A Performance Evaluation of Hive for Scientific Data Management

Taoying Liu, Jing Liu, Hong Liu, Wei Li
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
{lty, liujing2013, lh, liwei}@ict.ac.cn

***Abstract*—It is very important to evaluate the MapReduce-based frameworks for scientific data processing applications. Scientists need a low-cost, scalable, easy-to-use and fault-tolerance platform for large volume data processing eagerly. This paper presents an implementation of a scientific data management benchmark, SSDB, on Hive, a MapReduce-based data warehouse. A complete strategy of migrating SSDB to Hive is described in detail including query HQL implementation, data partition schema and adjustments of underlying storage facilities. We have tuned the performance using several system parameters provided by Hive, Hadoop and HDFS. This paper provides preliminary results and analysis. Evaluation results indicate that Hive achieves acceptable performance for some data analysis tasks even compared with some high efficient distributed parallel databases, but it needs subtle adjustments of underlying storage facilities and indexing mechanism.**

Keywords—benchmark; scientific data management; Hive; performance evaluation

I. INTRODUCTION

Distributed data processing frameworks like Hadoop[11] and its toolsets are widely used in business data processing field, especially in log analysis, data mining, business intelligence, and so on. There are less researches of scientific data processing on such platforms. However scientific data are expanding with an explosion speed. Scientists are among those who need a cheap, scalable distributed data processing platform urgently. Due to the following reasons, scientists prefer Hadoop-like technologies: (1) ease of use and learning, (2) function extensible, (3) simple yet feasible scalability solution, and (4) low costs.

Currently there are some efforts on scientific data processing on Hadoop platform, such as [26][27][32][33], and [6]. But there still lacks a benchmark of scientific data processing on distributed platforms. [1] is a work of evaluating the performance of astrophysics data queries implemented in Interactive Data Language [3], on an RDBMS and Pig, a data query engine based on Hadoop. In this work, 5 queries are extracted from the most often used operations by astrophysicists. Performance is compared, and it is found that Pig gets lower performance than database and IDL do on clusters with 10 or less nodes. Hadoop-like distributed frameworks are designed for clusters with hundreds or thousands of nodes, and their performance advantage over databases will be shown in that situation, and they are not efficient as databases are on small scale clusters. The work is done on Pig without any further optimizations. Technologies like column-store, partitions are adopted in many Hadoop-like frameworks, and they can bring much performance promotion for certain applications. What impact will these technologies bring to scientific data processing is not discussed.

SSDB [14], Standard Science DBMS Benchmark, was proposed in XLDB 2010 by MIT CSAIL research group. SSDB is proposed as a performance benchmark of a distributed scientific database that is the research output from this group. SSDB is a good typical workload sample of astronomy, remote sense and medicine image applications. It is implemented on SciDB [2] and MySQL. We implement this benchmark on Hadoop-based data warehouse Hive [10], and use optimization like partition and RCFile [8]. It shows that after optimization the performance of SSDB on Hive is acceptable.

Our contributions in this paper are three-fold. First, to be best of our knowledge, we are the first to comprehensively present how the SSDB can be implemented using the standard Hadoop and Hive software stack. Second, our detailed performance evaluation, with a complete coverage of various parameters tuning, reveals both the advantages and disadvantages of the Hadoop ecosystem when executing different SSDB workloads. Third, our evaluation results clearly point out the further research and development directions to improve the Hadoop/Hive system for scientific data management.

This paper is organized as follows. Background of standard Science DBMS Benchmark including the data model and queries and Hive are introduced in Section II. In section III, we illustrate in detail the implementation of SSDB on Hive, some modifications of the queries are made for the resilience to Hive. Experiment data and analysis are shown in Section IV. We run SSDB both on Hive and SciDB, performance is compared, and we also analysis the tuning parameters in Hive. Related work are listed and discussed in Section V, and finally we conclude in Section VI.

II. STANDARD SCIENCE DBMS BENCHMARK

A. SSDB Overview

SSDB [14] is a science data oriented benchmark proposed by MIT CIASIL Lab. The benchmark is composed of two

978-1-4799-1293-3/13/$31.00 ©2013 IEEE

parts: data and queries. Data of SSDB are abstracted from astronomy, remote sensing, medicine image processing. Those data are all location-based in a 2-dimensional space.

Data of SSDB simulate astronomy data of sky survey. Astronomical telescopes of a sky survey plan take pictures of their observation range, then slowly turn a very small angel, and continue taking pictures of the new observation area. Every picture is a rectangle image in the sky and each point of this image is transformed into a record with 11 physical attributes stored in 4-byte integers. The size of an image depends on the diameter of the telescope. There are 4 major scale configurations of SSDB dataset, see Table I. Each dataset is composed of sky images taken from telescopes. For example, for small scale, the image size is 3750*3750, and there are 160 images in this dataset. Actually, this is a 3-dimension array. Coordinates X, Y are the locations of a point in the observation area, and Z means the absolute position of the observation area. If you want to get the absolute position of a point, you should first look up a table to get the absolute position of the start point of this area, and then add X, Y. In fact, Z cannot cover the whole sky. It is picked randomly from the sky, with 80% from dense areas and 20% from sparse sky.

TABLE I. SSDB DATA SCALES

	Very_ small	**Small**	**Normal**	**Large**
Image width	1600	3750	7500	15000
# of images	40	160	400	1000
Total Size	4.5GB	99GB	0.99TB	9.9TB

Not all original raw data need to be processed after forth, SSDB first do the 'cook' process, and this process is called 'findstars'. Original data are scanned, if the value of one point is greater than the threshold, then the neighbors of this point are checked iteratively, points that have greater values than the threshold are included into this continued observation. The cook process removes most parts of the dark sky, and leaves data useful for future research. The second step of 'cook' performs a simple cluster algorithm. On the basis of the data of the first step, stars that are located within a short distance are grouped together into one polygon. This process is called 'groupstars'. The resulted areas from 'groupstars' could correspond to stars, boundaries of vegetation regions (satellite imagery), temperature regions (oceanography), or medical anomalies (medical imaging).

B. SSDB Queries

There are 9 queries in SSDB:

Q1: Scan all the original data and compute the average value of one attribute.

Q2: Recook one image of the raw data using a different threshold.

Q3: Alleviate noise and collapse the original array for a 10:3 proportion.

Q4: Compute the average value of observation attributes of all the stars that fall into a specified rectangle.

Q5: Get all records of observations whose polygons fall into a specified rectangle.

Q6: In the specified slab, find all the small rectangle tiles containing more observations than a threshold.

Q7: For all the observation groups, find those whose centers fall in the specified area at any time.

Q8: Find those observation groups whose trajectories intersect the specified area, and output raw data of those observation groups for small tiles. Trajectory is defined as the sequence of centers of the observations in a group.

Q9: Define trajectory as the sequence of the boundary of the observation group. Find those observation groups whose trajectories intersect the specified area, and output raw data of those observation groups for small tiles

These 9 queries can be divided into 3 categories. First category is the scan-type query, Q1, Q2, and Q3. They work on the whole original raw data. All data need to be scanned. But the queries are simple. The second type of queries, Q4, Q5, Q6 and Q7, query on the output of cooking process in order to find observation objects that intersect with a certain area of the sky. These queries scan only part of the cooked data. The third category includes Q8 and Q9, they are very complex queries. They first find object ids from the cooking results, and then get records detail corresponding to object ids from the original data. The input of Q8 and Q9 covers not only the cooked data but also the original raw data.

III. SSDB IMPLEMENTATION ON HIVE

SSDB is a built-in benchmark of SciDB. There are also SSDB for MySQL version, which can be downloaded from XLDB website [15]. This version is totally different from SciDB version. Data are distributed evenly among cluster nodes and are loaded as local tables into a MySQL instance on each node. Programs written in C++ are used for global control and data processing.

SSDB has not been ported to MapReduce platforms before. We choose Hive as the migration target because that Hive is a widely used MapReduce-based data warehouse, and it supports SQL-like query language and we can add user defined function into Hive. In this section, we will describe how SSDB data is loaded and stored in Hive, and how queries are programmed.

A. Hive Background

Hive [5][10] is an open-source Hadoop-based data warehouse framework. With series of tools that Hive provides, structured data files can be mapped to database tables. Hive also defines a SQL-like language called Hive Query Language, which is compatible to SQL. HQL statements can be translated into MapReduce tasks to run on multiple nodes.

Data in Hive can be stored in three formats, text, sequence binary and RCFile[8]. Text is the default data store format of Hive. It is often used in log processing and data loading. The drawback of text format is that the costs for parsing are much

higher than using the binary format and data should be parsed repeatedly for each time of query execution.

Sequence file is a binary format provided by Hadoop and also supported in Hive. Sequence file uses standard writable interface to implement serialization and deserialization. Data are stored as <key, value> pairs in sequence file.

RCFile, Record Columnar File, is designed for storing relational tables on clusters using the MapReduce framework. It divides table data first horizontally and then vertically. Table records are divided into row groups. Each row group block contains the sync marker, metadata header and the table data. Values of one column are stored continuously, and then the next column in this row group. Thus RCFile is able to read only necessary columns and skip those columns that are not needed in queries.

Hive does not support indexing. Data are stored in fixed chunks in HDFS. However Hive uses partition mechanism to narrow down query range, which can be seen as a very simple alternative to indexing. With this mechanism, data can be divided into partitions using a hashing based method. If the partition identification is specified in the where clause, only that partition of the data are scanned.

TABLE II. IMPLEMENTATION OF SSDB QUERIES IN HIVE AND SCIDB

Q#	SciDB Array Query Language	Hive Query Language
Q1	avg(subarray(normal,0,0,0,19,7499,7499),a)	select avg(a) from normalrc where z<=19 and y<=7499 and x<=7499;
Q2	findstars(subarray(normal,0,0,0,0, 7499,7499),a,900)	MapReduce program
Q3	thin(window(subarray(normal_reparted,0,0,0,19, 7499,7499), 0,1,0,4,0,4,avg(a)),0,1,2,3,2,3)	select * from(select cast(x/4 as int) as sj, cast(y/4 as int) as si, cast(avg(a) as int) from normal where z<=19 group by cast(x/4 as int), cast(y/4 as int))normal2 where (si-2)%3=0 and (sj-2)%3=0;
Q4	for (i=0..19) avg(filter(subarray(normal_obs_`printf $i`, 491000, 493000, 501000, 503000, center is not null),sumPixel)	select avg(sumpixel) from normal_obs_sumrc where j>=493000 and j<=503000 and i>=491000 and i<=501000 and center='true';
Q5	for (i=0..19) filter(subarray(normal_obs_`printf $i`, 491000, 493000, 501000, 503000, polygon is not null)	select * from normal_obs_sumrc where j>=493000 and j<=503000 and i>=491000 and i<=501000 and center='true';
Q6	for (i=0..19) filter(window(filter(subarray(normal_obs_`printf $i`, 491000, 493000, 501000, 5030000, center is not null), 0,12,0,12,count(center)),center_count>$count)	select x1,y1,cs from(select cast(j/12 as int) as x1,cast(i/12 as int)as y1, count(*) as cs from normal_obs_sumrc where j>=493000 and j<=503000 and i>=491000 and i<=501000 group by cast(j/12 as int),cast(i/12 as int))heihei where heihei.cs>1;
Q7	filter(aggregate(normal_groups,avg(x),avg(y),group), x_avg > 496000 and y_avg > 497000 and x_avg < 500000 and y_avg<510000	select * from normal_obs_sumrc where j>=493000 and j<=503000 and i>=491000 and i<=501000 and polygon is not null
Q8	aggregate(cross_join(small_groups, filter(sum (project (apply (cross (subarray (Points, 0,3), join (subarray (small_groups, null, null, null,18) as Pi, subarray (small_groups, null,1,null,null) as Pj)), crosses, iif (((((Pi.y <= Points.y) and (Pj.y > Points.y)) OR ((Pi.y > Points.y) and (Pj.y <= Points.y))) and (Points.x < Pi.x + ((Points.y - Pi.y) / (Pj.y - Pi.y)) * (Pj.x - Pi.x)) and (Pi.x is not null and Pi.y is not null and Pj.x is not null and Pj.y is not null)), 1, 0)), crosses), crosses,Pj.group), crosses_sum > 0)), avg(small_groups.x),avg(small_groups.y),small_groups.group) "	select /*+ mapjoin(q81)*/ cast(i/100 as int),cast(j/100 as int) from normal_obsrc join (select oid from normal_groupsrc where pinp(cast(x as int), cast(y as int),497000,498000,499000,500000)=1)q81 on (normal_obsrc.oid=q81.oid) group by cast(i/100 as int),cast(j/100 as int); select /*+ mapjoin(q8_1)*/ cast(normalrc1.y1*1000+normalrc1.x1 as int),normalrc1.x,normalrc1.y,normalrc1.a,normalrc1.b,normalrc1.c,normalrc1.d,normalrc1.e,normalrc1.f,normalrc1.g,normalrc1.h,normalrc1.i,normalrc1.j,normalrc1.k from normalrc1 join q8_1 on (normalrc1.x1=q8_1.x and normalrc1.y1=q8_1.y);
Q9	aggregate(filter(filter(cross(subarray(small_obs_0, 491000, 493000, 501000, 5030000) as A, small_groups),A.polygon is not null), A.oid=small_groups.oid),avg(small_groups.x),avg(small_groups.y),small_groups.group)	select oid, cast((avg(ax)-496211)/100 as int), cast((avg(ay)-497861)/100 as int) from (select normal_groupsrc.x as ax, normal_groupsrc.y as ay, normal_groupsrc.oid from normal_groupsrc join (select normal_obs1rc.oid as oid from normal_obs1rc where j>=498000 and j<=500000 and i>=497000 and i<=499000 and polygon is not null)C on(C.oid=normal_groupsrc.oid))haha group by oid; create table if not exists normalq9result like normalrc; insert overwrite table q9result select /*+ mapjoin(q9_1)*/ cast(normalrc1.y1*1000+normalrc1.x1 as int),normalrc1.x,normalrc1.y,normalrc1.a,normalrc1.b,normalrc1.c,normalrc1.d,normalrc1.e,normalrc1.f,normalrc1.g,normalrc1.h,normalrc1.i,normalrc1.j,normalrc1.k from normalrc1 join q9_1 on (normalrc1.x1=q9_1.x_avg and normalrc1.y1=q9_1.y_avg);

B. SSDB Queries in HQL

HQL is quite similar to the standard SQL, but HQL in Hive only has preliminary functions compared with MySQL. SSDB MySQL version [15] uses spatial query and indexing function in MySQL. Queries of Q4-Q9 all use this advanced feature. All the queries use C++ codes in distributing data and paralleling process. Although SQL and HQL are very similar, neither SQL statements nor C++ codes can be directly migrated from MySQL version to implement SSDB queries in Hive. In order to provide complete functionality of SSDB, we implement 9 queries in HQL from scratch. Queries for normal scale test in HQL and AQL are listed in TABLE II except Q8 and Q9, because the AQL scripts for normal dataset are too long to fit in the table space, we list HQL statement for small scale. SciDB AQL statements are listed here to help understanding and validating the queries.

Q2 is implemented as a MapReduce program and inserted into Hive as a user defined function.

Except for window and thin operators, HQL statements of Q1-Q7 are easy to understand. Because Hive provides distributed framework and parallel processing capability, there is no need to write any C++ codes or Java codes.

We find that some operators in AQL have no correspondence to any HQL operators, such as window, subarray, project, and thin. Window [16] operator in AQL is a fairly common operation in scientific data processing, it computes aggregates over moving window. Thin [16] operator is to select data from an array at fixed intervals along each dimension. Hive does not support multi-dimensional window operator. In Q3, the average value of property 'a' should be calculated on a running 5*5*2 3D window. Window operation needs to locate the neighbors of a record along all dimensions, and store these records in memory for aggregation. Because there is no indexing in Hive, it is very resource-consuming to locate neighborhoods of a point in 3D space. So we simplify query Q3 to calculate on a running 2D window with window size of 5*5. The aggregation inside a window is implemented by HQL operator 'group by', elements in this window are grouped together and calculated.

C. Data Partitions in Hive

There are two categories of data in SSDB. One is the original raw data. This data is the mapping of the sky, each record is corresponding to a point in the sky with 14 non-null properties. This category of data is uniform. The data are composed of serial pictures taken by the telescope. Each picture is an image of square shape, and many of such squares scatter over the sky. The coordinates inside the square image are local coordinates. Each square has its starting point; the coordinates of this point are stored in a standalone table. The absolute coordinates of points inside an image can be calculated by adding the starting point coordinates to the local coordinates. Many query statements are related with the local and absolute coordinates.

Although there is no array support in Hive, Hive has simple indexing mechanism partition to speed up locating array elements. It is natural to store an image in one partition indicated by its image number. We divide the original raw data into several partitions, and the sizes of each partition are totally equal. In the 9 queries, there are Q1 and Q3 in which Z coordinates are specified. For normal scale, Q1 checks the images with image numbers less than 20, which are only 20/400 of the total raw data. With this solution, we can greatly reduce the amount of the input data of Q1.

For data cooked and filtered, many of the records of dark areas are removed. Validated observations are unevenly distributed across the sky. In normal scale of SSDB, the size of the result of cook process 'findstars' is about 409GB. We have two methods to divide the data into partitions. First method is to put data into zones according to their different image numbers. But in fact, queries on the observations data are all related with the world-coordinate instead of local coordinate with area identifications. In fact, some queries scan data across area boundaries. Q4, Q5, Q6, Q7 use absolute coordinates. So, it will not bring any benefit to narrow the queried range using this partition solution. We adopt the second method. We convert the coordinates of the after-cooked data into absolute coordinates, and divide the data according to their absolute coordinates. The data are gridded into tiles or sliced into stripes, as shown in Fig. 1. There is no essential distinction between the two methods, but they will affect the data distribution in partitions. Some partitions may have much more data than others, and some may have much less data than others. It is the inherent characteristics of the application data who determines the data skew incurred. More details of partitions will be given in section IV.

Fig. 1. Two Partition Schemas for Cooked Data.

IV. Experiment Results and Performance Factors

In this section, we will report the results of SSDB running on Hive and SciDB, and analyze some tuning factors of Hive which could affect the performance.

A. Experimental Setup

We run the benchmark on a gigabit ethernet Linux cluster of 5 nodes. Each node has dual 6-core 2.0GHz Intel Xeon E5-2620 CPUs with the hyperthreading option enabled, and 3 3TB SATA HDDs and 32GB RAM. The version of SciDB we use is 13.2, and the Hive version is 0.9.0, Hadoop is 0.20.0.

B. System Performance

The experiment results are listed in TABLE III. We test the performance of "Load", "Cook" and 9 queries for both small and normal datasets. Each dataset is distributed among 5 nodes. Every query was run 10 times, and system cache was cleared each time before testing. The numbers in Table III are average values of one running. This is different from [14], in which the execution time is the sum of 15 times of running.

TABLE III. SSDB EXCECUTION TIME ON HIVE AND SCIDB (SECOND)

	Dataset	Load	Cook	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
SciDB	small	1031	44	2	4	72	4	2	13	0	5	4
	normal	10394	1014	10	14	533	5	11	387	0	19	17
Hive	small	1048	130	65	58	71	29	34	37	22	1039	869
	normal	12104	13936	236	168	250	51	35	40	21	13061	3802

We get the following observations:

1) For data loading: Data loading refers to the operation loading original raw data into the testing systems. Both systems load distributed data from several nodes in parallel. The bottleneck of reading data is the hard disk bandwidth; SciDB and Hive get approximately the same performance in this operation.

2) For query Q1: This is a simple query that aggregates one property of all records of the array, but with great performance difference on the two systems. Hive does not support array, it needs to scan all the data to get only one element of an array. While the basic data model of SciDB is array, it can quickly locate any array element according to its coordinates.

3) For complicated array operation Q3: There is not much difference on Hive and SciDB.

4) For Q7, the query with little data input and output: The size of the input file is less than 100MB, in this situation Hive is much slower than SciDB.

5) For Q4, Q5 and Q6: In the original SSDB released with SciDB, there are 20 queries accessing the same array in parallel. We change the concurrent queries into sequential ones because SciDB does not respond when processing concurrent user requirements in parallel in our test. This greatly reduces the performance gap between Hive and SciDB, compared with the astonishing gap between SciDB and MySQL reported in [14].

6) For Q8 and Q9: These two queries first do some data analysis based on the already cooked data and then fetch from raw data records, they deal with large volumes of data both input and output. In original SSDB, the output of Q8 and Q9 are not written back to the hard disk. But usually the output of Q8 and Q9 are quite large, we suppose scientists will like to save them for later research. So we add save statement into Q8 and Q9, this also greatly reduces the performance gap between Hive and SciDB. But there is another reason for the slowness. Q8 and Q9 need to read all fields of records and RCFile is not efficient when all fields of records are needed in a query. This is because when reading all columns, it needs to locate all row groups inside an HDFS block. The start of each row group must be found by scanning.

C. Performance Factors

Parallel Processing Slot Number

The nodes of our testing platform all have dual 6-core processors with the hyperthreading option enabled. There are 24 cores that can be used. But as for different workload, it is not always beneficial to get more processors. SSDB is a workload with heavy IO consumption, especially for Q1, Q2 (for all images) and Q3.

We test normal scale of Q1 on SciDB and Hive with different slot size. Q1 is a simple workload which just scans several images and computes the average value of one attribute. We start 4, 8, 16 and 24 instances of SciDB working daemon on each node, and find that Q1 gets best performance when there are 4 instances on one node. It is the hard disk bandwidth which limits the CPU processing speed. We also test Q1 with different mapper slot sizes on Hive, and get the same result, see Fig 2. So we could set the same size of parallel processing unit in these two systems. That is also a relatively fair configuration for our tests.

Fig. 2. Execution time of Q1 with different slot numbers

HDFS Block Size

The number of mappers is usually defined by the number of HDFS block size. In most situations, one mapper processes one block. So, multiple processors or nodes could be used simultaneously to process large volumes of data.

Data of SSDB could be stored using three file storage format, text, sequence, and RCFile. As shown in [13], different storage format greatly influence the application performance, usually binary format may get the best. In our experiments, we have tested queries under these three file formats, RCFile gets better performance than text and sequence file does for queries from Q1 to Q6, but gets worse for Q7, Q8 and Q9. RCFile is suitable for queries that only fetch few columns of the table. As Q8 and Q9 select all attributes of records that meet the conditions, RCFile cannot show benefit for this situation. But for the consistency of the configuration for our tests, we only show RCFile performance of all queries here.

We use RCFile as our data storage format. For queries Q1, only attributes ‘X’, ‘Y’, ‘Z’ and ‘a’ are extracted from all 13 attributes. Suppose we use the default HDFS block size 64MB,

the data that need to be read are only 14MB, that is to say, one mapper loads only 14MB for its processing. So for small scale (120GB) query Q1, there should be 1920 mappers. If we set HDFS block size to 256MB, only 480 mappers need to be started. Obviously, the latter configuration greatly decreases the task scheduling costs and the management costs. In the actual testing, 256MB block can achieve much better performance than the default 64MB.

Partitions and Data Skew

As discussed in section III, original raw data are naturally divided into partitions according to their image numbers. Partitions are all of the same size. This greatly improves the performance of queries like Q1 and Q3.

Cooked data can be partitioned using grid or slice method. We grid the cooked data of the small scale into square tiles with 1000*1000 size. There are totally 167 tiles containing observation records. Because the stars in the sky are not uniformly distributed, so every tile contains different numbers of stars. From our statistics, using 1000*1000 tiles make the data heavily fragmented. About 65% tiles have sizes below 32MB, see in Fig 3.

We have also tried tiles with size 10000*10000. This made partitions evenly distributed, but in our experiment we find that this configuration reads much more data from disk than the 1000*1000 configuration does. Big partitions limit the parallel degree, fewer mapper program are used for data processing. Many CPU cores are not involved; this results in a worse performance.

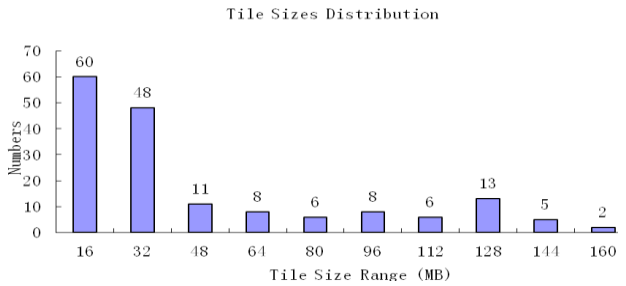

Fig. 3. Tile Sizes Distribution in Small Scale

Under SSDB normal scale, although 1000*1000 configuration can incur less data read from hard disks, it will produce too many more files in HDFS and will incur much more extra maintaining overhead, so we divide each image into several stripes along one coordinate. For a 7500*7500 image, there are 8 stripes in it, and every stripe has a size of 7500*1000. This method totally produces 3200 partitions.

Row Group Size

RCFile first divide the table horizontally into row groups. The parameter of row group is a block containing table rows, and inside these rows, data are stored in columnar style. Row group size can be configured according to different applications features. When row group becomes larger, whole storage space is decreased for smaller meta data space, and the same is the cost of data compression, but needs a larger consumption of RAM. While row group size becomes smaller, total storage space consumption is decreased, the time of data compression and decompression increases quickly, incurring extra CPU costs. At the same time, the time of data read from disk is increasing too, because RCFile blocks are small, whenever a row group is read, next row group meta data should be located and parsed. We should get balance of the query performance and the storage space, and find a fitful row group size.

In [9], the impacts of different row group size to query performance are investigated. For word grep application, row group size is configured as 1MB, 4MB and 16MB and the execution time and data read from hard disks are measured. Under the workload, the best execution time is achieved when row group is set to 16MB. This value is different from 4MB that is the RCFile designers' suggestion[8]. The designers get this value from the testing of RCFile under the Facebook workload. So the best row group size is relevant to the application logic.

We test the execution time and data volume read from disks of Q1, see TABLE IV. The data read from disks are measured using Linux system command iostat. As our expectation, the smaller row group size, the longer of execution time and larger data volume read from disks. When the row group size is set to 64MB, we get the best performance and the least data volume read from hard disks.

TABLE IV. EXECUTION TIME AND DATA READ FROM DISK OF Q1 FOR VARIED ROW GROUP SIZE

Row Group (MB)	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
Time (sec)	105	82	71	67	74	68	63	66	79
Data Read (GB)	13.4	11.2	9.8	9.1	9.0	8.6	8.4	8.5	8.9

V. RELATED WORK

Since MapReduce[12] has been introduced for massive data processing, many data management systems based on it come into birth. Hadoop[11] is one of the most popular open source distributed data processing systems. Upon Hadoop, there exist many data processing platforms, some of them provide query language, such as Pig Latin [17], Hive. Pig and Hive are both data warehouses based on Hadoop, both of them provide high level query language compliant with SQL standard, which translates user queries into mapreduce tasks running on underlying Hadoop and HDFS. DryadLINQ[19] is a parallel work from MicroSoft, it can also translate LINQ or C# into distributed Dryad tasks. Some argue that MapReduce framework is neither designed for scientific applications, nor for scientific data manipulation [18]. Due to its lack of support for complex embedded data type and index, many operations of array are very difficult to complete incurring great much extra IO overhead. In this paper, we aim to evaluate to what extent the state-of-the-art MapReduce-based data processing systems can support scientific data management applications.

As for scientific data processing benchmark, there is little published work in this field. [1] is the most relevant work with

ours. The authors collect real scientific data from astrophysics and extract 5 queries that are most often used. The goal of this is also to evaluate whether it is suitable for executing scientific data queries on MapReduce-based frameworks. Queries are implemented and executed on Pig [17] and a DBMS and IDL. It shows that in the small scale clusters DBMS significantly outperforms Pig, but not always in the same speedup. [6] is the first work on comparing the performance between MapReduce-based data processing platform and RDBMS. The data used in this benchmark are synthesized plain text and HTML documents. Benchmark operations are 'grep' task and complex data analysis tasks like selection, aggregation, join and user defined aggregation. Loading data and analytical tasks are executed on Hadoop, a column-store DBMS Vertica and another commercial DBMS. Testing platforms have 25-node, 50-node and 100-node configurations. Actually on all platforms Vertica has the best score, next is the commercial DBMS, and then Hadoop. But for grep task, the larger the scale, the less the performance difference. For analytical tasks, even on the 100-node cluster, Hadoop does not show competent performance. [4] compares the performance of astronomy application cross match on Hive and purely hand-written MapReduce program.

From the early days of the generation of cloud technologies, scientists began to evaluate the feasibility of migrating existed scientific applications into the cloud ecologies [30][31]. [28] compared the costs of storing parts of LSST [29] experiments datasets in Amazon S3 and in grids. They conclude that cloud is a competent alternative for hosting scientific datasets.

Some scientists already have done much work on customizing and optimizing these tools towards the special requirements from scientific data processing. [7] designed a storage format for scientific array operations. [20][21][22][23][24][25] all work on the spatial indexing support for Hadoop and Hadoop-based data processing systems. Hadoop-GIS [25] is a high performance spatial data query engine supporting large volumes of spatial data built on Hadoop and Hive. Hadoop-GIS supports global partition indexing and customizing local indexing on demand. These features may accelerate some location related operations in SSDB, e.g., Q4-Q9. Twister[33] is also an improved Hadoop-based platform supporting iterative scientific computing job. These works are all based on Hadoop. SciDB[2] is a new distributed parallel database. It is a shared-nothing architecture much like MapReduce framework. Data are stored distributed on the local space of every SciDB instances. These instances could be configured on each node of the cluster, or could be dispatched on each core of one server. SciDB is designed for scientific array data usage models. It supports multiple dimensional arrays, and provides many specific array operators. In our test, it shows good performance score for a 5-node cluster.

VI. CONCLUSIONS

SSDB is a standard science DBMS benchmark. It is first implemented on a distributed science database SciDB and on MySQL. We port this benchmark to the mapreduce-based data warehouse Hive and test small and normal scale of SSDB on SciDB and Hive. This paper is a preliminary work, we will do further analysis and tests with new optimization techniques like Hadoop-GIS. Currently we find that Hive/Hadoop could achieve acceptable performance for some data analysis tasks with subtle system parameters tuning. There is great potential for further improvement with good support of scientific data management. First, a specific array-oriented storage format helps a lot. Such storage format should provide quick array elements locating and fetching, boundary checking, and support flexible partition mechanism to deal with data distribution and data skew. Second, some special scientific data array functions could be added into Hive as user defined functions, such as window, thin, subarray, etc. These user defined operators could take full advantage of the underlying storage for scientific data. Third, indexing is needed especially for sparse array and spatial space data. And last, for some complex data queries, there are many intermediate data that are written into hard disk, incurring redundant IO overheads. This could be simplified and customized for scientific data management.

ACKNOWLEDGMENT

This work is supported in part by the Hi-Tech Research and Development (863) Program of China (Grant No. 2013AA01A212, 2013AA01A209), the National Basic Research (973) Program of China (Grant No.2011CB302502, 2011CB302803), and the major national science and technology special projects (Grant No. 2010ZX03004-003-03). We like to thank Dr. Rubao Lee and Dr. Yin Huai, for their insightful suggestions. We would also thank Dixin Tang, Liechun Zhou and Shufang Wang, they did valuable work to support this research.

REFERENCES

[1] Loebman, S.; Nunley, D.; Yong-Chul Kwon; Howe, B.; Balazinska, M.; Gardner, J.P., "Analyzing massive astrophysical datasets: Can Pig/Hadoop or a relational DBMS help?," IEEE International Conference on CLUSTER, vol., no., pp.1,10, Aug. 31 -Sept. 4 2009

[2] Paul G. Brown, Overview of sciDB: large scale array storage, processing and analysis, Proceedings of the 2010 international conference on Management of data, June 06-10, 2010, Indianapolis, Indiana, USA

[3] "IDL – Data Visualization Solutions," http://www.ittvis.com/ ProductServices/IDL.aspx

[4] Cuncang Mi; Qian Chen; Taoying Liu "An Efficient Cross-Match Implementation Based on Directed Join Algorithm in MapReduce", Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, On page(s): 41 – 48

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in ICDE, 2010, pp. 996-1005.

[6] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in SIGMOD Conference, 2009, pp. 165-178.

[7] Joe B. Buck, Noah Watkins, Jeff LeFevre, Kleoni Ioannidou, Carlos Maltzahn, Neoklis Polyzotis, and Scott Brandt. 2011. SciHadoop: array-based query processing in Hadoop. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA

[8] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao;,Jain, N., Xiaodong Zhang, Zhiwei Xu, "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," Data Engineering (ICDE), 2011 IEEE 27th International Conference on , vol., no., pp.1199,1208, 11-16 April 2011

[9] Avrilia Floratou, Jignesh M. Patel, Eugene J. Shekita, and Sandeep Tata. 2011. Column-oriented storage techniques for MapReduce. Proc. VLDB Endow. 4, 7 (April 2011), 419-429.

[10] Hive. http://hive.apache.org/.

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

[12] Jeffrey Dean, Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM, v.51 n.1, January 2008

[13] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. 2010. The performance of MapReduce: an in-depth study. Proc. VLDB Endow. 3, 1-2 (September 2010), 472-483.

[14] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stanley B. Zdonik, Paul G. Brown. SS-DB: A Standard Science DBMS Benchmark. XLDB 2010, Stanford University, CA, Oct. 6-7, 2010

[15] "Standard Science Database Benchmark," http://www.xldb.org/science-benchmark/

[16] "SSDB User's Guide", http://www.scidb.org/forum/download/file.php?id=56

[17] C. Olston, B. Reed, U. Srivastaba, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreignlanguage for data processing," in Proc. Of the SIGMOD Conf., 2008, pp. 1099-1110.

[18] M. Stonebraker, J. Becla, D. DeWitt, K.-T. Lim, D. Maier, O. Ratzesberger, and S. Zdonik, "Requirements for science data bases and SciDB, " in Fourth CIDR Conf. Perspectives, 2009.

[19] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. 2008. DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language. In Proceedings of the 8th USENIX conference on Operating systems design and implementation (OSDI'08). USENIX Association, Berkeley, CA, USA, 1-14.

[20] Zhang, Shubin, Jizhong Han, Zhiyong Liu, Kai Wang, and Zhiyong Xu. "Sjmr: Parallelizing spatial join with mapreduce on clusters." In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, pp. 1-8. IEEE, 2009.

[21] Zhang, Shubin, Jizhong Han, Zhiyong Liu, Kai Wang, and Shengzhong Feng. "Spatial queries evaluation with mapreduce." In Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on, pp. 287-292. IEEE, 2009.

[22] Cary, Ariel, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe. "Experiences on processing spatial data with mapreduce." In Scientific and Statistical Database Management, pp. 302-319. Springer Berlin Heidelberg, 2009.

[23] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, Joel Saltz: Hadoop-GIS: A Spatial Data Warehousing System Over MapReduce. Accepted. The 39th International Conference on Very Large Databases (VLDB'2013), Trento, Italy, August 26-30, 2013.

[24] Aji, Ablimit, and Fusheng Wang. "High performance spatial query processing for large scale scientific data." In Proceedings of the on SIGMOD/PODS 2012 PhD Symposium, pp. 9-14. ACM, 2012.

[25] "Hadoop-GIS". http://confluence.cci.emory.edu:8090/display/HadoopGIS/Home

[26] Wiley, Keith, Andrew Connolly, Simon Krughoff, Jeff Gardner, Magdalena Balazinska, Bill Howe, Y. Kwon, and Yingyi Bu. "Astronomical image processing with hadoop." Astronomical Data Analysis Software and Systems(2010).

[27] Wiley, Keith, Andrew Connolly, Jeff Gardner, S. Krughoff, Magdalena Balazinska, Bill Howe, Y. Kwon, and Yingyi Bu. "Astronomy in the cloud: using mapreduce for image co-addition." Astronomy 123, no. 901 (2011): 366-380.

[28] Palankar, Mayur R., Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. "Amazon S3 for science grids: a viable solution?." In Proceedings of the 2008 international workshop on Data-aware distributed computing, pp. 55-64. ACM, 2008.

[29] "Large Synoptic Survey Telescope," http://www.lsst.org/.

[30] Foster, Ian, Yong Zhao, Ioan Raicu, and Shiyong Lu. "Cloud computing and grid computing 360-degree compared." In Grid Computing Environments Workshop, 2008. GCE'08, pp. 1-10. Ieee, 2008.

[31] Deelman, Ewa, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. "The cost of doing science on the cloud: the montage example." In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, p. 50.

[32] Qiu, Xiaohong, Jaliya Ekanayake, Scott Beason, Thilina Gunarathne, Geoffrey Fox, Roger Barga, and Dennis Gannon. "Cloud technologies for bioinformatics applications." In Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, p. 6. ACM, 2009.

[33] Ekanayake, Jaliya, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. "Twister: a runtime for iterative mapreduce." InProceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 810-818. ACM, 2010.