Performance Comparison of Big-Data Technologies in Locating Intersections in Satellite Ground Tracks

Khoa Doan[1,2], Amidu Oloso[2,3], Kwo-Sen Kuo[2,4], Thomas L Clune[2]
University of Maryland, Department of Computer Science[1]
NASA Goddard Space Flight Center[2]
Science Systems and Applications, Inc[3]
Bayesics, LLC[4]
Email: khoadoan@cs.umd.edu, {amidu.o.oloso, kwo-sen.kuo, Thomas.L.Clune}@nasa.gov

Abstract

The performance and ease of extensibility for two Big-Data technologies, SciDB and Hadoop/MapReduce (HD/MR), are evaluated on identical hardware for an Earth science use case of locating intersections between two NASA remote sensing satellites' ground tracks. SciDB is found to be 1.5 to 2.5 times faster than HD/MR. The performance of HD/MR approaches that of SciDB as the data size or the cluster size increases. Performance in both SciDB and HD/MR is largely insensitive to the chunk size (i.e., granularity). We have found that it is easier to extend HD/MR than SciDB at this time.

Keywords: Multidimensional arrays; MapReduce; intersection algorithm; SciDB.

1. Introduction

Several emerging Big-Data technologies offer cautious hope to scientists facing the daunting challenge of analyzing datasets of unprecedented volumes in the era of Big Data. While none of these technologies is yet mature enough for routine operational use in scientific research, several are sufficiently robust to warrant further investigation into their potential role in a typical research environment.

Although most scientists now have access to powerful computational resources ranging from multi-core laptops to petascale clusters, their personal data analysis workflows seldom exploit the full capabilities of these resources. Performance hence remains largely constrained by serial processing, because exploiting parallelism generally requires additional software engineering skills and resources that typical researchers rarely possess. Further, because the workflows are often unique to each scientific investigation, generic support for parallelism is limited to only a handful of very common analysis patterns.

One of the common processes in a scientific workflow is "subsetting", i.e. the extraction of subsets of research interest from vast volumes of relevant datasets. Parallel database systems are especially adept in such process. While these systems, such as Vertica or Oracle, also facilitate various data analysis tasks, developing analytic capabilities in these systems is often too arduous for many scientists. More recent frameworks provide simple, yet powerful, high-level abstractions and tools that makes it possible for different types of users to work with data efficiently without detailed knowledge of the underlying implementation.

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) [2], 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 [5], which specifically target multidimensional arrays, are providing an attractive alternative to Hadoop/MapReduce (HD/MR) for scientific data analysis. SciDB, a next-generation array-model parallel database system, not only indexes the data it ingests for fast extraction and retrieval, but also provides an attractive, albeit still basic, mathematical/statistical toolbox for data analysis. Like HD/MR, SciDB exploits the affinity of compute and data.

We compare two technologies in this paper, Hadoop and SciDB, in the aspects of 1) performance and 2) ease of implementations, using a common use case in Earth science remote sensing. We first describe our use case scenario in section 2. We elaborate in Section 3 a few key considerations regarding processing ground track arrays, then describe the array data used in Section 4. The Big-Data algorithms used for our evaluation are introduced in Section 5. In Section 6, we describe our hardware platform, detail our experiments, and report results. We conclude the paper with a discussion and our plan for future works.

2. Use Case Description

The problems we are facing today with our Earth's future are complex and carry grave consequences. We need long-term and comprehensive observations of Earth's conditions to understand this complex system of systems. However, approximately two-thirds of Earth are oceans where direct and dense measurements are difficult to obtain. Remote sensing hence becomes the more cost-effective means for obtaining the measurements required to monitor Earth's current health and to provide data for the prediction of its future.

Remote sensing problems, however, are usually under-constrained. That is, its problem space is often of a higher dimensionality than that covered by the observations of the instruments. To gain better constraints and to reduce ambiguity, scientists strive to obtain as much simultaneous, co-located and independent information as possible concerning the problem space. Our use case is thus to find nearly coincident spaceborne radar measurements of two NASA Earth science

 ISBN: 978-1-62561-003-4

missions: CloudSat [6] and Tropical Rainfall Measuring Mission, or TRMM [7, 8].

Both satellite platforms, CloudSat and TRMM, carry radars capable of characterizing precipitation vertical profiles, but at different microwave frequencies. These radars' high-resolution vertical profiling capabilities are in general superior to other instruments for understanding the vertical structure of precipitation systems. CloudSat's W-band (94 GHz) Cloud Profiling Radar, CPR, is sensitive to smaller precipitation particles or lighter precipitation than TRMM's Ku-band (13.8 GHz) Precipitation Radar, PR, but gets attenuated and rendered less useful than TRMM's PR in heavier precipitation. Combining near-coincident and complementary measurements from both CloudSat CPR and TRMM PR, therefore, has the potential to provide a more complete three-dimensional picture of precipitation events.

The complication, however, is that CloudSat and TRMM do not fly in formation with similar orbit characteristics. CloudSat is in a sun-synchronous orbit with local overpasses roughly constant in solar time, while TRMM, is in an orbit with approximately 35-degree inclination, designed to attain better temporal sampling. As a result, they do not routinely observe the same location of Earth at the same time. Thus, to obtain nearly coincident observations from both the 94-GHz CPR and 13.8-GHz PR, we need to first identify intersections of their ground tracks. There are multiple approaches to the solution of this problem, but not all of them are appropriate for the shared-nothing architecture of the Big-Data technologies used in our comparison.

3. Array Considerations

The array concept used in the study for the Big-Data technologies is slightly more abstract than the typical array understood by a science programmer. It is important to understand this distinction in order to follow subsequent discussions.

An array here can be thought of as a grid of cells, such as that of a numerical weather prediction model, which is often multidimensional. Each array cell in this case may contain multiple attributes, e.g. pressure, temperature, humidity, etc., whereas the typical array concept in a programming language like C or Fortran normally contains a single attribute, e.g. $p[i, j, k]$ for pressure at location x_i, y_j, z_k. This latter concept is only equivalent to the former when it has only one attribute in each cell, i.e. pressure. A cell of the 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 it attributes are absent or invalid, otherwise it is *partially filled*. Our discussion below, however, involves only filled or empty array cells.

As mentioned above, most scientists work primarily with multidimensional arrays. 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, SciDB directly manipulates multidimensional arrays and supports many array operations out-of-the-box.

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. Most satellite ground tracks, where the natural dimensions are time, latitude, and longitude, are sparse arrays in the spatiotemporal coordinate space. Moreover, a satellite ground track often exhibits irregular spatial pattern over time, further complicating the situation. Although it is possible to create a denser array for a satellite ground track 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 dimensions).

Upon consideration, we choose the REG-REG chunking scheme for HD/MR in this study, because 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.

4. Ground-track Data

In our study, we use data from CloudSat, which has sun-synchronous orbits and whose mission is to measure the vertical structure of clouds from space [6], and TRMM, whose mission is to understand tropical rainfall and its feedback with the global climate [7, 8]. Our multidimensional arrays are created using CloudSat's 2B-GEOPROF product, which contains cloud mask and reflectivities data, and TRMM's 1C21 product, which contains TRMM PR 3-D reflectivity. Although these missions are not the most data-intensive ones launched by NASA, they still generate considerable amounts of data while orbiting Earth; for example, TRMM 1C21 product holds ~270GB of data per year in compressed format. Our study employs 1-, 2-, or 4-year temporal subsets of both data products. For intersection identification, we search for matches

between ~5×10^5 geolocations per day from CloudSat and ~1.5×10^5 from TRMM.

5. Intersect Algorithm

Our challenge is to identify all locations where the ground tracks of two satellites intersect. An intersection, in our case, is defined as a pair of geolocations (each a 2-tuple of latitude and longitude) corresponding to the two tracks, whose spatial distance is within half of the spatial resolution of either instrument (i.e. CloudSat CPR or TRMM PR), and whose temporal difference is within a predefined time window, as following:

> Given s_1 and s_2 as half of the spatial spacings of the neighboring data points in the 2 respective ground tracks, and Δt as the temporal constraint, the intersection of each orbit of the first ground track, $i(p_1^*, p_2^*)$, where p_1^* and p_2^* are data points of the 2 tracks, is defined as:
>
> $$i(p_1^*, p_2^*) = argmin_{p_1,p_2} dist(p_1, p_2),$$
> $$where\ |t_1 - t_2| \le \Delta t\ and\ dist(p_1, p_2) \le \max(s_1, s_2)$$

Intersections so identified, in turn, can serve as the centers around which adjacent observations of a predefined neighborhood area can be extracted from respective datasets to be used for Earth science remote sensing applications.

For finding intersections between the ground tracks of CloudSat and TRMM, we first transform the latitude-longitude geolocations to [*x*, *y*, *z*] of a Cartesian coordinate system, yielding two ground-track arrays of data points with dimensions [*x*, *y*, *z*, *t*], where *t* is for time. Although each data point can be uniquely identified with *t*, having [*x*, *y*, *z*] as dimensions (instead of as attributes) generally allows more efficient processing of the arrays in systems such as SciDB because dimensions are faster in look-up as a result of their column storage system, and more amenable to partitioning strategies.

A simple algorithm is to partition the coordinate space into a regular mesh of subspaces, i.e. partition the corresponding arrays into subarrays. This configuration enables efficient "join" operation within each subspace and hence subarray. This algorithm, henceforth referred to as the *Baseline algorithm*, essentially consists of two steps. First, identify potential intersections by cross-comparing all possible data points within the corresponding subarrays to locate the pair with closest spatial distance. Because the two ground tracks can only intersect with each other at most two times per orbit, the actual intersection can thus be found by choosing the closest pair among the potential intersections identified in the first step for each half of its orbit. It is possible for the pair of ground track positions of the actual intersection to fall in neighboring subspaces rather than the same subspace. This complication is easily addressed by a slight modification to use an overlapping mesh system.

The Baseline algorithm utilizes all dimensions, i.e. *x*, *y*, *z*, and *t*, of the ground track to allow for easier parallel computation of intersections within each partition of the data. One of its advantages is that it does not require pre-partition or pre-indexing of the original data. However, the cross-matching step takes a considerable amount of computation to examine all possible pairs of points in a subspace.

On a database system like SciDB, indexing is performed at the ingestion time of the data. For SciDB, we partition the arrays according to one of the chunking schemes introduced above. Two arrays with the same schema, i.e. the same coordinate space and chunk size for each of its dimensions, are partitioned into two similar sets of chunks. Corresponding chunk pairs of the two arrays are thus located on the same physical node. In other words, each corresponding pair of chunks on a given node is contained by the same subspace, like in the Baseline algorithm. This partitioning and co-location allow each pair of corresponding chunks to be processed locally, independently, and efficiently, often without necessitating any cross-node communication.

Our SciDB's "bestMatch" operator implements exactly this idea, as in Figure 1. The implementation allows for the ability to constrain the search radius in each dimension, thus permitting more flexible search of the potential intersections. We use k-d tree to find the best match in the second array for each point in the first array.

In Hadoop file system (HDFS), there is no such notion of co-location. The storage system automatically determines the physical placements of the HDFS blocks of a file. Therefore, in order to have a fairer evaluation of these two dissimilar analytic systems, we first generate a sequence file for the pair of ground-track 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 Hadoop with the similar advantage of chunk co-location as that in SciDB.

```
1)  bestMatch step - finding the probable
    pairs of intersection
1.1)  for a pair of corresponding chunks of
      CloudSat and TRMM build a k-d tree for
      TRMM-chunk
1.2)  for each data point in CloudSat
1.2.1)  find the best match in TRMM that
        satisfies the dimension constraints.
        if there is a match
1.2.1.1)  output the pair, re-dimensioned
          by CloudSat's orbit's ID

2)  aggregate step - finding the actual
    pairs of intersection
2.1)  for each CloudSat's orbit
2.1.1)  find the 2 pairs of data points that
        are spatially closest, one on each
        half side of the earth
2.1.2)  this is the center of intersection
        that satisfies the problem's con-
        straints
```

Figure 1. Algorithm for bestMatch.

6. Experiment Description and Results

6.1 Computing Environment

We run both the Hadoop and SciDB experiments on the same computing cluster. We ensure that the cluster is "clean" for each run. The cluster has a total of 36 compute nodes. Each node has the following features: 32GB of main memory, Intel E5-2670 Sandy Bridge processors clocked at 2.60 GHz, 2 sockets with 8 CPU cores per socket for a total of 16 cores, and a total of 36 TB of local disks consisted of twelve 3-TB disks. The local disks are divided into two equal halves, one half for HDFS and the other for SciDB. Each half is configured as 5+1 RAID5. The whole cluster runs Centos 6.3. The nodes are connected by Infiniband (Mellanox MT27500 FDR IB). We use Cloudera 2.0.0-cdh4.1.2 for Hadoop and the open-source version 13.12 for SciDB.

For both Hadoop and SciDB, we conduct our experiments using two cluster configurations of 6 and 30 nodes. Hadoop has a replication factor of 3. The open-source version of SciDB, however, does not support replication.

Our bestMatch SciDB User Defined Operator (UDO) incorporates the KDTreeSingleIndexParams kd-tree implementation of FLANN [10] for its speed and robustness. In Hadoop, we use Rednaxela's Bucket PR kd-tree, as described in [11].

6.2 Dataset

We evaluate the performance of Hadoop and SciDB on a pair of satellite ground tracks: those of CloudSat and TRMM. TRMM orbits the earth around the equator, between approximately -35 and 35 degrees latitude, while CloudSat travels up and down close to the poles in a sun-synchronous orbit. Both satellites complete an orbit around Earth in approximately 100 minutes and their ground tracks intersect about 30 times a day. However, they normally go over an intersection location in their respective orbits at different times.

We extract the ground track geolocations from the HDF4 files of the dataset for each satellite and convert them into comma-separated values (CSV). These are used as input to our Baseline algorithm first. The data are indexed into sequence files in Hadoop as described above and ingested into the SciDB clusters.

Table 1 summarizes the data point distribution of chunks for the best optimal chunking configuration for our algorithm in SciDB.

Table 1. Distribution of the number of data points (non-empty cells)

	Min ($\times 10^6$)	Max ($\times 10^6$)	Mean ($\times 10^6$)	Std. Dev. ($\times 10^3$)	Average Chunks Per Node
TRMM	2.7	2.7	2.7	4.9	47,854
CloudSat	9.7	9.7	9.7	16.8	44,801

per node for 1-year of data in a 30-node cluster.

Although our array configuration/schema yields approximately 10^{16} possible array cells per chunk, the number of filled cells for each ground track is considerably smaller, i.e. $O(10^6)$, resulting in very sparse multidimensional arrays. While it is convenient in SciDB to distribute actual data evenly over the cluster nodes, aiding its parallel performance, the partitioning in Hadoop has to be done more laboriously through a sequence file. The data distribution over nodes, however, depends on HDFS distribution of blocks, which should be reasonably even, and the availability of nodes at runtime, which are scheduled by Hadoop's jobtracker.

A year's worth of data is approximately 20 GB for CloudSat and 10 GB[1] for TRMM. Evaluations are carried out for 1, 2 and 4 years of data.

6.3 Performance Evaluation

In this section, we present the results of our experiment for different cluster configurations and array chunking choices.

Figure 2 shows the average running time (in log scale) of our experiment on a cluster with 30 nodes. The Baseline algorithm takes significantly longer to finish because it includes the time to partition our raw data into the cluster's nodes. In addition, the Pig Script results in multiple iterations of Hadoop jobs, which is a primary disadvantage of Hadoop's 2-stage computation.

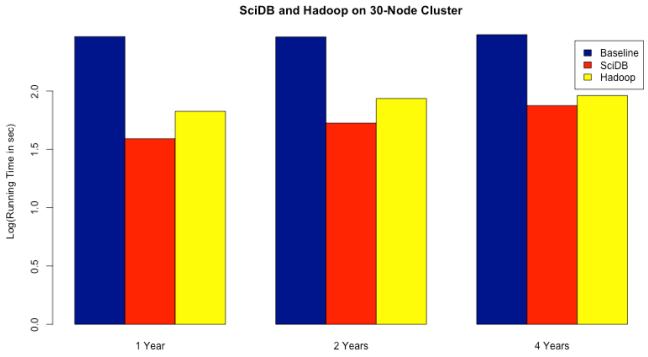

Figure 2. Optimal running time on a 30-node cluster.

Overall, SciDB performs better than Hadoop by a factor of ~2.5 in a cluster of 30 nodes, which aligns with our expectation. As the data volume increases with the number of years, this performance advantage diminishes, as shown in Table 2, in which the right-half column under each year heading provides the Hadoop-to-SciDB timing ratios. As the initial overhead cost of Hadoop becomes further amortized, we expect this trend to continue, with Hadoop's performance approaching that of SciDB.

Table 2. Optimal running time (in seconds) in a 30-node cluster, for both Hadoop (HD) and SciDB (Sci).

Nodes		1 Year		2 Years		4 Years	
30	HD	65.5	2.5	114.9	2.6	125.4	1.5
	Sci	25.6		43.6		83.7	
6	HD	89.8	0.7	155.8	0.6	254.9	0.3
	Sci	112.6		249.0		699.7	

In a smaller cluster of 6 nodes, where resources become more limited, SciDB becomes less efficient than Hadoop, as exhibited in Table 2. In the same cluster, Hadoop scales out better than SciDB as the data volume increases. It is also important to note that, in our experiments, data handling in SciDB becomes difficult when resources are limited, e.g. it takes several retries to partition the arrays. This is due to a known memory management problem in SciDB version 13.12 and earlier; however, it is reported that in its latest version, 14.8, SciDB has improved the efficiency of memory management, utilizing well-known techniques that are used in the database literature, such as memory arenas.

Table 3 shows the running time of our experiments with various chunk size configurations in both SciDB and Hadoop. Since the arrays are rather sparse and the distribution of data points is far from uniform, it is very difficult to come up with an optimal chunk configuration perfectly aligned with the recommendations by SciDB. Nevertheless, the experiments show that all the configurations exhibit similar running times, for both SciDB and Hadoop.

Table 3. Effect of chunk sizes using 1-year data set in Hadoop (HD) vs. SciDB (Sci) in 30-node cluster.

	22,500		45,000		90,000		720,000	
HD	73.84	2.50	65.50	2.50	66.50	2.60	75.97	2.52
Sci	29.51		26.21		25.61		30.10	

7. Discussion and Future Works

Our experiments have demonstrated the potential of an array database system, SciDB, especially for the Earth Science domain. SciDB gives us better performance than general analytic systems such as Hadoop, but its consistency raises a few concerns. It takes considerable effort to create sparse arrays in clusters with limited resources. In addition, we find that the ease of writing a SciDB operator depends on adequate documentation of its C++ class API for developers to reference. Presently this documentation exists primarily as comments in the source code, examples provided in the SciDB distribution, and through responses to developers' questions by Paradigm4 [14]. This is insufficient, because it is practically impossible to provide a complete set of relevant use cases through examples alone for a software as complex as SciDB. It is also worth noting, however, that SciDB is aimed at end users who are interested in using high-level queries rather than developing complex analytic algorithms.

For future works, we plan to compare Hadoop and SciDB with a broader range of problems, including both single-pass and iterative operations, in Earth science domain. Moreover, instead of using HDFS as our backend storage system, we plan to experiment with ones that have better data organization such as Parquet Columnar Store or Cassandra key-value Store. We would also like to explore indexing and chunking strategies that have promise for better efficiency. In addition, we are aware of other Big-Data frameworks such as Spark, which allows for multi-stage in-memory computation with similar advantage of MapReduce functional features as Hadoop, and plan to explore the possibility of array store in these systems.

Acknowledgement

We are sincerely thankful to the Advanced Information Systems Technology (AIST) program of the NASA Earth Science Technology Office (ESTO), which provided the funding for this study, and the NASA High End Computing (HEC) program, which provided the computing resources at the NASA Center for Climate Simulations (NCCS). We would also like to express our gratitude to our database administrator Gyorgy Fekete, and system administrators Hoot Thompson and Scott Sinno, without whose dedicated effort, we could not have done the study.

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 May 12, 2014, from http://www.paradigm4.com