Scalable Parallel Data Loading in SciDB

Sangchul Kim Junhee Lee Taehoon Kim Bongki Moon
Seoul National University
Seoul 08826, Korea
{stdio@, jvl@tcs., thkim@dbs., bkmoon@}snu.ac.kr

***Abstract*—SciDB is an array-based DBMS popularly used for scientific data analysis. One major hurdle in processing large-scale scientific data is pre-processing before loading data, which includes extraneous file conversion from a raw data format to a software-specific format, which causes a significant I/O overhead. Moreover, data loading is typically followed by array transformation, which requires data to be sorted and redistributed by hashing. In order to reduce the overhead, we streamline the conversion process and modify the distribution method in loading stages. In addition, we eliminate two heavy-duty steps, namely sort and redistribution, which account for a dominant portion of the redimensioning cost. Our experiments show that the data loading time can be reduced up to 65% compared with the vanilla loader of SciDB.**

***Index Terms*—Array Database Model; SciDB; Data Loading;**

I. Introduction

Scientific experiments, simulations, and observations generate scientific data, commonly stored in dedicated structured file formats such as Hierarchical Data Format version 5 (HDF5) [1] and Network Common Data Form (NetCDF) [2]. Since those files contain their metadata including dimensions and attributes, it helps scientists interpret logical data structures in different platforms, thereby improving portability. In addition, most of the scientific data have dimensions or values represented in coordinates. They are commonly stored in arrays where elements close to each other can be stored physically in close proximity, so that the locality of coordinate systems is preserved.

For the reasons, an array-based model is considered more appropriate for scientific data management rather than a traditional relational data model [3]. Over the past few decades, much research has been done to develop query languages [4], [5], [6] and database management systems [7], [8] specialized for array manipulation and analysis. One of the popular array DBMSes is SciDB [8]. It stores data as multi-dimensional arrays and shows great performance on scientific workloads and complex analytics. Since most of the scientific applications need mathematical functions and algorithms for scientific data analysis, data loading is a beginning step to process these repeated and iterative operations in SciDB.

The initial format of loaded data is a one-dimensional array, inadequate to compute linear algebra operations and utilize the array model, even though scientific data have a few dimensions. In order to reshape one-dimensional arrays to multi-dimensional arrays, data conversion is required. SciDB provides the conversion operator, $redimension()$, transforming attributes to dimensions or vice versa. It is analogous to both indexing and clustering, making data objects close to each other stored together. Since redimensioning ensures the physical locality of data, it is an essential function and considered a core operator. Hence, we mean by data loading the entire process containing the loading and redimensioning. However, the data loading takes a significantly long time and causes extreme overhead. Our experiment shows that the entire process takes about 151,300 seconds (about 42 hours) on RRS data. In addition, when a raw file format is not supported for SciDB, the situation is even worse since it incurs additional cost to convert the file format.

In this paper, we propose a new scalable loader, *Scalable Loader for SciDB* (*SLS*), which accelerates data loading on SciDB. In the loading stage, data are distributed by Message Passing Interface (MPI) and a consistent distribution method. Since the vanilla-SciDB loader does not regard how to partition data in the redimensioning stage, it causes communication overhead in the stage. The consistent distribution method, using the identical function of the redimensioning stage for re-distribution, can minimize the overhead by removing all-to-all exchange in the stage, which also leads to scalable data loading. In addition, as simplifying the conversion process and modifying the distribution method, we can reduce the loading time substantially. Especially, redimensioning, which occupies a significant portion of time in data loading, is optimized by removing sort and redistribution. We implement a new redimension operator using user-defined operators, plug-in method.

The main contributions are as follows.

- We simplify the process of file format conversion in loading stage such that data can be loaded directly from HDF5 data format.
- Instead of Secure SHell (SSH), we use MPI to distribute the data to each instance of SciDB, reducing the workload of the master node.
- In the redimensioning stage, sort is eliminated explicitly by pre-sorting data in the loading process.
- The all-to-all data exchange is removed from the redimensioning stage by distributing data using hash in the loading stage.

The rest of this paper is organized as follows. Section II explains an existing loading method. In Section III, we de-

This research was partly supported by KISTI, Korea (K-15-L03-C01-S03) and PF Class Heterogeneous High Performance Computer Development (NRF-2016M3C4A7952587).

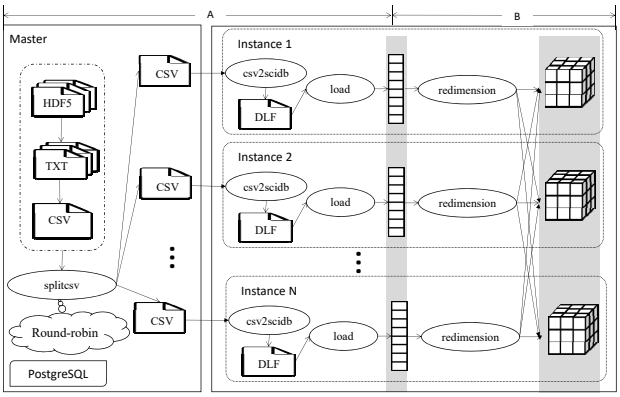


Fig. 1: Data loading in SciDB:
(A) Loading stage, (B) Redimensioning stage

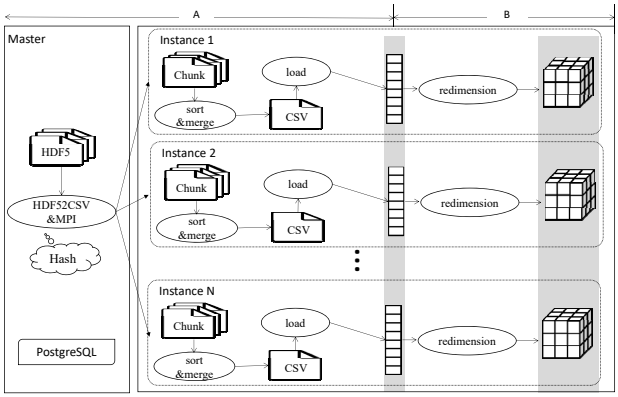


Fig. 2: Improved data loading by *SLS*:
(A) Loading stage, (B) Redimensioning stage

scribe the design and implementation of *SLS*. Section IV shows the experiments and the results. Lastly, we summarize and conclude our work in Section V.

II. Data Loading in SciDB

In this section, we describe data loading process in vanilla SciDB. The overall process is described in Figure 1 and explained below in more detail.

A. Loading

As data format is various, data should be converted into an input file format for SciDB loader. Pre-processors are used to convert raw data into CSV which is an input file format accepted by SciDB. The master node converts raw data (*i.e.*, HDF5) to intermediate files (*i.e.*, TXT) and then converts them to a Comma-Separated Value (CSV) files.

Afterwards, files are distributed by round-robin fashion because load operator supports distributed loading. This process is only executed on the master node alone. Afterwards, the files are distributed to corresponding instances using SSH and pipes, and each instance receives the data, converting them into DLF (Data Loading Format) formatted text file describing a one-dimensional array of SciDB. After the conversion is done, DLF files are loaded as a single one-dimensional array.

B. Redimensioning

The one-dimensional array is transformed to an n-dimensional array to make data close physically as well as to use dimensions as indexes. There are four major steps in the redimensioning stage. Data are prepared in the first step, which involves converting the array into a one-dimensional array and matching each attribute to a corresponding coordinate. The reason for the conversion is to facilitate all cells to be sorted in the next step.

The next step is to sort cells. If one of the cells is not the sorted order, then all elements should be sorted, which accounts for half of the total redimensioning time. The third step aggregates sorted elements into a temporary array, performed before distributing data to actual target instances. Last, the prepared data in the third step are redistributed based on hash values, and then each instance stores the received data. After redistribution, the instances synchronize with each other.

III. Scalable Loader for SciDB

In this section, we illustrate how the loading and redimensioning processes are improved. Figure 2 shows the whole picture of data loading process. We assume that the relevant schemas of one-dimensional and n-dimensional arrays are already defined. That is, the metadata for a one-dimensional or n-dimensional array is available, and we use this information to optimize the loading process. Our new approach to data loading is named *Scalable Loader for SciDB*(*SLS*), scalable by eliminating all-to-all exchange in the redimensioning stage.

A. Loading

Initially, because the raw data (i.e., HDF5) resides only in the master node, they should be split and sent to corresponding instances for parallel loading. Before sending the data, *SLS* loader reads HDF5 files and gets the metadata (i.e., properties of data). The files are then converted into CSV format. Unlike to vanilla SciDB, a CSV file is not stored on disk, and the next process is directly started. When the conversion is done, *SLS* executes the process being cell-to-slave mapping. We use a consistent distribution method that uses the same method that a redimension operator uses. Since this operator distributes data by hash, *SLS* calculates a hash value corresponding to each cell. The hash value comes from a chunk ID and the number of instances determining where SciDB instance the data should reside.

Cells are sent to the corresponding instances in an MPI packet using hash. MPI is a de facto standard interface to assist in communicating among processes or machines. With MPI, the master node sends data to the corresponding instances, converting from HDF5 to CSV file format on the fly. Consequently, we can minimize I/O overhead and reduce the loading time.

Slave instances are in standby mode for receiving cells merged into local CSV files called *chunk* files. Each SciDB

instance receives the data through MPI, and each cell is stored in a corresponding chunk file. In this process, the cell-to-chunk mapping which determines the chunk file to store each cell is executed following the array schema, and the cells are collected to each chunk file that has a unique chunk ID indicating the chunk sequence order.

SciDB explicitly checks whether all cells are sorted by the specified order during the redimensioning process. To avoid sorting which consumes the significant time, we execute the inter-bucket sorting ahead of redimensioning. The chunk files are read by increasing order with chunk ID and merged simultaneously. While merging the files, the cells in the chunk file are sorted internally by intra-bucket sorting. Through these sorting processes, especially bucket-sorting, we can reduce the time and bypass the sorting in the redimensioning stage. After the prior work is done, load() is executed for loading the sorted files to a one-dimensional array.

B. Redimensioning

The steps in the redimensioning stage are the same as vanilla SciDB except for sort and redistribution (all-to-all exchange) steps. The right portions of Figure 1 and Figure 2 illustrate the redimensioning stage of the vanilla SciDB loader and *SLS*, respectively.

Before executing load(), a master mode sends data using hash, and data are sorted. These make no further sorting and redistribution in the redimensioning stage. In addition, the third step is also simpler because the preparation for aggregating data is not necessary. Consequently, in the redimensioning stage, the first step remains for converting a redimensioning form and mapping attributes to corresponding coordinates. The third and fourth steps still exist without preparation and redistribution of the data. The parallel loading is improved because all nodes need not communicate each other for redistribution, a barrier to scalability. This performance is described in Section IV-B. The operator, variant of *redimension()*, is implemented by user-defined operators.

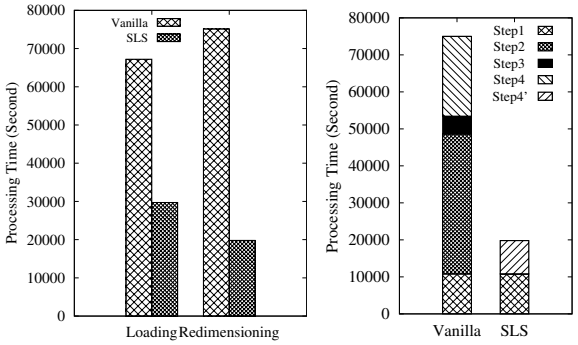
IV. Evaluation

In this section, we analyze the performance of *SLS* by comparing it with that of the vanilla loader of SciDB. The datasets used in the experiments were real scientific data from MODIS. In all the experiments presented in this section, data loading times were measured in the wall-clock time.

A. Experimental Environment

We used ten nodes on an Intel i7-4770 3.40GHz CPU, 8GB DDR3 1600MHz RAM, 128GB SSD and 1TB 7200RPM HDD connecting to 1Gbps network switch. They run Ubuntu 14.04.2 with Linux kernel 3.13, and the installed version of SciDB is 14.12.0.8739, where a vanilla SciDB and our modified *SLS* concurrently reside. We used MPICH2 3.0.4 as an implementation of MPI. Each node has a SciDB instance.

Data used for experiments were MODIS remote-sensing reflectance (in short, RRS). As a collection of the earth images of the satellite SeaStar, the data have three dimensions



(a) Loading and redimensioning time (b) Breaking down the time on redimensioning

Fig. 3: Comparison between vanilla and *SLS*, using RRS

(*longitude, latitude, time*) and values measured at each unit (*chlorophyll*). RRS has 4km $\times$ 4km resolution with a range 8,640 $\times$ 4,320. This data are stored into the HDF5 format. The number of files in total is 545, and the number of cells is 20,342,016,000. For a one-dimensional array by a loading stage, we set the chunk size as a million rows.

B. Evaluation

We evaluate *SLS* compared with vanilla SciDB loader by loading and redimensioning time. Especially, in a redimensioning stage, we break down steps to analyze more details. In addition, as increasing data size, we measured the processing time of loading and redimensioning. After that, we evaluated scalability while increasing the number of nodes.

1) Loading and Redimensioning: We measured the loading and redimensioning performance of *SLS* regarding processing time. Figure 3 shows the loading and redimensioning time for the RRS data. The loading time is reduced about 65%.

There are two reasons. First, *SLS* reduces the processing time because we remove redundant I/O overhead to make temporary files. Specifically, in vanilla SciDB, since the master node makes temporary files (*i.e.*, TXT) alone, slaves should wait for creating the file to finish, which leads to delay. However, *SLS* bypasses the temporary files and sends files for loading on the fly from the master node to slave nodes (distributing workload).

Second, we use MPI instead of SSH. MPI has higher speed than SSH and can make the master node send data directly to slave nodes, which are also related to minimizing I/O overhead. As files are distributed while data conversion is performed, the I/O is not incurred to make intermediate files.

In the redimensioning stage, the data are sorted before loading and are not necessary to be sorted. Likewise, since the master node already distributes data to the instances which the data eventually should arrive, the redistribution process is removed. The third step does not almost take place because preparation work for redistribution process is not required. As a result, the loading and redimensioning times

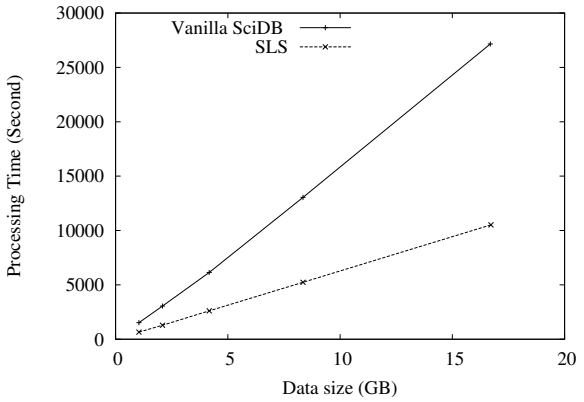


Fig. 4: Trend of time with respect to the data size.

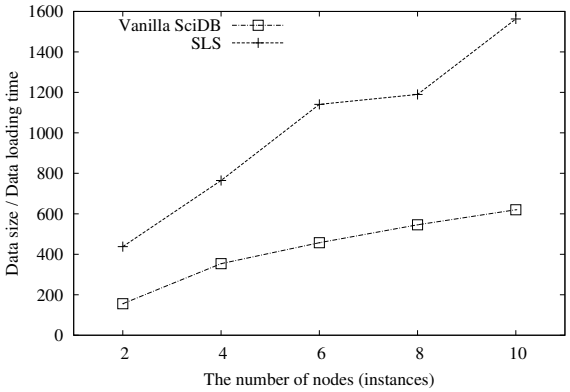


Fig. 5: Comparing with vanilla-SciDB and *SLS* for scalability

are reduced compared to vanilla SciDB by optimizing and removing redundant steps.

2) Data Size: Since the scientific data is huge and SciDB is for handling massive data, we evaluate the performance, varying the data size. We changed the data size to observe the processing time trend, and it can show the benefit of *SLS*.

Figure 4 shows the processing time for whole processes with different data size. Both of two lines increase linearly, but they have various aspects of the performance. Compared to vanilla SciDB, *SLS* achieves about two-thirds of the processing time because we subtract and simplify the processes and thus essential steps remain. Based on this, we conclude that when processing bigger data, *SLS* would gain better performance for the bigger data size.

3) Scalability: As SciDB has shared-nothing architecture, scalability is a significant concern for the system. As minimizing the number of shared data or interferences, systems can handle an amount of data simultaneously. That is, if each node can run without synchronization, the system could be scalable and fully use its resources. Unless there is a dependency with each node, adding extra nodes improves the performance. In order for evaluating scalability, we increased the number of nodes and measured the elapsed time of data loading. We used CHL data, and the nodes increased as 2, 4, 6, 8 and 10.

Figure 5 shows the scalability of *SLS* and the vanilla SciDB. The x-axis is the number of nodes (instances), and the y-axis is the amount of data size divided by elapsed time (i.e., data processing rate). Compared with the vanilla SciDB, *SLS* is scalable because of removing all-to-all exchange in redimensioning and simplifying processes. It reduces network overhead and barriers to wait for communications with all nodes.

However, when the number of nodes is eight, the graph does not grow linearly. Because the system should synchronize all nodes, synchronization consumes significant time and hinders the performance improvement even though distributing workload reduces the whole processing time.

The gradient means the amount of processed data per second dealt with by each node. As the average of *SLS* gradient is higher than that of the vanilla SciDB, the data processing rate of each node increases. Consequently, *SLS* is scalable better than the vanilla SciDB.

V. Conclusion

We propose a new scalable approach to improving the data loading process for SciDB. We use MPI to send data from the master node to slaves in the loading stage such that the I/O overhead can be reduced significantly compared with SSH used by the vanilla loader of SciDB. Moreover, sorting is done separately from the redimensioning stage so that the all-to-all communication for data redistribution can be removed altogether. That is, two major bottlenecks namely, sort and redistribution, in the redimensioning stage are eliminated. The results from our experiments also show that the reduction in loading time by *SLS* is indeed significant.

References

[1] T. N. C. for Supercomputing Applications, "Hierarchical Data Format Version5," https://www.hdfgroup.org/HDF5/, Feb 2017.

[2] R. Rew and G. Davis, "NetCDF: an Interface for Scientific Data Access," *Computer Graphics and Applications, IEEE*, vol. 10, no. 4, pp. 76–82, 1990.

[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, 2010.

[4] A. P. Marathe and K. Salem, "A Language for Manipulating Arrays," in *Proceedings of the 23rd VLDB Conference*, Athens, Greece, Sep. 1997, pp. 46–55.

[5] N. Widmann and P. Baumann, "Efficient Execution of Operations in a DBMS for Multidimensional Arrays," in *the 10th International Conference on Scientific and Statistical Database Management*, Capri, Italy, Jul. 1998, pp. 155–165.

[6] M. Kersten, Y. Zhang, M. Ivanova, and N. Nes, "SciQL, a Query Language for Science Applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. Uppsala, Sweden: ACM, 2011, pp. 1–12.

[7] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann, "The Multidimensional Database System RasDaMan," in *ACM SIGMOD Record*, vol. 27, ACM. New York, NY, USA: ACM, 1998, pp. 575–577.

[8] P. G. Brown, "Overview of SciDB: Large Scale Array Storage, Processing and Analysis," in *Proceedings of the 2010 ACM SIGMOD Conference*. Indianapolis, indiana, USA: ACM, 2010, pp. 963–968.