BigDataBench-S: An Open-source Scientific Big Data Benchmark Suite

Xinhui Tian[1, 7], Shaopeng Dai[1, 7], Zhihui Du[2], Wanling Gao[1, 7], Rui Ren[1, 7], Yaodong Cheng[3], Zhifei Zhang[4], Zhen Jia[5], Peijian Wang[6] and Jianfeng Zhan[1, 7]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Tsinghua University, Beijing, China
[3]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, China
[4]Capital Medical University, Beijing, China
[5]Princeton University, Princeton, USA
[6]Xi'an Jiaotong University, Xi'an, China
[7]University of Chinese Academy of Sciences, Beijing, China

***Abstract*—Data generated from modern scientific instrumentation have grown up to an unprecedented scale. Moreover, data formats and computational behaviors of scientific big data workloads are much more complex than those in Internet services. These two facts pose a serious challenge to scientific data management and analytics. Among many concerns, the first one is how to build a comprehensive and representative scientific big data benchmark suite. Previous benchmark efforts either focus on Internet areas (i.e. BigDataBench) or pay attention to a specific area (i.e. GeneBase). This paper presents our preliminary work on building a comprehensive scientific big data benchmark suite---BigDataBench-S. Also, we use BigDataBench-S to evaluate several general-purpose big data management systems specifically designed for Internet services applications. Our evaluation shows: these systems cannot achieve expected performance for many scientific workloads, especially for complex matrix computation, for the lack of appropriate mechanisms and policies on data storage, query optimization and support of distributed matrix computation.**

Keywords*—*Big Data Benchmark, HPC or Scientific computing, Management and Analytics, Evaluation.

I. Introduction

Scientific researches now face great challenges on how to efficiently manage and analyze massive data generated from scientific equipment and devices, e.g., the Large Synoptic Survey Telescope (LSST), collision accelerator and high throughput gene sequencers. Up to 2014, the large collision accelerator had collected about 140 PB raw data [1], and the size of the image data collected by LSST is predicted to be 100 PB or even more after its use in 2020 [2].

Scientific big data pose a serious challenge to scientific data management and analytics. General-purpose big data management systems such as Hadoop and Spark are originally designed for Internet service workloads. However, the data formats and computational behaviors of the scientific workloads are much more complex than those in traditional data management scenarios, i.e. internet services. In many scientific fields, raw data are stored in a multidimensional array format [3] and computation requires many complex operations, such as matrix decomposition, frequency variation, and so on. For instance, for satellite imagery collection, each satellite collects image data on its orbital. The smallest unit of the image size is between 10 to 100 megabyte. The data collected by a single satellite form an image vector consisting of multiple small units, and the vectors collected by the multiple satellites will compose the whole image of the entire earth surface. For the regions covered simultaneously by several satellites, they have multiple coordinate systems, which need further coordinate transformations and a series of linear transformation operations.

On the other hand, scientists always need to perform different types of operations on the same set of data (query operations or complex analysis), and data sharing is common among different kinds of workloads. In general, scientists do not care about data formats; however the latter impacts the performance significantly. Even for much simpler Internet service workloads, previous work has developed diverse storage formats, e.g. Parquet [4], ORCFile [5] and demonstrates obvious performance gaps.

We believe there are great design and implementation spaces of management and analytics systems for scientific big data. Before exhaustedly exploring the space, we have to answer two fundamental issues. First, what is a comprehensive scientific big data benchmark suite? Second, do general-purpose big data management and analytic systems perform well in the context of scientific big data?

This paper focuses on answering the above two questions. Firstly, on the basis of our previous work---BigDataBench, we build an open-source scientific big data benchmark suite--BigDataBench-S [1]. BigDataBench-S involves many typical scientific scenarios and provides different implementations on many widely-used big data systems. To ensure the completeness of the benchmark, we analyze typical workloads

[1] We will release BigDataBench-S on the BigDataBench homepage: http://prof.ict.ac.cn/BigDataBench soon.

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.111

and data sets in high energy physics, astronomy, and genomics in detail. In GenBase, each workload represents a mix of data management, linear algebra and statistical operations, while each workload in BigDataBench-S contains either data management operations or complex analysis operations. The purpose of this design decision is to facilitate apple-to-apple comparison among the components of different big data systems. Also, we will add workloads from other scientific fields in near future.

Using BigDataBench-S, we evaluate several widely used general-purpose big data analytics systems including Hadoop (MapReduce and Tez engine) and Spark, considering the impact of different data size and storage format. Through the analysis of experimental results, we have the following findings: 1) For the query workloads in scientific fields, the execution parallelism degree and the workload balance are the main factors that affect the efficiency of the implementation. 2) Current column-based storage formats help decrease the space used for data storage, but cannot help achieve better performance for data filter operations, for lacking of appropriate data structure for scientific data sets. 3) For the complex analysis workloads in scientific researches, the distributed linear algebra operations provided by the current systems cannot achieve good enough performance, for lacking of efficient matrix partitioning and execution strategy for matrix computation. The results of this paper can help explore the design and implementation spaces of scientific big data management and analytics.

The contributions of this paper are as follows:

1) We construct a new benchmark suite, BigDataBench-S, which contains workloads and data sets from typical scientific research areas.

2) Using BigDataBench-S, we comprehensively evaluate several widely used general-purpose big data analysis systems with different data sizes and storage formats.

3) From the experimental results, we find there are great design and implementation spaces for scientific big data, including new storage format and index, query optimization strategies, distributed matrix structure and parallel linear algebraic computation support. We also provide advices for further performance improvement.

II. Related Work

A. Scientific Big Data Benchmarks

Building a scientific big data benchmark is a key issue to evaluate big data systems. Most of previous work focuses on a specific research area. However, as a benchmark suite, it should include diverse and representative workloads and data sets.

SS-DB [6] is a scientific benchmark proposed by MIT, which simulates an astronomical data management scenario, and provides queries including raw data cooking and observation data analysis. The main data structure is a mutlidimensional array.

GenBase [3] is another scientific analysis benchmark proposed by MIT, which simulates genomics research. The benchmark contains five queries, including linear regression, covariance calculation, bi-cluster calculation, singular value decomposition and statistical test. Each query [3] represent a mix of data management, linear algebra and statistical operations that are representative genomic workloads. Supported platforms include R language on a single machine, SciDB and database management systems combined with R language.

Our BigDataBench-S is different from the above two benchmark suites. First, BigDataBench-S contains diverse data sets and workloads in a variety of typical scientific fields. Second, we include two classes of workloads: query processing and complex analysis, rather than mixed workloads presented in GenBase.

B. Big Data Management and Analysis Systems Evaluation

There have been a lot of research efforts in evaluating big data systems. In [7], data management workloads including data import, regular queries and UDF queries are selected to compare the performance between traditional parallel database systems and Hadoop. The experiment results show that Hadoop performs better for data load but cannot achieve better performance on regular and UDF queries, for the reason of lacking enough strategies on query optimization.

In [8], TPC-H and TPC-DS are used to evaluate Hive and Impala. This work considers the performance differences when using different types of file formats, which is also considered in this paper. However, these researches only consider query operations of traditional data management systems without including different workloads of diverse scientific scenarios.

Wang et al. [9] propose a big data benchmark suite from internet services, and it does not consider scientific scenarios.

III. Introduction of General Purpose Big Data Analysis System

In this section, we give a brief introduction of the general purpose big data systems mentioned in this paper.

A. Hadoop

Apache Hadoop [10] is an open source implementation of Google GFS and MapReduce, which includes a distributed file system called HDFS and a distributed batch-oriented computation engine called Hadoop MapReduce. In HDFS, files are spilted into blocks and distributed to machines in the cluster. Each file block may have multiple replications on different machines for fault tolerance. The MapReduce engine is responsible for the distribution and execution of batch processing jobs. The engine simplifies the development of parallel and distributed programs via providing a simple programming model to programmers, while hiding the low level details of task scheduling, parallelization, inter-machine communcation and fault-tolerance. A typical MapReduce job is made up of two operations called map and reduce. The map operation generates key value pairs according to a user-defined map function in parallel, and then all the values belonging to one key are sent to the same reduce task for later reduce operation.

To achieve good scalability, all the intermediate data generated by the map functions are first written to the local disks before transmission. Moreover, the results of each MapReduce job are written to the HDFS. If an application is represented as multiple MapReduce jobs running sequentially, and the data sharing between two consecutive jobs have to rely on HDFS. Those facts make MapReduce very inefficient to run several complex or latency sensitive workloads, such as iterative machine learning and interactive database query. Tez [11] is another execution engine where jobs are represented in the form of directed acyclic graphs (DAG), in order to eliminate the extra overhead of data sharing between different MapReduce jobs, especially map-only jobs.

B. Spark

Spark [12] is another distributed big data analytics system developed by UC Berkeley AMPLab, which originally targets at iterative machine learning algorithms. The main data and programming abstraction is Resilient Distributed DataSets (RDD). Unlike MapReduce, Spark provides developers with a variety of computation primitives, such as group by, join and filter. Primitives are categorized into two classes---transmission and action. Computation jobs are also organized in the form of DAGs, where each vertex represents a transmission operation, and each edge represents the data exchange. A job will not be submitted to the engine for real execution until an active operation is called. After a job is submitted, it is separated into different stages according to whether a data shuffle is needed or not, and tasks in each stage are executed in parallel. The data exchange between operators within one stage is directly through memory.

The DAG job structure and RDD based data cache mechanism help Spark eliminate unnecessary disk IOs between operators, which makes it suitable for iterative machine learning algorithms and other workloads that need multiple computing rounds on the same data sets. On the other side, since Spark relies heavily on the memory resources, the computation efficiency can be impacted if the garbage collection happens frequently.

C. Other Related Systems

Hive [13] is a data warehouse query processing component atop on Hadoop. It includes a SQL-like query language called HiveQL, a query translation layer, and a query optimization layer. In the earlier versions of Hive, each HiveQL query was translated into a MapReduce job sequence for execution. Now Hive has also added support for Tez and Spark.

Mahout [14] is a distributed machine learning algorithm library on top of Hadoop, which involves classification, clustering, matrix decomposition and other commonly used algorithms. Mahout also provides several basic linear algebraic operations. The current Mahout version also contains the support for Spark and Flink.

Spark also has its own data warehouse query processing layer and machine learning library, named SparkSQL [15] and MLlib [16], respectively. SparkSQL supports all the HiveSQL statements and provides a scalable query optimization layer for users to customize optimization strategy based on the specific scenarios.

MLlib provides many machine learning algorithms implemented on Spark. Unlike the unitary matrix data structure in Mahout, MLlib provides varieties of distributed matrix data structures, including row based matrix, block matrix and indexed row matrix, to facilitate matrix computation on different scale of problems.

The efficiency of data storage and access is also a non-trivial factor for computation performance. Parquet and ORCFile are two widely used storage formats that address this problem. Both formats use a column-based way to organize data, and a series of serialization and compression strategies to reduce the cost of data storage. Column-based indexes are also considered for efficient data filter and access. In addition, Parquet provides particular support for nested data, while ORCFile provides a specific strategy to serialize data of integer and string type.

IV. BIGDATABENCH-S

BigDataBench-S is a scientific big data benchmark suite, which extends from a mature big data benchmark suite called BigDataBench [9] and adds support for typical workloads in scientific fields. In this section, we first present the methodology of BigDataBench-S, and then give a brief introduction of the scientific research domains currently involved in it.

For workloads selection, we choose several important scientific fields and select typical workloads from these fields as shown in Fig. 1. The current version of BigDataBench-S involves three scientific fields: high energy physics, astronomy and genomics.

In high energy physics, the raw data are derived from many detectors of Large Hadron Collider (LHC). The data size can reach up to 3PB each year [17]. Through analyzing the event data collected by detectors, researchers can find experimental proof for physical theories or explore unknown physical phenomena. Each event record contains multi-variant values associated with the event, which is stored in the ROOT [18] format. Typical high energy physics data analysis process includes several steps: event data collection, experiment-related event filter, specific experimental event analysis, and final result extraction. LHC's detectors are responsible for the event collection, and the data filter process needs to separate specific events from others for later analysis. The data size always can be reduced down to the extent that a single machine can handle after filtering. To achieve efficient and accurate event discrimination (i.e. to separate events needed for one specific research from others), a lot of rule-based event selection and classification operations need to be proceeded in this phase, which is the most complex phase in the whole pipeline [19]. Therefore, we mainly consider this phase when we build the high energy physics benchmark.

Events associated with a specific physical experiment are called the signal events of the experiment (Signal), and the other unrelated events are called background events (Background). During the data filter process, researchers first give simple filter conditions, to select event data using database query operations, then use complex analysis algorithms to identify specific signal events. Algorithms involve typical operations of classification and regression, including Maximum Likelihood Estimation, Function Discriminant

Analysis, Boosted Decision Trees, Support Vector Machine, and so on.

Scientists from astronomy try to explore the universe structure and understand how the universe and different celestial bodies evolve. The raw data mainly come from observation devices such as the Large Synoptic Survey Telescope (LSST). The collected data are images at different time periods. For each period, the images represent the observations of universe at this moment. Each image is stored in the form of two-dimensional array. Furthermore, a complete image is composed of many small images from a number of sensors, where each small image records the information of every pixel also in the form of two-dimensional array [6].

In a typical astronomical image analysis scenario, raw images are first cooked into a lot of observations, which then are processed through several analysis operations such as classification and trajectory analysis.

The goal of genomics is to study the genomes of lives, analyze functions and think about how to use them. Genomic research relies on large amounts of genomic data. The size of gene data collected each year are more than 15 PB [20]. Besides, as the cost of gene collection technology continue to decrease, the amount of gene sequence data will grow at an even faster rate.

The DNA microarrays can simultaneously measure the expression data of thousands of genes in a single experiment, and is one of the major techniques used in current genome research [3]. The collection of microarray data relies on a gene chip that contains a large number of complementary deoxyribonucleic acid (cDNA) or oligonucleotide probes. The expression data of the gene is obtained by the technique of fluorescein labeling, which can reflect the activity information of each gene and provide data for studying the physiological state of the cell.

In practical research scenarios, such as the correspondence between diseases and genes, biologists collect thousands of gene samples from a large number of patients, which result in a large-scale dense floating-point matrix with gene as column and patient sample as row. Biologists can perform data selection queries, statistics, complex analysis and other operations on this matrix structure data.

Based on the above analysis of data model and typical applications in each scientific field, we construct BigDataBench-S. The chosen data sets include the ATLAS data set from high-energy physics [21], astronomical and genomic simulation data sets using SS-DB and GenBase data generators. The workloads contain 17 typical operations in these three scientific fields. The Intersection operation from astronomy is a common operation for trajectory analysis. Sigma-Clipping [22] is a common image processing operation in astronomy. Each workload of BigDataBench-S is either data operation query or complex analysis, which is different from GenBase and other existing scientific benchmarks. This design decision is convenient for apple-to-apple comparison among components of different big data systems. The detail of BigDataBench-S is shown in Table I.

V. Data Sets and Workloads For Evaluation

We choose genomics as the scenario for our evaluation as a case study, and describe data structure and workloads in details in this section.

A. Data Structure

The data set is from GenBase's data generator, which contains four tables representing microarray data, patient metadata, gene metadata and gene ontology information, respectively.

The microarray table records the expression value of each gene (also called gene expression omnibus, in short, GEO) in different patient samples. Each patient is assumed to collect only one gene sample. The microarray data is organized in a matrix structure where columns represent genes and rows represent patient samples. Let the matrix be M and the matrix element from row i and column j be M_{ij}, which represents the expression value of gene j in the patient sample i. The data are stored in a file in the form of *(row, column, expression value)*.

The patient metadata table (Patients) records the basic information of each patient, including age, gender, and drug response.

The gene metadata table (Genes) records the information of each gene, including the corresponding target gene (via protein), chromosome number, function information and so on.

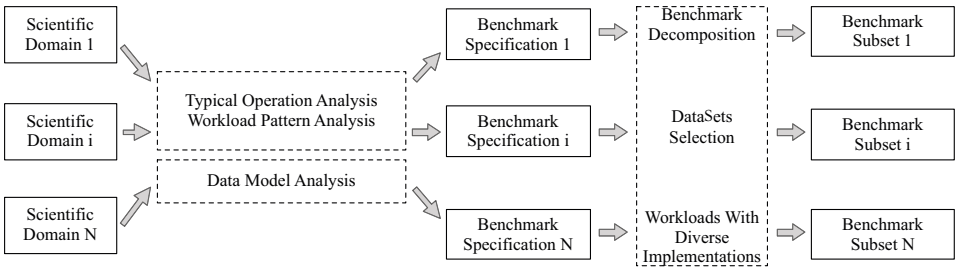


Fig. 1 BigDataBench-S Methodology, which is also used in BigDataBench [9]

Table I. Datasets and Workloads in BigDataBench-S

Domains	Datasets	Workloads	
High Energy Physics	ATLAS Dataset [21]	**Data Manipulation Queries**	**Selection**: select events based on filter conditions
		Classification	SVM
			k-Nearest Neighbor
			LDA
		Regression	Boosted decision trees
			Maximum likelihood fit
Astronomy	Simulated Dataset Using Generator from SS-DB[6]	**Data Manipulation Queries**	**Selection**: Select images in given time and space ranges
			Aggregation: Compute average value of cells of images to find average background noise
			Join: Join each image cell using the average value of each cell on the same position to perform co-addition image process
		Complex Analysis	Intersection of images
			Sigma-Clipping
Genomics	Simulated Dataset Using Generator from GenBase[3]	**Data Manipulation Queries**	**Selection**: Select genes based on filter conditions
			Aggregation: Compute average expression value of genes
			Join: Join genes with gene ontologies
		Complex Analysis	QR decomposition
			SVD
			Covariance

The gene ontology table (GO) records the biological function of each gene, which means what role a gene will play in a biological function. For example, Gene1 may be related to respiratory function, and Gene2 and Gene3 may be associated with a certain cancer. The specific information of the table is shown below:

```
CREATE TABLE geo(
   geneid INT,
   patientid INT,
   expr_value FLOAT);

CREATE TABLE go(
   geneid INT,
   goid INT,
   belongs INT);

CREATE TABLE genes(
   geneid INT,
   target INT,
   pos BIGINT,
   len INT,
   func INT);

CREATE TABLE patients(
   patientid INT,
   age INT,
   gender INT,
   zipcode INT,
   disease INT,
   response FLOAT);
```

In order to test the scalability of each system under different data scales, three data sizes are selected for testing, namely, small, medium and large. The dimensions of the gene microarray data are 5000*5000, 12000*20000, 30000*40000, respectively, and the other table scales are adjusted accordingly.

B. Selected Workloads

As described previously, each query workload provided by in GenBase is a mix, which firstly perform database query to select data satisfying some specific rules, and then perform complex analysis on preprocessed data. However, for big data analysis systems such as Hadoop and Spark, queries and complex analysis computations are handled by different subsystems. For example, the query processing layer of Hadoop is Hive, and complex machine learning and matrix operations are included in Mahout. Users and system designers not only care about the efficiency of the overall data processing system, but also the performance of each subsystem. Performing apple-to-apple comparison can provide valuable information to help them select more appropriate components and find the bottlenecks, which is also one of our main purposes to propose this benchmark. In BigDataBench-S, we separate each typical query in GenBase into several workloads in BigDataBench-S, which is either database queries or complex analysis. The differences of selected workloads between BigDataBench-S and GenBase are shown in Table II.

Table II. Comparison between GenBase and BigDataBench-S Genomics

	GenBase	BigDataBench-S
Workload Category	Mix	Either data queries or complex analytics
Workload Number	5 mixed workloads	3 data manipulation queries and 3 complex analytics workloads
Supported Systems	Traditional row and column stores + R/Madlib, Hadoop (Partial support), SciDB	Large scale distributed systems, including Hadoop(MapReduce and Tez) and Spark

C. Queries

1. Selection

Selection is the most common query operation. In genomics scenario, researchers often use selection with some filter conditions to get a much smaller data set for further analysis. The selection query used in this paper is shown as below:

```
SELECT geo.* FROM genes
JOIN geo ON (geo.geneid=genes.geneid) WHERE
genes.func < X;
```

The selection query requires accessing data in two tables. A subset of data is first selected from the genes table on a given condition, and then this data subset is used to filter data from the geo table whose geneid equals to that of the genes subset. Such query requires a join operation to perform the real data filter operation, but the size of genes subset is always very small. Consequently, during the real execution of this query, all

evaluated systems choose firstly broadcasting the genes subset to all the data nodes containing geo data, and then perform local join operation. Consequently, the cost of join is much smaller than that of the selection operation of geo data. The main purpose of this query is to evaluate how well a system performs table scan operation.

2. Aggregation

Biologists often need to perform statistical operations on gene data, for example, to get the mean value of expression value. Such operations always can be expressed as an aggregation query, which contains a group by operation and a specific aggregation operation. The query is as follows:

```
SELECT geneid, avg(expr_value) as avg_expr_value
FROM geo
GROUP BY geneid;
```

The result of this query is an array of key-value pairs with geneid as the key and the average expression value as the value. The execution consists of a table scan and a data merge operation. The main purpose of this query is to evaluate the performance of systems on aggregation- like operations.

3. Join

When a biologist performs biological function related research, each gene data needs to be correlated with its ontology data---requiring a join operation between the geo and go tables. The query is as follows:

```
SELECT go.goid AS go_col, go.pid AS pid,
go.belongs AS cat, gp.ev AS val
FROM
(SELECT g.geneid AS gid,
         p.patientid AS pid,
         g.expr_value AS ev
  FROM geo g, patients p
  WHERE p.patientid < 5
  AND g.patientid = p.patientid
) gp, go
WHERE go.geneid = gp.gid;
```

The join operation is complex, which requires a lot of data transmission during the execution. This query can be used to test how efficient the system performs from perspectives of communication and job parallelism.

D. Complex Analysis Workloads

For complex analysis, we choose three kinds of calculations as follows: QR decomposition, SVD decomposition, and matrix covariance calculation.

1. QR Decomposition

QR decomposition is a common matrix decomposition, which can be used for linear regression, eigenvalue calculation, and minimum dichotomy. The QR decomposition of the m*n matrix M is shown below:

$$M = QR$$

Where Q is an orthogonal matrix of m*n and R is an upper triangular matrix of n*n.

In the genomics scenario, QR decomposition can be used to solve a linear regression problem of predicting the pharmacological response of a patient.

2. SVD

During the studies of disease correlation analysis, it is always necessary to eliminate the interference data existed in the raw data, or perform dimension reduction to obtain key semantics. Such kinds of operations rely on the Singular Value Decomposition (SVD). The SVD of the matrix M is calculated as follows:

$$M = U \Sigma V$$

Where Σ is a diagonal matrix containing singular values, U and V are left and right singular vectors

3. Covariance Calculation

Covariance calculation is used to analyze the relevance of multidimensional data. In genomic studies, it is often used to discover the biological relevance of different genes.

VI. Experiments and Analysis

We use a 10-node cluster with Huawei RH2285 server. Each node is equipped with two Intel Xeon E5645 processors, including 12 physical cores. The memory is 32GB, and the disk size is 1TB.

The version of Hadoop and Hive is 2.7.1, 2.0.0, respectively. The version of Spark and Tez is 2.0.1, 0.8.3, respectively. Every software stack uses optimal configurations according to specific workloads. Among numerous parameters, the degree of parallelism is a key factor that influences processing performance. However, when testing query statements, the system will automatically adjust the number of parallel tasks according to workloads, thus we don't set the degree of parallelism manually. We evaluate every workload three times and report the average value.

A. Evaluation and Analysis of Query Workloads

We mainly evaluate three systems including SparkSQL, HiveOnMR and HiveOnTez. The data storage formats are Text, Parquet and ORCFile. We evaluate three systems from the perspectives of file format compression efficiency, query performance, and scalability under different data scales.

1. File Format Compression Efficiency

We first compare the storage space using different data formats, which reflects serialization and compression efficiency of data formats. Both Parquet and ORCFile provide optimization strategies for serialization and compression, so they save more storage space compared to the Text format.

Fig. 2 lists the data size of geo and go tables using three file formats. We find that Parquet and ORCFile can significantly reduce the storage size of two tables. Especially, ORCFile can reduce the data size of go table 40 times. The data in go table are all integer data, and ORCFile provides special serialization method for the integer data type.

Summary: using column-based data format reduces storage space notably. The specialized serialization methods for scientific data types can further reduce data storage space.

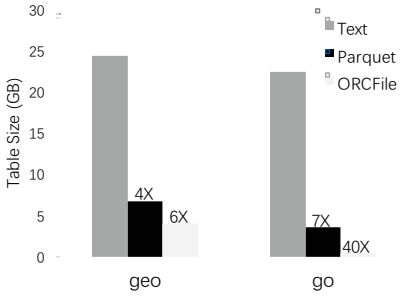


Fig. 2 Data size using different formats

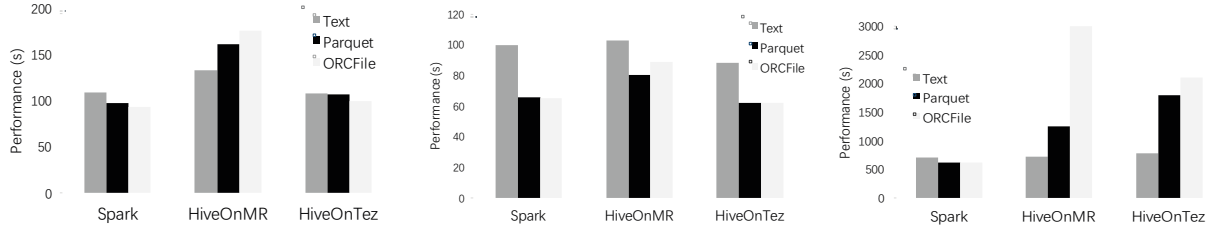
2. The Effect of File Format on Query Performance

We further compare the impacts of data formats on query performance, and evaluate whether using various column-based data format can improve the query efficiency in science fields.

We choose data size of 30000 x 40000. Fig. 3 presents query processing performance using different formats. For selection query, Spark and HiveOnTez have few performance gains using the Parquet and ORCFile formats, while HiveOnMR gets worse performance. Through analyzing the execution logs, we find that the main reason for the poor performance of HiveOnMR is the reduction of parallelism degree. The parallelism degrees of selection query using Text, Parquet and ORCFile are 98, 25 and 17, respectively. However, the data scales which need to be processed by each task gradually increase. For example, the records processed by every Map task is 13.31 million when using Text format, while the number is 51.36 and 73.52 million for the Parquet and ORCFile formats, respectively. In addition to the variation of execution time for Map tasks, the reduction of parallelism degree and the increase of data scale for each Map task indeed influence the processing performance of selection query on HiveOnMR.

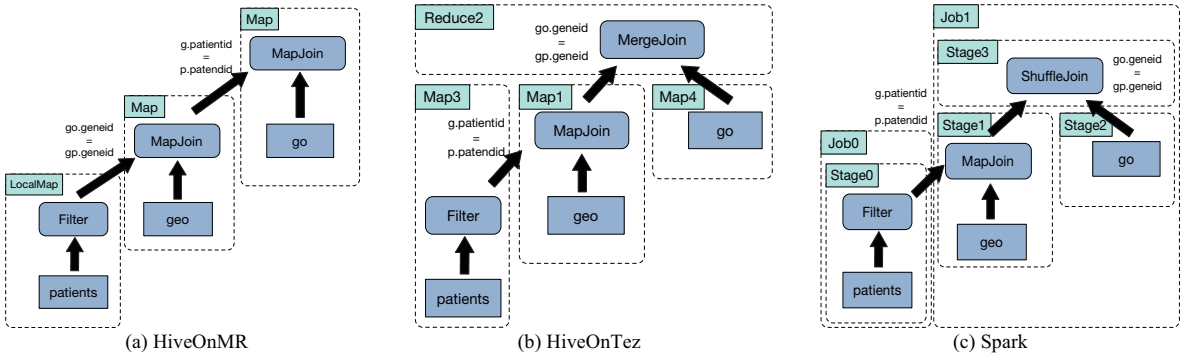
The negative impact of increased data scale processed by Map tasks on processing performance reflects that the column-based storage formats provided by Parquet and ORCFile don't filter data effectively. The reason is that the filter information of selection query is from genes table, which need join operations to filter data, rather than proper filter conditions for column-based storage format. This is also the main reason why Spark and HiveOnTez don't benefit from column-based storage format.

For aggregation query, all three systems can gain performance improvement using the Parquet and ORCFile formats. The aggregation query execution includes two stages. The first stage reads data from geo table sequentially, and generates key-value pairs based on geneid. Local aggregation is performed, such as average calculation. For the second stage, the data are transferred to different nodes according to geneid and then complete the final aggregation. Consequently, the aggregation query performance largely depends on data shuffle process. When using Parquet and ORCFile formats, because of low read parallelism, local data can be fully aggregated, and further reduce the data amount which need to be transferred. From the log information, we find that the data transfer time using the Parquet format reduces about 5 times than using the ORCFile format.



(a) Selection (b) Aggregation (c) Join

Fig. 3 Performance comparison of query processing using different formats



(a) HiveOnMR (b) HiveOnTez (c) Spark

Fig. 4 Execution plan of join query in different systems

In order to analyze the root cause of inefficiency in the join query on HiveOnMR and HiveOnTez, we first describe the execution plans of join queries on all systems, as shown in Fig. 4. HiveOnMR translates join query into three MapReduce jobs which only have Map tasks. The first job is responsible for processing patients table on the local node. The second job joins geo and patients tables, and then sends the join result to the node containing go table for Map-side join operation. The data transmission among different nodes uses HDFS, and the filtered geo data are written to HDFS. The third job reads the filtered geo data on HDFS and joins them with go table. The third job accounts for a large proportion since every Map task need to read go table and execute join query concurrently. Through analyzing execution logs, we find that the proportion of execution time of the third job is 88.2%, 95.4% and 98.8%, respectively, using three data formats. Meanwhile, the Map parallelism degree during the third job is 91, 14 and 4, respectively. Therefore, the inefficiency of HiveOnMR using the Parquet and ORCFile formats is due to the load imbalance caused by too low task parallelism, which largely relies on optimization strategy and task partition mechanism.

The execution plans generated by HiveOnTez and Spark on join query statement are similar. First, they execute the Map Join operation of patients and geo tables, to obtain the intermediate data using the key of geneid, handle go table to get the intermediate data using the key of geneid similarly, and then re-combine the data according to geneid for distributed join operations. So there are two phases when jobs run: (1) read from tables, and (2) distributed join operations (Compared with the overall overhead, the overhead of patients and geo's Map join can be ignored).

In order to analyze the performance difference between Spark and HiveOnTez, we compare the overhead of the above two phases, as shown in Fig.5. We see that, the main performance overhead of Spark occurs in the join operations phase, and the overhead of table read operations is very low, when using the three formats. Meanwhile, the table read overhead of HiveOnTez is longer than Spark, and especially the table read of ORCFile accounts for more than 70%. By checking the monitoring logs, we find that read and write on go table consume a lot of time in HiveOnTez, and the parallelism degree of this task is 7. This is much smaller than that in the case of the Text and Parquet formats, which is 188 and 90, respectively. In addition, because HiveOnTez needs local sort when dealing with Map tasks, which is same like that in MapReduce, the pressure of a single Map task further increases. In summary, the low parallelism degree of table read operations is still the main reason for that HiveOnTez runs slowly when using the ORCFile format.

On the other hand, when dealing with the Parquet format, the join operation parallelism degree of Reduce job in Tez is only 14, in spite of the table-read parallelism is high. Because join operations don't have a large number of local message merging such as in aggregation operations, and few reduce jobs receive large amounts of messages, which results in high overhead and poor join performance.

Summary: When running the above three queries, three big data processing systems fail to achieve significant performance improvements using the column storage, and even have worse performance in some cases. It is the side effects of imperfect index design of column storage, systems' query optimization, and cost estimation mechanism. On one hand, data filtering for column storage fails to consider filtering through join operations of other tables; On the other hand, the parallelism degree of HiveOnMR and HiveOnTez is adjusted only with data size, and fail to consider the computation overhead of a single Map task, which leads to workload imbalance. In contrast, when handling queries, Spark does not relate the parallelism degree to the data size, thus the parallelism degree does not decrease as the data size become smaller. Spark also gets performance boost when using the column storage format. However, Spark's optimization mechanism is also inadequate. For example, the parallelism degree of join operations needs to be set manually, and there are no automatic matching strategies based on load characteristics.

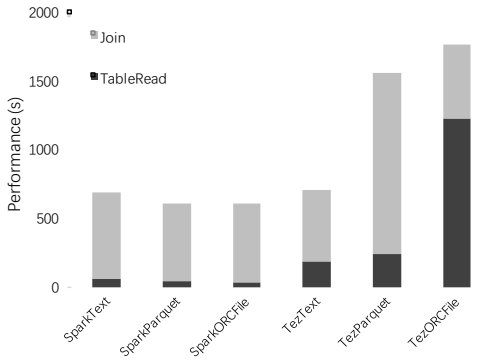


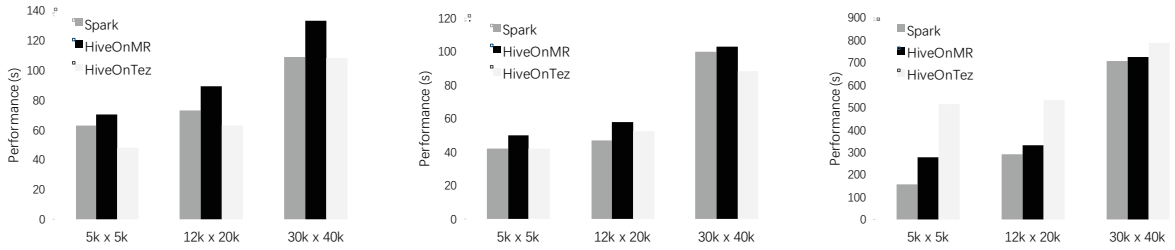
Fig. 5 Run-time distribution of join queries using different formats in Spark and Tez

3. Comparisons of System Query Processing Performance

We further compare the performance of different systems, as shown in Fig.6, 7, 8. For simple selection operations, the performance of HiveOnMR is worst when using the three formats. The main reason is that HiveOnMR will translate the queries into MapReduce tasks, which are executed by the MapReduce engine. With respect to Spark and Tez, starting MapReduce tasks takes a longer time because of the heartbeat mechanism for task allocation. In addition, Map tasks execute sort operations before output, which results in unnecessary overhead. Aggregation operations are similar to selection ones, and the performance of the three systems is close for small data sets. In other cases, the performance of HiveOnMR is worse than the others.

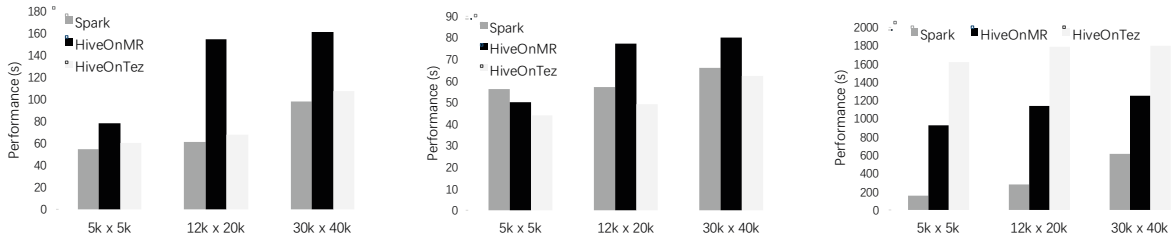
The performance difference of join operations varies widely. The performance of Spark is significantly better than the other two systems, because in Spark, there are faster table reading operations and better concurrency of join operations. When reading from tables, Spark reads geo and go tables at the same time, while HiveOnMR and HiveOnTez both need to read two tables serially. In addition, the join operations of Spark have high parallelism, which helps achieve calculation balance.

The join query performance of HiveOnTez with the Parquet format is worse than that of HiveOnMR. The main reason is that HiveOnTez starts few Reduce processes when executing join operations, and hence few nodes are used to process large amounts of data, which reduces communication overhead.



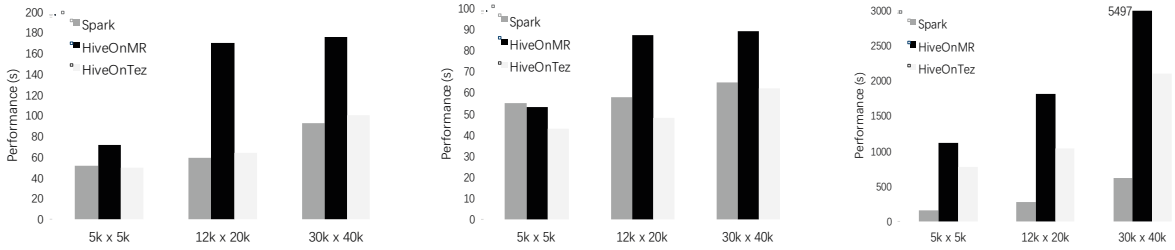
(a) Selection (b) Aggregation (c) Join

Fig. 6 Performance comparison of query processing in three systems with the text format



(a) Selection (b) Aggregation (c) Join

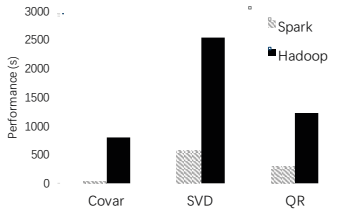
Fig. 7 Performance comparison of query processing in three systems with the Parquet format



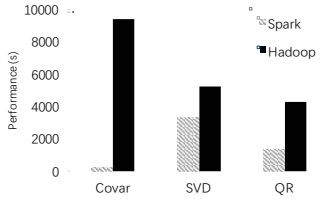
(a) Selection (b) Aggregation (c) Join

Fig. 8 Performance comparison of query processing in three systems using the ORCFile format

Summary: The efficiency of selection and aggregation queries on HiveOnMR is affected by longer time of job's start-up with respect to the other systems. The low concurrency of Map and Reduce tasks results in the inefficiency of HiveOnTez and HiveOnMR when executing complex join queries.



(a) 5k x 5k case



(b) 12k x 20k case

Fig. 9 Performance comparison of complex analytics in Spark and Hadoop

B. Evaluation and Analysis of Complex Analytical WorkLoads

In this section, we compare the performance of Hadoop and Spark for complex analytics. The load is rewritten according to Mahout and MLlib's library functions, matrix structure and interface. We still consider three data sizes. We find that these two systems cannot complete within the acceptable time when using large-scale data. Therefore, we use two smaller data size (5000*5000 and 12000*20000) for comparisons, and the performance data is shown in Fig.9.

The performance of Spark's covariance calculation is much better than that of Hadoop. The reason is that the main part of covariance calculation is matrix multiplication, and Spark's MLlib provides a block-based matrix distributed structure for matrix multiplication. Specifically, the matrix will be divided into multiple matrix blocks. So calculation has better parallelism and load is much more balanced. Mahout (Hadoop) provides only row-based matrix partitioning method, so it is difficult to achieve good parallelism for complex matrix calculations.

Spark's distributed block matrix structure only provides matrix multiplication and other basic operations, and much complex matrix calculations, such as SVD, QR are implemented with the row-based distributed matrix structure,

which will result in serious workload imbalance. In addition, during QR's execution process, we find that most calculations are performed in just few nodes and consume huge memory. The memory reclamation on these nodes also results in significant performance loss.

VII. Discussion And Suggestions

Through above experiments, we find that the current big data systems have many drawbacks in performing scientific analysis workloads, which can be summarized as follows:

1) Adopting column-based storage format does not help achieve good performance on data filter and table scan. The main reason is that the data structure used in genomics is actually a matrix structure, which requires both the row and column information for data filter.

2) When adopting column-based storage format, the parallelism degree of the final execution plan is very low in HiveOnMR and HiveOnTez, which results in load imbalance and slow execution. The reason is that the optimization layer of Hive only considers the total data size when deciding the parallelism degree of final execution plan, without considering of the complexity of query. This indicates that the cost model of these query processing layers are still simple. The query execution complexity should be considered in addition to the scale of data. Currently, the parallelism degree of join operation in SparkSQL is a fixed value set by the user, needing a better optimization strategy.

3) Both Spark and Hadoop currently do not provide an efficient algorithm library for complex linear algebra and matrix computation, which significantly reduce the applicability of these systems in scientific fields.

We believe that there are a lot of room for improvement in the field of scientific research for the current big data management and analytics systems: 1) an efficient storage structure for varieties of data structures in the scientific field, which not only provides a more efficient index structure for row and column data, but also facilitates data filtering. 2) More complex and efficient query optimization strategy, considering both the data size and computation complexity. 3) A rich linear algebra library that supports complex matrix partitioning strategies and provides corresponding parallel operations.

VIII. Conclusion

In this paper, we first present BigDataBench-S---a scientific big data benchmark suite. The current version of BigDataBench-S includes data sets and corresponding data manipulation queries and complex analytical workloads from three different scientific areas: high energy physics, astronomy, and genomics. We will open-source BigDataBench-S and include other typical data sets and workloads from other areas soon. Also, we use BigDataBench-S to evaluate general-purpose big data management and analytics systems. The results indicate that the current big data systems do not provide enough support for typical scientific data analysis workloads, and there is a lot of space for improvement from perspectives of storage format, query optimization, matrix computation parallelism and so on.

IX. Acknowledgements

This work is supported by the National Key Research and Development Plan of China (Grant No. 2016YFB1000600 and 2016YFB1000601).

References

[1] G. Farmelo, Smashing Physics review – how the Higgs particle was found[M]. THE GUARDIAN/Science, May 2, 2014.

[2] A. Wright, Big Data Meets Big Science. Communications of the ACM , Vol. 57, No. 7, July 2014.

[3] E. Taft, M. Vartak, N. R. Satish, N. Sundaram, S. Madden, and M. Stonebraker, GenBase: A Complex Analytics Genomics Benchmark. SIGMOD '14, 177-188.

[4] Parquet. http://parquet.io/

[5] Y. Huai, C. Ashutosh, G. Alan, H. Gunther, N. H. Eric, O. M. Owen, P. Jitendra, Y. Yuan, L. Rubao, and X. Zhang. Major Technical Advancements in Apache Hive. SIGMOD, 2014, 1235-1246.

[6] P. Cudre-mauroux, H. Kimura, KT. Lim, J. Rogers, S. Madden, M. Stonebraker, S. B. Zdonik, et al. SS-DB: A Standard Science DBMS Benchmark. http://www-conf.slac.stanford.edu/xldb10/docs/SSDB_benchmark.pdf

[7] A. Pavlo, E. Paulson, A. Rasin, et al. A comparison of approaches to large-scale data analysis. ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, Usa, June 29 - July. 2009:165--178.

[8] A. Floratou, U. F. Minhas, Özcan, Fatma, SQL-on-Hadoop: full circle back to shared-nothing database architectures. Proceedings of the Vldb Endowment, 2014, 7(12):1295-1306.

[9] L. Wang, S. Zhang, C. Zheng, et al. BigDataBench: A big data benchmark suite from internet services. International Symposium on High PERFORMANCE Computer Architecture. IEEE, 2014:488-499.

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

[11] Apache Tez. http://tez.apache.org/

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, et al, Spark: cluster computing with working sets. Usenix Conference on Hot Topics in Cloud Computing. USENIX Association, 2010:1765-1773.

[13] A. Thusoo, J. S. Sarma, N. Jain, et al. Hive: a warehousing solution over a map-reduce framework. Proceedings of the Vldb Endowment, 2009, 2(2):1626-1629.

[14] Apache Mahout. http://mahout.apache.org/

[15] M. Armbrust, R. Xin, C. Lian, et al. Spark SQL: Relational Data Processing in Spark. ACM SIGMOD International Conference on Management of Data. ACM, 2015.

[16] R. B. Zadeh, X. Meng, B. Yavuz, et al. MLlib: Machine Learning in Apache Spark. Computer Science, 2015.

[17] D. Malon, J. Cranshaw, P. van Gemmeren, Q. Zhang, Emerging Database Technologies and Their Applicability to High Energy Physics: A First Look at SciDB. Journal of Physics: Conference Series 2011 (Vol. 331, No. 4, p. 042016).

[18] R. Brun, F. Rademakers, ROOT—an object oriented data analysis framework. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 1997 Apr 11;389(1):81-6.

[19] A. Lazzaro, Algorithm Challenges in the LHC data analysis..https://indico.cern.ch/event/130322/contributions/119655/attachments/91993/131514/lazzaro.pdf

[20] M. C. Schatz, B. Langmead B, The DNA data deluge. IEEE Spectrum, 2013, 50(7):28-33.

[21] Open Data ATLAS. http://opendata.atlas.cern/

[22] M. Moyers, E. Soroush, SC. Wallace, et al. A demonstration of iterative parallel array processing in support of telescope image analysis. Proceedings of the VLDB Endowment. 2013 Aug 28;6(12):1322-5.