Performance Evaluation between Hive on MapReduce and Spark SQL with BigBench and PAT

Van-Quyet Nguyen, Kyungbaek Kim
Dept. Electronics and Computer Engineering, Chonnam National University
E-Mail: quyetict@utehy.edu.vn, kyungbaekkim@jnu.ac.kr

Abstract

Big data systems have been proposed to address the challenges of big data such as collecting, storing, and analyzing data. Recently, Hive has been the most popular data warehouse for the big data systems by supporting HiveQL, which is compiled into MapReduce jobs executed on Hadoop; meanwhile, Spark SQL has emerged as a leading big data framework by using in-memory based distributed computing. There are several studies have been performed to evaluate these two frameworks and showed that in most cases Spark SQL is faster than Hive on MapReduce, but in some cases related to joining large tables Spark SQL is slower than Hive on MapRedue. Recently the latest version of Spark SQL has many improvements which can provide better performance of handling SQL queries such as catalyst optimizer. In this paper, we present the new results of performance evaluation between Hive on MapReduce and the recent Spark SQL on our big data system by using a benchmarking tool, called BigBench, and performance analysis tool (PAT). Our experiments illustrate that the recent Spark SQL outperforms Hive on MapReduce with all of 30 BigBench queries. Moreover, we observed that Spark SQL consumes less network traffic and keeps higher utilization of memory usages than Hive on MapReduce.

1. Introduction

Big data systems have been designed in order to deal with a huge generated data from a variety of aspects in our life such as network services, agriculture development, and scientific research areas [1]. These systems face challenges of collecting, storing, and analyzing big data. During the last decade, Hadoop [2] has been the most popular framework for big data processing. It provides a parallel computation model MapReduce [3] and Hadoop distributed file system (HDFS) module. However, the MapReduce programming model requires developers to write custom programs which are hard to be maintained and reused. To simplify storing and accessing big data, Hive [4] is proposed to support queries expressed in a SQL-like declarative language, HiveQL, which is compiled into MapReduce jobs executed on Hadoop.

Recently, another big data framework, Spark [5] has emerged as a leading distributed computing framework for real-time analytics with memory-oriented architecture and flexible processing libraries. In which, Spark SQL is a component on top of Spark Core for processing structured data. It reuses the Hive frontend and metastore giving full compatibility with existing Hive queries. In our work, we investigate in evaluation of the two big data engines' performance: Hive on MapReduce and Spark SQL.

The characteristics of big data (i.e. volume, velocity, variety and veracity) not only made the design and implementation of big data systems to be the complex but also was difficult in evaluating these systems. Therefore, big data benchmarks have been developed to evaluate and compare the performance of big data systems and engines [6]. BigBench [7] was proposed as the first end-to-end benchmark for big data offline analytics. It supports evaluating both Hive on MapReduce and Spark SQL. *Ivanov et al*. [8] has evaluated Hive and Spark SQL version 1.4 with BigBench which the 8 queries from group of 14 pure HiveQL queries run faster on Spark SQL than Hive on MapReduce, while other queries were executed slower because of joining issue.

Currently, the last stable version of Spark is 2.1 which has many improvements in the Catalyst optimizer for common workloads in of Spark SQL. Spark 2.X, which used for common operators in SQL; and DataFrames via a new technique called whole stage code generation, has a substantial 2 to 10 times performance speedups comparing to Spark 1.X. In this paper, we conduct performance

evaluation of Hive on MapReduce and Spark SQL 2.1 by using BigBench. To understand better about the differences between these two engines, we use a performance analysis tool called PAT to capture the resource utilization metrics.

2. BigBench and PAT

2.1 BigBench

BigBench is developed to evaluate and compare the performance of big data systems. It consists of two major components: a data model for synthetic data generator and a collection of workload queries.

There are 3 categorized parts of the data model: structured, semi-structured, and unstructured one. To generate the data, an existing Parallel Data Generation Framework (PDGF) is used with a large range of scale factor. It can scale the data to large volumes by a scale factor from 1 to 1000000 (1 scale factor equals 1 GB).

The workloads are major parts of BigBench which include 30 business queries. These were made underlying business questions relating online shopping areas, such as marketing; merchandising; and supply chain. The workloads were made to be sure that different algorithms (classification, clustering, etc.), which use three different data types, had been implemented, and such queries can be evaluated by Hive on MapReduce or Spark SQL.

2.2 Performance Analysis Tool (PAT)

Although BigBench supports the performance evaluation of big data systems, but it only provides the response time of each query after evaluation is finished. To support measuring resource utilization metrics of the big data system, we used PAT running simultaneously with BigBench during the time of evaluating the queries. For each query executed on MapReduce/Hive and Spark SQL, CPU utilization; disk input/output; memory utilization and network input/output are provided.

PAT is enabled to deploy on a master/slave system. It consists of two main modules: collecting data and post-processing. For the former one, PAT master performs installing scripts to slaves and start measuring the resource utilization on each slave. Next, PAT master executes a command which contains a script to run BigBench queries. Finally, the resource utilization information is stored in each slave. For post-processing module, PAT collects all files which contain resource utilization metrics from all slaves to master, then performs extracting and averaging each kind of metrics. The measurements of the utilization metrics are depicted as graphs to show their distribution over the query's runtime.

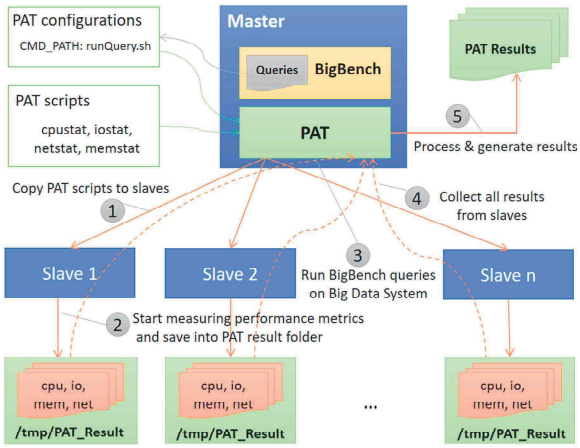

Figure 1. Running BigBench with PAT

2.3 BigBench queries execution with PAT

To measure resource utilization during evaluating each BigBench query, we execute the query from PAT.

Figure 1 depicts the process of executing BigBench queries with PAT and getting the resource utilization of the big data system for evaluating each query. For evaluating each BigBench query, the process includes five steps as follows.

Step 1: we first start PAT collecting data module by installing PAT scripts to slaves.

Step 2: once PAT scripts are installed on a slave; they will start measuring resource utilization of that slave.

Step 3: After starting resource utilization measurement, PAT master will execute a shell file “runQuery.sh” which contains a command running a BigBench query. By this way, master submitted MapReduce (or Spark) jobs to run on the big data system. Once the BigBench query evaluation is finished, the response time will be returned, and each slave will generate its PAT results into CSV files, then go to next step.

Step 4: Run PAT post-processing module on master to collect PAT results from all slaves.

Step 5: PAT combines all results to compute and generate final PAT results, which consist of graphs showing the average of resource utilization of the system for a BigBench query.

3. Performance Evaluation

3.1. Evaluation Settings

To evaluate Hive on MapReduce and Spark SQL, we deployed both Hadoop and Spark on five machines: one machine for master node, and four others for compute nodes [1]. Each machine has 4 CPUs and 16GB of RAM. The main configuration settings of our big data system are shown in Table 1.

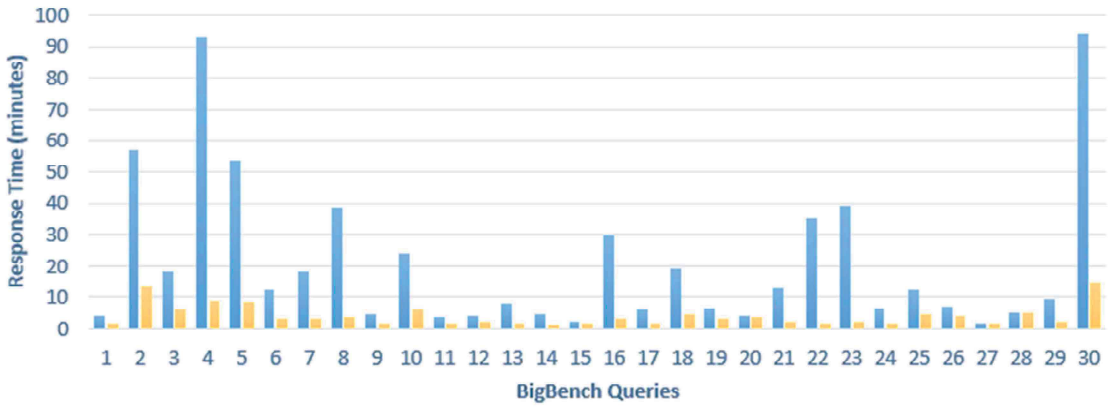

Figure 2. Performance comparison of Hive and Spark SQL using 100GB dataset

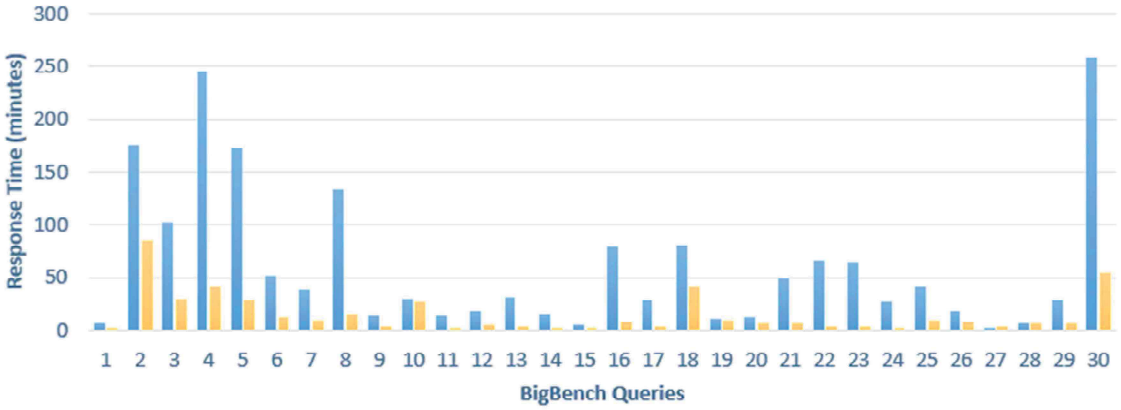

Figure 3. Performance comparison of Hive and Spark SQL using 500GB dataset

Table 1. The configuration settings of big data components

Component	Parameter	Value
YARN	yarn.nodemanager.resource.memory-mb	14GB
	yarn.scheduler.maximum-allocation-mb	14GB
	yarn.nodemanager.resource.cpu-vcores	4
MapReduce	mapreduce.map.java.opts.max.heap	2GB
	mapreduce.reduce.java.opts.max.heap	2GB
	mapreduce.map.memory.mb	4GB
	mapreduce.reduce.memory.mb	4GB
Spark	spark.executor.cores	3
	spark.executor.memory	9GB
	spark.driver.memory	1GB
	spark.executor.instances	5
	spark.kryoserializer.buffer.max	256m
Hive	hive.auto.convert.join	false

Experiments were performed by using all of the 30 BigBench queries for both Hive on MapReduce and Spark SQL. We evaluated these engines by using two different scale factor (100GB and 500GB).

3.2. Experimental Results

This section presents the response time of the 30 BigBench queries running on Hive/MapReduce and Spark SQL. Each query is executed twice then an average of the response time is got. The performance comparison of Hive on MapReduce and Spark SQL is depicted in Figure 2 and Figure 3.

In both of two scale factors (100GB and 500GB), all of 30 queries on Spark SQL run faster than Hive on MapReduce. Especially, with some queries (Q2, Q4, Q5, Q8, Q30) which contain “JOIN” clause among big tables or relate to iterative algorithms, Hive on MapReduce performed slower, for instance, Q4 being 10 times slower than on Spark SQL with 100GB dataset.

4. Resources Utilization Analysis

In this section, we analyze the resource utilization of Hive on MapReduce and Spark by considering the measurements of the metrics: CPU, memory, and network traffic. We measured the resource utilization of all 30 BigBench queries on 100GB dataset.

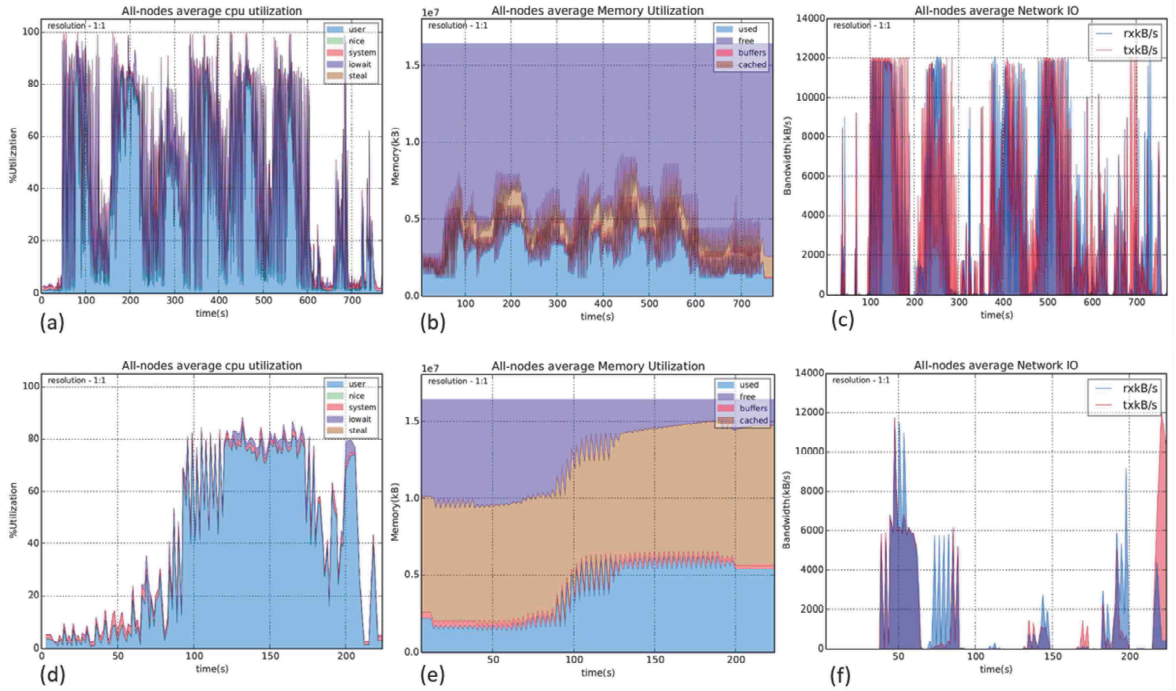

Figure 4. Query Resource Utilization: (a)-(c) for Hive on MapReduce; (d)-(f) for Spark SQL

Overall, we observed that in the most queries, there is no considerable difference on CPU utilization between Hive on MapReduce and Spark SQL, but the utilization of memory of Spark SQL is much greater than Hive on MapReduce, and the amount of data transferring of Hive on MapReduce over the network is much larger than Spark SQL. For instance, we presented the results of resources utilization of query 6, which were generated by PAT, as shown in Figure 4. Both Hive on MapReduce and Spark SQL utilized on average 45% CPU. However, Spark SQL utilized much more of memory, which is 75%; while Hive on MapReduce used only 35%. Especially, Spark SQL used 50% of the memory for caching data. For network traffic, we observed that Hive on MapReduce needs to transfer a large amount of data over network which on average 2.2MB/sec for both transmission and reception bandwidth used, whereas the Spark SQL used average 0.8MB/sec.

5. Conclusion

In this paper, we presented the results of performance evaluation between Hive on MapReduce and the recent Spark SQL on our big data system by employing BigBench and PAT. We also compared the performance and the resources utilization of Hive on MapReduce and SparkSQL. Our experiments proved that the recent Spark SQL outperformed Hive on MapReduce on all of 30 BigBench queries with higher resource utilization.

In our future work, we intend to optimize Spark SQL, where rule-based and cost-based models for improving performance of joining big tables would be investigated.

Acknowledgement

This work was carried out with the support of "Cooperative Research Program for Agriculture Science and Technology Development (Project No. PJ01182302)" Rural Development Administration, Republic of Korea. This work was supported by the National Research Foundation of Korea Grant funded by the korean Government(NRF-2014R1A1A1007734).

References

[1] Van-Quyet Nguyen, Sinh Ngoc Nguyen, Kyungbaek Kim. “Design of a Platform for Collecting and Analyzing Agricultural Big Data.” JDCS vol. 18, no.1, 2017. pp. 149-158.

[2] Hadoop, Apache. "Apache Hadoop". http://hadoop.apache.org, 2017.

[3] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters. " Communications of the ACM Vol.51, No.1, 2008. pp.107-113.

[4] Thusoo, Ashish, et al. "Hive: a warehousing solution over a map-reduce framework." Proceedings of the VLDB Endowment Vol2, No.2, 2009. pp.1626-1629.

[5] Zaharia, Matei, et al. "Spark: Cluster Computing with Working Sets." HotCloud 10.10-10, 2010 pp.95.

[6] Han, Rui, Xiaoyi Lu, and Jiangtao Xu. "On big data benchmarking." Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware. Springer International Publishing, 2014.

[7] Ghazal, Ahmad, et al. "BigBench: towards an industry standard benchmark for big data analytics." Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM, 2013.