Evaluating the Impact of Data Placement to Spark and SciDB with an Earth Science Use Case

Khoa Doan
University of Maryland, College Park, MD, USA
khoadoan@umd.edu

Amidu Oloso, Kwo-Sen Kuo,
Thomas Clune
NASA GSFC, Greenbelt, MD, USA,
SSAI, Greenbelt MD, USA
{amidu.o.oloso, kwo-sen.kuo,
thomas.l.clune}@nasa.gov

Hongfeng Yu
University of Nebraska, Lincoln, NE, USA
yu@cse.unl.edu

Brian Nelson
NOAA National Climatic Data Center, Asheville, NC, USA
brian.nelson@noaa.gov

Jian Zhang
NOAA National Severe Storms Laboratory,
Norman, OK, USA
jian.zhang@noaa.gov

***Abstract*— We investigate the impact of data placement for two Big Data technologies, Spark and SciDB, with a use case from Earth Science where data arrays are multidimensional. Simultaneously, this investigation provides an opportunity to evaluate the performance of the technologies involved. Two datastores, HDFS and Cassandra, are used with Spark for our comparison. It is found that Spark with Cassandra performs better than with HDFS, but SciDB performs better yet than Spark with either datastore. The investigation also underscores the value of having data aligned for the most common analysis scenarios in advance on a shared nothing architecture. Otherwise, repartitioning needs to be carried out on the fly, degrading overall performance.**

Keywords- SciDB; Spark; multimensional arrays; SciDB; data layout

I. Introduction

Several Big Data technologies based on shared-nothing architecture (SNA) [12] offer cautious hope to scientists facing the daunting challenge of analyzing datasets of unprecedented volumes in the Big Data era. Among SNA's important advantages are: 1) it provides a simpler distributed programming model better suited for data parallelism than that of traditional network programming such as MPI; 2) it can take advantage of relatively inexpensive commodity hardware; and 3) it supports redundancy and resiliency resulting in better system availability (long system uptime).

A very common category of analysis in Earth Science is comparing values of the same geophysical variable obtained by different means. These comparisons may involve observations from different instruments or observations and model results that require binary operations such as join. While parallel database systems, e.g. Vertica or Oracle, are especially adept at such operations and facilitate various data analysis tasks, developing analytic capabilities in these systems is generally too arduous for Earth scientists. More recent frameworks, however, provide simple, yet powerful, high-level abstractions and tools that make it possible for various types of users to work with data efficiently without detailed knowledge of the underlying implementation.

SciDB is one such recent technology development that specifically targets multidimensional arrays, providing an attractive alternative to general purpose analytic platforms such as Hadoop/MapReduce [1] or Spark [17], for scientific data analysis. As a next-generation parallel database system based on the array data model, SciDB not only indexes the data it ingests for fast retrieval, but also provides an attractive, mathematical/statistical toolbox for data analysis. Similar to Spark and Hadoop/MapReduce, SciDB also exploits the affinity of compute and data, with arguably better effectiveness.

In this study, we compare two technologies that are designed with different analytic purposes in mind, Spark and SciDB, in the aspects of 1) performance 2) flexibility, and 3) impact of data placement using a typical use case in Earth science. In Spark, we also explore an alternative datastore to HDFS, i.e. Cassandra. We first highlight the needs and requirements for high-level distributed computing systems in Section II. We describe our use case scenarios next in section III, and then present our evaluation in Section IV. We conclude the paper with a discussion and our plan for future works.

II. Data Intensive Applications In Earth Science

A. *Shared Nothing Computing Frameworks*

Since the publication of MapReduce (MR) [1], data scientists and technologists have tried to adapt and extend it to many data analysis applications in various domains. Hadoop (HD), the open-source version of MapReduce, has thus become the default choice for almost every Big-Data analysis application. But, its sub-optimal performance has been noted in a number of scenarios [3, 4]. Recent technological developments, such as SciDB and Spark, are providing attractive alternatives to Hadoop/MapReduce (HD/MR) for scientific data analysis.

While HD/MR is simple and arguably laudable for one-pass computations, it is inherently inefficient for multi-pass algorithms. The reason for this is that HD/MR lacks appropriate primitives for sharing intermediate states of the calculation between passes and instead sends/retrieves intermediate states to/from a distributed file system. The overhead from communication and I/O of this approach often dominates overall performance. For this reason, HD/MR is poor for complex applications and algorithms that are typically composed of simpler calculations, which in-and-of-themselves are well-suited to HD/MR. SciDB, on the other hand, not only indexes the data it ingests for fast extraction and retrieval, but also provides an attractive, mathematical/statistical toolbox for data analysis. Like HD/MR, SciDB exploits the affinity of compute and data.

One of the primary disadvantages of Spark, compared to a full DBMS solution such as SciDB, is its loose coupling between the datastore and the execution framework. While Spark provides several primitives for efficient data manipulation such as sharing and partitioning, we can leverage these primitives only after data are loaded into Spark's execution engine. Therefore, Spark cannot directly exploit regularities in structured data. Instead, it must effectively "rediscover" such structure every time data are accessed, leading to a certain degree of unavoidable overhead. Fortunately, there are datastore systems, such as Cassandra, that allow better integration with Spark into a semi-DBMS solution.

Cassandra is an in-memory, "distributed storage system for managing large amount of structured data spread out across commodity servers, while providing highly available service with no single point of failure" [20]. Cassandra implements column datastore that ensures data locality for its partitions. A partition in Cassandra, however, may be split across multiple files locally and data locality within a partition is not guaranteed. That is, it may need to read multiple local files in order to arrive at the specific rows or columns of interest. This has pertinence in analytic processing because, unlike SciDB (also using column store), operations requiring the full list of attributes or columns (such as `projection` in SciDB) may not be optimal.

B. *Multidimensional Arrays*

Scientists typically work with multidimensional arrays. An array can be thought of as a grid of cells, such as that of a numerical weather prediction model, which is often multidimensional and each array cell often contains multiple attributes, e.g. pressure, temperature, humidity, etc. A cell of this more abstract array, hitherto referred to simply as "array", is *filled*, if all of its attributes are present and valid, or *empty* if all of its attributes are absent or invalid, otherwise it is *partially filled*.

There have been efforts to build array data processing on top of table-based RDBMS, but the process of translating array-specific operations to RDBMS' primitives is nontrivial, and such efforts generally fail to exploit effective multidimensional partitioning and indexing [9]. In contrast, array-oriented systems such as SciDB directly manipulates multidimensional arrays and supports many array operations out-of-the-box with efficiency.

Finding an optimal strategy for indexing arrays has been a major focus of existing works on multidimensional array data. The standard approach is to partition an array into "chunks", each of which typically resides on a node in a cluster and can be considered as a unit of work for parallel processing [9]. Selection of partitioning, or chunking, heuristics is important as they affect the efficiency of the storage system.

There are two basic chunking approaches: regular and irregular [9]. Regular chunking (REG) partitions the array cells into uniform chunks regardless whether they are filled or empty, whereas irregular chunking (IRR) may partition them into different sizes, where each chunk often holds roughly the same number of filled array cells to even the workload of an operation over nodes and achieve better overall performance. One advantage of REG is its amortization of seek times and any fixed-costs associated with processing a chunk, but IRR can avoid straddling over skewed data, which occurs often with sparse arrays [9].

More complex chunking approaches can be derived from these basic ones, such as REG-REG, where each chunk of the array is subdivided into smaller regular chunks that can be used to efficiently determine the relevant set of data for an operation, or IRR-REG where each chunk of roughly the same data volume of the array is partitioned into smaller, equally spaced chunks to take advantages of the regular chunking scheme. Sparse arrays, often unevenly distributed in their coordinate space and thus poorly skewed, are particularly hard to partition for efficient parallel processing.

Existing works have shown that the majority of Earth Science data, where the natural dimensions are time, latitude, and longitude, are sparse arrays in the spatiotemporal coordinate space. Moreover, data often exhibits irregular spatial pattern over time, further complicating the situation. Although it is possible to create a denser array by using only the temporal dimension for indexing, this causes inefficiency in other array operations, such as aggregation operations over spatial areas (aggregate value grouped by spatial subsets). Upon consideration, we choose the REG-REG chunking scheme for Spark in this study, because our arrays are dense and it has been reported to be optimal for various types of array processing workload [8]. For fairness sake, a similar regular chunking scheme is also used in SciDB.

C. *Importance of Data Layout*

Queries on multidimensional arrays typically involve scan, dicing, join and overlap operations [9]. Array scans, such as `filter`, processes all chunks of an array. Array dicing, such as `subsample`, may involve a subset of an array. While array scans and subsamples are trivial to parallelize because they operate on each chunk independently, it is more efficient if only relevant chunks are processed. Binary operations, such as `join` and `merge`, can get more complex depending on the `join` predicate [9,15]. But what is critical to efficient binary operators is that the same *logical* chunks from both input arrays are co-located on the same instances, which avoids computationally expensive repartitioning to align the corresponding pairs of chunks. Overlaps, such as clustering or connected component labeling, is difficult to implement efficiently and its implementation details often depends on the problem at hand [16].

The queries in this paper focus on scan and binary operations. Our goal is to evaluate the performance characteristics the selected technological approaches. As mentioned previously, performance of binary operations can vary, depending on how corresponding arrays are initially partitioned. Evaluations are carried out for both scenarios: When the arrays are aligned and mis-aligned for a `join` operation. Alignment of arrays requires not only they have the same *shape*, i.e. same dimensionality and same range in each dimension, but also the same partition configuration. In SciDB, the same array *schema* ensures both and guarantees that corresponding chunks physically reside on the same node, or aligned.

D. *Multidimensional arrays in Spark/HDFS and Spark/Cassandra*

Since Hadoop distributed file system (HDFS) automatically determines the physical placements of the HDFS blocks of a file, in order to have a fairer evaluation of these two dissimilar analytic systems, we first generate a sequence file for the pair of aligned arrays, a step that resembles the re-dimensioning and chunking operation in SciDB. A "chunk" so-emulated is represented as a key-value pair, where its key is the coordinate space of the chunk and its value holds the data inside each chunk of the compatible arrays. This endows Spark+HDFS with the similar advantage of chunk co-location as that in SciDB.

With a column datastore such as Cassandra, however, co-location of data records with similar keys is possible. Because Cassandra consistently hashes a user-defined partitioning key, if we use the coordinate space of chunks as partitioning key, aligned arrays can have their chunks distributed in similar ways in the cluster. We implement each array in Cassandra as a table, whose dimension ranges of a chunk are partition keys, and each row encapsulate data for a chunk, whose attribute is a column with its value being a binary representation of data for this attribute within this chunk.

III. USE CASE DESCRIPTION

In order to study the performance characteristics between SciDB and Spark, we use a concrete set of typical queries in Earth Science domain. The queries operate on 2 multidimensional datasets described below.

A. *Datasets*

Two regularly gridded datasets for the period of Winter 2010, i.e. from 1 December 2009 to 28 February 2010, are used to conduct our experiments. The first one is extracted from hourly datasets of the NASA Modern Era Retrospective-analysis for Research and Applications (MERRA) [22] data collection and the second from National Mosaic and Multi-sensor QPE (NMQ, where QPE stands for quantitative precipitation estimate) [23].

MERRA has global coverage, whereas NMQ is only available for the contiguous United States (CONUS), specifically 20°N-55°N in latitude and 130°W–60°W in longitude. They are also of different resolutions. For MERRA it is ⅔°×½° (longitude×latitude) in space and hourly in time, whereas it is 0.01°×0.01° in space and every 5 minutes in time for NMQ. Therefore, homogenization of these datasets is the necessary first step of our data placement experiment.

In the preparation of the MERRA array, we first resample the original MERRA array in the longitude direction to ½° equivalent and then replicate the resampled array by a factor of 5×5 for each ½°×½° grid cell in space, from which a CONUS subset is extracted. The CONUS subset is subsequently replicated 12× in time. The resultant MERRA array effectively mimics 0.1°×0.1° resolution in space and 5-min in time. Accordingly, we perform a 10×10 average in space for the NMQ array, bringing its resolution down to 0.1°×0.1° in space and 5-min in time as well. After the homogenization, both arrays have array dimensions of 700×350 in space and 25,920 (=12×24×90) in time, i.e. the same shape.

B. *Regridding queries*

One of the typical queries that Earth scientists perform on spatiotemporal data is averaging over predefined intervals in each dimension. Similar queries are used in this case to homogenize the two arrays to the same shape. In SciDB, a query can be written in Array Query Language (AQL) and/or Array Functional Language (AFL) [21]. We use AFL in our SciDB queries. (AFL code for the queries can be obtained from us upon request.)

Q1. *Resample MERRA array to 0.1°, in both latitude and longitude, and 5-minute resolution.*

Q2. *Average NMQ array also to 0.1° and 5-minute resolution.*

Platform	Rpl.	Q1	Q2	Q3		Q4	Q5	
				cross-join	join		cross-join	join
SciDB	**None**	6.40	**100.91**	**33.17**	13.47	9.40	**28.87**	5.84
SciDB	**2×**	6.72	113.47	49.34	**11.70**	**8.39**	40.85	**5.41**
Spark+HDFS	**2×**	5.76	193.67	42.43	—	11.49	32.89	—
Spark+Cassandra	**2×**	**3.80**	157.50	38.15	—	10.53	29.36	—

Table 1. Average running time of queries for 5-day data on 3-node cluster (in minutes).

Q1 selects the *total precipitation* attribute (PRECTOT) first. The outer xgrid-regrid-xgrid operations transform MERRA into the 0.1° and 5-minute resolution. Similarly, Q2 changes the original NMQ array into the same resolution as that of MERRA.

C. *Comparison queries*

Another common type of operations in earth science is comparing the values of the same geophysical quantity (in this case, precipitation) obtained with different means.

Q3. *Repartition resampled MERRA array on the fly to compare precipitation rates with coarsened NMQ array.*
Q4. *Repartition to align resampled MERRA with coarsened NMQ array.*
Q5. *Compare precipitation rates using aligned arrays.*

The purpose of Q3 is to find the difference in precipitation rates between corresponding cells of the 2 arrays. Since the arrays' placements are not aligned, repartitioning of one array (in this case the MERRA array) to align with the corresponding chunks of the other is necessary, which is done via `repart` in SciDB. Since SciDB partitions intermediate arrays of queries and offline arrays in similar manner, only 1 array needs to be repartitioned. In Spark, however, this is not possible unless the coarsened NMQ array has been previously repartitioned by Spark itself.

Q4 and Q5 attempt to study the effect of placement alignment by aligning the arrays in advance. Q4 saves the results of the alignment operation of Q3 and Q5 performs the join operation in order to calculate the difference in precipitation rates between each pair of corresponding cells.

Since Spark does not have a similar high-level query language as SciDB, we have to implement our own. In implementing these queries we strive to use operators that are as similar as possible to the data flow pattern of SciDB. Spark code for the queries are also available upon request.

We describe our computing environment and report evaluation results of the evaluation experiments in the next section.

IV. Performance Evaluation Experiments And Results

A. *Computing Environment*

Two virtual clusters, one with 3 nodes and the other 28 nodes, are set up to carry out our experiments. Each node resides on a separate, unique physical container. All the nodes have the same features: 32 GB of main memory (or random access memory, RAM), an 8-core CPU, and ~16 TB of local spinning disk storage and they all run Centos 6.5 Linux operating system. The nodes are interconnected with Infiniband (Mellanox MT27500 FDR IB).

We use the enterprise edition of SciDB release 15.7, which supports replication and advanced linear algebra operations, such as singular value decomposition (SVD), that are not available in the community edition. As to Spark, version 1.6 of the Cloudera CDH5.4.2 distribution is used. To ensure fairness of the comparison, all the technological approaches are configured on and use the same clusters.

The HDFS configuration employs a replication factor of 2 and 3, respectively, on the 3-node and 28-node clusters. The same replication factors are used for SciDB as well on respective clusters. However, performance numbers are also obtained for SciDB with no replication (equivalent to replication factor of 1) because the community version of SciDB does not offer replication. For the Spark+Cassandra integration, we use Datastax Cassandra version 3.07.1159 with MurmurHashPartitioning strategy. We also configure Cassandra to have the same replication factors as those used with HDFS on the respective clusters. Finally, our Spark's implementation of the array models is written in Java.

For all technological approaches, we configure as much as possible to have the same resources for utilization. For example, each Spark job is executed with 2 8GB, 4-core executors. Similarly, there are 2 SciDB instances running on each node. Each query is repeated three (3) times for each technological approach and on each cluster. The averaged performance of the 3 runs are reported below. The timing variation of the 3 runs is always around only few percent.

B. *Evaluation*

In this section, we present the results of our experiment for different configurations of our clusters on the queries

Platform	Rpl.	Q1	Q2	Q3		Q4	Q5	
				cross-join	join		cross-join	join
SciDB	**None**	**23.99**	245.44	148.43	75.65	**77.28**	90.18	**14.85**
SciDB	**3×**	24.48	**241.69**	**146.64**	**72.67**	78.79	**90.01**	14.96
Spark+HDFS	**3×**	30.26	297.90	257.76	—	94.72	159.13	—
park+Cassandra	**3×**	24.50	255.07	202.24	—	84.14	156.36	—

Table 2. Average running time of queries for 3-month data on 28-node cluster (in minutes).

mentioned above. Since the data volume of the entire 3-month is too much for the 3-node cluster, a 5-day subset is used instead. The 28-node cluster, however, operates on the entire 3-month (or 90-day) datasets.

There are two operators available for joining operations in SciDB: `cross-join` and `join`. The former, i.e. `cross-join`, is a generic join operator that makes no assumptions about the schemas of the array operands, while the latter, i.e. `join`, requires the array operands to be aligned (i.e. corresponding chunks co-located on the same nodes). The performances of both are reported for SciDB. Spark, however, does not have similar options.

Table 1 tabulates the average running times (in minutes) of the queries on the 3-node cluster. Best performing number in each query is boldfaced in the table. Spark+HDFS is consistently the worst performer. This aligns well with our earlier discussion regarding Spark+HDFS that it does not leverage resident memory as SciDB or Spark+Cassandra does. Spark+Cassandra performs the best only in Q1. SciDB is consistently the best performer. For Q2, SciDB (both with or without replication) is ~50% faster than the next best, Spark+Cassandra. However, it is curious that SciDB with 2× replication is significantly slower than without replication on `cross-join` for both Q3 and Q5. We currently have no good explanation to this. The most striking result is perhaps the efficiency of SciDB join operator, which offers 3 to 5 times the speed than that of the otherwise best performer, i.e. SciDB `cross-join`, in Q3 and Q5.

The performance advantage of SciDB is even more apparent on the 28-node cluster, for which Table 2 tabulates the timing results. Unsurprisingly, Spark+HDFS is still the worst performer. Both SciDB configurations, i.e. with or without replication, beats Spark+Cassandra in all 5 queries, often with wide margins. However, while the performance advantage margin of SciDB over Spark-based approaches has narrowed for Q2, it has grown for Q3 and Q5 on `cross-join`. More interestingly, we do not see the large disparity between the two SciDB configurations on `cross-join` in Q3 and Q5 anymore. SciDB `join` still offers the best performance and considerable advantage over `cross-join`.

To compare how the technological approaches scale from 3 nodes to 28 nodes, roughly an order of magnitude, we compute the following ratio as a measure for scalability:

$$(t_3/5/3)/(t_{28}/90/28),$$

where t_3 and t_{28} are performance times of the 3-node and 28-node clusters, respectively. Basically, it is the ratio of the 3-node performance time to 28-node performance time, after both have been normalized by the data volume (number of days) and number of nodes. Larger ratios thus correspond to better scalability. The results of this comparison are tabulated in Table 3, with the best performer of each query boldfaced as before. It is apparent that SciDB is still the best overall performer in scalability. SciDB without replication scales better for `join,` whereas SciDB with replication scales better for `cross-join`. Spark+HDFS breaks its losing streak by having the highest ratio in Q2. However, due to limited computing resources available to us, we are unable to perform scalability comparisons to larger clusters up to, more realistically, thousands or even tens of thousands of computing nodes. The results presented here therefore should not be considered conclusive.

V. Discussion And Future works

Our experiments have demonstrated the potential of an array-based DBMS system, SciDB, particularly for the Earth Science domain. In most of the experiments, SciDB yields better performance than general analytic systems such as Spark, and provides more convenient, mature analytic toolbox for working with multidimensional arrays. It is worth noting here that SciDB aims at end users who are interested in using high-level queries rather than in developing complex analytic algorithms. SciDB also appears to scale better. In addition, we have demonstrated that array data processing can be built on top of general purpose analytic systems such as Spark, with respectable efficiency.

Based on the evidence presented, we conclude that best performance is achieved when data placements are aligned for the query because it eliminates the need for computational expensive repartition operation. For a Big Data analysis system, however, it is impossible to have data placements aligned for all possible queries. Therefore, to reap the greatest benefit and attain the greatest value, the Big Data analysis system should align data placements for the most common and frequent queries.

In Earth Science, analysis scenarios predominantly require spatiotemporal coincidence. For example, when investigating precipitation events or systems, we almost always need the environment conditions for the same location and same time (i.e. spatiotemporal coincidence) of the events or systems. Unfortunately, this simple concept of data placement alignment presents a formidable challenge in

Platform	Rpl.	Q1	Q2	Q3		Q4	Q5	
				cross-join	join		cross-join	join
SciDB	**None**	0.51	0.79	0.43	**0.34**	0.23	0.62	**0.76**
SciDB	**3X/2X**	**0.53**	0.91	**0.65**	0.31	0.21	**0.88**	0.70
Spark+HDFS	**3X/2X**	0.37	**1.25**	0.32	—	0.23	0.40	—
Spark+Cassandra	**3X/2X**	0.30	1.19	0.36	—	**0.24**	0.36	—

Table 3. Approximate throughput comparison.

practice and in implementation, because a diverse set of data models exists for representing Earth Science data. One of our research goals is thus to find and identify an effective indexing scheme that 1) can be applied to, preferably, all data models, 2) supports data placement alignment, and 2) exerts little or no negative impact on data analysis performance, all at the same time. Our research indicates that Hierarchical Triangular Mesh (HTM) [24] is a promising approach that meets the requirements. The evaluation of HTM performance in various Earth Science data analysis scenarios is thus an important component of our future efforts.

Acknowledgement

This work was primarily funded by the NASA Earth Science Technology Office (ESTO) through its Advanced Information Systems Technology (AIST) Program. It is also partially supported by the National Science Foundation's (NSF) EarthCube program.

References

[1] Dean, J., Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. clusters. In Proceedings of Operating Systems Design and Implementation (OSDI). San Francisco, CA. 137-150.

[2] Apache Hadoop. (2014, September 12). In Wikipedia, The Free Encyclopedia. Retrieved 03:21, August 23, 2014, from http://en.wikipedia.org/w/index.php?title=Apache_Hadoop&oldid=625239666

[3] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., & Stonebraker, M. (2009, June). A comparison of approaches to large-scale data analysis. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (pp. 165-178). ACM.

[4] Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., Rasin, A. (2010 January). MapReduce and Parallel DBMSs: Friends or foes? Comm. of the ACM, 53(1), 64-71.

[5] Stonebraker, M., Brown, P., Zhang, D., and Becla, J., (2013 May-June). SciDB: A Database Management System for Applications with Complex Analytics. Computing in Science & Engineering, 15(3), 54-62.

[6] Stephens, G. L., Vane, D. G., Boain, R. J., Mace, G. G., Sassen, K., Wang, Z., Illingworth, A. J., O'Connor, E. J. Rossow, W. B., Durden, S. L., Miller, S. D., Austin, R. T., Benedetti, A., Mitrescu, C., and the CloudSat Science Team (2002 December). The CloudSat mission and the A-Train: A new dimension of space-based observations of clouds and precipitation. Bull, Amer. Meteor. Soc., 83(12), 1771-1790

[7] Simpson, J., Adler, R.F., North, G.R. (1988, March). A proposed Tropical Rainfall Measuring Mission (TRMM) satellite. Bull, Amer. Meteor. Soc., 69(3), 278-295.

[8] Kummerow, C., Barnes, W., Kozu, T., Shiue, J., Simpson, J. (1998 June). The Tropical Rainfall Measuring Mission (TRMM) sensor package. J. Atmos. Oceanic Technol. 15, 809-817.

[9] Soroush, E., Balazinska, M., & Wang, D. (2011, June). Arraystore: a storage manager for complex parallel array processing. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 253-264). ACM.

[10] FLANN - Fast Library for Approximate Nearest Neighbors, http://www.cs.ubc.ca/research/flann/

[11] kd-tree. (n.d.). - RoboWiki. Retrieved May 12, 2014, from http://robowiki.net/wiki/Kd-tree.

[12] Stonebraker, M. (1986). The case for shared nothing. IEEE Database Eng. Bull., 9(1), 4-9.

[13] Stonebraker, M., Brown, P., Poliakov, A., & Raman, S. (2011, January). The architecture of SciDB. In Scientific and Statistical Database Management (pp. 1-16). Springer Berlin Heidelberg.

[14] Paradigm4, Inc. | Big Analytics. (n.d.). Paradigm4 Inc. Retrieved July 12, 2016, from http://www.paradigm4.com

[15] K. Doan, A. Oloso, K.-S. Kuo, T. Clune, (2014), Performance Comparison of Big-Data Technologies in Locating Intersections in Satellite Ground Tracks, 2014 ASE BigData/SocialInformatics/PASSAT/BioMedCom Conference, December 14-16, 2014, Harvard University, Cambridge, MA, USA.

[16] A. Oloso, K.-S. Kuo, T. Clune, P. Brown, and A. Poliakov, "Implementing Connected Component Labeling as a User Defined Operator for SciDB," 9th Extremely Large Databases Conference (XLDB 2016), 24-26 May 2016, Menlo Park CA http://www-conf.slac.stanford.edu/xldb2016/

[17] K.-S. Kuo, A. Oloso, Doan, K., T. Clune, Yu, H. Implications Of Data Placement Strategy To Big Data Technologies Based On Shared-Nothing Architecture For Geosciences. IGARSS 2016.

[18] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: cluster computing with working sets. HotCloud, 10, 10-10.

[19] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012, April). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 2-2). USENIX Association.

[20] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44(2), 35-40.

[21] SciDB Documentation. Retrieved July 12, 2016 from http://www.paradigm4.com/resources/documentation/

[22] Bosilovich, M. G., R. Lucchesi, and M. Suarez. "MERRA-2: File specification." (2015).

[23] Zhang, Jian, et al. "National Mosaic and Multi-Sensor QPE (NMQ) system: Description, results, and future plans." *Bulletin of the American Meteorological Society* 92.10 (2011): 1321.

[24] Rilee, M., Kwo-sen, Kuo., Clune, T., Oloso, A. Addressing the Big-Earth-Data Variety Challenge with the Hierarchical Triangular Mesh. IEEE BigData (2016).