Selective Scan for Filter Operator of SciDB

Sangchul Kim* Seoung Gook Sohn† Taehoon Kim†
Jinseon Yu† Bogyeong Kim† Bongki Moon*
Department of Computer Science and Engineering
Seoul National University, Seoul, 08826, Korea
*{stdio, bkmoon}@snu.ac.kr
†{sgsohn, thkim, yujinee, bgkim}@dbs.snu.ac.kr

ABSTRACT

Recently there has been an increasing interest in analyzing scientific data generated by observations and scientific experiments. For managing these data efficiently, SciDB, a multi-dimensional array-based DBMS, is suggested. When SciDB processes a query with *where* predicates, it uses *filter* operator internally to produce a result array that matches the predicates. Most queries for scientific data analysis utilize spatial information. However, filter operator of SciDB reads all data without considering features of array-based DBMSs and spatial information. In this demo, we present an efficient query processing scheme utilizing characteristics of array-based data, implemented by employing coordinates. It uses a selective scan that retrieves data corresponding to a range that satisfies specific conditions. In our experiments, the selective scan is up to 30x faster than the original scan. We demonstrate that our implementation of the filter operator will reduce the processing time of a selection query significantly and enable SciDB to handle a massive amount of scientific data in more scalable manner.

CCS Concepts

•Information systems → Query optimization; Query operators;

Keywords

Array Database Model; Query optimization; Performance

1. INTRODUCTION

Scientific technology development over the past few decades has made remote explorations using satellites effortless leading to the prevalence of satellite applications such as weather forecast system, red tide detection, and global warming estimation. Consequently, the need for managing and analyzing unprecedented sizes of scientific data has emerged.

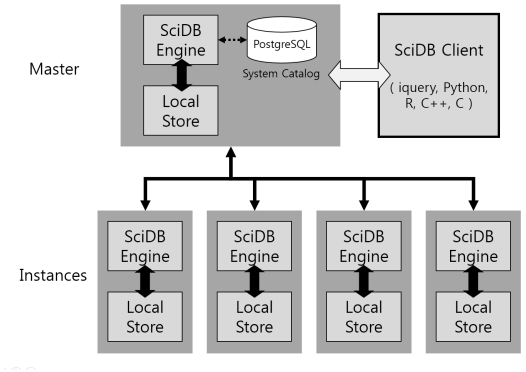


Figure 1: The architecture of SciDB

At first, RDBMS was taken into account for such needs, but it was soon verified that relational data model is not appropriate for processing scientific data which have geometric properties defined in space and time [3]. To this end, multi-dimensional array-based DBMSs were suggested, SciDB [2] being one of them.

SciDB is widely used in scientific research areas such as bioinformatics [9] and experimental physics [6], and it can analyze data simply based on the regularity of scientific data. It provides arithmetic and logical operations via a language called Array Functional Language (AFL) and an SQL-like language called Array Query Language (AQL). When executing an AQL query, it is translated into an AFL query in a preparing phase. While processing queries, chunks by which SciDB stores and manages each array are read. Chunk metadata such as coordinates that represent chunk boundaries is saved in a chunk map table.

Such features are commonly found in other array DBMSs as well. RasDaMan [1, 8], for example, has the unit of storage and access called *tile*. RasDaMan Query Language (RasQL) offers *trimming* to reduce the spatial domain and *section* to slice the dimensionality from an array.

In this demo, we present a new implementation of the *Filter* operator in order for SciDB to accelerate the processing of selective queries. Currently, as is shown below, SciDB unconditionally runs a full scan for most range queries written in AQL, which is clearly inefficient. The new implementation avoids full scans by leveraging range predicates and

This work was supported in part by the Ministry of Science ICT and Future Planning (Grant No. K-16-L03-C01-S03), and the National Research Foundation (Grant No. 2015R1A5A7037372) of the Korean Government.

SSDBM '16 July 18–20, 2016, Budapest, Hungary

DOI: 10.1145/2949689.2949707

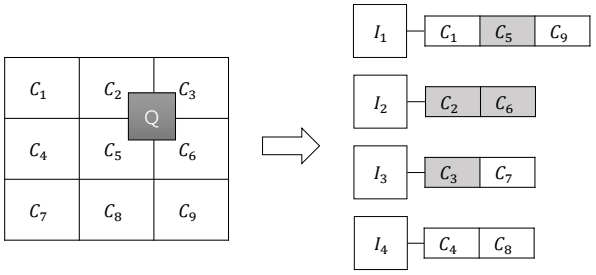


Figure 2: An example of windowing (I_n: n-th instance, C_n: n-th chunk)

```
Data: chunks, boundary
Result: position
initialization;
if chunk is not null then
    current_position = chunk.getPosition();
    if the chunk is not in a boundary then
        chunk++;
    end
    position = current_position;
end
```

Algorithm 1: The pseudo code of windowing

array indexes. Needless to say, our selective scan significantly reduces the overall query processing time, allowing SciDB to better handle massive amounts of scientific data.

Operator	Full Scan Performed
Between	No
Filter	**Yes**
Lookup	Yes
Slice	No
Subarray	No
Thin	Yes
Scan	Yes

Table 1: Selection and scan operators

Table 1 compares six selection operators and a scan operator with respect to the types of data access performed by SciDB. While three operators, *Between*, *Slice* and *Subarray*, take advantage of coordinates of data chunks to read them selectively, the others including the *Filter* operator, which most AQL range queries are transformed into, do not.

Section 2 describes internals of SciDB briefly. In Section 3, we compare the original query processing scheme to our scheme, *windowing*, and demonstrate our experimental results. Section 4 describes our demonstration scenarios and we conclude our work in Section 5.

2. INTERNALS OF SCIDB

SciDB is a parallel DBMS based on a multi-dimensional array model which supports *ragged* arrays because the system does not require that arrays be rectangular. The model makes data objects that are close to each other to be stored together. SciDB uses a shared-nothing architecture, where each node connected to the network can communicate with

Query 1	SELECT ∗ FROM *CHL* WHERE *longitude* ≤ 1000 AND *latitude* ≤ 1000 AND *time* = 1997241 (0.18% of whole cells)
Query 2	SELECT ∗ FROM *CHL* WHERE *longitude* ≥ 500 AND *longitude* ≤ 1500 AND *latitude* ≥ 500 AND *latitude* ≤ 1500 AND *time* ≤ 2000365 AND *chl* > 0 (0.57% of whole cells)
Query 3	SELECT ∗ FROM *CHL* WHERE *chl* > 0.001
Query 4	SELECT ∗ FROM *CHL* WHERE *longitude* ≥ 4000 AND *latitude* ≤ 500 (1.71% of whole cells)
Query 5	SELECT ∗ FROM *CHL* WHERE *longitude* ≥ 500 AND *longitude* ≤ 1500 AND *time* < 1998365 (2.65% of whole cells)
Query 6	SELECT ∗ FROM *CHL* WHERE *time* ≥ 2009001 AND *chl* > 0.1 (8.5% of whole cells)

Table 2: Representative Queries

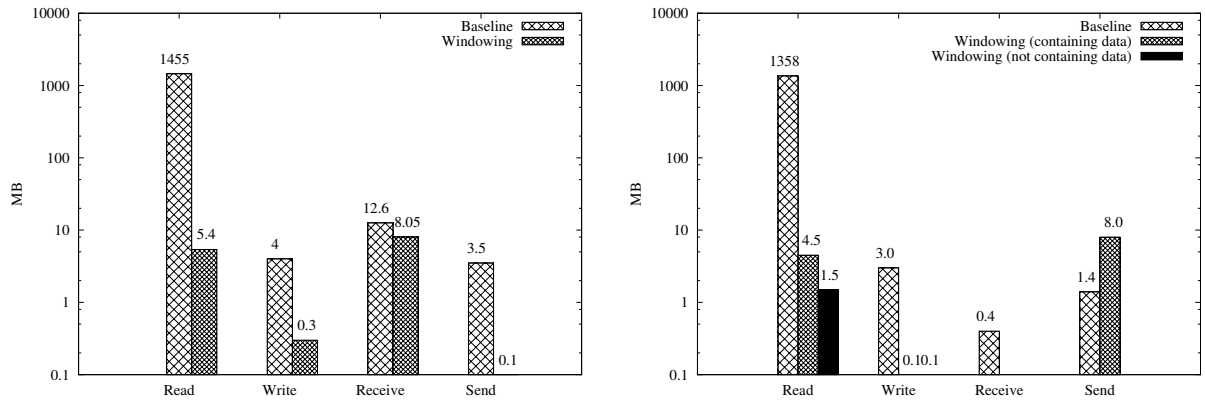
every other node and has its own independent storage. A node can have several **instance**s, each of which contains a storage manager and a query processor. The system consists of one or more slave nodes and a single master node that distributes input queries to the slave node(s). Figure 1 illustrates the architecture of SciDB.

The master node stores a **system catalog**, metadata of SciDB, in PostgreSQL [4]. While the master activates the entire SciDB processes, it retrieves relevant information from the catalog when processing queries. During **chunking**, SciDB partitions multi-dimensional arrays into **chunks** which is the physical unit of I/O. SciDB does not store indexes explicitly because it stores coordinates of data in the system catalog. Other chunk metadata such as boundaries and chunk lengths is saved in a chunk map table. Each instance is responsible for managing a subset of arrays.

In SciDB, there are two distinct query languages: Array Functional Language (AFL) and Array Query Language (AQL). AFL is optimized for array calculations which derived from APL (A Programming Language) whose central data type is a multi-dimensional array. By default, queries are issued using AFL and the language supports functions that work with arrays such as *filter()*, *between()*, *aggregate()*, etc. AQL follows the grammar of SQL [5] and is translated into AFL in query preparing phase.

3. FILTER OPERATOR

As mentioned in Section 2, AQL queries are translated



(a) Master node (b) Slave nodes

Figure 3: Amounts of I/O and network communication (Query 1)

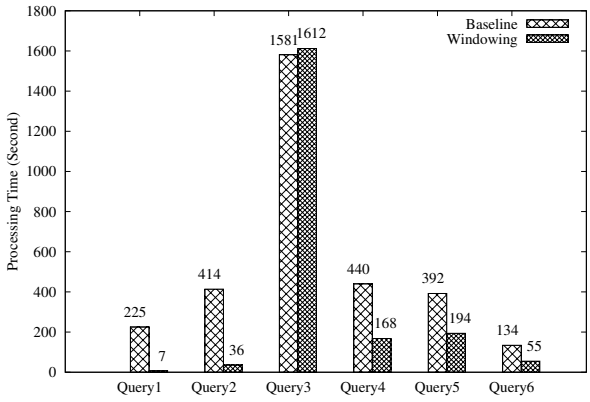


Figure 4: The elapsed time for processing queries

into AFL queries. If an AQL query contains a *where* clause, it is internally converted to a **filter** operator which filters out data based on *where* predicates. Precisely, the filter operator examines the attributes and the coordinates of data. Then it produces an array with the same schema of the input array that consists of cells that satisfy *where* predicates. Even for range queries which can utilize indexes, filter operator naively executes full scans and checks whether each cell is in the range or not, yielding poor performance regardless of the range.

In our approach, we utilize metadata which contains information of arrays such as starting and ending coordinates, chunk length of each dimension, etc. By exploiting the regularity of coordinates in arrays, each instance can easily identify which chunks are relevant to an input range query thus skips heavy full scans.

Figure 2 shows an example of windowing and how selective scans are processed. There are 4 instances and 9 chunks. Q represents a range query, and C_2, C_3, C_5 and C_6 are relevant chunks of Q. In the case of current filter operator, all instances fully scan the entire data (i.e., 9 chunks) and examine the data read using query predicates. This step contains redundant processes because the chunks including data requested by the query Q are relevant only to 4 chunks. However, using our approach, we can reduce the time to search data as well as irrelevant I/Os. Specifically, in this example, the 4th instance does not need to search all chunks it has, and chunks unrelated to the query (C_1, C_4 and C_7) are not read.

The pseudo code of windowing is in Algorithm 1. If boundaries of a query are given (i.e., predicates contain boundary conditions), we bypass unrelated and empty chunks, and only read the relevant chunks.

The experiment was carried out on a cluster of 10 nodes, where all nodes are connected to a 1Gbps network switch. Each node has Intel i7-4770 3.40GHz CPU and 8GB RAM. Data are stored in 1TB 7200RPM HDD attached to each node. Linux kernel version is 3.5.0 and SciDB version is 14.12.

For the experiment, we used SeaWiFS L3 Chlorophyll (CHL) data, provided by NASA [7]. As a collection of the earth images of the satellite SeaStar, the data have 3 dimensions (*longitude, latitude, time*) and values measured at each unit (*chlorophyll*). Each cell is an 8-day average of chlorophyll of the earth, with the resolution of 9km × 9km. Each chunk has 1 million cells and the chunk size is 1000 (*longitude*) × 1000 (*latitude*) × 1 (*time*). The total number of chunks in database is 8,115. The number of cells is 5,048,179,200. The schema we declared is $CHL < chl : double > [longitude = 1 : 4320, 1000, 0, latitude = 1 : 2160, 1000, 0, time = 1997241 : 2009361, 1, 0]$ and the array is partitioned using hash.

We ran 6 queries as shown in Table 2, and measured the elapsed time, amounts of I/O and network traffic. These queries represent range searches based on a different combination of attributes and coordinates. Query 1, 4, and 5 includes a predicate based only on coordinates. Query 3 includes a predicate based only on attribute *chl*. Query 2 and 6 include a predicate combined with dimensions and an attribute.

Figure 3 indicates amounts of I/O and network traffic in the master node and slave nodes when processing Query 1. Data located in slave instances are sent to the master node and this result is depicted in Figure 3a. Figure 3b shows amounts of I/O and network traffic comparing current

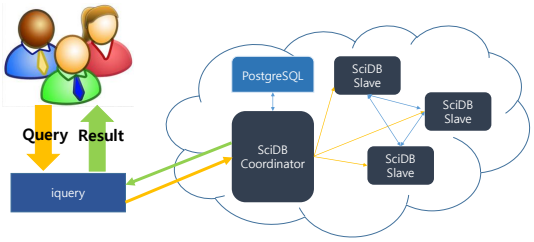


Figure 5: Processing queries using iquery

filter operator to *windowing*, and results are distinguished whether data are relevant to queries or not. When windowing is used, practical read and send occur in nodes where relevant data exist.

Figure 4 shows the elapsed time for processing 6 queries. The overall performance improved, only exception being Query 3. Especially, the elapsed time of Query 1 is reduced by 96%. As the selectivity of data goes up, so did the performance. However, if a query includes predicates that do not use spatial information (coordinates) such as Query 3, the performance is hardly improved, and even worse because *windowing* consumes additional time for computing the query boundaries.

4. DEMONSTRATION SCENARIO

In this section, we demonstrate of windowing step by step. Also, we describe the main scenario and how users can command to the system. At first, we introduce what an expected influence of our approach is. Users access our remote cluster via SSH and connects the master node (SciDB coordinator). They will have the chance to choose from two methods: vanilla and windowing filter operators.

In this demonstration, we will use two data: CHL and MODIS remote-sensing reflectance (in short, RRS). CHL was used in our experiment in Section 3 and RRS is the bigger data than CHL. RRS has higher resolution (4km × 4km and 8,640 × 4,320) than SeaWiFS but measures different experimental value known as remote-sensing reflectance. The number of cells is 20,342,016,000. The schema of the array is $RRS < chl : double > [longitude = 1 : 8640, 1000, 0, latitude = 1 : 4320, 1000, 0, time = 2002185 : 2014128, 1, 0]$ and the array is partitioned using hash. We loaded two types of data (CHL and RRS) in advance, because it takes a significantly long time.

Users use *iquery*, one of the client APIs, and input AQL or AFL queries for using filter operator and results are shown on a screen. The workflow for processing queries using iquery is in Figure 5. For analyzing the elapsed time, it is measured by *time* command. The screenshot of results for using one of the queries in Table 2 is shown in Figure 6. In addition, to show the selective scan more clearly, disk I/O will be measured and displayed by a graph.

5. CONCLUSION

In SciDB, most queries for scientific data analysis utilize spatial information. However, filter operator for processing queries reads all data without considering features of array-based DBMSs and spatial information. In this demonstration, we propose a selective scan method, *windowing*, which only retrieves data selectively corresponding to a range. It

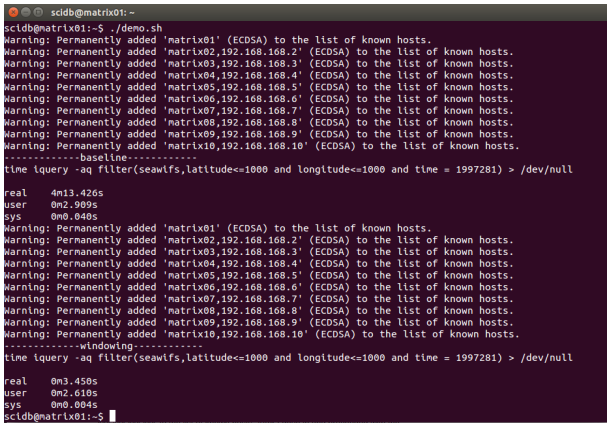


Figure 6: The screenshot of results

reduces the processing time of a selection query significantly when queries contain range search. Our implementation of the filter operator will accelerate scientific analysis, and relieve the hesitation of using AQL statements. It will also enhance SciDB to handle a massive amount of scientific data in more scalable manner.

6. REFERENCES

[1] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The Multidimensional Database System RasDaMan. In *ACM SIGMOD Record*, volume 27, pages 575–577. ACM, 1998.

[2] P. G. Brown. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. In *Proceedings of the 2010 ACM SIGMOD Conference*, pages 963–968, 2010.

[3] P. Cudre-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, S. Madden, M. Stonebraker, S. B. Zdonik, and P. G. Brown. SS-DB: A Standard Science DBMS Benchmark. *Extremely Large Databases Conference 2010*.

[4] http://www.postgresql.org/. PostgreSQL.

[5] L. Libkin, R. Machlin, and L. Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. In *ACM SIGMOD Record*, volume 25, pages 228–239, 1996.

[6] D. Malon, P. van Gemmeren, and J. Weinstein. An exploration of SciDB in the context of emerging technologies for data stores in particle physics and cosmology. *Journal of Physics: Conference Series*, 368(1):012021, 2012.

[7] NASA Goddard Space Flight Center, Ocean Ecology Laboratory, Ocean Biology Processing Group. SeaWiFS Ocean Color Data.

[8] N. Widmann and P. Baumann. Efficient Execution of Operations in a DBMS for Multidimensional Arrays. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 155–165. IEEE, 1998.

[9] Y. Yao, T. Sun, T. Wang, O. Ruebel, T. Northen, and B. P. Bowen. Analysis of Metabolomics Datasets with High-Performance Computing and Metabolite Atlases. *Metabolites*, 5(3):431–442, 2015.