Storage Management for Multidimensional Data

Xiaodong Qi
East China Normal University
1259560280@qq.com

ABSTRACT

Multidemsional array data is an significant data in science field. It is very convenient to adopt array data model to express query operators used by scientists. How to store array data is a foundational work. In the past two decades, RDBMS is chose to store array data. Duo to the feature of RDBMS, it is not suitable for scientific data management very well. Hence, new system which adopts array data model is developed recently.

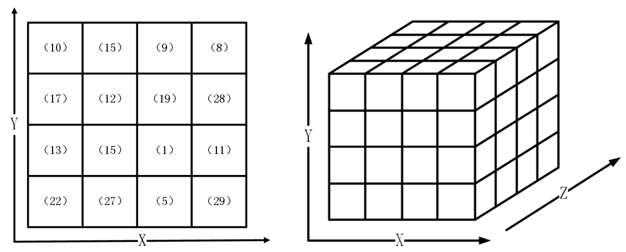
Scientific data management system is far different from RDBMS, such as version control, uncertainty and provenance, so it needs new design to satisfy particular requirements. In this paper, the common query operators of array data model is introducted. The requirements of scientific data management is analysised and the details of storage with array data model is discussed later.

CCS Concepts

•Computer systems organization → **Embedded systems;** *Redundancy;* Robotics; **•Networks** → Network reliability;

Keywords

Array data; Requirement; Storage



(a) 4×4 array (b) $4 \times 4 \times 4$ array

Figure 1: Example of arrays

WOODSTOCK '97 El Paso, Texas USA
© 2017 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

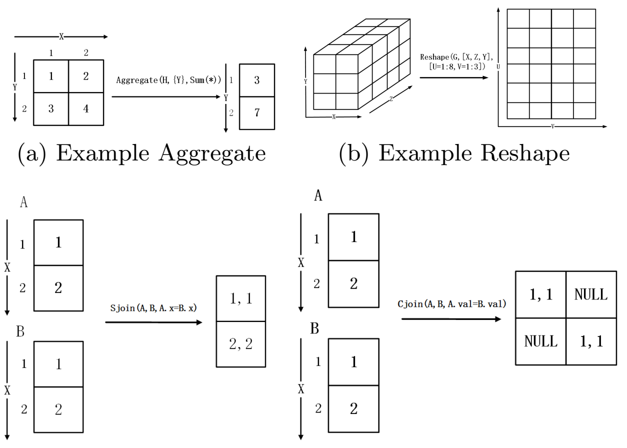
1. INTRODUCTION

With the development of science and technology, scientists can now generate large amounts of data at unprecedented scale and speed [15, 2]. Scientists typically work with array data or simulated array data, because in particle physics, biology, astronomy, oceanography and other fields of science, it can naturally express the scientific data used in the calculation, such as matrix, picture, video. Although commercial database communities have done a lot of work for scientific database system for years, producing a series of scientific database system based on relational databases, such as C-STORE[25], Sequoia[11], Paradise[10], MonetDB/SQLSky Server[16].However, due to data structure and the characteristics of scientific computing, the storage or process of scientific data with a RDBMS causes a number of disadvantages and inconveniences.Commercial relational database and Key-Value database when they are used to store and process scientific data change the array model structure of the data itself, making subsequent development of data management and analysis software become too complex. In the field of science ,many specific array operations need to be performed by scientists, such as feature extraction [17], smooth [8], cross-matching [18], and the Fourier transform, are not built-in by Key-Value database or RDBMS. Therefore, the traditional commercial database systems no longer meet the needs of such scientific data, and new data management system need to be developed.

In this paper we introuduce *Array Data Model* formally. The special requirements of scientific data management which are different from RDBMS are analysised detailedly. To support these requirements efficiently, we focus on the discussion of latest techniques which used by new systems.

1.1 Array Data Model

Array data model is widely used by scientists to express scientific data. The definition of an array in this papaer is similary to Furtado and Baumann[13]:Given a discrete coordinates set $S = S_1 \times ... \times S_d$,where each $S_i, i \in [1, d]$ is a finite totally ordered discrete set, an array is defined by a d-dimensional domain $D = [I_1, ...I_d]$, where each I_i is a subinterval of the corresponding S_i. Each combination of dimension values in D define a *cell*. All cells in a given array have the same type T, which is a tuple as in relational DBMS. Each T may have some attributes defined by schema. In particular, an attribute of an array can be an array as well. As shown in Figure 1, (a) is a 4×4 array with an attribute, and (b) is a $4 \times 4 \times 4$ array.


Aggregate (H, {Y}, Sum(*))
Reshape (G, [X, Z, Y], [U=1:8, V=1:3])
Sjoin (A, B, A. x=B. x)
Cjoin (A, B, A. val=B. val)
1, 1
2, 2
NULL

(a) Example Aggregate (b) Example Reshape

(c) Example Sjoin (d) Example Cjoin

Figure 2: Example of Operators

1.2 Operations

In this subsection, some operators that based on array data model are explained . There are two broad categories of these operators[26]. Duo to *join* is an very special operator, it is introudced separately.

1.2.1 Structural Operators

The first operator generates the result array just based on the structure of input arrays.This is means that this operator do not have to access the value of every cell to produce the terminal result array. The characteristic of this operator provides opportunity for optimization.

Subsample is a typical operation of this category. Subsample takes two inputs, an array A and a predicate over A's dimensions. Subsample returns return a result array *A'* with the same number of dimemsions as A.But value of each dimension is smaller than A's.

Reshape is another common operator of this category. This operator translates the input array into a new array which has a different shape. The new array may have more or fewer dimensions, possibly making new dimension names created. At the same time, new array must have the same of cells with the input. For example, in Figure 2(b), a $2 \times 3 \times 4$ is translated into a 4×6 array. This translation make the array reduced a dimension.

1.2.2 Content-Dependent Operators

Another important kind of operators is *Content-Dependent Operators*,whose result depends on the data in each cell. Sometimes, these operators also generate result by the value of input's demension.

A simple example of this kind of operator is *Filter*,$A' = FILTER(A, P)$. Here, A is a input array and P is a predicate over cell values of A. To accomplish this operator, all cell will be accessed to judege whether it's value satisfied the predicate P. If so, the cell will exist in the output array *A'*. The output array *A'* has the same dimensions with A including the interval of each demension which is different from *Subsample*.

Aggregate is a significant *content-dependent operator*. Similary to relational model, the *Aggregate* of a array also takes an n-dimensional array A, k differnt grouping dimensions G, and an aggregate function *AggFun* as input. Different from the operators mentioned before, the result array *A'* of *Aggregate* only has k dimensions which contained in G. Each value of cell in result array is computed by *AggFun*.In Figure 2(a), a *Aggregate* is appled over an array with function *Sum*.

1.2.3 Join

Join is a important operator whether in *Relational Data Model* or in *Array Data Model*. But join is also time-consuming operator. There are two kind of *Join* in *Array Data Model*: Structured-Join(or Sjoin) and Content-Join(or Cjoin).

Sjoin . It only restricts its join predicate over dimension value. When an Sjoin is excuted on an m-dimension array and an n-dimension array which involves k dimensions from each array in the join predicate, the result array will has $(m + n - k)$ dimensions. In each cell of result array, its value is the concatenation of cell tuples of source arrays if the JOIN-predicate is **true** as shown in Figure 2(c).

Cjoin. It only restricts its join predicate over data value. When an Cjoin is excuted on an m-dimension array and an n-dimension array which involves k attributes from each array in the join predicate, the result array will has $(m + n)$ dimensions. In each cell of result array, its value is the concatenation of cell tuples of source arrays if the JOIN-predicate is **true** as shown in Figure 2(d).

In a word, if we treat each dimension as an attribute of an array, the join of array is very similary to the join of a relational table. So some researchers expand join by relaxing the join condition[12]. For example, in Figure 2(c), the join condition can be $A.x = B.val$.

1.3 Storage

In order to extract useful information from scientific data, fisrt we need to store these data.Becasue array data is the major type of scientific data, how to storage array data become a foundation work. The 90's of the last century, the RDBMS already had considerable development.So it's very natural to use RDBMS to store array data. But RDBMS has many disadvantages and inconveniences mentioned before to process and analysis array data. In recent years, some new systems are developed to store and process array data. These systems usually treat array as first class citizen, implementate spcial query language for array and is built into many common operation used in field of science.Besides, with the rise of big data, some researchers also attemp to use Haddop and Spark to process and analysis array data.

The rest of this paper is organized as follows. In Section 2, we introduce two typical system which stores array data with RDBMS simply. In Section 3, the requirements of scientific data management will be represented first. Then we describe the details of array chunking, chunk overlap and named version. Last, we include in Section 4.

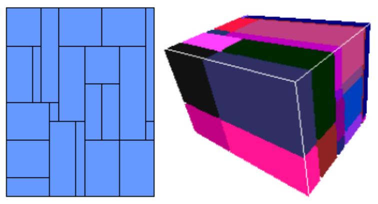


Figure 3: Tile in Rasdaman

2. STORAGE WITH RDBMS

Earlier data management system of an array are usually based on RDBMS[6, 7, 9]. These systems use a relation database for data storage, by extending the relational database for the storage and analysis of array data. A naive implementation is treating the value of each dimension as a attribute for a cell and the store each cell as a record in a relational table.Rasdaman, MonetDB/SciQL is one of the two. We will introduct them in next subsection.

2.1 Rasdaman

Rasdaman[7] is a commercial database running on a Linux system, developed by the University of Bremen and the Rasdaman company, announced open source in September 2008 (www.rasdamn.org). Rasdaman [6] is an array database system, its data can be stored in a standard relational database array data, such as astronomical data, image data, financial statistics, and so on. Rasdaman divide the Rasdaman array into different block called *tile*. Then Rasdaman stores each tile in PostgreSQL and establishs and index to speed up quries to the tiles [7]. As shown in Figure 3, arrays are splited into tiles which are irregular. Rasdaman is particularly suitable for handling common 2-dimensional raster images, but rasdaman does not hava a a limit of dimension. It can store 1-dimensional measurement data, 2-dimensional satellite image data, 3-dimensional time images data, 4-dimensional Ocean and weather data. At the same time, Rasdaman provides a query language similary to SQL called Raster Query Language (rasql) to handle these scientific data.

2.2 MonetDB/SciQL

MonetDB[28] is an open source column-oriented database management system. MonetDB is a in-memory relational database. It is designed for larger-scale data (such as tables with hundreds of columns and millions of rows) to provide high-performance query support. At present, the database system has been successfully used for applications with high performance requirements, such as data mining, OLAP, text search, multimedia search, and so on. MonetDB is originally developed by Peter Boncz and Martin Kersten. It was announced open source on September 30, 2004. Due to MonetDB is a relational database, it can't drecitly support the storage of array data. SciQL[9] is an extension of MonetDB database engine component. It provides a query language based on SQL, to handle scientific applications. Arrays were treated as first-class citizens in Rasdaman. Currently SciQL has been used by Teleios project providing interfaces for large-scale scientific data access.

3. STORAGE WITH NEW SYSTEM

In recent years, array model technical research has gradually expanded. Some new database systems are developed. Although relational table can store array data, it can not keep the spatial proximity of cells in origin array. These new systems are not built on the relational database, but use array to store scientific data directly. This means that cells close to each other logically are stored together physically in array database. This property can make many operations optimizated such as subsample, clustering and sjoin.

3.1 Requirements for Storage

Duo to scientific data management is different from traditional data management, there are some special requirement for storage of array[27].

3.1.1 Named Versions

Named Versions is an important requirement of most science users. Different people has different algorithm towards raw data. But these algorithms turn raw data into derived data sets. Sometimes scientists would like to reprocess raw data with new algorithm. Although they can store derived data sets separately, it would consume a large amount of storage space since enormous data volume. It is wildly more efficient to delta their copies off of the conventional one. As a result, the raw data only appears once. On the other hand, scientific data are usually collected by sensors. Obviously data of two contiguous timestamp is very similar. So named version would result in less space when it is used to store these data. Besides, if a data item is shown to be wrong, the update to it will create a new version, because wrong data is also meaningful for scientists.

3.1.2 "In Situ" Data

General speaking, we have to load data into system if it is needed processed more. In the most cases, however, it costs large amount of time to load data. These needed data often stored in various popular external formats such as HDF-5[5] or NetCDF[3]. In another hand, these data sometime is only processed once by scientific. So it's inefficient and inflexible to load data into system. An alternative approach is to define a self-describing data format and then develop particular adoptors to external formats. If an adoptor exists for user's data, then he can process it without requiring a load process.

3.1.3 Compression

Scientific data is very huge. For example, the Hubble Space Telescope collects 5GB data everyday and Sloan Digital Sky Survey (SDSS) has produced 25TB data over the past decade. But the space of disk and memory is limited even though the hard disk capacity is very lary now.Although compression reduces overall system performance to some extent. Becasue system has to extract data when accesses data. But compression is a very necessary way to save space. On the other hand, many arrays sparse such that most value of cell in array is default or empty. In this case, it very easy to compress data by only store useful cell.

3.1.4 Provenance

Sometimes scientists want to repeate the process of data derivation. In order to achieve this goal, system needs a repository to record run-time parameters including source arrays, operators and parameters for each operator. Two basic requirements for this repository are:

1. For a given data element D, find the operators that created it from source data elements.
2. For a given data element D, find all the data elements created later whose value is impacted by the value of D.

When a scientist suspects a data element wrong, he wants to know cause of the possible incorrectness. Obviously, this is the first requirement. On the other hand, the data elements computed based on the wrong data elements may be wrong as well. Searching these data elements is the second requiremnt. Unfortunately, there are no system that has implemented this function so far, due to various reasons.

3.1.5 *Uncertainty*

Duo to most scientific data collected by sensors, it is usually imprecise. Current commerical RDBMS is designed for business users, where are is a much smaller need for this requiement. Hence, current systems do not support uncertainty. Two different kinds of uncertainty in an array are[14, 20]:

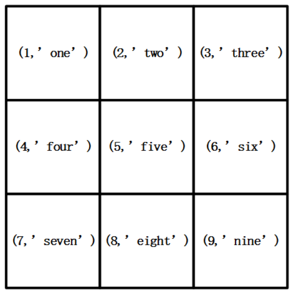
1. *structure uncertainty*:One data element may belongs to a set of cells in an array. We can not define which cell is the exact location of the data element.

2. *content uncertainty*:The value of each data element is imprecies. The real value may close to the collected value.

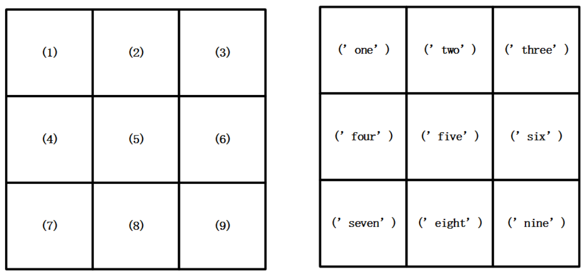
The challenge to support uncertainty for a system is how to describe uncertainty. Sometimes probability-distribution functions are used to describe uncertainty of arrays, such as Gaussian distribution. But there is no a universal way to describe uncertainty becasue different field normally has their own way. There is no doubt that it increas the difficult to process uncertain data in a universal framework.

3.1.6 *Distribution and Parallelization*

Huge data is generating by scientists every day, so it's very hard to store and process these data within one node. It is easy to think about using a cluster consist of several nodes to finish all tasks. It needs addtional consideration while data stored within several nodes. For example, how to distribute data among nodes effects the performance of system. We also hope that least data migration happens to finish a task because data migration between two nodes decreases the whole performance and data is distributed evenly across all nodes.

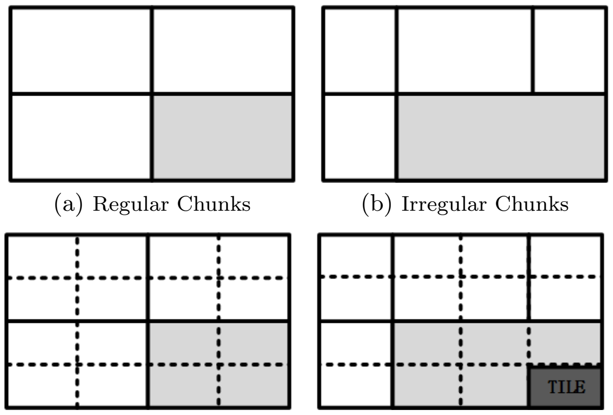


(a)



(b)

Figure 4: Columnar Organization of Array



(a) Regular Chunks (b) Irregular Chunks (c) Two-Level Chunks (d) Two-Level Chunks

Figure 5: Four different chunking strategies applied to the same rectangular array.Solid lines shows chunk boundaries of the logical array(sample chunks shaded). Inner-level tiles are represented by dashed lines(one tile is textured).

3.2 Storage Management

In this subsection, we present the current design for storage.

3.2.1 *Column Store*

Row store is the major approah in RDBMS because it is suitable for OLTP. But analaysised opeartors are performed by scientists mostly. Some systems adopt column-store approach and take the record of attributes in each cell and split them up to store each attribute's values in separate physical blocks. In fact, the data of logical array is stored with several single attribute arrays corresponding each attribute. Figure 4 shows the basic column-store idea. This approach has many advatages over traditional row-store DBMSs. The benefits of column store are:

1. Typical scientific data has many attributes. But a typical analytic query might only use two or three of them. This approach lets us read attributes directly, without reading the entire cell. It means that I/O for many queries is minimized.

2. Storing data by column makes compression easier because data of same attribute is usually physically close. Values of many attributes vary by small amounts. Techniques like run-length encoding or Adaptive Huffman are used to compress data efficiently, further reducing the I/O required to analytic quries.

Of course, column storing also has its own disadvantages such as updates. Column orientated systems might spend more time finishing updates compared to row-store DBMSs. Fortunately, in most applications, data is appended to the database continuously, and updates are a small part of overall query workload.

3.2.2 *Basic Array Chunking*

Generally,current systems based on array data model take the approach of breaking an array into fragments called *chunks* and storing these chunks on disk[24, 19]. There are three major techniques to chunk an array until now. Chunk is the unit of IO. All cells in a chunk is stored disorderly. If the cells of a chunk are emtpy, it would not be stored. In the same way, if values of all cells are default, only one value needs recorded. To finish a query, we must scan all cells in chunks involved in the query. Chunking is very benificial in parallel array processing. We can distribute all chunks to several nodes and then accumulate the resulte computed by each node generating the terminal result.

Regular Chunks(REG). The first approach breaks an into chunks that have the same size in terms of the coordinate space. For example, consider a 3D array with dimensions(X, Y, Z) such that X=[1:8], Y=[1:8], and Z=[1:8]. Then the array can be broken into 256 regular chunks, splited each X, Y, and$Z$ dimension into 8, 8, and 4 respectively. Regular chunks are commonly used for storing arrays on disk. The biggest advantage of this approach is that the implementation of it is very simple. We can also quickly determine a cell located in which chunk without any index. Once an array is sparse, this approach would cause data skew in parallel environment. Becuase the real number of data elements in each chunk varies greatly despite they have same size in terms of the coordinate space. Figure 4(a) illustrates this approach.

Irregular Chunks(IREG). In order to overcome the disadvantages of regular chunking, irregular chunking is proposed by researchers. The key idea is to chunk an array such that each chunk contains the same amount of cells. This may make every chunk covering a different amount of the logical coordinate space. This approach can speed-up range-selection queries when the chunk size is just suit for the specied workload. There is a approach using a kd-tree which splites a multidimensional space into small partitions ensuring load balance between partitions. For dense arrays,this approach is identical to regualar chunks. If chunks are irregular, index over chunks is needed to support efficient access to each chunk. There are several indexes can be used here such as R-tree. This approach is shown in Figure 4(b).

The biggest problem of the approachs mentioned before is settling chunks with appropriate size. If the size of a chunk is too large, additional cells would be loaded while some operators only a part of cells in a chunk such as subsample. It not only consumes more memory, but needs more time to be processed. We have to examine all cells in a chunk. But large chunks amortize seek times when reading data from disk. On the other hand, if the size of a chunk is too small, we need more time to read all chunks as each chunk becomes smaller. But it would reduce time to process data because less additional cells are loaded. Hence, a new chunking strategy was proposed.

Two-level Chunks(REG-REG or IREG-REG). An alternate approach is to create *two-level chunks*. The key idea is to splite a larger chunk into smaller chunks called tiles. The larger chunk can be either regular chunk(REG-REG) or irregular chunk(IREG-REG) as illustrated in Figure 4(c)(d). The larger chunks are the unit of disk I/O, while the smaller *tiles* can be the unit of array processing. So largers can reduce the times of I/O, and only the involved *tiles* will be exacted quickly reducing the time of processing. REG-REG can efficiently support binary operators such as Sjoin because the matching cells locates in corresponding chunks across two arrays. In contrast, IREG-REG can help address data skew during parallel processing.

All chunking strategories mentioned above is called aligned chunking. It still exists unaligned chunking shown in Figure 5. Compared to aligned chunks, unaligned chunks are more flexible. We can splite an array into chunks with any shape and size acoording different requirements.In practice, aligned chunks are used more frequently because the management of unaligned chunks is more complex and compared the advantage of unaligned is not very obvious proved by many experiments.

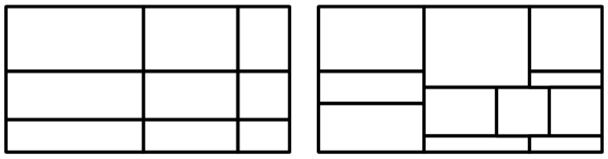


Figure 6: Two different kinds of chunking.

3.2.3 Data Distribution

To support parallel array processing, we can spread array chunks across several indepent nodes. For this, an array is partitioned into N disjoint segments. Each node only holds a subset of the array chunks. Different partitioning strategies effect the performance differently. As shown in Figure 7, the array is splited into 16 chunks and these chunks are spread across four nodes. The strategies includes [24](1) random(assign each chunk to a randomly select segment), (2)round-robin(iterate over chunks in some order and assign them to each segment in turn), (3)range(splite the arrray into n disjoint ranges of chunks and assign all chunks within a range to a segment), (4)hash(adopt a hash function to hash each chunk to segment), (5)block-cyclic(split the array into M regular *blocks* of N chunks each. Iterate over the chunks of a block in some pre-defined order and assign them to each of the N segments in turn). In order to ensure the would not lost when none node shutdowns, copies of one data should be stored across multi nodes.

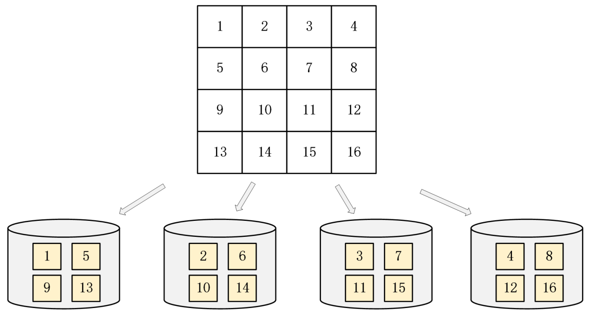


Figure 7: Distribution of array chunks

3.2.4 Overlap Data Support

In scientific field, when a cell is being processing, we need cells surrounding it such as clustering[1]. But these cells may not be facilited in the same chunk with it. If there are no other techniques, we have to read adjacent chunks to finish the computation. When processing an array in parallel, the adjacent chunks may stay at other nodes. This force systems to access remote nodes to get data. This is not economical because we hope each node to process its own chunks independently as possible. To increase the performance of these operators, a post-processing step is needed. The post-processing step often passes related data to each node. So there is no transfer among nodes during processing step.

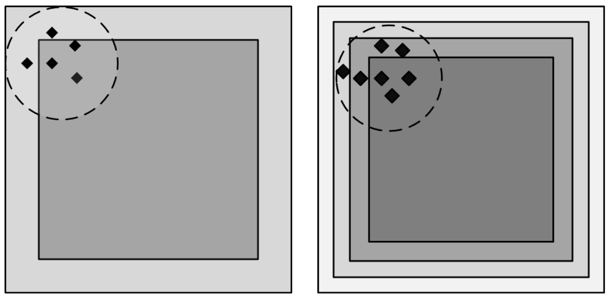

(a) Single-Layer Overlap (b) Multi-Layer Overlap

Figure 8: Example of overlap.

Single-Layer Overlap. Although this problem can be alleviated to a certain extent by post-processing step, it can not be solved absolutely[21]. Overlap is a efficient to satisfiel the needs of these operators. For each array chunk, overlap is area ϵ from neighboring chunks stored with the original chunk. When need data in adjacent chunks, we only read the overlap data instead of neighboring chunks. In Figure 8(a), the data in dashed circle is need and the light region is overlap. When a chunk is read, the overlap is also read at same time. Overlap data can be stored separately from the core array for optimization and is only read on demand. However, a problem with this approach is that ovlerlap casuses significant overhead for arrays. For example, if chunks become 10 larger along each dimension to cover the overlapping area, the space of storage will increases 33% for a 3D chunk and 75 for a 6D chunk. Besides, the size of overlapping area is very difficult to determine. If it is small, the data needed by operators is not covered by overlapping area. Otherwise, it may lead to I/O overhead. Fortunately, the overlap required is small compared with chunk. With the cost of disk decreasing, this approach is alternative.

Multi-Layer Overlap. As mentioned earlier, the key chanchallenge with overlap is selecting a appropriate size. This problem is very similary with the size of a chunk. So a approach called *Multi-Layer Overlap* is proposed naturally[24]. A multi-layer overlap is defined like a set of onion-skin layers around chunk. A multi-layer overlap takes the form $(n, w_1, ..., w_2)$, where n is the number of layers requested and each w_i is the thickness of a layer along dimension i. Here, overlap is store separately for optimization as well.

For a given query, we first judge whether it needs overlap. If so, only the layers covering the requested region will be loaded. It results in the least amount data read and processed. As shown in Figure 8(b), only two layers will be read with the chunk for clustering. Through multi-layer overlap, we get more freedom when access data. Ususally, the size of each layer is small relatively and more layers are provided. As well, multi-layer overlap impose storage overhead and the cost is same with single-layer overlap.

Overall, overlap is significant meaningless to single node because all data is store in a node. But in parallel processing, it is very vital to operators which needs neighboring data. It avoid a lot of network traffic. Some binary operators like join does't benefit from overlap as well.

3.2.5 *Version Control*

As described in section 4.1.1, versioned array storage model is an important feature needed by scientists. In addtion, version is the base of provenance. Next, we will present some

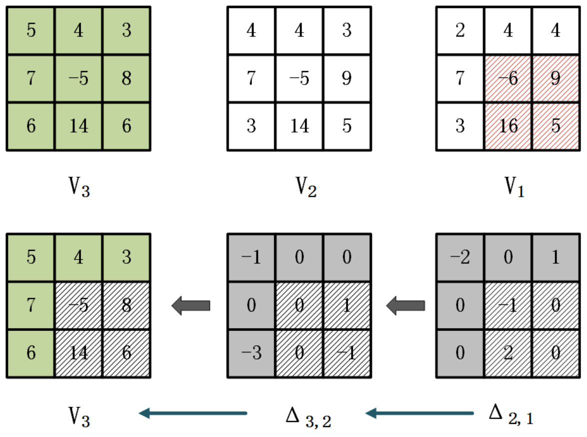


Figure 9: Illustration of a chain of backward delta versions for a 3×3 array.

details about version control.

From the start, we introduce two different kinds of storage. The first storage is describe as basic chunking called materialized storage. Values of all cells can be acquired directly. The second called delta storage just stores delta between two arrays, but the base chunk must be stored materially. Usually, the delta chunks are sparse chunks. So it provides a chance to compress them. As shown in Figure 9, version 3 adopts materialized storage while version 2 and 3 only store the delta. For delta storage, only values that are not zero will be stored because most values are zero. In above, chunks are stored oredered by the time by default. Because successive chunks are more likely similary.

For several versions, we only have to select latest version as base because it is accessed most frequently and store other versions with delta. This forms a chain over versions as shown in Figure 8. When the data of dashed area is requested in V_1, all data of dashed area in $delta_{2,1}$,$delta_{3,2}$ and V_3 is read to recover origin data. If the chain is long, we will spend more time and I/O iterating from the requested chunk to the base chunk. Hence, we hope the path to the base chunk is as short as possible. But this is not always, sometime the delta between nonconsecutive arrays is smaller. For example, $delta_{5,2}$ is smaller than $delta3, 2$ in Figure 9. So it is efficient to store V_2 based on V_5.

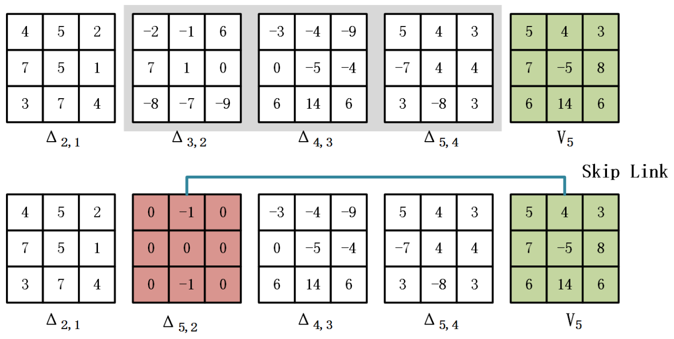


Figure 10: Skip Link

$$sizeof(\Delta_{j,i}) < \alpha \times sizeof(\Delta_{i+1,i}) \qquad (1)$$

$$V_{target} = V_{base} + \Delta_{base,target} \qquad (2)$$

Skip links are proposed to store arrays as shown in Figure

11. In general speeking, skip links are created over chunks or tiles to a better outcome. A naive implementation is enumerating all possible $\Delta_{j,i}$ combinations and verifying the condition in Equation 1. Where *sizeof* returns the sizeof an object in bytes and $\alpha \in [0, 1]$ is a factor that ensures skip links are created between similar chunks or tiles. Obviously, this approach would not be economical for computation because of large number of version combinations. A simple but efficient approach is that only skip links of the form $\Delta_{r,i}, \forall i < j$ are considered[22, 23]. The steps of this approach are:

(1) Storing the latest array materially as the base one.

(2) For a previous version, considering all the versions that was added later to compute Δ and choosing the smallest one.

(3) Processing all versions recursivley in the same way until no one left.

When we want to read data from particular version, we only traverse from the base version to the target version through skip links and then form the origin data. To fininsh the traverse, we first start at base version and then jump to the version formed by Equation 2 which is cloest to target version through skip link . The new version treated as base version,repeating the above step until achieve the target version. This requires that there are no links that across with each other. This approach makes path to base array shorter, and uses less space.

The approach mentioned above only considers delta like $\Delta_{r,i}, \forall i < j$,but there is another approach relaxing this condition. Here we introduce a notion called layout which is a storage of all versions. There are four oberservation of layout which are:

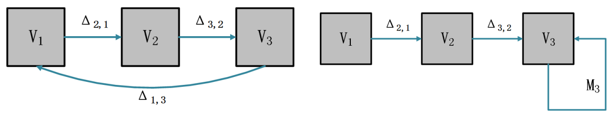
1. A layout of n versions always contains n edges.

2. A layout containing a set of connected components where each component has one and exactly one materialized version is valid.

3. A layout without any(undirected) cycle of lenghth $>$ 1 is always valid; hence, without considering the materialization edges, a valid layout graph is actually a polytree.

According to oberservation 3, a simple algorithm is designed based on tree. Firstly, we calculate the edges between every pairs of nodes whose weight are delta between corresponding nodes. Then we get a completed graph. Secondly, any minimal spanning tree algorithm can be used over the graph to achieve a MST such as Kruskal and Prim. Last but not least, we choose a version as the base version which has the smallest materialed storage cost. There are details about this approach as well. Obviously, it is very expensive to calculate delta between all versions. Hence, in practice, we often adopt sample method to estimate the delta between two version. Besides, it is more efficient to store a version materially rather than storing the delta. This step will splite the MST into a minimal spanning forest. When we want to read data from particular version, we read its base version and all delta to achieve the real data.

Update is a very common operation in science. It means that new versions well be added with the change of time.



(a) Invalid (b) Valid

Figure 11: Two ways three versions.

It is a diaster that restoring is performed when a version is added while the number of versions existed is large. So we do not want to restore all previous versions. As a more sophisticated update, we can accumulate K new versions as a batch and then only compute the optimal layout over the batch. In practice, K is between 10 and 100 to the best performance. Regardless of which approach above is used, we often materlize the latest version because it is accessed most frequently.

3.3 Others

Although some new system is developed, some requiremets is not addressed very well such as uncertainty, provenance and "In Situ" data. SciDB[4] is relative complete array data model system. It is also a open source system. It adopts coulum stores and support overlap data, compression and named version. It is based on a shared-nothing architecture. However, duo to various reasons, it does't support uncertainty data and provenance. There are some research on uncertainty data and provenance based on SciDB. SciDB can run on a distributed environment and build in many common operators in scientific field. So it is a definitely worthwhile system when you research in scientific data management. In another hand, more time is spent waiting for the result with the volume of data becoming more huge. So approximate queries are more useful for users. Some work is been done on this point[23], but approximate queries in array is less.

4. CONCLUSION

Although the research of scientific data management has developped for many years, it is still in a relatively early stage. There are still some key problems unsolved until now. The operators that has been implemented is also limited. Distributed system has played an very important role in management of science data. It is a foundational work to design a good storage to support the operations based on this better. How to design parallel algorithm to finish task efficiently is very meaningful. Join is a very trouble query because it can not be finished when data size is very large in SciDB. Besides, there are still some other type of data in science. This is a also an interesting point which can be do something with.

5. REFERENCES

[1] http://mahout.apache.org/.

[2] Large synoptic survey telescope. http://www.lsst.org/.

[3] Netcdf user's guid. http://www.unidata.ucar.edu/software/netcdf/guide.txntoc.html/.

[4] Scidb. http://www.paradigm4.com/.

[5] Hdf5:api specification reference manual. national center for supercomputing applications(ncsa). http://hdf.ncsa.uiuc.edu/, 2004.

[6] A. V. Ballegooij, R. Cornacchia, A. P. D. Vries, and M. Kersten. Distribution rules for array database

queries. *Lecture Notes in Computer Science*, 3588:55–64, 2005.

[7] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The multidimensional database system rasdaman. *Acm Sigmod Record*, 27(2):575–577, 1998.

[8] P. G. Brown. Overview of scidb: large scale array storage, processing and analysis. In *ACM SIGMOD International Conference on Management of Data*, pages 963–968, 2010.

[9] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *Proceedings of the Vldb Endowment*, 2(2):1481–1492, 2009.

[10] D. J. Dewitt, N. Kabra, J. Luo, J. M. Patel, and J. B. Yu. Client–server paradise. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 558–569, 2001.

[11] J. Dozier, M. Stonebraker, and J. Frew. Sequoia 2000: a next-generation information system for the study of global change. In *Mass Storage Systems, 1994. 'Towards Distributed Storage and Data Management Systems.' First International Symposium. Proceedings., Thirteenth IEEE Symposium on*, pages 47–53, 1994.

[12] J. Duggan, O. Papaemmanouil, L. Battle, and M. Stonebraker. Skew-aware join optimization for array databases. In *ACM SIGMOD International Conference*, pages 123–135, 2015.

[13] P. Furtado and P. Baumann. Storage of multidimensional arrays based on arbitrary tiling. In *International Conference on Data Engineering*, pages 480–489, 1999.

[14] T. Ge and S. Zdonik. Handling uncertain data in array database systems. In *IEEE International Conference on Data Engineering*, pages 1140–1149, 2008.

[15] T. Hey. *The Fourth Paradigm-Data-Intensive Scientific Discovery*. 2012.

[16] M. Ivanova, N. Nes, R. Goncalves, and M. Kersten. Monetdb/sql meets skyserver: the challenges of a scientific database. In *International Conference on Scientific and Statistical Database Management, SSDBM 2007, 9-11 July 2007, Banff, Canada, Proceedings*, pages 13–13, 2007.

[17] Y. C. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *ACM Symposium on Cloud Computing, SOCC 2010, Indianapolis, Indiana, Usa, June*, pages 75–86, 2010.

[18] M. A. Nietosantisteban, A. R. Thakar, and A. S. Szalay. Cross-matching very large datasets. *Santisteban*, 2010.

[19] E. J. Otoo, D. Rotem, and S. Seshadri. Optimal chunking of large multidimensional arrays for data warehousing. *Information Systems*, 31(3):25–32, 2008.

[20] L. Peng and Y. Diao. Supporting data uncertainty in array databases. In *ACM SIGMOD International Conference on Management of Data*, pages 545–560, 2015.

[21] K. E. Seamons and M. Winslett. Physical schemas for large multidimensional arrays in scientific computing applications. In *In Proceedings of the 7th International Working Conference on Scientific and Statistical Database Management*, pages 218–227, 1970.

[22] A. Seering, P. Cudre-Mauroux, S. Madden, and M. Stonebraker. Efficient versioning for scientific array databases. In *IEEE International Conference on Data Engineering*, pages 1013–1024, 2012.

[23] E. Soroush and M. Balazinska. Time travel in a scientific array database. In *IEEE International Conference on Data Engineering*, pages 98–109, 2013.

[24] E. Soroush, M. Balazinska, and D. Wang. Arraystore: A storage manager for complex parallel array processing. In *ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June*, pages 253–264, 2011.

[25] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, and E. O'Neil. C-store: a column-oriented dbms, in: Vldb'05. 2010.

[26] M. Stonebraker, J. Becla, D. J. Dewitt, K. T. Lim, D. Maier, O. Ratzesberger, and S. B. Zdonik. Requirements for science data bases and scidb. *Cidr*, 2009.

[27] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of scidb. In *Scientific and Statistical Database Management - International Conference, SSDBM 2011, Portland, Or, Usa, July 20-22, 2011. Proceedings*, pages 1–16, 2011.

[28] M. Vermeij, W. Quak, M. Kersten, and N. Nes. Monetdb, a novel spatial column-store dbms. 2008.