5)	(2, 0.15)	(2, 0.4)

Figure 2: Array with jagged edges

decomposition of cells to obtain a need specific information.
Hence, an example array specification in SciDB is:

```
CREATE ARRAY Example
( A::INTEGER, B::FLOAT ) [ I=0:5, J=0:7];
```

The above example contains an array with attribute A and B along with dimensions M and N. The array can be addressed as:
`Example[ 4, 5]` — indicates the contents of the (4, 5)th cell
`Example[M = 4, N = 3]` — more verbose notation for the (4, 3)th cell
`Example[4, 5].A` — indicates the A value of the contents of the (4, 5)th cell

3.2 Data Manipulation

There are two types of query language supported by SciDB(2):

1. Array Funtional Language (AFL).
   Array functional language has a collection of operations like join, filter, subsample etc. The user can use combination of various operations to obtain the needed result. The AFL consists of around 100 functions to perform array processing. Some of the major and frequently used functions include:

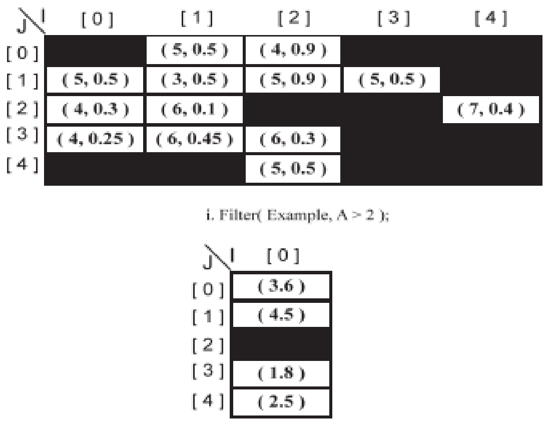
   (a) Filter() is applied to obtain a part of array depending upon the specifications applied. It produces an output array of same size but the cells for which condition is false are set to empty.

   ```
   temp = filter (A, c = value)
   result = join (B, temp:  I, J)
   ```

   or it can be written in composite form as

   ```
   result = join (B, filter (A, k = number), M, N)
   ```

where A and B are array with dimension M and N, k is the attribute of A.



J \ I	[0]	[1]	[2]	[3]	[4]
[0]		(5, 0.5)	(4, 0.9)		
[1]	(5, 0.5)	(3, 0.5)	(5, 0.9)	(5, 0.5)	
[2]	(4, 0.3)	(6, 0.1)			(7, 0.4)
[3]	(4, 0.25)	(6, 0.45)	(6, 0.3)		
[4]			(5, 0.5)		

i. Filter(Example, A > 2);

J \ I	[0]
[0]	(3.6)
[1]	(4.5)
[2]	
[3]	(1.8)
[4]	(2.5)

ii. Filter(Slice() , A * B);

Figure 3: Array in SciDB

(b) Slice() is used to obtain a part of the array for a particular index in a single dimension. In the example given below the slice() operator will return a single column, where all the cells have column index value is 2.

```
Slice ( Test, M = 2);
```

(c) Subsample() is a generalization of Slice(). Instead a slice of the array it will extract a region of the array.

```
Subsample (
Example,
I BETWEEN 1 AND 3 AND J BETWEEN 2 AND 4 );
```

(d) SJoin() combines cells from two input arrays. The cells with same dimensional index values are combined. The input array of Sjoin() does not necessarily have identical dimension. It is a binary array operator.

```
SJoin (
Subsample (
Example,
I BETWEEN 1 AND 3 AND J BETWEEN 2 AND 4 ),
Slice ( Example, I = 2) );
```

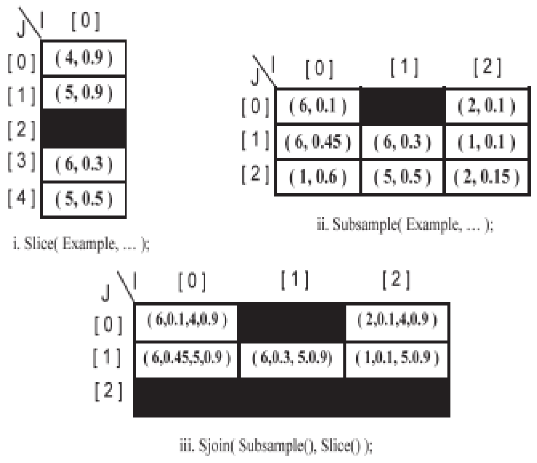


Figure 4: Array in SciDB

(e) Cjoin operator will combine two arrays. For example, if there are two arrays with dimension M and N then Cjoin would return an array with M + N dimension with concatenated cell tuples wherever the join condition is true.

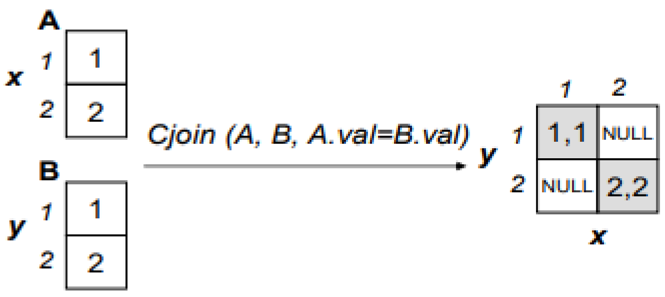


Figure 5: CJoin

(f) Apply() is used to perform calculations on the attribute values in the array and it would return an array with new values.

```
Apply( Slice ( Example, I = 2), A * B );
```

(g) Reshape() converts the array to a newer dimension with the same number of cells.

(h) Exists() is used to find out whether a given cell (e.g., [7,7]) is present or not in a 2-dimensional array A. [A, 7, 7] which returns true if [7,7] is present and false otherwise

(i) Aggregate takes input as a n-dimensional array A, and also a list of k grouping dimensions G, and an aggregate function Agg as arguments. The Aggregate function takes an argument that is an (n-k)-dimensional array since there will be one array for all combinations of the grouping dimension values.

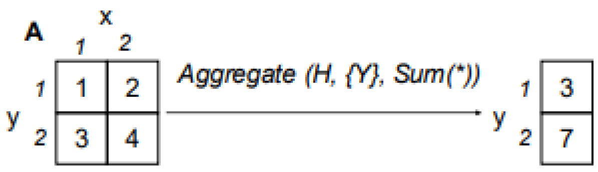


Figure 6: Aggregate

Further the operators can be classified in two types structural and content-dependent. The first operator category creates new arrays based on the structure of the inputs. Thus we can say that these operators are data-agnostic. These operators can be used for optimization as there is no need for them to read the data produced. Structural operators include Subsample, Reshape, SJoin, add dimension, remove dimension, cross product, and concatenate. While content dependent operator are those whose results relies upon the stored data in input array. Content-dependent operators includes filters, CJoin and Aggregate.

2. Array Query Language(AQL).
   AQL is compiled into AFL. The AQL language includes two types of queries:

   - Data Definition Language (DDL) : commands to define arrays and load data.
   - Data Manipulation Language (DML) : commands to access and operate on array data.

   The above example can be written in query form as:

```
Select * from A, B
where A.I = B.I and A.J = B.J and A.c = value
```

3.3 Extensibility

Database Management System should have an inbuilt functionality for processing data instead of external computation. Each scientific domain have their own set of algorithms and unique data types for data, so it is intuitional that the DBMS should support user-defined extensibility. In order to satisfy the non business requirements of the users SciDB supports user defined data type and user defined functions similar to the type found in SQL DBMS systems. With the help of extensibility the user will be able to add data types that are domain specific. These data types will be able to inter operate with the data types of SciDB and also the different operators. There are 4 extensions available to support the extensibility.

1. **User-defined Data Types:** These data types provide an extension over the normal data types available in SciDB like the integer, string and float. Thus the attribute values of the cell in SciDB could be user defined.

2. **User-defined Functions:** The database management system should also provide a functionality to perform operations on the user defined data types. These functions should accept one or more arguments of various data types and produce an output accordingly. These functions are written in C++.
   For example a function Multiple10 can be used to multiply array dimension with 10 as(3):

   Define function `Multiple10 (integer M, integer N)`
   returns `(integer I, integer J)`
   `file`

   The file contains the object code for the function and it would contain the link to call concerned function when needed.

3. **User-defined Aggregates:** S User defined aggregates can be written for both user defined as well as the default data types. These aggregates require 4 functions similar to Postgres. Out of this, 3 functions are Increment(), Init() and Final(). They are required to calculate single node user defined aggregates. SciDB is a multinode system. The functions mentioned above will be executed at each node. The rollup() function would pull combine various aggregates into a final outcome.

4. **User-defined Array Operators:** These functions are such that they have arrays as their input arguments as well their return type. A functionality called Gaussian Smoothing, which produces a 'weighted average' of a particular cell's neighbourhood for each cell in the array with the average is more weighted towards the value of nearby cells is an example of user-defined array operator.

4 Architecture

The storage of a SciDB instance has its local, and is spread over a large network of computers. Each of the node can undergo query processing,can support communication, and has a storage manager. SciDB has a **shared nothing design** type of system architecture like all the recent parallel DBMS's where each node interacts with its locally attached storage.
Portions of query are executed on the data on local basis the query optimizer. The operations in scientific applications are computation intensive, such as matrix multiplication, which cannot be executed parallelly. So they cannot be executed faster on an architecture like SciDB. As an important objective of SciDB is petabyte scalability, SciDB was decided to be a shared nothing engine.

A shared centralized system database that stores data of every node can be accessed by the processes which are running on each node and also stores information about data distribution. SciDB uses a Map/Reduce model like the modern distributed computing systems. Physical nodes can leave and join, but until a query operates on data stored on an offline node, it remains unaffected.

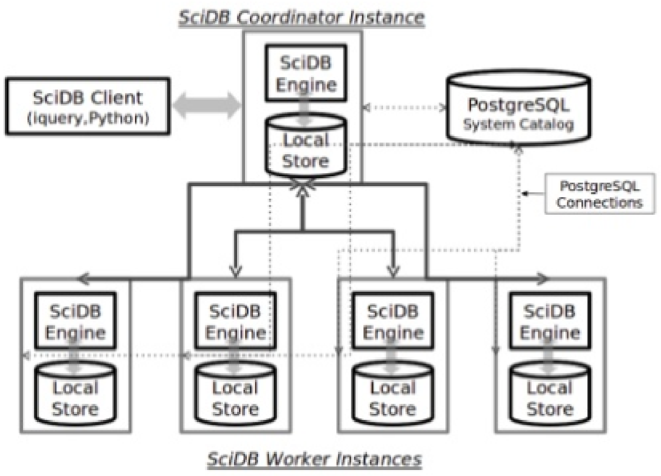


Figure 7: Architecture of SciDB

4.1 STORAGE MANAGER

SciDB has a no-overwrite, distributed storage manager. Data in a SciDB array is not updateable. It provides the functionality of a storing an array as well as the result of a query to the storage. SciDB obeys only the 'A' and 'D' of transactional ACIDity i.e Availability and Durability. In addition to this it also provides a mechanism to access external data with the help of operators,in order to avoid the cost of loading the data. It is really useful when a particular part of data is not repeatedly used and the efforts worth loading it are more than result.

1. **Chunking and Vertical Partitioning:**

In column-store storage managers, the data in arrays is vertically partitioned. The storage manager splits attributes in one logical array and handles values for each attribute individually. The basic operations deal with arrays that contain a single value in each cell. In SciDB the partitioning changes over time: e.g., one partitioning scheme is used for time less than t and another partitioning scheme for time greater than t. This is useful in a queries, on a sub-set of attributes in the logical array hence reducing the I/O costs. The storage manager takes data from each attribute, and decomposes the array into a number of equal sized and overlapping, chunks.

The chunks are the physical unit of I/O, processing, and inter-node communication. Chunks are quite large and are on an order of 64 megabytes. The chunk's location is stored as a range of dimensional indices within the logical array within the system catalog. The chunks are distributed among nodes using hashing, range partitioning, or a block-cyclic algorithm.

Also the chunks should be sufficient enough to serve as the unit of I/O between the disk and the buffer. In such a case, the CPU time can often be reduced by splitting a chunk internally into tiles. So now the sub-queries may be able to operate only a part of a chunk, and reduce the total time. Hence it is implemented here in a two level chunk/tile scheme.

2. **Fixed and Variable Size Chunks:**

   One possibility is to fix the size of the stride in each dimension, and hence creating logically fixed size chunks. The size of data in each chunk can vary because of data skew and different compression schemes. The first option is fixed logical size but variable physical size chunks. The second possibility is to support variable size chunks. In this case, we fill a chunk to a fixed capacity, and then close it, and so a chunk has a variable amount of logical array data. Variable chunk schemes implementation would require an R-tree or other such indexing schemes to keep note of chunk definitions. R-tree is a tree data structures used for spatial access methods, i.e., for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons.

3. **Overlapping chunks:**

   If the data is partitioned into chunks which are non-overlapping, the system would have to stitch together adjacent chunks to reconstruct the boundaries of the chunks. The breaking of of the arrays into overlapping chunks, and by choosing the right extent of overlap, we can parallelize various operations. So in this case. the data which is needed to filter is there within the same chunk. The flipside of this strategy is that it increases the storage management overhead, and also presents a challenge in configuration of the SciDB Databases. As shown in the figure below is an 11x5 array which has been decomposed into three, overlapping 5x5 chunks. With this scheme, an operation that requires the examination of any complete 3x3 subsample of A need only consult the data in a single chunk and this operation can be parallelized. The darker grey areas are regions of the array that are replicated and a particular attribute value in a cell stored in more than one chunk. Lighter gray areas represent regions of the array which are non-core in the sense that any algorithm considering a 3x3 subsample of A will be unable to compute results for these boundary cells.

4. **Distribution:**

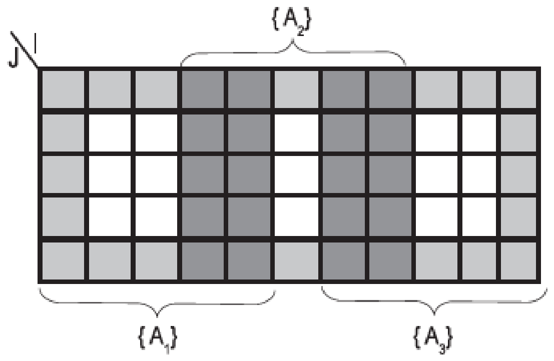


Figure 8: Overlapping Chunks

Chunks are allocated to nodes in such a way that all the chunks containing data values for a particular cell of the array are allocated on the same node. In SciDB the records in its system catalog adds an entry for each chunk. The query processor gets to knows from the system catalog what region of the array each chunk spans and what attribute it contains. Postgres is an open-source DB which is used as a system catalog repository manager because this uses R-Trees which helps to quickly identify chunks that contain data relevant to a particular subsample of the array and improving query planning and execution.

5. **Compression:**
In SciDB, the chunk size can be very large like 64 megabytes or so. In many of the scientific use-cases, the storage requirements are met with using compression. Chunk is individually compressed when writing it to disk, and uncompressed as read. It is possible to work on uncompressed data directly for some operators. It is also used to cut down the requirements of bandwidth of the Scatter/Gather operationsin networks. The different types of compression methods include delta encoding, run-length encoding, subtracting off an average value, Null Suppression, Run-Length Encoding, Lempel-Ziv, Delta Encoding and Adaptive Huffman. If the chunk undergoes frequent updates or too small geographic queries, then much overhead will be spent in decompressing and recompressing chunks. In this case, it is a good idea to divide a chunk into tiles, and compress each tile independently. So, only relevant tiles need to be decompressed and recompressed to support these kinds of queries and updates.

6. **Version Control:** The issues in which SciDB involves version management are:

   (a) A huge amount of data is based upon time

   (b) The previous old data cannot be thrown away

   (c) The transformation of raw data into derived information

   To support the no overwrite model, versioning is imposed in all SciDB arrays. Data is loaded into the array at the time indicated in the loading process. There are inserts, updates, or loading of new data at the time they are run, also keeping the previous data. The versioned data which is identified by the version number can be scanned by the query.

The latest version of the chunk are stored contiguously. The earlier versions of the chunk are used by a sequence of deltas referenced from the base chunk. A given chunk is saved as a base plus a sequence of "backwards deltas". It is always the case that users want the latest version of the chunk and also in an optimized manner.
The processing of a query should start with searching for the version having relevant data with respect to a given query. If we don't get the data in the current version, then its parent is searched, eventually searching in the stored array from which the version was derived. This architecture is somewhat similar to the configuration management systems, which also implement similar functionality.

4.2 Query Processing

The plan generator and the parser should consult the system catalogs to proceed with the analysis of the query expressions in SciDB. Each coordinate node in SciDB performs various operations for solving the queries like parsing, type checking, lookup of user defined extensions, semantic checks etc. The execution of fragments and the query pan is then sent to the other nodes. A physical plan would be generated by the query planner at the node with the help of a logical plan tree.

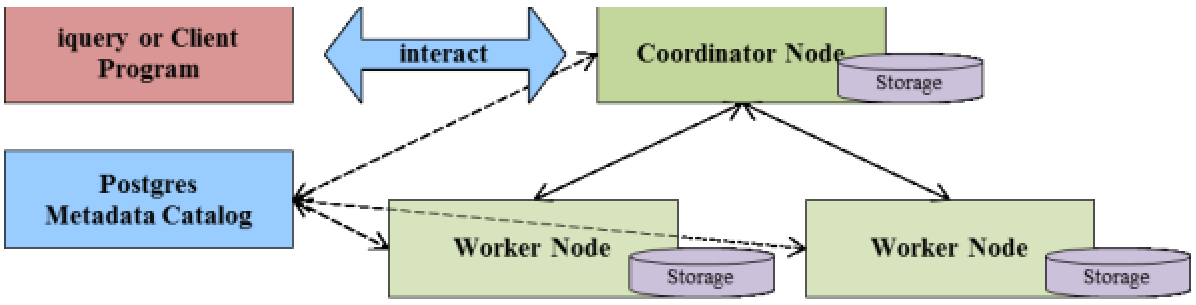


Figure 9: SciDB Query plan execution

Queries are solved using optimizer and executor. AQL is used by the users for queries and updates. There are several guiding principles in the design of this components in SciDB.

1. Obtain parallelism with minimal data movement.
   In case of the operations not being conducted in parallel fashion, redistribution of data is done. The optimizer processes the query parse tree in two stages.
   (a) Firstly, the tree is examined for commutative operations. It is a common operation as filter and join all are commutative operators. SciDB optimizer pushes the cheaper commuting operation down the tree.
   (b) The next step is to examine for blocking operations in the tree. A blocking operation is one that either requires a redistribution of data in order to execute, or cannot be pipelined from the previous operation, in other words it requires a temporary array to be constructed.

It is not possible to parallelise all the queries. For this, SciDB has implemented a special operator pair called the Scatter/ Gather (S/G).

2. Better query plans can be constructed using the incremental optimizers which have more accurate information.
   The SciDB optimizer is incremental, such that the best first subtree is picked and executed. After this sub-tree is executed, there is a perfect approximation for the size of the result, and the next subtree is chosen based on this. The disadvantage is that SciDB has a run-time optimizer.

3. Use a cost-based optimizer.
   Since SciDB only plans sub-trees, the cost of exhaustive evaluation of the options is not onerous.

5 Additional Information

5.1 Why Hadoop cannot be used?

Hadoop alone is not a DBMS. Modules like Pig, Hive, HBase are loosely integrated an require a lot of glue code(9). Hadoop is comparitively slower than a parallel distributed database. Furthermore, linear algebra operators are hard to operate as map and reduce. Thus SciDB is a better choice for scientific applications than Hadoop.

5.2 Non-Science Usage

A notable observation is that the given requirements are not just applicable for science applications. For example, eBay notes all the clicks on various websites done by the users. One of their use cases is "How relevant is the keyword search engine?" In other words, an eBay user can type a collection of keywords into the search box, for example "Coronavirus". it returns all the relevant items that match the request. The user might click on a particular item lets say item 5, and then follow a sub-tree of subsequent links under this item. Returning back to the previous level, the user might then click on item 4 and may follow a similar subtree under this. After this, the user may probably exit the system or go for another search request. eBay then extracts, the fact that items 4 and then 5 were selected, and they also infer that their strategy for search of Coronavirus is somewhat flawed, since the top 3 items were not of that much user's interest.
The user-ignored content is also important in this case along with the selected items information. E.g., how frequently did a particular object which came as a search result but was never clicked on? The algorithms and the web pages are getting dynamic, where the conventional weblog analysis cannot provide the required results as nothing is static. A detailed information about the content, needs to be collected and processed during the analysis. This application is next to impossible in the current RDBMS's; however, it can be very well modelled as a 1D array similar to a time series which will have embedded arrays to represent the results which we get at each step. Also there are user defined functions which complement the builtin search functionalities of SciDB to support this use case. The combination of an array based data model with fundamentals of uncertainty will provide a new platform for operations on petabytes of data. (3)

6 Conclusion

This paper has sketched the capabilities of SciDB-a science oriented database, along with the features and functionalities. SciDB has developed and progressed considerably in its functionality and robustness since its earliest incarnations. The earlier development stages had array data models for the storage purposes and its operations. This is highly appropriate for many types of sciences, i.e.it is designed to go well with image data, with LSST(Large Synoptic Survey Telescope) as a major target client. Support for array data structures, array partitioning (chunking) across nodes, overlap specifics, various types of array operations are very significant. The system is being developed by a team of developers in stages at different universities. Over the next two years, there will be great development in the open-source system for the science community.

7 Future Works

7.1 How the work can be extended further?

It is highly evident that scientists never throw away any data, and this results into a "grow only" data store. An increase in data is observed and is obtained from the various experiments and data from sensors. So, a science DBMS should support and enhance both data and processing elasticity. Elasticity should be implemented without large down time. The elastic partitioners balance the storage load efficiently, also reducing the reorganization time during the expansion of cluster operations. Some of the partioning schemes that are currently being developed are(7):

1. **Elastic Array Partitioning**

   Elastic array partitioning is designed to incrementally reorganize an array's storage, and it moves only the necessary data to balance load.

2. **Hash Partitioning**

   There are two elastic hash partitioners. The first, Extendible Hash, is optimized for skewed data, and the alternative, Consistent Hash, targets arrays with chunks of uniform size. Both algorithms assign chunks to nodes one at a time

3. **Range Partitioning**

   Range partitioning is used for queries that have clustered data access, such as grouping by a dimension or finding the k-nearest neighbors for a cell.

7.2 Research that can augment the current study

The current research areas of such array oriented databases that are being improved include(6):

- A major challenge that exists for array databases is the data loading into the system. Currently, only RasDaMan and Google Earth Engine (GEE) support reading of geospatial satellite imagery. GEE is the first platform to support reading of all datasets supported by the Geographic Data Abstraction Library

(GDAL). The SciDB community has created an extension for GDAL that convert any GDAL read dataset into SciDB format. An active area of ongoing research is the development of tools to read and export spatial data into the database.

- The increase in production and availability of big data presents an opportunity for the GIScience community. The development of combined big computation infrastructures and spatial data like array databases are the future in which analysis on big data are performed in parallel.
- Also the ongoing research in Earth Science is to find and identify an effective indexing scheme that(5):
    1. Can be applied to, preferably, all data models
    2. Supports data placement alignment
    3. Exerts little or no negative impact on data analysis performance, all at the same time.

7.3 Some interesting applications that can be build upon SciDB

SciDB has several interesting applications where it is currently being used and is under research in the life sciences as well as other industries. They are as follows(8):

1. Data cleaning
2. Clonality
3. Tumor Purity
4. Genomic Analysis
5. Digital Biomarker Discovery
6. Bioinformatics
7. Manufacturing Analysis

References

[1] "An Overview of SciDB Large Scale Array Storage, Processing and Analysis", The SciDB Development, Team `http://www.scidb.org`

[2] "The Architecture of SciDB" Michael Stonebraker, Paul Brown, Alex Poliakov, Suchi Raman, Paradigm4, Inc.

[3] "Requirements for Science Data Bases and SciDB", Michael Stonebraker MIT, Jacek Becla SLAC, David Dewitt Microsoft, Kian-Tat Lim SLAC David Maier Portland State University, Oliver Ratzesberger eBay, Inc., Stan Zdonik, Brown University

[4] "A Demonstration of SciDB: A Science-Oriented DBMS", P. Cudre-Mauroux MIT, H. Kimura Brown U, K.-T. Lim SLAC.

[5] "Evaluating the Impact of Data Placement to Spark and SciDB with an Earth Science Use Case", Khoa Doan, Hongfeng Yu, Amidu Oloso, Kwo-Sen Kuo, Thomas Clune, University of Maryland, College Park, MD, USA.

[6] "DM-81 - Array Databases", Haynes, D. (2019)

[7] "SciDB DBMS Research at M.I.T.", Michael Stonebraker, Jennie Duggan, Leilani Battle, Olga Papaemmanouil

[8] "Life Sciences-Paradigm 4"
`https://www.paradigm4.com/life-sciences/`

[9] "SciDB"
`https://www.slideshare.net/tsarevoleg/scidb-42136034`