Master's Thesis
석사 학위논문

A Distributed In-situ Analysis Method for Large-scale Scientific Data

Dong Hyoung Han (한 동 형 韓 東 亨)

Department of Information and Communication Engineering

정보통신융합공학전공

DGIST

2016

Master's Thesis
석사 학위논문

A Distributed In-situ Analysis Method for Large-scale Scientific Data

Dong Hyoung Han (한 동 형 韓 東 亨)

Department of Information and Communication Engineering

정보통신융합공학전공

DGIST

2016

A Distributed In-situ Analysis Method for Large-scale Scientific Data

Advisor : Professor Min-Soo Kim
Co-advisor : Professor Won-Seok Kang
Professor Jihwan Choi

by

Dong Hyoung Han
Department of Information and Communication Engineering
DGIST

A thesis submitted to the faculty of DGIST in partial fulfillment of the requirements for the degree of [Master of Science] in the [Department of Information and Communication Engineering] . The study was conducted in accordance with Code of Research Ethics[1]

12. 1. 2015

Approved by

Professor Min-Soo Kim (Signature)
(Advisor)

Professor Won-Seok Kang (Signature)
(Co-Advisor)

Professor Jihwan Choi (Signature)
(Co-Advisor)

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of DGIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

A Distributed In-situ Analysis Method for Large-scale Scientific Data

Dong Hyoung Han

Accepted in partial fulfillment of the requirements for the degree of [Master of Science]

12. 1. 2015

Head of Committee 김 민 수 (인)
Prof. Min-Soo Kim

Committee Member 강 원 석 (인)
Prof. Won-Seok Kang

Committee Member 최 지 환 (인)
Prof. Jihwan Choi

Degree
201422020

한 동 형. Dong Hyoung Han. A Distributed In-situ Analysis Method for Large-scale Scientific Data. Department of Information and Communication Engineering. 2015. 43p. Advisor Prof. Min-Soo Kim, Prof. Co-Advisor Won-Seok Kang. Co-Advisor Jihwan Choi

ABSTRACT

The size of scientific data has been increasing rapidly in a variety of domains. The scientific data is represented as array data and is managed by a diverse scientific data format such as HDF, NetCDF and MDSplus. Even though the existing array DBMSs such as SciDB and RasDaMan manage array data, there are challenges in loading data into the array DBMS. The data loading process of the distributed array DBMS incurs the significant overheads since the inefficient four transformation steps of file format incur the expensive disk I/O.

In this paper, we propose a distributed in-situ analysis method DISCAN that can process a scientific query efficiently and directly over raw scientific array data in distributed array DBMSs. Our approach eliminates unnecessary write operations during the data loading and processes only the data required in query. Our in-situ processing consists of two phases, HDF merger and DISCAN. HDF merger is responsible for managing raw scientific data in order to distribute the scientific data to nodes. DISCAN is composed of Local Map that transforms the raw scientific data into the internal data representation of DBMS and Global Map that replaces the transformed data according to a partitioning policy of the DBMS. DISCAN reads only the data required during query processing using the well-defined scientific data format libraries. We evaluate the performance of DISCAN across real-world scientific dataset. Experimental results show that DISCAN outperforms the processing query after data loading of the distributed array DBMS by up to more than 60 times.

Keywords: In-situ processing, data loading, array DBMS, scientific data format

Contents

Lists of figures

Lists of tables

Ⅰ. INTRODUCTION

In the era of information, massive data is explosively generated in the business as well as the science since scientific observation instrument and computing simulation has been developed rapidly. For example, in the case of the physics, Large Hadron Collider experiments observe particles generated by the collision. It generates the 30 petabytes of data annually and stores dozens of gigabytes of data per second for future analysis. At the astronomy, Large Synoptic Survey Telescope (LSST) that observes the sky to find stars, planets and other objects produces more than the 30 terabytes of data every night.

Most scientific data can be represented as an array. For example, one-dimensional array data is DNA sequence data. Two-dimensional array data is the map data, the graph data and MS/MS spectrum data. Three-dimensional array data is the satellite data that has two-dimensional map data and one-dimensional time data. In the fields of analyzing scientific data, larger numbers of scientific data formats are used to manage scientific data and support array. The Hierarchical Data Format (HDF) [1], the Network Common Data Form (NetCDF) [2], MDSplus [20] and ROOT [21] are file format to manage scientific data. Especially, NASA supports satellite data as HDF. The array data stored in these file formats is accessed through a high-level interface that is optimized to utilize the maximum bandwidth of disk I/O. Scientific file format libraries also optimize to store and retrieve consecutive data. However, the libraries do not support query interface like

SQL. Thus, users should code a custom application for simple analysis of scientific data. Compare with Database management system (DBMS), users are able to easily analyze scientific data through declarative SQL queries.

Analyzing array data is barely suitable for relational DBMSs since the DBMSs are developed to analyze relational data. The various array DBMSs have been developed to efficiently manage and manipulate array data, such as Titan [3], T2 [4], RasDaMan [5], ArrayDB [6], (S)RAM [7, 8], and SciDB [9]. Since the scale of data has been increasing, processing the data is beyond the single array DBMS. Hence, the distributed array DBMS has been researched like SciDB. The distributed array DBMS has approach is similar to Hive [11] supporting SQL query processing based on Hadoop [10]. The architecture of the distributed array DBMS consists of a master node and slave nodes called shared-nothing architecture. The one of major characteristics of shared-nothing architecture is scale-out approach which is able to add the nodes easily. Scale-up approach upgrading the performance of a machine has the limitation of cost-efficiency [12]. However, scale-out approach solves the limitation since low-priced commodity machine is added in the cluster.

In the distributed array DBMS, The master node generates a query plan using submitted a query and assigns the job to slave nodes. Each slave node executes the appropriate operators for query plan and sends results to master node. In order to analyze data in a DBMS, DBMS should load the data to the database. Data loading is to store data as an optimized format to process the query on the DBMS. Transformation of file format causes disk I/O since the transformation process reads the existing file and then cre-

ates the new file. Thus, the data loading becomes significant overhead depending on the increase of data size. For example, it takes about 5 hours to load 1TB data into DBMS using HDD. Typically, the valuable information of scientific data is the part of the entire data. Although users want to analyze the part of scientific data, the entire data should be loaded to DBMS. Relational data is generated slowly and is used by lots of queries. On the other hand, scientific data is generated rapidly and the number of queries is less than relational DBMS. Thus, loading all the scientific data to DBMS is inefficient. In addition, array DBMS does not support loading scientific data to the DBMS. Scientific data should be transformed to data loading format in order to load scientific data. Data loading process is time-consuming since it needs multiple disk I/O.

In order to solve the overhead of data loading, there have been a number of efforts to execute query on raw file without data loading called in-situ processing. The existing in-situ processing is able to process text files like CSV without data loading. Since the text file does not be optimized to process a query on DBMS, the in-situ processing uses the ways to index data position in a text file in order to perform in-situ processing effectively [13]. Accessing scientific data format which stores binary data is optimized by scientific data format libraries.

In addition, the distributed array DBMS is in no condition to adapt the existing in-situ processing since the in-situ processing has been developed on RDBMS of single system. Adapting in-situ processing to distributed array DBMS considers the characteristics of the DBMS. The distributed array DBMS uses the chunking mechanism which splits the array into sub-array having the same shape. In order to adapt in-situ processing on

the distributed array DBMS, it is needed to distribute data from storage server to SciDB instances and partition the data properly to process queries on the distribute array DBMS.

In this paper, we propose Distributed In-situ SCAN operator (DISCAN) that allows the distributed array DBMS to execute query over raw scientific files without data loading. DISCAN removes the inefficient portion of the existing data loading process of the distributed array DBMS and is able to process queries over raw scientific files. The one of the important characteristic of array is to manage sorted data based on dimensions. While transforming scientific data file into a DLF file, the characteristics of array is removed. We design DISCAN as in-situ processing without the preprocessing that removes the characteristic of the array. In order to read data needed in the query, DISCAN modifies query plan and uses well-defined scientific data format libraries. We evaluate the performance of DISCAN on the cluster consisting of 10 nodes and experiment with a real satellite observation data of NASA, called Moderate Resolution Imaging Spectroadiometer (MODIS). Our experiments show that DISCAN outperforms the existing data loading process of the distributed array DBMS by more than 60 times on the same MODIS dataset.

Our contributions are summarized as follows:

- We design and develop DISCAN that is the scan operator for in-situ processing over raw scientific data format on the distributed array DBMS.
- We modify the query plan and utilize well-defined scientific data format libraries in order to efficiently access raw data needed in the query.

- We implement DISCAN in the SciDB and evaluate its performance across real-world dataset. DISCAN shows that remarkable performance improvements can be achieved.

The remainder of the paper is structured as follows. First, we introduce background about array DBMS and data loading in Section 2. Section 3 presents related work about in-situ processing. We then describe DISCAN in Section 4. Section 5 follows with a description of the experimental setup and presents the performance evaluation. We conclude and discuss future work in Section 6.

Ⅱ. BACKGROUND

2.1 Array DBMS

Since Typical business DBMS is developed to analyze efficiently business data which represents relational table, it is unsuitable about analyzing scientific data that represents an array. Various array DBMSs are developed using an array as a data model to handle efficiently array. RasDaMan and SciDB are array DBMSs which are widely used in the scientific data analysis.

RasDaMan has been developed ahead of SciDB. The system supports multi-dimensional array and has a client/server architecture as shown in figure 1, unlike SciDB has a shared-nothing architecture. The architecture consists of RasDaMan engine processing data and base DBMS storing data. When a client submits query to RasDaMan, RasDaMan reads data needed in the query from base DBMS and then executes the query. At this time, base DBMS stores and manages only data and query is executed in RasDaMan.

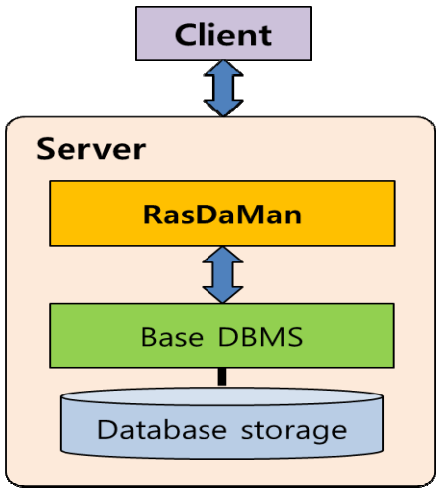

Figure 1. Architecture of RasDaMan.

RasDaMan stores array data as chunk which splits an array into a sub-array. There are two ways to create chunk. The regular chunking splits array into the same shape of all the chunks. The irregular chunking creates chunks to the different shape. RasDaMan stores chunk using the irregular chunking method. Figure 2 shows that the irregular chunking method is more efficient than other method. When attempting to access no.1 area indicated by a green color in order to process queries in Figure 2, Figure 2(A), 2(B) and 2(C) are the three strategies of each the linear subdivision, the regular chunking, and the irregular chunking for accessing the green area.

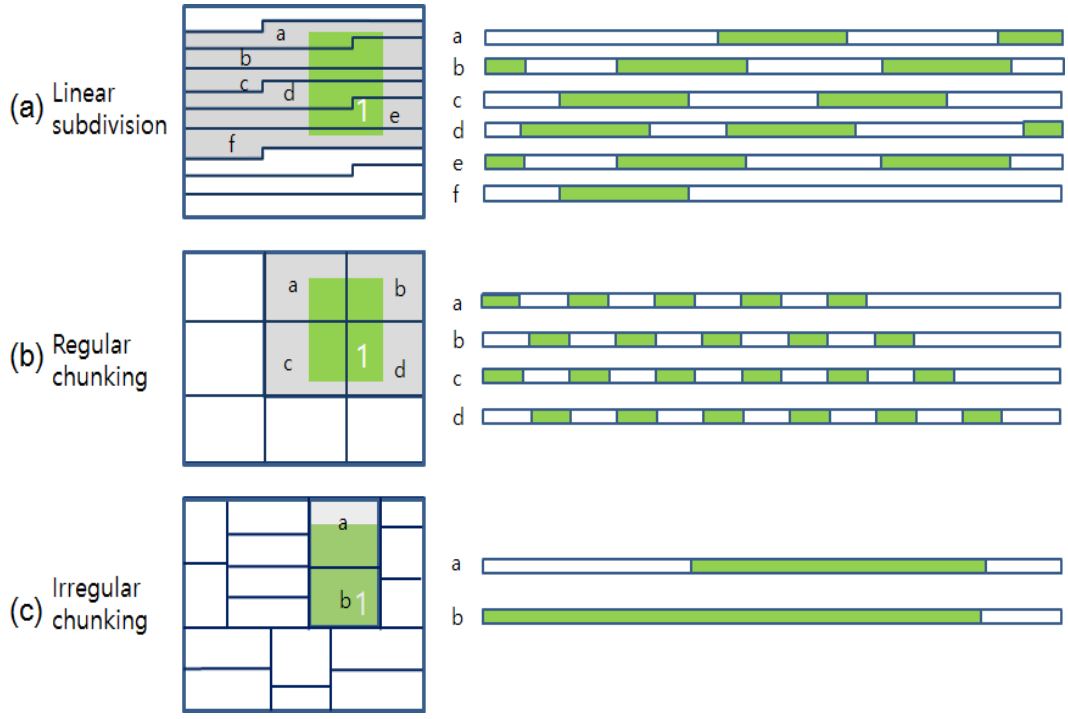

Figure 2. The example of chunking strategies.

Linear subdivision strategy reads data sequentially without the use of chunking mechanism. Thus, linear subdivision strategy reads six chunks in order of {a, b, c, d, e, f} to access green area. Although the regular chunking strategy reads four chunks in

order of {a, b, c, d}, unused data is read from disk. The irregular chunking strategy used in RasDaMan accesses two chunks of {a, b} to read number 1 region. Thus, the irregular chunking strategy is more efficient than other strategies since the strategy occurs the least number of disk I/O.

In addition to the requirement of using an array as a data model, other requirements that scientists hope at DBMS for analyzing scientific data is as follows[14].

- Array data model
- Massive scale of data processing
- No overwrite storage system that does not update result after query processing
- Provenance method tracking process of data analysis
- Processing error data involved in scientific data
- Version control for scientific data generated periodically

SciDB has been developed to satisfy all the requirements for DBMS which scientist hope to analyze scientific data. Figure 3 represents the architecture of SciDB. The system uses an array as a data model and has shared-nothing architecture using at Hadoop Distributed File System (HDFS) [15]. In common with Other array DBMS, SciDB also stores chunk when storing data. In order to handle query failure at a node, chunk is replicated to multiple nodes. SciDB uses regular chunking strategy when splitting data into chunk. SciDB instances is a group of processes to operate query in SciDB. Multiple SciDB in-

stances can exist on one node depending on configuration. SciDB consists of coordinator node and work nodes in order to execute the query. Coordinator node is responsible for managing position information of the loaded chunk and setting up a query plan for submitted query. Work node executes operation of query plan using the chunk stored in each node.

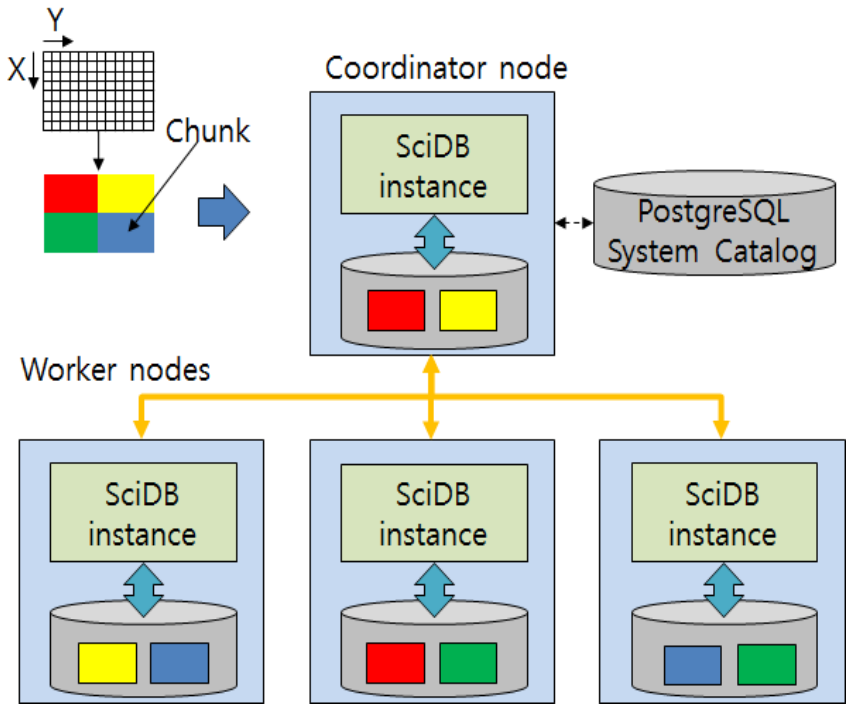

Figure 3. Architecture of SciDB.

2.2 Data loading

In order to analyze data in the raw file, DBMS transforms the raw file into data representation used in the DBMS and then store the data in database storage. The process is called data loading. Data loading separates into five components of figure 4. Each component is as follows.

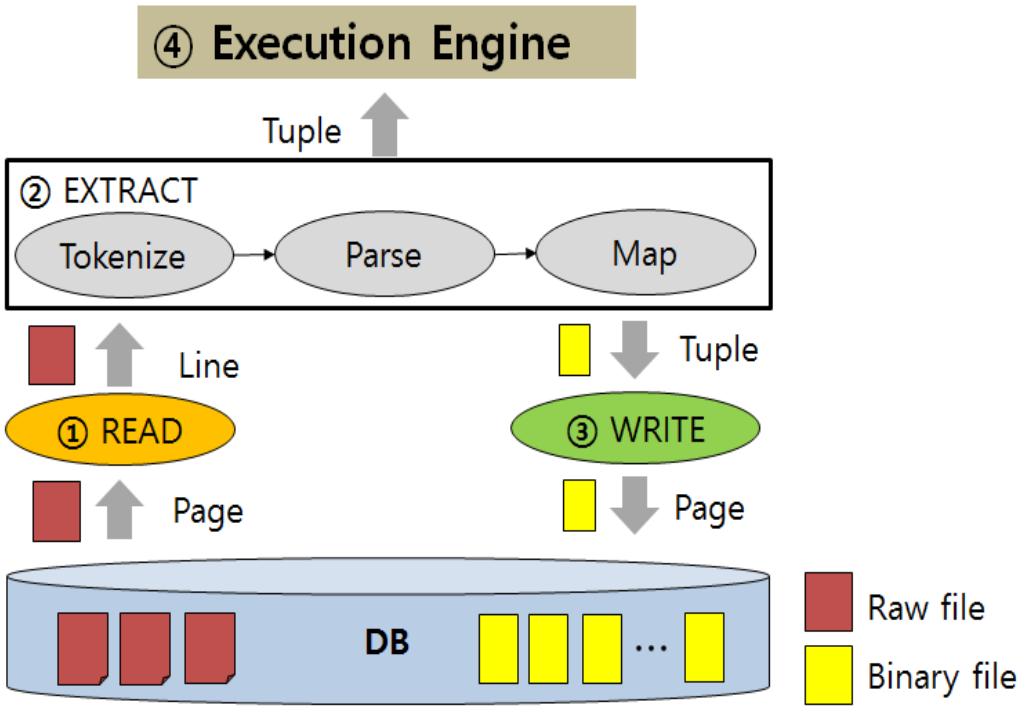

Figure 4. Raw file access processing in DBMS.

- READ: reads data from the raw file. In order to optimize READ component, there is the way to read data with page as a unit and the way to cache pages in memory buffer.
- TOKENISE: splits data read at READ component into attributes. Indexing attribute position in data and reading data until required attribute, called selective tokenizing[13], are used to optimize TOKENISE component
- PARSE: converts data of each attribute into binary representation that is able to process in DBMS. This component uses selective parsing [13] that converts only required tuples.
- MAP: maps converted binary representation on data representation used in DBMS. For example, DBMS that uses column storage maps data on data representation divided by column.

- WRITE: stores converted data in disk. In the case that READ component and WRITE component access disk simultaneously, both components should be executed asynchronously to remove interference about disk access.

While three components of TOKENISE, PARSE, MAP are considered the Extract phase, data loading process of DBMS is to execute components in order of READ → Extract → WRITE. In other words, data created in Extract phase is stored in the storage of DBMS by WRITE component. DBMS can execute queries after finishing data loading. WRITE component takes up the largest part of the data loading time.

Ⅲ. RELATED WORK

There have been a number of efforts to execute query on raw files without data loading, called in-situ processing. There are *external tables* [16, 17] as the basic example of in-situ processing on relational DBMS. *External tables* is the way to use a CSV format file to process the query as if the CSV format file is loaded in DBMS. Although schema of the raw file is defined in database catalog, the actual data does not be loaded into the DBMS. The raw file remains the same. *External tables* passes data converted by Extract phase to execution engine of DBMS. The in-situ processing is executed in order of READ → Extract → Execution engine in Figure 4. Disk I/O of in-situ processing is less than disk I/O of data loading since in-situ processing removes disk I/O when writing data converted by Extract phase and reading data to process query.

In order to overcome limitations of *external tables*, NoDB [13] improves performance of in-situ processing through improvement of each component that processes raw files in Figure 4. In-situ processing adapted to NoDB reduces the execution time through three optimizations as follows:

- Selective tokenizing: reduces the tokenizing costs by tokenizing data until the required attributes for a query.
- Selective parsing: reduces raw file access costs. NoDB delays converting data into binary representation until determining required data for a query.
- Positional map indexing: reduces parsing and tokenizing costs. The posi-

tional map is created on-the-fly during query processing. Position information about data in raw file is maintained to navigate and retrieve data faster. Figure 5 shows an example of a positional map.

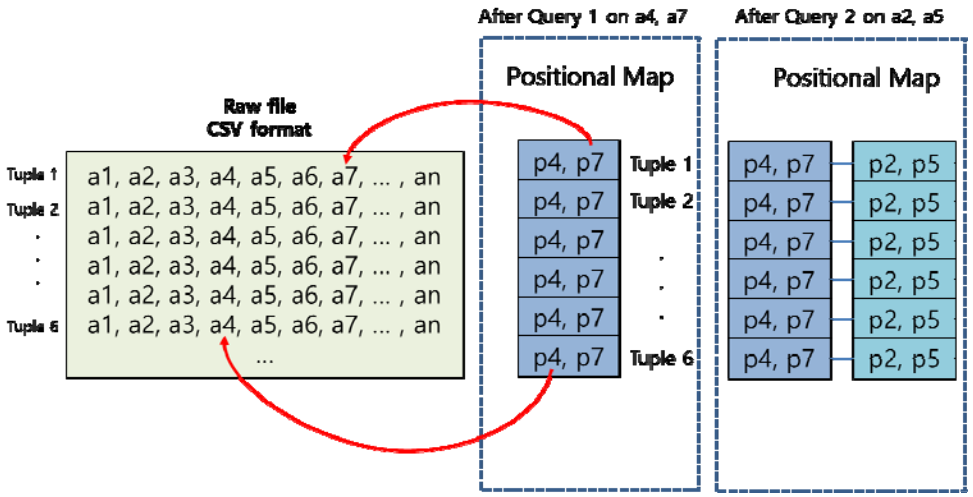

Figure 5. Positional map indexing [13].

SCANRAW [18] presents the performance improvement since each component of the Extract phase processing raw file is implemented using multiple threads suitable for the multi-core processor environment. In order to solve the limitation that in-situ processing is worse than query processing after data loading in the case of continuous query processing about the same raw file, SCANRAW also apply speculative loading that loads a slight data before or after query processing. Figure 6 represents a parallel in-situ processing of SCANRAW. TOKENISE and PARSE components are executed by multiple threads accessing other parts of the data. PARSE component involves MAP component of processing raw file in order to show simply a structural representation. Basically, SCANRAW is to execute *external tables* method in parallel. However, if a query uses CPU resource intensively, SCANRAW loads data using idle I/O bandwidth by speculative loading.

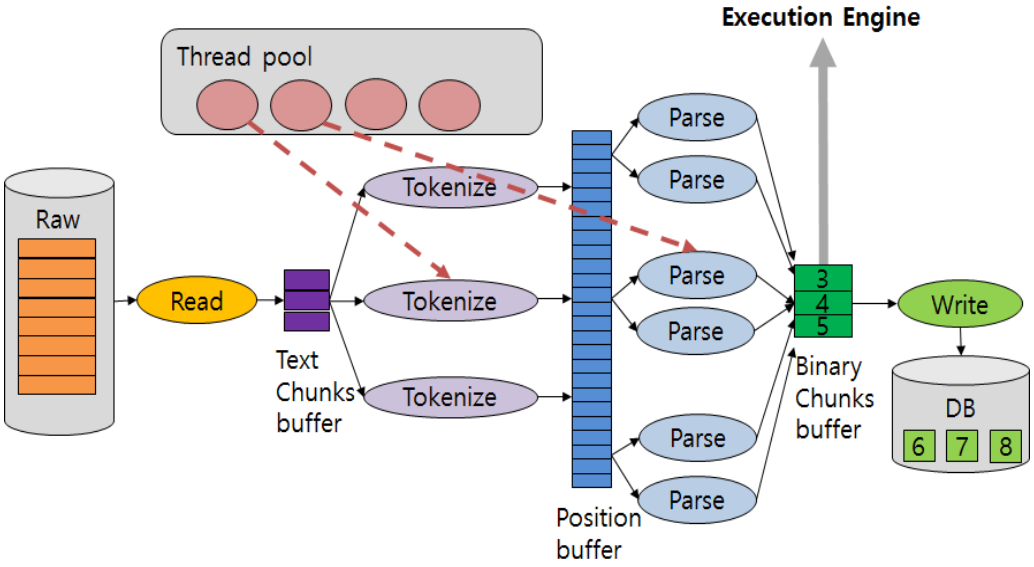

Figure 6. The data processing of SCANRAW [18].

SCANRAW implements three buffers among READ, TOKENIZE, PARSE, and WRITE components in order to execute *external tables* as a pipeline. The role of each buffer is as follows:

- Text Chunks Buffer: stores data read from raw file with chunk as a unit and then provides input data used by TOKENIZE threads.
- Positional Buffer: has text chunk which is output of TOKENIZE threads and the positional map which is position information about data in the raw file. Data in positional buffer is passed as input of PARSE threads. The input data is converted to binary representation and is mapped on data representation of DBMS.
- Binary Chunks Buffer: passes data in this buffer to the execution engine by priority. If processing query requires intensive CPU resource, the data is stored in DBMS storage since speculative loading loads data using idle

disk I/O. The remaining data in this buffer can be used as in-memory caching.

However, in the case that a query requires disk I/O intensively, no data is loaded into the DBMS. Thus, SCANRAW executes safeguard mechanism [18] to load a small amount of data after finishing a query. When SCANRAW starts next query, Safeguard mechanism stores data in binary chunks buffer into DBMS after the data is used as cache. For example, we assume that there are raw file which consists eight chunks of (1, 2, 3, 4, 5, 6, 7, 8) and binary chunks buffer which is able to have three chunks like figure 6. After first query, the remaining chunks in binary chunks buffer are chunk 6, 7, and 8. Thus, safeguard mechanism writes them into DBMS after SCANRAW passes them to the execution engine. A second query processes the chunks in the order of {6, 7, 8, 1, 2, 3, 4, 5}. The status of binary chunks buffer after finishing the second query equates to figure 6. Since chunk 6, 7, and 8 are delivered from the cache, the second query is faster than first query. The third query processes the chunks in the order of {3, 4, 5, 6, 7, 8, 1, 2}. If the same work is performed repeatedly, all the chunks are loaded into DBMS after the fourth query.

Scientific Data Service /Query (SDS/Q) [19] is the in-situ processing system for HDF5 that is the one of standard scientific data formats. The system is implemented suitably on the supercomputer environment that has the large capacity of main memory and parallel file system. Since SDS/Q is able to use the large capacity of memory, the philosophy of SDS/Q is to use in-memory engine that is faster than execution time of the disk based engine. Bitmap indexing used in SDS/Q is more suitable for analyzing multi-

dimensional array than positional map indexing used in NoDB and SCANRAW.

Ⅳ. DISCAN

In this section, we propose the distributed in-situ scan operator processing query on raw scientific data without data loading. First, we introduce to modify the query plan to adapt DISCAN. In addition, we present components and logic of DISCAN.

4.1 In-situ processing

In order to analyze scientific data, the array DBMS like SciDB should load the data into the storage of the DBMS. Data loading is typically a time-consuming process since multiple disk I/O is used. Since SciDB is the most popular distributed array DBMS, We explain The data loading of SciDB to verify overhead when loading data. Figure 7 shows the data loading process of SciDB. The data loading process is composed of three steps of distribution, loading, and redimension steps.

- Distribution step: splits CSV file which has n lines into the number of SciDB instance and then distributes them into each SciDB instance.
- Loading step: transforms the distributed CSV files which have n/k lines into data loading format and then loads them into SciDB as 1-D array format.
- Redimension step: shuffles the loaded 1-D array in order to match 1-D array to the schema of the existing scientific data.

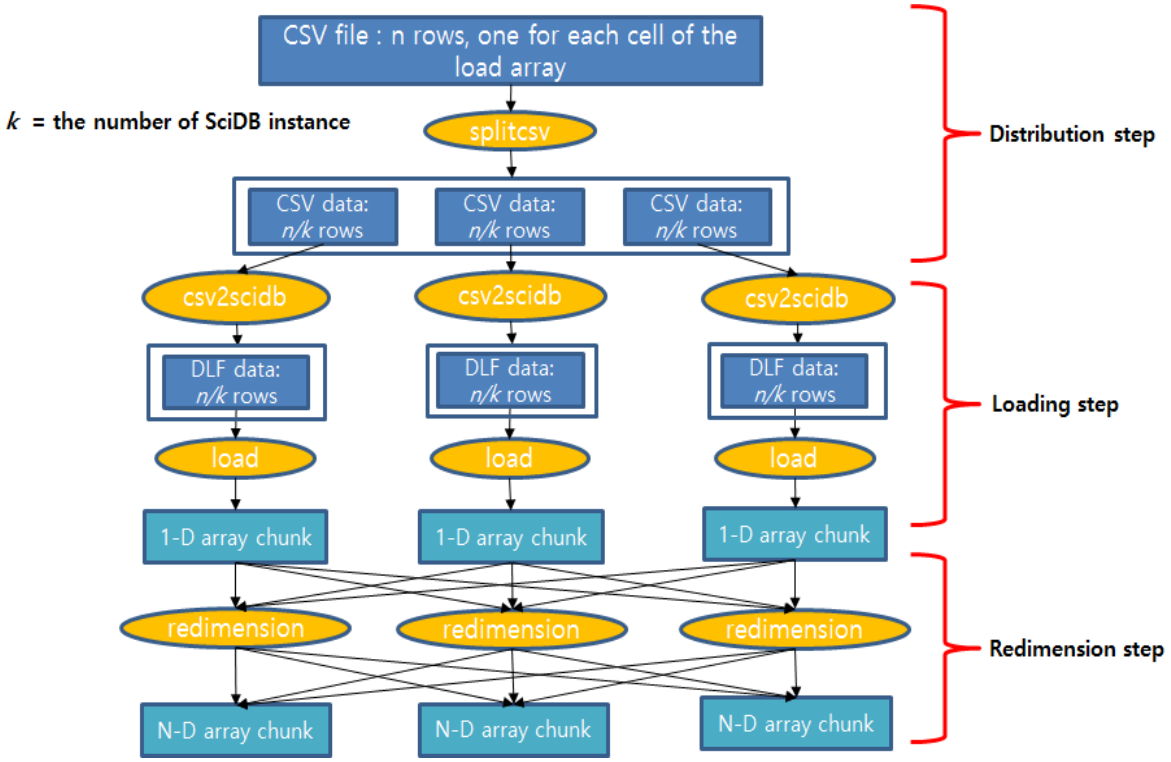

Figure 7. Data flow of SciDB data loading.

For example of loading the distributed data to SciDB, figure 8 represents the transformation of data formats during data loading. The csv2scidb operation which is the external utility transforms CSV file which has four attributes of (event, year, person, time) to DLF data. The DLF data has a line number of CSV file as the dimension of a 1-D array and contents of a line as the cell of the 1-D array. SciDB starts the load operation after executing csv2scidb. The load operation of SciDB loads DLF data into 1-D array since DLF data has a 1-D array schema. If the load operation finishes, SciDB data loading is progressed until loading step at figure 7. After loading 1-D array in SciDB, users should perform redimension step in order to analyze array as the schema of the existing scientific data. In the figure 8, Redimension step is the process converting (year, person) of 1-D array into dimensions of the 2-D array.

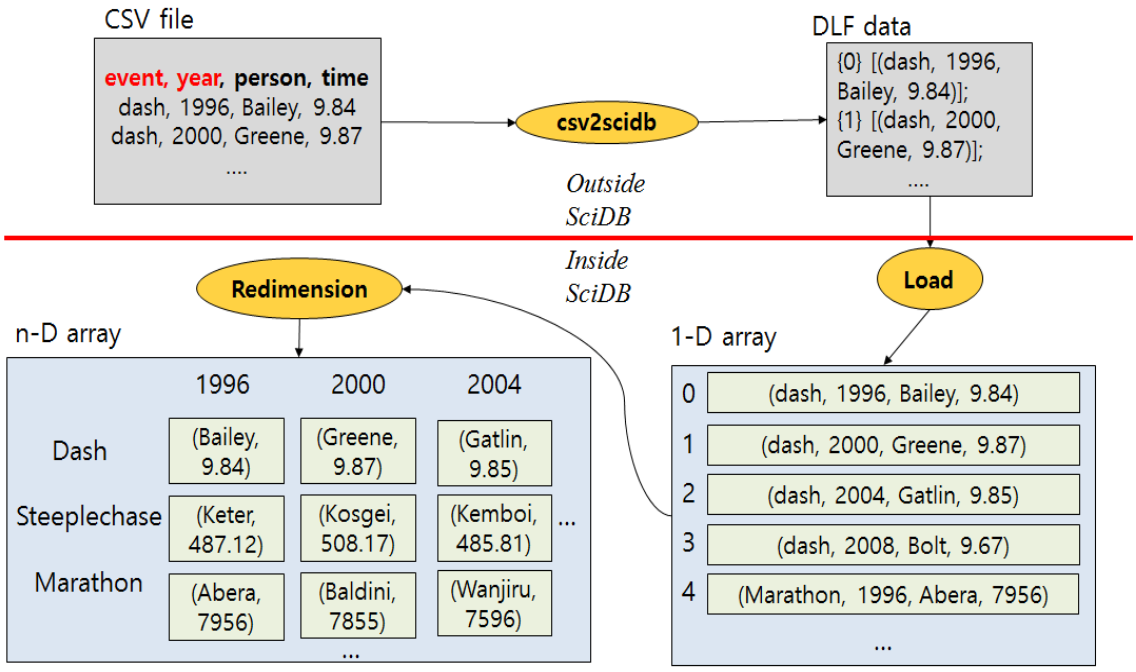

Figure 8. Data layout transformation in SciDB data loading.

In order to evaluate the performance of SciDB data loading, the experiments use cluster consisting of 11 nodes. Each node has Intel Xeon E3-1240 4 core 3.4GHz CPU, 34GB RAM and 4TB HDD of 7200RPM. The network uses 1Gbps Ethernet. Table 1 shows the data used in experiments. The kinds of data is SST, chlor_a, Par, NSST and Rrs_667 of MODIS Aqua Level 3 and is converted to CSV file. Figure 9 shows the performance of SciDB data loading.

Table 1. Data information.

Data type	The number of lines	Size
SST	17.3B	450GB
Chlor_a	20.2B	580GB
PAR	20.2B	580GB
NSST	13.6B	310GB
RRS	20.2B	580GB

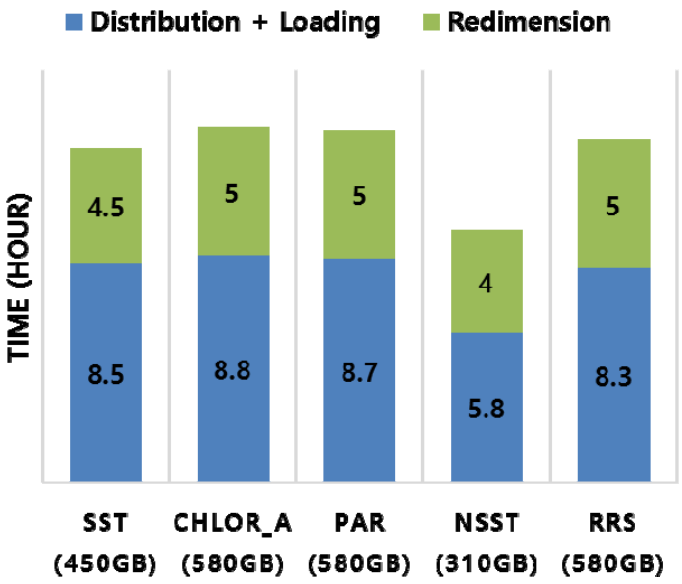

Figure 9. The performance of SciDB data loading.

Each data is loaded to a 3-D array which has dimensions of latitude, longitude and date after loading CSV file to a 1-D array. Figure 9 presents that X axis and Y axis represent the kinds of data and the elapsed time of data loading. Since each data is stored in a file, CSV file is split and distributed appropriately in order to load to SciDB that has distributed system. And then SciDB loads them to the 1-D array. The elapsed time of the two process is the blue box in figure 9. The green box represents the elapsed time of rediemnsion step. Loading 580GB of chlor_a takes about 14 hours. Consider a scientist wants to analyze the red tide phenomenon using chlor_a data. Although the data wanted to analyze is the part of the entire data, the scientist should wait about 14 hours to analyze the red tide phenomenon. Thus, in-situ processing is needed to reduce the time meaningless to scientists.

In order to load scientific data to SciDB, raw scientific file needs to transform to an intermediate format like CSV format called preprocessing. The process that scien-

tific data is loaded in SciDB is showed at Figure 10.

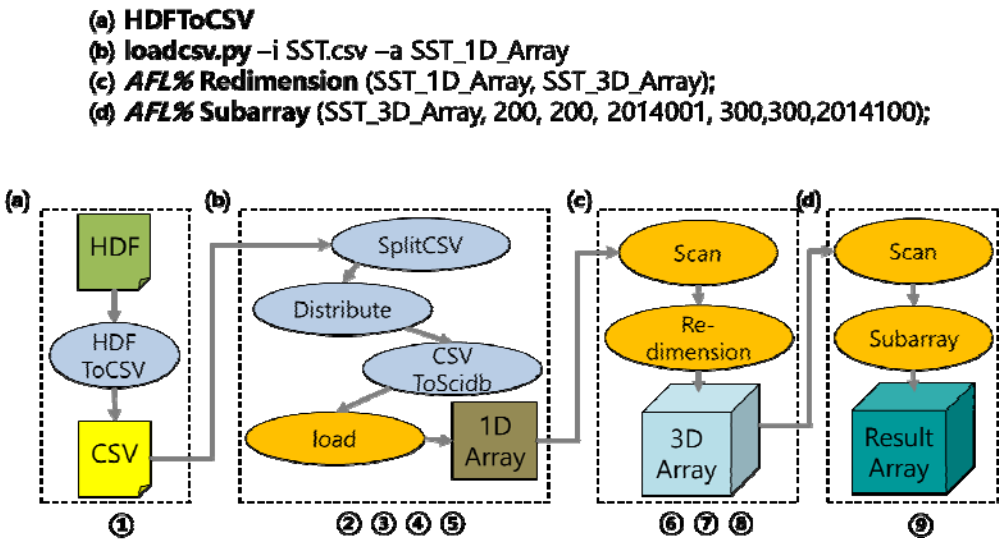

Figure 10. The entire process of SciDB data loading.

Preprocessing transforms raw scientific data to CSV file. Raw scientific data loses the characteristic of multi-dimensional array since CSV file has the 1-D array schema. Redimension step restores the characteristic of scientific data. In order to load scientific data to SciDB, Users execute the commands of (a), (b), (c), and (d) in figure 10. The command (a) is to use the external custom utility that transforms raw scientific data to CSV file. And then users can execute the utility supplying by SciDB. The utility loads CSV file to SciDB as 1-D array. To restore the existing scientific data schema, users should execute a query for redimension step. From now on, users can analyze data using to submit queries to SciDB. In the case that scientific data is generated frequently, scientists want to analyze the data always performs the work which inputs the four commands of (a), (b), (c), and (d). However, in-situ processing requires the less number of commands to users. Figure 11 shows distributed in-situ processing.

(a) HDF_Merger
(b) *AFL%* **Subarray** (SST_3D_Array, 200, 200, 2014001, 300,300,2014100);

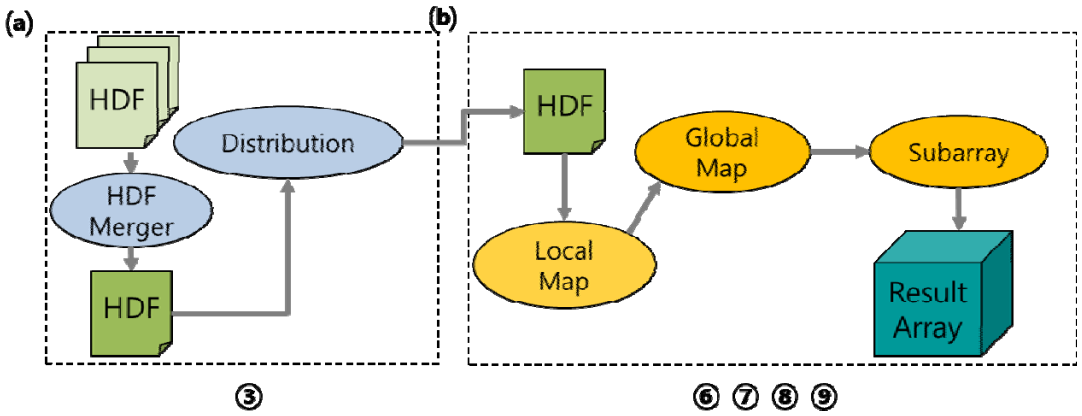

Figure 11. The proces of in-situ processing.

In-situ processing adapted to SciDB can process queries without transforming raw scientific data to data representation of SciDB. Commands (a) and (b) of figure 11 is to perform queries without data loading in SciDB. HDF merger executed by the command (a) merges multiple scientific data as the k files, k means the number of SciDB instances. And then HDF merger distributes them to SciDB instances. After distribution, users can analyze scientific data using command (b) that is the same command (d) of figure 10. If a user submits a query, SciDB executes Distributed In-situ SCAN operator (DISCAN) which consists of Local Map and Global Map components. Local Map, which maps scientific data existing on the local to the internal data representation of SciDB, is performed by SciDB. SciDB stores chunks to SciDB instance depending on the chunk placements policy, called chunk partitioning. Since chunk placement is guaranteed to process queries in SciDB, DISCAN performs Global Map to replace chunks. SciDB performs the remaining work using replaced chunks. DISCAN supplies the convenience to users and outperforms the ex-

isting data loading by more than 60 times.

4.2 Modification of a query plan

In order to execute in-situ processing on SciDB, in-situ analysis path and in-situ analysis layer are implemented on SciDB. Figure 12 shows the architecture of SciDB adapting in-situ processing. In-situ analysis path is responsible for controlling physical data flow. In-situ analysis layer optimizes data access for a query. After client submits query to SciDB, SciDB instance reads scientific data through in-situ analysis path without transforming to data format. If all data is not required to process a query, in-situ analysis layer supply information for the required data and SciDB instance can read only the required data.

Distributed array DBMS like SciDB processes a query using multiple operators. The operator performs the specific function needing to query in DBMS. For example, scan operator is responsible for reading data in database and multiplication operator executes to multiply two arrays. When a query is submitted, Distributed array DBMS generates a query plan composing the tree of operators. Processing a query is to execute operators from the bottom of the tree to the top of the tree. The bottom operator of tree is always the scan operator. To perform in-situ processing on SciDB, the scan operator should change in-situ scan operator that is able to read scientific data and map the data on the internal data representation of SciDB.

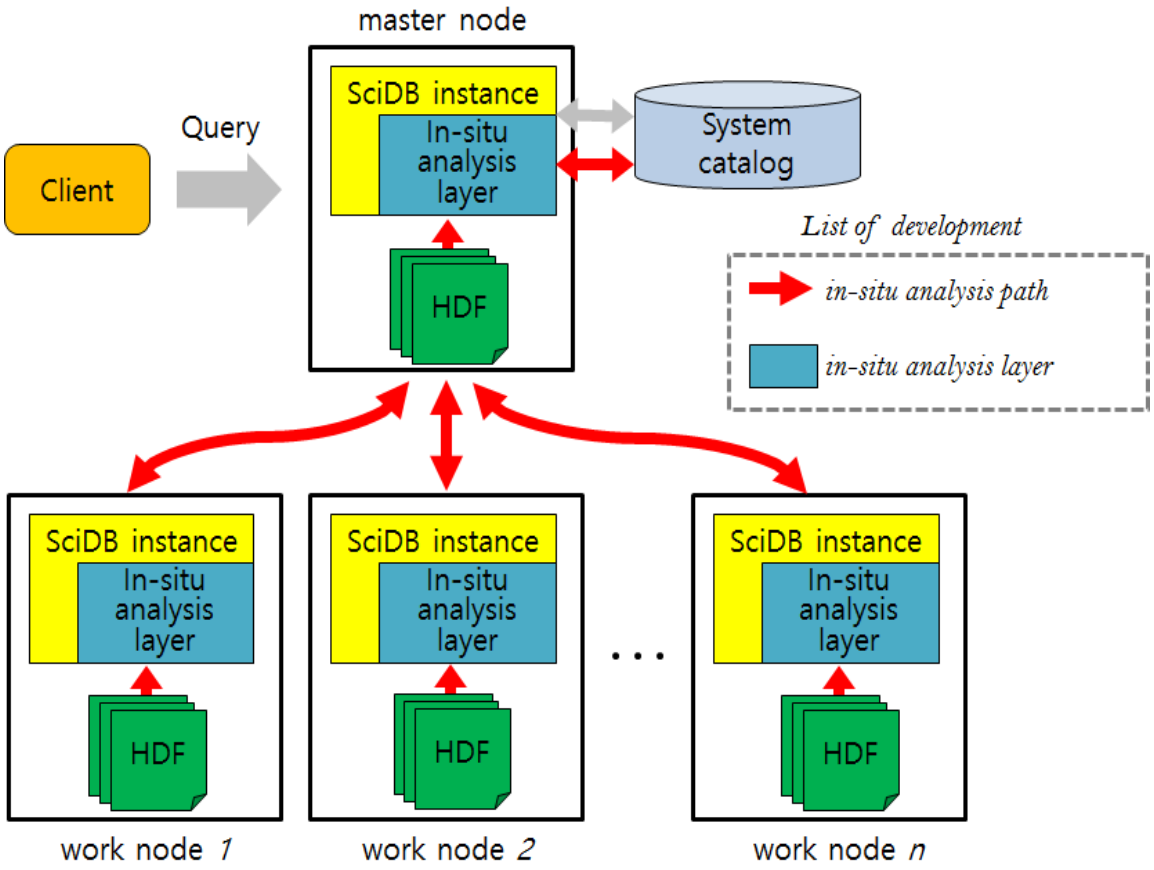

Figure 12. The architecture of SciDB with in-situ processing.

Figure 13 presents to change scan operator to in-situ scan operator in the query plan. When a query joins two partial arrays, query plan of figure 13(a) is generated. The bottom of the query plan consists of scan operators that reads an array required during query processing. In order to process a query without data loading, query plan should change scan operator to in-situ scan operator like figure 13(b). The operator of SciDB guarantees that both the input and the output of operator should be an array. Thus, in-situ scan operator generates an array using data read in scientific data files.

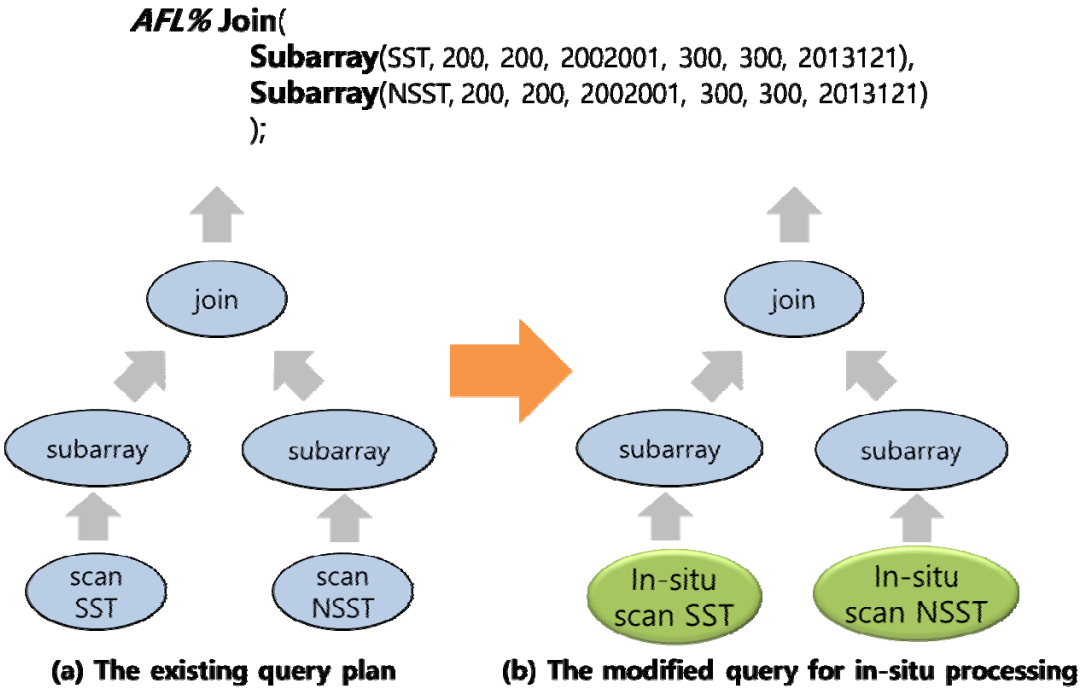

Figure 13. The modifying query plan for in-situ processing.

Typically, raw scientific data is stored in single machine. The scientific data should be distributed to the each node of SciDB since SciDB is the distributed array DBMS. If processing a query with in-situ scan operator requires multiple scientific files, in-situ scan operator opens and closes the file stream for all the files. Opening and closing the file stream is a significant overheads if the number of files increases. Thus, the merging scientific data is needed to guarantee the opening and closing file once. HDF merger merges scientific data as files of a number of SciDB instance. And then the merged files are distributed to SciDB instances. Figure 14 shows the need of HDF merger.

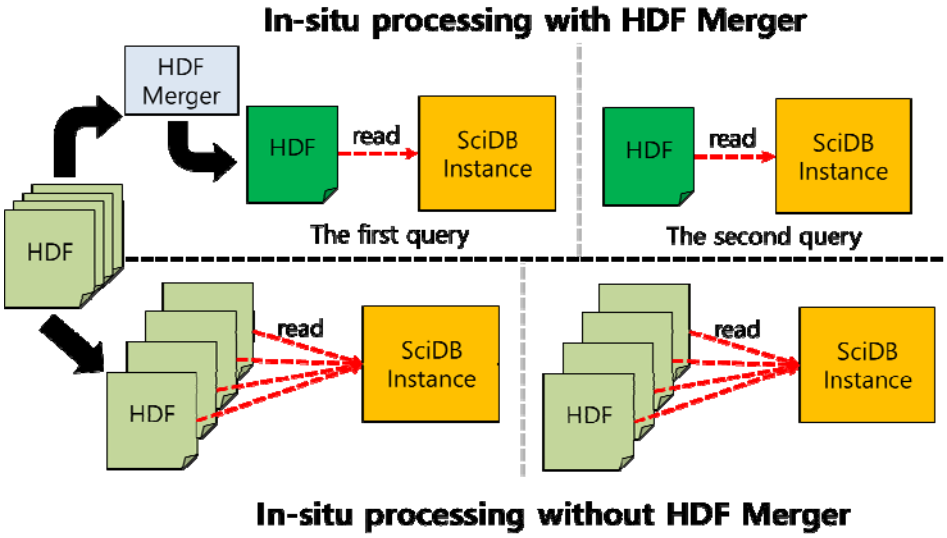

Figure 14. The need of HDF Merger.

HDF merger is responsible for merging multiple HDF files which are scientific data. Figure 15 shows the example of HDF merger. Consider a raw file has 2-D array data that has the dimensions of latitude and longitude and multiple files which has the date from 1/1/2014 to 1/12/2014. We assume that The number of SciDB instances is three. HDF merger merges the twelve raw files as three HDF files. A merged HDF file has 3-D array data and information of the dimensions.

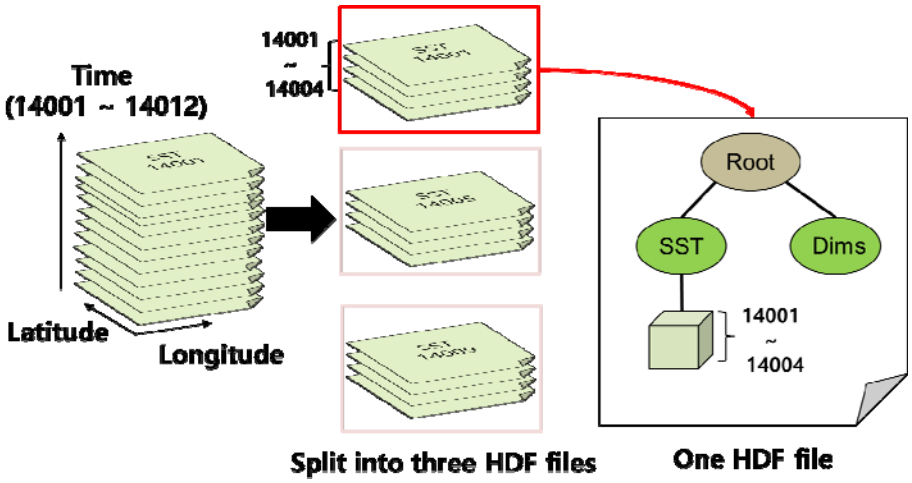

Figure 15. The example of HDF merger.

4.3 The distributed in-situ scan operator

The merged files are distributed and stored in each SciDB instance. Each SciDB instance has one HDF file that stores the part of the entire scientific data. If query is submitted to SciDB after distribution, SciDB generates a query plan that the bottom of the tree is DISCAN and then assigns the work to each node. Each node executes DISCAN to read scientific data. DISCAN starts Local Map. The object of Local Map is to map scientific data on array which is SciDB internal data representation. Figure 16 shows the components of DISCAN. Local Map is consists of the three steps of Filtering, READ, and, Building.

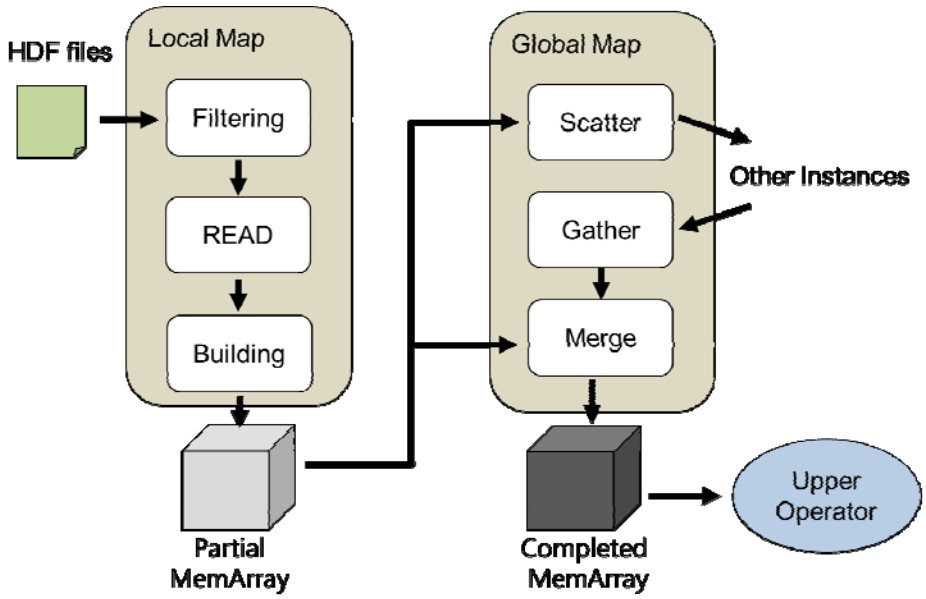

Figure 16. The components of DISCAN

The filtering step gets the filter information of query. When processing array data, query requires commonly the part of the entire data. These queries are called selective query. Figure 17 presents the selective query retrieving the partial data. *Subarray*

(A, 2, 1, 3, 3) is a query to retrieve 3 X 2 array indicating the green region in Array A. If the query is executed after data loading, DBMS can retrieve 3 X 2 array after loading array A. However, DISCAN is able to process the query with accessing 3 X 2 array data. To determine the data required by the query, the Filtering step of Local Map is executed. First, Filtering step searches the query plan of SciDB to find a filtering operator which scans the partial array. The kinds of filtering operator are Between, Subarray, and Slice. The middle picture of figure 17 is the query plan generated when *Subarray (A, 2, 1, 3, 3)* is submitted to SciDB. The Filtering step acquires the filtering operation and provides the information to READ step.

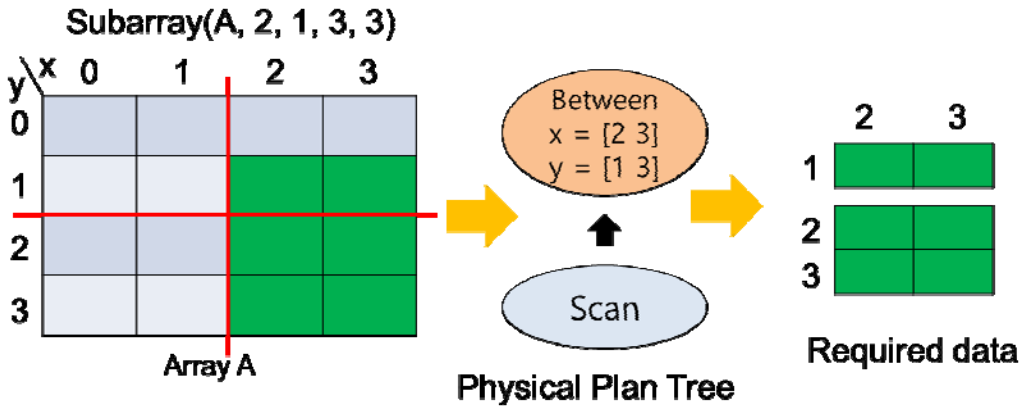

Figure 17. The example of selective query.

READ and Building step is the important step of in-situ processing. Figure 18 shows the processing of READ and Building step. READ step reads only the needed data using the filtering information provided to Filtering step. When reading scientific data, the unit of the read is a chunk. If data is read sequentially, distributed in-situ processing needs the sort phase. SciDB stores array data as a chunk that array is divided and the chunk stores the values in order of dimensions. Thus, DISCAN should sort data

based on chunk Id and position in the chunk.

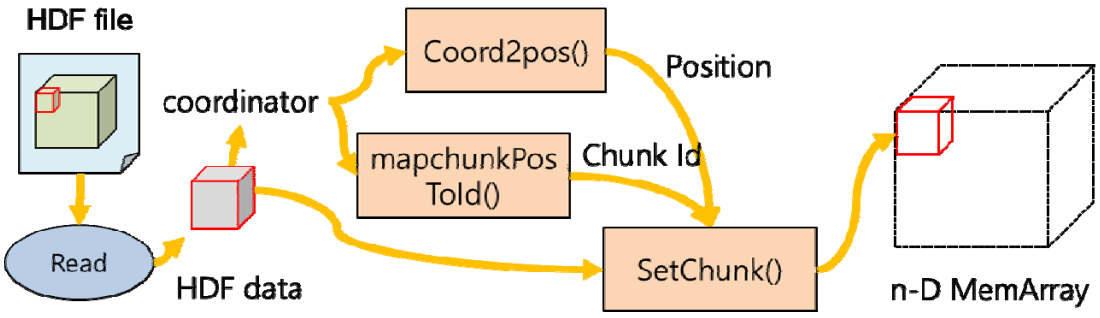

Figure 18. READ and Building step of Local Map

Building step is responsible for mapping chunk read in scientific data on SciDB array. In order to map the read data on SciDB, DISCAN calculates chunk Id and position in the chunk. The information is calculated by Coord2pos() and mapchunkPosToId(). READ and Building step finishes Local Map after converting all the scientific data to SciDB array. The created SciDB array in Local Map does not be used in other operator since the SciDB array does not be guaranteed depending on chunk partitioning. After finishing Local Map, DISCAN executes Global Map to replace chunks according to chunk partitioning based on Hashing. Figure 19 shows the work of Global Map. In the SciDB array created after Local Map, the certain chunks should be in other instance that since the chunks are unnecessary in the local instance. The chunks are sent to other instance corresponding to chunk partitioning via network. Global Map is implemented by scatter/gather of MPI in order to transmit chunk. Scatter step sends chunks to other instance. Gather step is responsible for receiving the chunks. After receiving the chunks, the chunks are merged if chunk Id of the chunk is the same. Finally, completed array can be used in other operator.

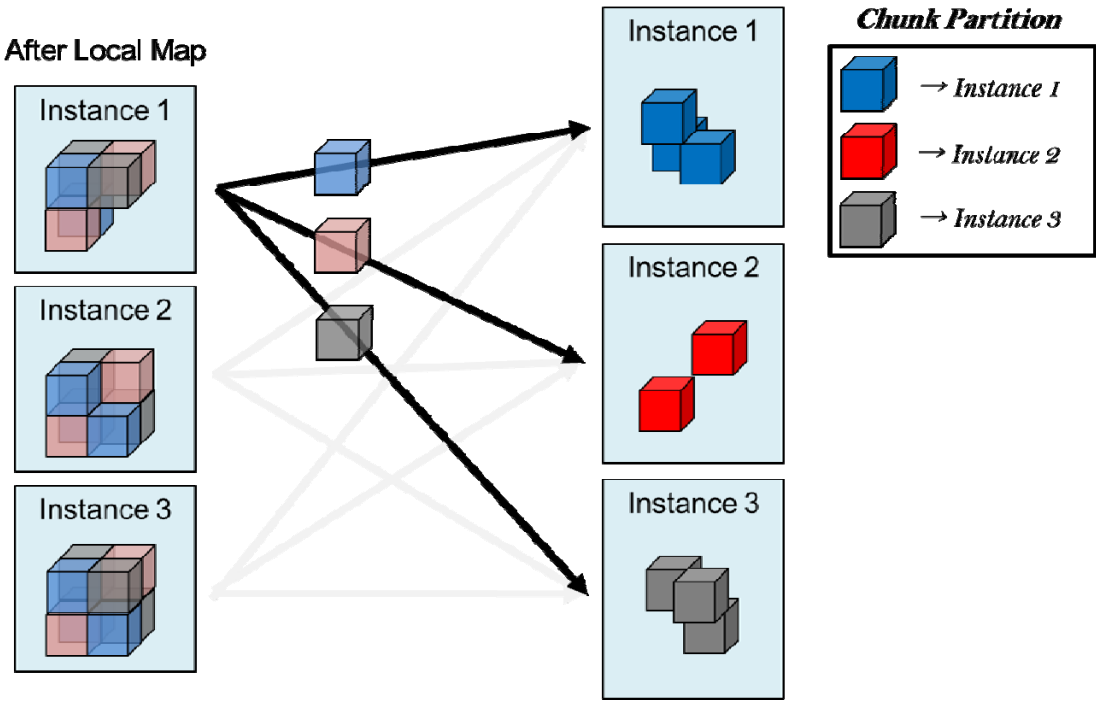

Figure 19. The work of Global Map.

Ⅴ. PERFORMANCE EVALUATION

In this section, we evaluate DISCAN that is the distributed in-situ scan operator processing query on raw scientific data without data loading. First, we show the performance comparison results of DISCAN and query processing after the data loading. In addition, we compare with MATLAB which is the popular matrix processing system.

In order to evaluate the performance of DISCAN, we perform the experiments in our cluster. The cluster consists of ten nodes. Each node has Intel Core i7-5930K and 32GB memory. All the nodes are connected via 10G Ethernet. The used data is 12 datasets of MODIS Aqua L3M which is satellite data. The data format is HDF format and data size is 4.2GB per a dataset. Figure 20 shows to compare with DISCAN and query processing after data loading. The query is *SELECT count(*) FROM Array*, counts the number of cells in Array. CSV data transformed a dataset has the size of 33GB. In figure 20, data loading time takes 60 minutes. Each step of data loading consists of preprocessing step, split & distribution step, DLF transformation step, loading 1-D array step and redimension step. The most time-consuming step of the data loading is split & distribution step. While the data loading process takes a long time, query processing time just takes 0.01 minutes. Our in-situ processing takes 0.98 minutes to complete all the processing. DISCAN outperforms the processing query after the data loading by more than 60 times.

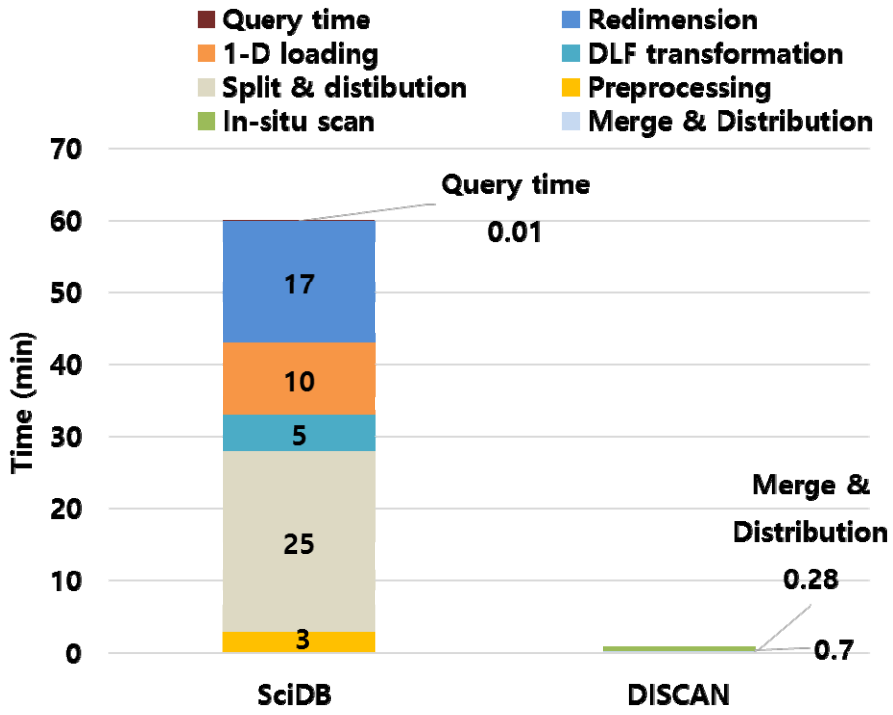

Figure 20. The performance of DISCAN and query processing with data loading.

Since query of figure 20 scans all the data, DISCAN does not expect to improve the performance with selective query. Thus, we evaluate real selective queries used to find the red tide phenomenon of the coast of Korea in order to verify the performance with selective query. Our data has the entire data of earth. The real queries requires about 0.7% of the data. Figure 23 shows the performance of selective query. The query finding red tied needs to access multiple dataset. SciDB loading of figure 20 is the elapsed time about loading dataset required during query processing. After completing data loading, query processing takes just 1 ~ 2 seconds. In order to process four queries finding red tide, Queries FRTD and morel need two datasets, query SS needs three datasets and query TSF needs four datasets. SciDB data loading takes 360 seconds to load a dataset. Thus, the loading datasets takes 720, 1080, and 1440 seconds according to the number of

dataset. Typically, in the case of real query processing, the 12 datasets are loaded to process four queries. DISCAN takes 9.5, 14.7 and 22.2 seconds according to the number of datasets required. In the case of selective query, DISCAN show that remarkable performance improvements can be achieved by more than 75 times.

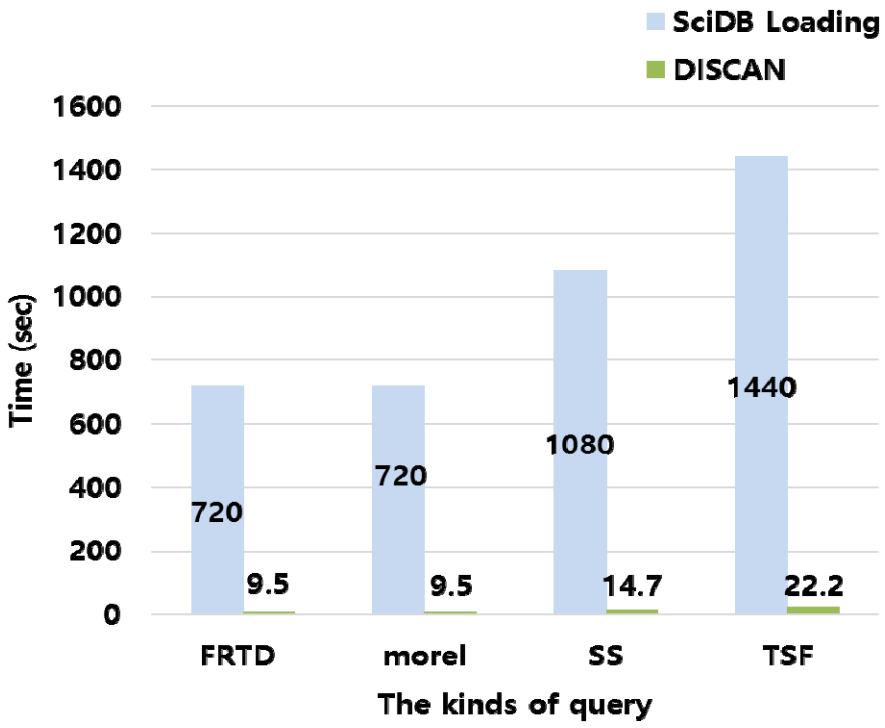

Figure 21. The resutls of selective query finding red tide.

MATLAB is the most popular matrix processing system on a single machine. Since MATLAB supports to access HDF data directly, we compare with DISCAN and MATLAB. Figure 22 presents the experimental results for various selective queries comparing with DISCAN and MATLAB. The size of data is 4.2GB scientific data. The query is *SELECT avg(*) FROM Array GROUP BY first_dim*. MATLAB does not support to access the partial data for selective query even if it supports to read HDF data directly. Thus, MATLAB always takes the

same time. DISCAN takes 9 seconds more than MATLAB when processing a query that has selectivity 100%. Since distributed system is suitable for processing large-scale data, the initializing the job on the system becomes overheads. However, in the case of other query that has less selectivity than 100% selectivity, DISCAN outperforms MATLAB by more than 6.6 times.

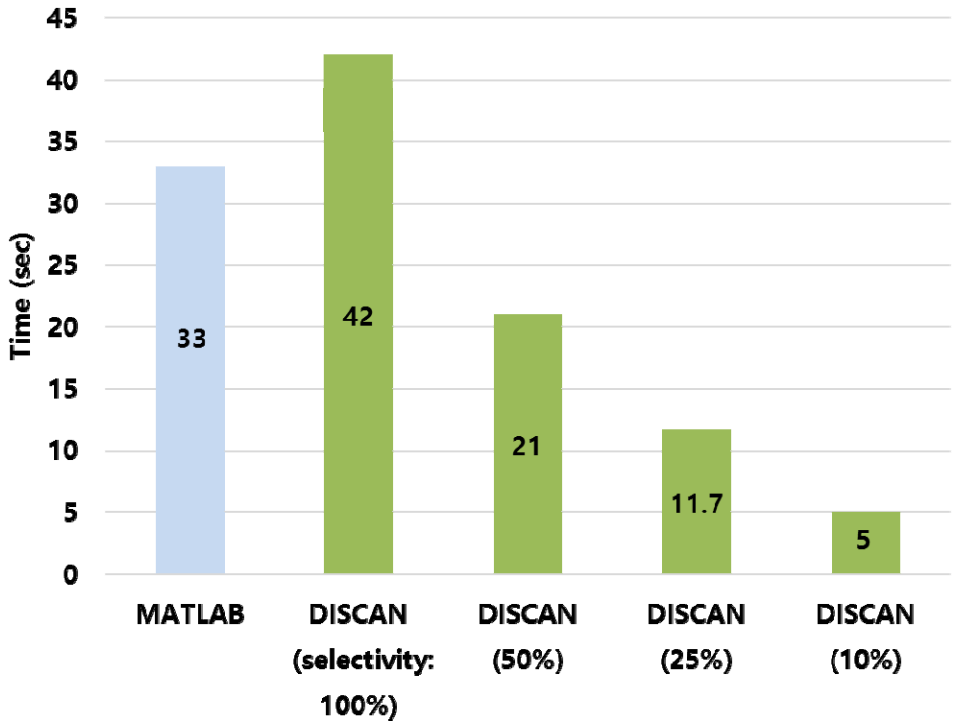

Figure 22. The performance comparing with MATLAB for selective query.

We verify that DISCAN shows better performance than MATLAB as the size of data increases. Figure 25 shows the experiment about the data scalability. The experiments uses data of 4.2GB, 8.4GB, and 12.8GB. Green line in figure 23 is the results of MATLAB. Blue and red lines are the results of DISCAN that has selectivity 50% and 100%. Although MATLAB shows the most performance in the case of 4.2 GB data, DISCAN outperforms MATLAB

as the size of data increases. Especially, DISCAN shows the performance improvement more than 13 times when analyzing 12.8 GB data.

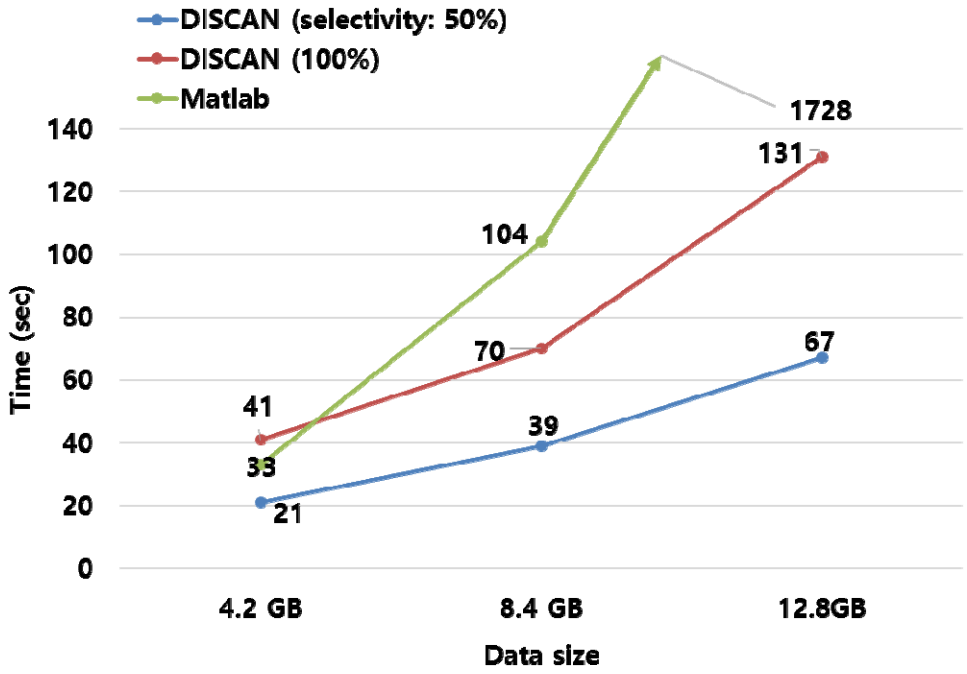

Figure 23. The perfomance according to the data scalability.

We evaluate the performance of fundamental operation when analyzing matrix. The synthetic 2-D array data is used in this experiment. The kinds of data are 0.4GB of 10000 X 10000 array, 1.5GB of 20000 X 20000, and 3.4GB of 30000 X 30000 array. In the cast of multiplication, the same size of two data are used to multiply the data. The queries used on the experiments are the transpose, the multiplication and the SVD. The transpose is less computation than other queries. We measure the scan performance of DISCAN, the operation performance of SciDB and the performance of MATLAB. This experiment of transpose result in figure 24. MATLAB does not process 30000 X 30000 array since the array does be assigned at the memory. Both SciDB with DISCAN and MATLAB have similar

performance at the results of 0.4GB and 1.5GB. In the case of transpose operation, SciDB operation finishes in 1 ~ 2 seconds since transpose has less computation. In other words, processing transpose operation takes up the most time to scan data. Since hard-ware is the same, SciDB with DISCAN and MATLAB have the same performance of disk scan. However, SciDB with DISCAN is able to process large-scale data which is processed in MATALB.

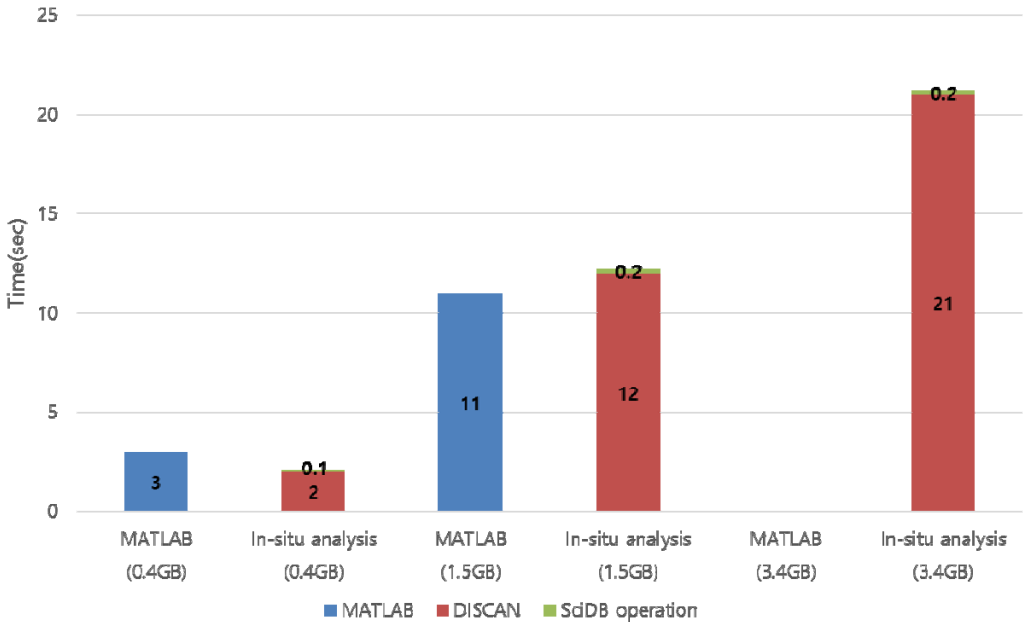

Figure 24. The result of transpose operation.

In the case of lots of computation, MATLAB outperforms SciDB with DISCAN. Figure 25 and 26 shows the results of multiplication and SVD. SciDB operation has the most time of the elapsed time. We can know that SciDB operation is excessively slower than MATLAB. Thus, SciDB operation needs the study for optimizing operator.

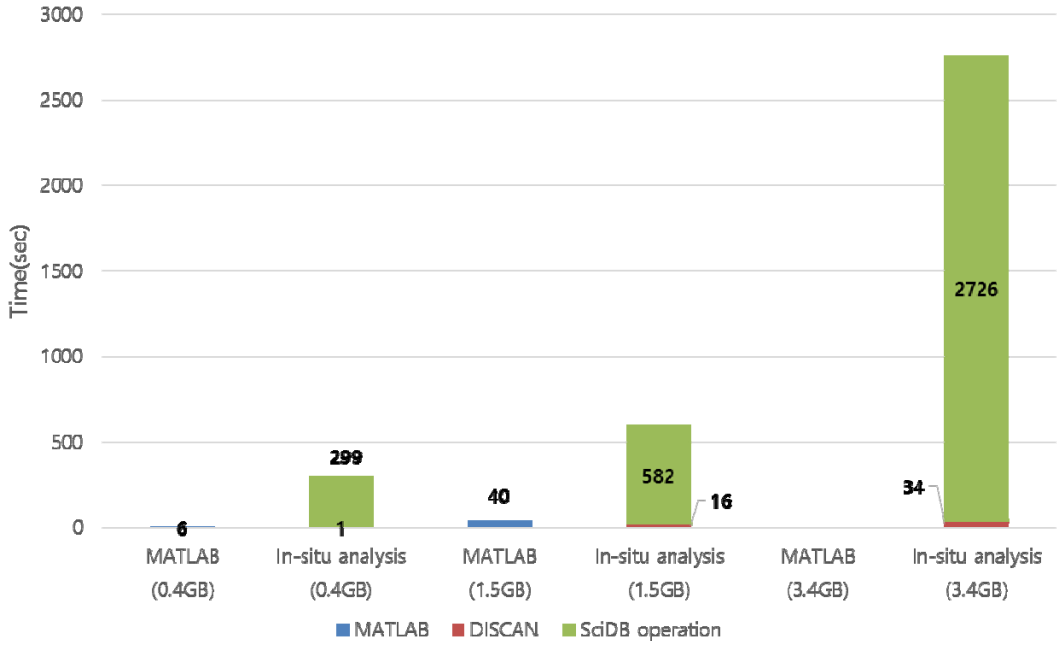

Figure 25. The results of multiplication operation.

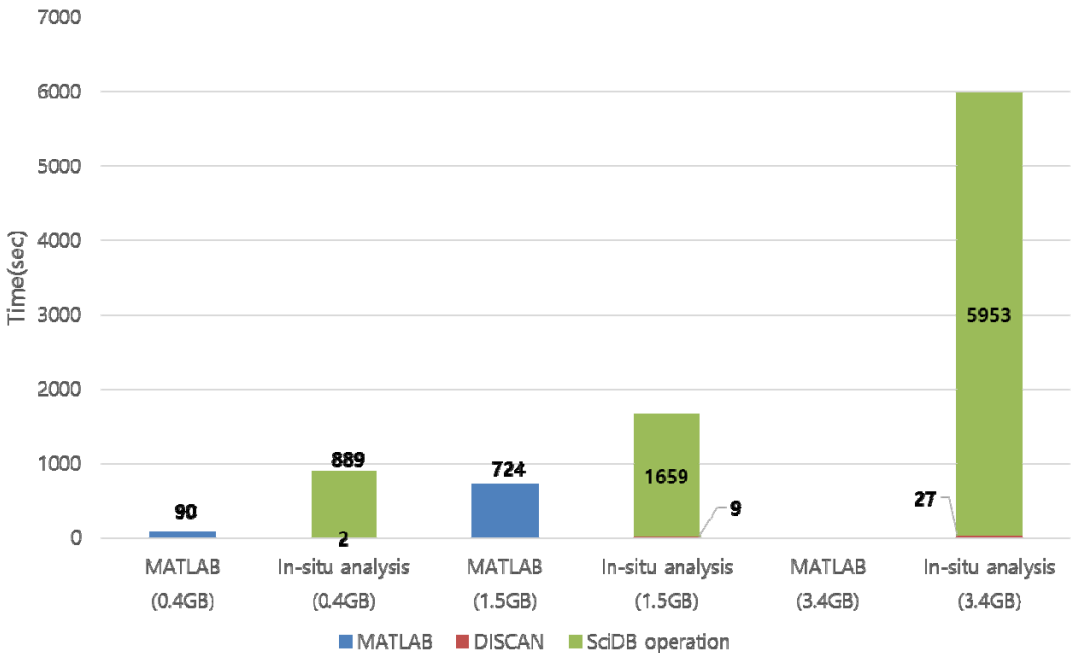

Figure 26. The results of SVD operation.

If DISCAN is used continuously for the same data, the accumulated time of in-situ analysis exceed the data loading time. Figure 27 presents the accumulated time when us-

ing DISCAN continuously for the same data. The accumulated time of in-situ analysis is larger than the data loading time if in-situ analysis for the same data is executed more than 62 times. The data loading is to transform raw data into the data format optimized to process queries on the DBMS. Thus, query processing after data loading takes less time than in-situ analysis. However, the existing data loading takes a long time since the raw data is transformed four times unnecessarily. DISCAN can process queries on DBMS without the transformation of data formats. If the result of in-situ analysis is stored in the DBMS, it is to load data as one data format transformation.

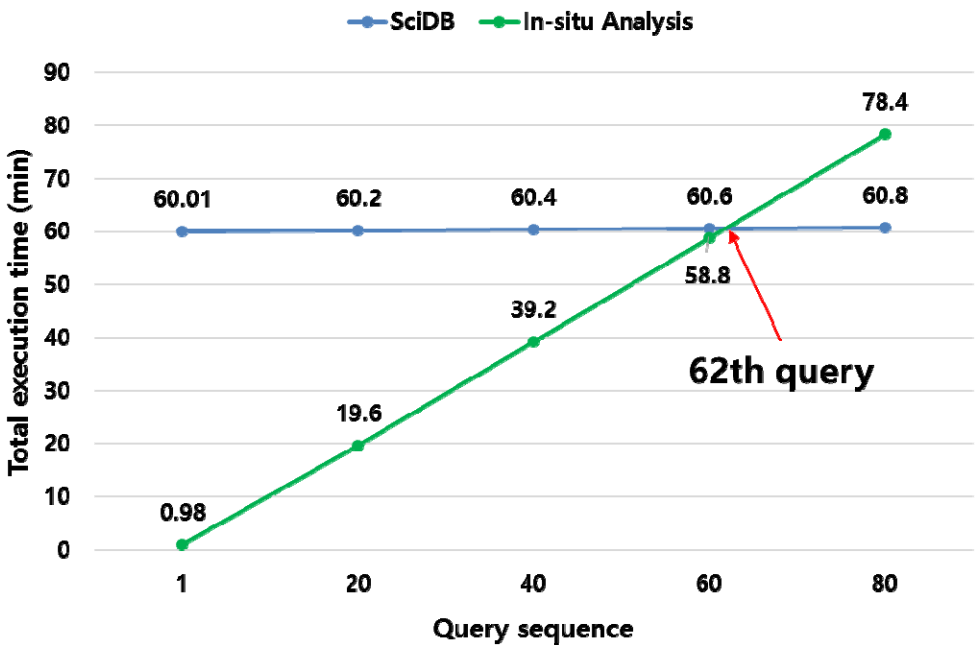

Figure 27. The cumulative time for a sequence of queries.

The performance of data loading using DISCAN is shown in figure 28. The data used in the experiment of figure 28 is 300GB of HDF data. If the data is transformed into CSV data, the size of the CSV data becomes about 2.3TB. The SciDB data loading of figure 28 is evaluated proportionally for the data loading of small data since the data loading of

large data takes a long time. The data loading using in-situ analysis completes the loading using 10% time of the existing data loading time. Thus, DISCAN shows superior performance compared to the query processing after data loading. In other words, The time until the first query processing is reduced about 61 times. And Data loading time is reduced about 10 times in the case of large data.

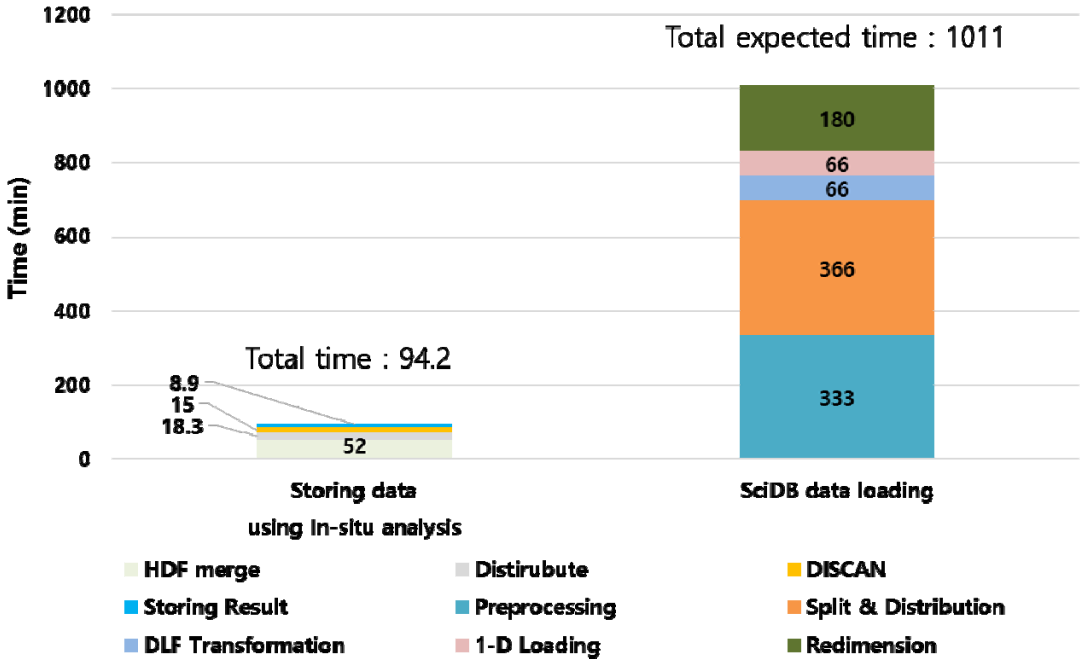

Figure 28. The data loading using in-situ analysis.

VI. CONCLUSION

In this work, we propose DISCAN that is the distributed in-situ analysis method in the distribute array DBMS. Our in-situ processing considers the approach to improve the performance of data loading in SciDB. DISCAN supports the in-situ processing over raw scientific data and optimization about selective query. The in-situ processing of DISCAN ensures to process a query at upper operator since the two important components, Local Map and Global Map, is implemented. Local Map maps scientific data on SciDB array and Global Map shuffles the data generated after Local Map depending on chunk partitioning. DISCAN also takes advantage of well-defined scientific data format libraries in order to process the partial data in scientific data.

We implement DISCAN in a state-of-the-art distributed array DBMS and evaluate its performance across real-world datasets. DISCAN responds more 60 x faster than the query processing after data loading.

Ⅶ. REFERNECE

[1] HDF4. www.hdfgroup.org/products/hdf4.

[2] NetCDF. www.unidata.ucar.edu/software/netcdf/docs.

[3] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman and J. H. Saltz, “Titan: A High-Performance Remote Sensing Database.”, In Data Engineering, pp.357-384, 1997.

[4] C. Chang, A. Acharya, A. Sussman and J. H. Saltz, “T2: A Customizable Parallel Database for Multi-Dimensional Data”, ACM SIGMOD Record, vol. 27, pp.58-66, 1998.

[5] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch and N. Widmann, “The Multidimensional Database System RasDaMan.”, ACM SIGMOD Record, vol. 27, pp. 575-577, 1998.

[6] N, Widmann and P. Baumann “Efficient Execution of Operations in a DBMS for Multidimensional Arrays.”, In SSDBM, pp.155-165, 1998.

[7] A. P. Marathe and K. Salem, “Query Processing Techniques for Arrays.” The VLDB Journal, vol. 11, pp.68-91, 2002.

[8] A. R. Van Ballegooij, “RAM: A Multidimensional Array DBMS.”, In Current Trends in Database Technology-EDBT Workshops, pp.154-165, 2004.

[9] SciDB Development Team, “Overview of SciDB: Large Scale Array Storage, Processing and Analysis.”, In SIGMOD, pp.963-968, 2010.

[10] Hadoop. http://hadoop.apache.org

[11] Thusoo, Ashish, et al. "Hive: a warehousing solution over a map-reduce framework." Proceedings of the VLDB Endowment 2.2 (2009): 1626-1629.

[12] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. (2013). “Scale-up vs scale-out for Hadoop: time to rethink?”. In Proceedings of the 4th annual Symposium on Cloud Computing (SOCC ’13). ACM, New York, NY, USA, Article 20, 13 pages.

[13] I. Alagianis, R. Borovica, M. Branco, S. Idreos and A. Ailamaki. “NoDB: Efficient Query Execution on Raw Data Files”, In SIGMOD, pp.241-252, 2012.

[14] M. Stonebraker, J. Becla, D. Dewitt, K. T. Lim, D. Maier, O. Ratzesberger and S. Zdonik, “Requirements for Science Data Bases and SciDB”, In CIDR, vol. 7, pp.173-184, 2009.

[15] Shvachko, Konstantin, et al. "The hadoop distributed file system." Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010.

[16] A. Witkowski, M. Colgan, A. Brumm, T. Cuanes and H. Baer, “Performant and Scalable Data Loading with Oracle Database 11g.”, An Oracle White Paper, 2011.

[17] MySQL CSV Storage Engine. http://dev.mysql.com/doc/refman/5.0/en/csv-storage-engine.html.

[18] Y. Cheng and F. Rusu, “Parallel In-Situ Data Processing with Speculative Loading.”, In SIGMOD, pp.1287-1298, 2014.

[19] S. Blanas, K. Wu, S. Byna, B. Dong and A. Shoshani, “Parallel Data Analysis Directly on Scientific File Formats.”, In SIGMOD, pp.385-396, 2014
[20] MDSplus. www.mdsplus.org.
[21] R. Brun and F. Rademakers, “ROOT-An Object Oriented Data Analysis Framework”, In AIHENP’96 Workshop, 1997

요 약 문

분산 환경 기반 시스템에서 과학 기술 빅데이터 in-situ 분석 방법

최근 과학 기술 데이터에 대한 분석 수요가 크게 증가하면서 SciDB 와 같은 array DBMS 들이 널리 사용되기 시작했지만, 이들 시스템들은 데이터 로딩의 오버헤드가 매우 크고 로딩을 완전히 끝내기 전에는 데이터 분석을 할 수 없다는 문제점을 가지고 있다. 데이터를 DBMS 에 로딩하지 않고 RAW 데이터에 대해 in-situ 분석방식을 적용한다면 과학 기술 데이터의 분석의 속도와 편의성을 크게 향상 시킬 수 있다.

본 논문은 분산 환경 기반의 과학 기술 데이터에 대한 in-situ 분석 방법을 the-state-of-the-art distributed array DBMS 에 적용하는 방법을 다룬다. In-situ 분석방법을 적용하기 위해 HDF merger 와 in-situ scan operator 인 DISCAN 을 구현한다. HDF merger 는 다수의 RAW 파일들을 SciDB 의 instance 에서 disk I/O 성능을 최대한 활용하기 위해 in-stance 당 하나의 파일로 병합하다. DISCAN 은 SciDB 에서 질의를 분석 시 동작하는 것으로 로컬에서 RAW 파일의 데이터를 SciDB 내부 자료 구조로 변환하는 Local Map 과 instance 들 간에 chunk 들을 재배치하는 Global Map 으로 구성된다.

DISCAN 은 질의에 따라 데이터 로딩 후 질의처리 하는 것에 비해 최대 6123% 성능을 개선한다. 또한, 대한민국 근해 적조 탐색을 위한 실제 질의에 대해서도 모든 데이터 셋을 로딩하지 않아도 되며 부분적인 데이터 접근을 통해 75 배 이상의 성능을 개선한다.

핵심어: 데이터 로딩, 과학 기술 데이터, 분산 환경 시스템. In-situ 분석방법, array 데이터 베이스