Benchmarking Performance for Migrating a Relational Application to a Parallel Implementation

Krishna Karthik Gadiraju, Karen C. Davis, and Paul G. Talaga

Electrical Engineering and Computing Systems
University of Cincinnati
Cincinnati, OH- 45221-0030
gadirakk@mail.uc.edu, karen.davis@uc.edu, talagapl@ucmail.uc.edu

Abstract. Many organizations rely on relational database platforms for OLAP-style querying (aggregation and filtering) for small to medium size applications. We investigate the impact of scaling up the data sizes for such queries. We intend to illustrate what kind of performance results an organization could expect should they migrate current applications to big data environments. This paper benchmarks the performance of Hive [20], a parallel data warehouse platform that is a part of the Hadoop software stack. We set up a 4-node Hadoop cluster using Hortonworks HDP 1.3.2 [10]. We use the data generator provided by the TPC-DS benchmark [3] to generate data of different scales. We use a representative query provided in the TPC-DS query set and run the SQL and Hive Query Language (HiveQL) versions of the same query on a relational database installation (MySQL) and on the Hive cluster. We measure the speedup for query execution for all dataset sizes resulting from the scale up. Hive loads the large datasets faster than MySQL, while it is marginally slower than MySQL when loading the smaller datasets.

Keywords: Hive, Hadoop, benchmarking, big data, SQL, queries.

1 Introduction

Big data refers to petabyte scale datasets that cannot be managed and analyzed using traditional database management systems and data warehouses. The last decade has seen an enormous rise in the size of data collected by different organizations. According to Baru et al. [1], studies have indicated that enterprise data is estimated to grow from 0.5 ZB in 2008 to 35 ZB in 2020. Traditional relational database systems are considered incapable of handling data of such scale. However, it is not always clear to a smaller organization what performance gains they can expect to achieve for their application with a modest investment in additional hardware. One such organization approached us with this question. After analyzing their current configuration and typical queries, we selected a hardware/software configuration and test query to enable us to provide some indication of what they could expect if they scaled up their data. Their application only runs one data intensive query at a time, and they wished to own their own hardware, so our experiments target this scenario. There are no other results in the literature that address this question.

M. Indulska and S. Purao (Eds.): ER Workshops 2014, LNCS 8823, pp. 55–64, 2014.

In this paper, we discuss the features of Apache Hive, and compare its performance against a relational database system. We use a query and data that are a part of the TPC-DS [19] benchmarking standard. In the following sections, we briefly describe the features of Hive. We present our experimental setup, procedures, and results obtained. We analyze the results and offer conclusions. We also suggest some future work.

2 Apache Hive

Apache Hive is a distributed data warehouse that is a part of the Hadoop software stack. Queries in Hive are written in HiveQL (Hive Query Language), which follows a syntax similar to SQL. Queries written in HiveQL are translated into a series of MapReduce [2] jobs. The HiveQL query is translated into a DAG (Directed Acyclic Graph) of MapReduce jobs. MapReduce is a parallel programming framework. The MapReduce jobs defined in the DAG are executed in parallel on the different nodes in the cluster where the data is stored to obtain the results. A typical Hive installation has a Metastore [20], which is used to store the metadata related to the tables and columns, a Thrift server [20], which provides a client API for executing the HiveQL statements, external interfaces such as command line interface (CLI), a driver [20], which is responsible for management of the life cycle of a HiveQL statement, a compiler [20] which is used to translate a HiveQL statement it receives from the driver into a DAG of MapReduce jobs. Once the DAG of MapReduce jobs is prepared by the compiler, the driver invokes the execution engine [20] (which in the case of Hive is Hadoop) to execute the MapReduce jobs. A Hive data model consists of tables, partitions and buckets [20]. A Hive table is similar to a table in a relational database and is made up of rows and columns. Each time a table is created in Hive, a new directory is created on HDFS (Hadoop Distributed File System) and data related to that table is stored in that folder. A table can have one or more partitions. Each partition is stored as a separate directory within the table directory and the data is stored in whichever partition directory it belongs to. The partitions can be further sub-divided into buckets, depending upon the hash of a column in the table [20]. The buckets are stored as individual files within their respective partition directories.

3 Experimental Setup

3.1 Hardware Configuration

A 4-node Hadoop cluster was set up using Hortonworks HDP 1.3.2 [10]. We were limited to a small cluster because of the feasibility of an academic setup wherein we used the available departmental hardware. Each machine has dual quad-core Intel Xeon processors for a total of 16 hyper-threaded cores per machine, 48 GB RAM and 3.08 TB HDD. All four machines communicate with each other using a gateway machine. The MySQL machine shared the same processor and RAM, but had a 2.05 TB HDD installed. The six machines mentioned here are connected together using a Cisco SG 200-26 26-Port Gigabit Smart Switch.

3.2 Software Configuration

The Hadoop cluster was set up using Hortonworks HDP 1.3.2. The version of Hive used in this study is 0.11, and the version of MySQL used is 5.1.71. Centos 6.4 minimal operating system was used to set up the Hadoop cluster, which ran Hadoop version 1.2.

3.3 Experimental Procedure

There are several benchmarking standards defined to benchmark the performance of Hadoop such as such as Sorting programs (Hadoop Sort Program [17], TeraSort [7]), GridMix [5] and HiBench Benchmarking Suite [12], but none of them have well-defined queries or a schema necessary for evaluating the run time performance of a big data management system such as Hive. BigBench [4] and Hive Performance Benchmark [9] both define a schema and dataset for benchmarking Hive, but BigBench is based on the TPC-DS benchmark and provides a larger variety and scale of datasets and queries. While the structured part of BigBench is based on TPC-DS, it also adds several semi-structured and unstructured data components [4]. Since we are dealing with how Hive performs against a relational system, we use the TPC-DS benchmark to analyze the performance of Hive.

TPC-DS (Transaction Processing Performance Council–Decision Support) is a benchmark for evaluating decision support systems. It defines 99 distinct queries that serve a typical business analysis environment [13]. TPC-DS uses a snowstorm schema [13], which is a collection of several snowflake schemas. The schema has been created to model the decision support functions of a retail product supplier [13]. We selected Query 7 [19] from the TPC-DS benchmark as a representative OLAP-style query. It joins 5 tables and contains 4 aggregation operations, 1 group by operation, and 1 order by operation. We modified the original version of the query to remove the "top 100" expression in order to focus on aggregation and filtering over a fact table and several dimension tables. The modified and HiveQL version of the query are as shown below.

The modified SQL query is:

```
select i_item_id,
  avg(ss_quantity) agg1,
  avg(ss_list_price) agg2,
  avg(ss_coupon_amt) agg3,
  avg(ss_sales_price) agg4
from store_sales, customer_demographics, date_dim, item,
promotion
where ss_sold_date_sk = d_date_sk and
  ss_item_sk = i_item_sk and
  ss_cdemo_sk = cd_demo_sk and
  ss_promo_sk = p_promo_sk and
  cd_gender = 'F' and
  cd_marital_status = 'D' and
```

```
  cd_education_status = 'College' and
  (p_channel_email = 'N' or p_channel_event = 'N') and
  d_year = 2001
group by i_item_id
order by i_item_id;
```

Since HiveQL uses joins rather than listing tables using the ',' operator in the FROM clause as in the TPC-DS query above, the revised query is shown below:

```
select i_item_id,
  avg(ss_quantity) agg1,
  avg(ss_list_price) agg2,
  avg(ss_coupon_amt) agg3,
  avg(ss_sales_price) agg4
from store_sales ss join date_dim d on
(ss.ss_sold_date_sk = d.d_date_sk)
join item i on ( ss.ss_item_sk = i.i_item_sk )
join promotion p on (ss.ss_promo_sk = p.p_promo_sk)
join customer_demographics cd on (ss.ss_cdemo_sk =
cd.cd_demo_sk)
  where
  cd_gender = 'F' and
  cd_marital_status = 'D' and
  cd_education_status = 'College' and
  (p_channel_email = 'N' or p_channel_event = 'N') and
  d_year = 2001
 group by i_item_id
 order by i_item_id;
```

The tables mentioned in the above query are defined in both SQL in HiveQL. Since Hive 0.11 does not support variable types such as varchar and date, they are substituted by string and timestamp, respectively.

4 Results and Analysis

4.1 Results

We use dbgen2 [3] the data generator that is provided as a part of the TPC-DS framework to generate datasets of different scales-100 GB, 300 GB and 1 TB (D1, D2 and D3). Tables 1, 2, and 3 display the sizes of datasets used for each experiment and the amount of time taken to load the datasets into Hive and MySQL. Table 4 displays the amount of time taken to execute the query mentioned in the previous section for all the dataset sizes.

Table 1. Amount of time taken to load datasets (100GB)- D1

Table name	Size	Data load time (MySQL)	Data load time (Hive)
customer_demographics	77 MB	5.78s	2.764s
item	56 MB	1.73s	2.045s
promotion	123 KB	0.01s	0.41s
store_sales	39 GB	14h 25m 47.22s	21m 43.907s
date_dim	9.9 MB	0.5s	0.6s

Table 2. Amount of time taken to load datasets (300 GB) – D2

Table name	Size	Data load time (MySQL)	Data load time (Hive)
customer_demographics	77 MB	5.82s	2.507s
Item	72 MB	2.24s	2.271s
Promotion	159 KB	0.03s	0.407s
store_sales	116 GB	1d 19h 42m 48.84s	1h 2m 36.776s
date_dim	10 MB	0.37s	0.718s

Table 3. Amount of time taken to load datasets (1 TB) – D3

Table name	Size	Data load time (MySQL)	Data load time (Hive)
customer_demographics	77 MB	5.74s	3.28s
item	82 MB	2.53s	2.168s
promotion	184 KB	0.02s	0.458s
store_sales	390 GB	6d 3h 17m 20.76s	2h 58m 8.888s
date_dim	10 MB	0.36s	0.686s

Table 4. Query execution times for datasets D1, D2, and D3

Dataset	Original dataset size	Query dataset size	Query execution time (MySQL)	Query execution time (Hive)
D1	100 GB	39 GB	8m 49.18s	4m 12.816s
D2	300 GB	117 GB	31m 59.77s	11m 57.084s
D3	1 TB	390 GB	1h 35m 46.23s	38m 0.41s

4.2 Analysis

From Tables 1-4, we offer two observations and conclusions.

1. Regarding data loading: consider the amount of time taken by both MySQL and Hive to load the item dataset from Tables 1, 2 and 3. While the dataset size is small (D1), MySQL loads data faster than Hive. But as the size of the dataset increases, the difference in time decreases. In D3, Hive is in fact faster in loading the dataset than MySQL. By observing the amount of time taken to load the *store_sales* dataset for the three datasets shown in Tables 1, 2, and 3 it can be observed that for our scenario, Hive loads large datasets faster than MySQL. Since Hive copies the files

verbatim onto a folder on HDFS and does not parse the file [21], and MySQL parses the data file, the difference becomes apparent as the size of the dataset increases. Figures 1 and 2 show a comparison of the amount of time taken to load the *item* and *store_sales* datasets. From the snowstorm schema defined by TPC-DS, *store_sales* is categorized as a fact table, while *item* is categorized as a dimension table [13]. In other words, while the *item* dataset can be categorized as a small/medium scale dataset, with its size ranging between 56-72 MB, the *store_sales* dataset can be categorized as a large dataset, with its size ranging between 39 GB to 390 GB. By considering both these datasets, we are able to analyze how Hive loads datasets of different sizes.

2. Regarding query execution: consider the amount of time taken to execute the query as shown in Table 4. It can be observed that for all the three datasets, Hive executes the query faster than MySQL. Figure 3 shows a comparison between the query execution times for three datasets shown in Table 4. As the size of the dataset increases, the difference in query execution times between MySQL and Hive increases. This difference in query execution speed can be attributed to the fact that a query written in HiveQL is translated into a series of MapReduce jobs as explained in Section 2. Since the query is executed in parallel on different machines based on where the data has been stored by the HDFS, Hive is able to execute the query faster than MySQL.

Based on our studies, we conclude that an organization working with aggregate queries in an RDBMS environment can benefit from scaling up their applications to significantly larger datasets. One limitation, however, is that HiveQL currently only supports equijoins.

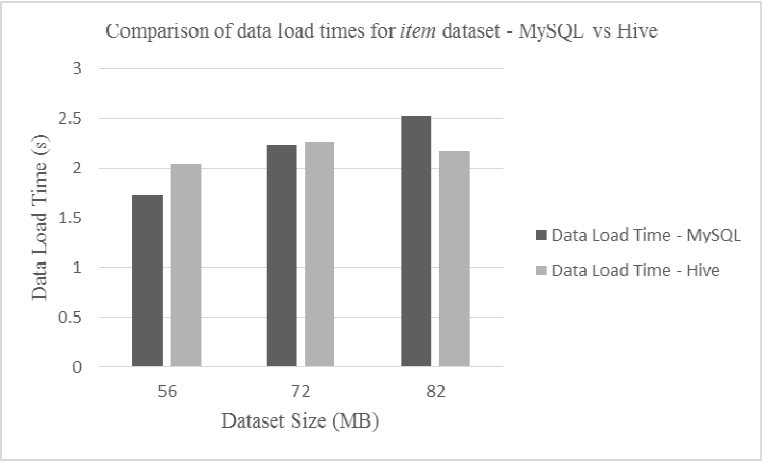

Fig. 1. A comparison of data load times for *item* dataset: MySQL vs. Hive

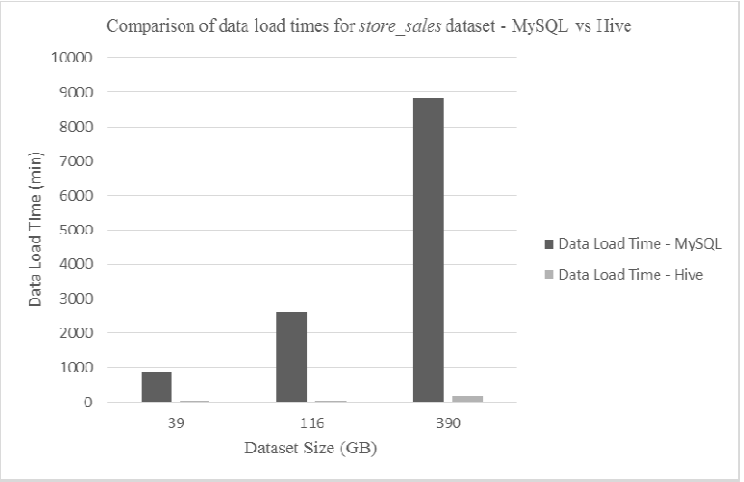

Fig. 2. A comparison of data load times for *store_sales* dataset: MySQL vs. Hive

Fig. 3. A comparison of query execution times for MySQL and Hive

5 Related Work

While there have been other studies that have benchmarked the performance of Hive, this study differs from them in terms of the number of nodes used, the version of Hive used, the relational database used, the benchmarking standard used and the size of data used. Table 5 gives a comparison of the different studies conducted on benchmarking Hive and how they differ from our study.

Table 5. Comparison of features with related studies

Study	Benchmark	Dataset size	Hive version	Number of nodes used	Additional Differences
Hortonworks Stinger Initiative [11]	TPC-DS (Query 27, 95)	200GB, 1 TB	0.11, 0.12, 0.13	Not specified	• Not compared to a relational database • Does not consider data load time
Shi et al. [18]	Queries and datasets provided by Pavlo et al. [15]	110 GB	0.6	20	• Not compared to a relational database • Uses queries that are not as complex as the one used in this study
Hive Performance Benchmark [9]	Queries and datasets provided by Pavlo et al. [15]	110 GB	Trunk version 786346	11	• Not compared to a relational database • Uses queries that are not as complex as the one used in this study
Jia et al. [16]	TPC-H	100 GB	Trunk version 799148	11	• Does not consider data load times
Pansare et al. [14]	TPC-H	10 GB	Not specified	4	• Focuses on mid-level data analysis and uses only 10GB dataset • Not compared to a relational database
Gadiraju et al. (current study)	TPC-DS	390 GB	0.11	4	• Compares performance with a relational database • Considers both data load time and query execution time • Focuses on large scale data analysis

6 Future Work

Future performance studies could investigate additional queries as well as increase the number of nodes in the cluster. There are several directions in which research can be conducted on Hive to achieve further improvement. Some of them include:

- Improved connection to the Hive Metastore: one of the issues we observed while working with the cluster is that the Metastore used by Hive to store its metadata is a single point of failure. Once the relational database service running as the Metastore crashes, Hive has no other means of accessing its metadata. There is a need to define a backup mechanism through which Hive can still access its metadata when it is unable to access its Metastore.
- Optimizing performance: while several steps have been taken to ensure query optimization [20] in Hive, there are several ways in which Hive queries can be optimized. Storing statistical data in the form of metadata in the Metastore at column, partition, and table level as suggested by Gruenheid et al. [6] is one way of improving query performance.
- Herodotos et al. [8] conduct research to indicate how placement of data within the HDFS would govern how well a MapReduce job would run. Their model, StarFish [8], defines a means to place data optimally to enhance MapReduce performance. Further research can be conducted to study the efficiency of the StarFish model, and benchmark how the StarFish model would improve the performance of Hive.

Acknowledgements. Thanks to Dr. Thomas Wilson and James Newman for bringing this interesting problem to our attention and to Trajectory HealthCare (www.trajectoryhealthcare.com) for supporting the investigation.

References

1. Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., Rabl, T.: Setting the direction for big data benchmark standards. In: Nambiar, R., Poess, M. (eds.) TPCTC 2012. LNCS, vol. 7755, pp. 197–208. Springer, Heidelberg (2013)
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
3. DSGen v1.1.0, data generation tool for TPC-DS, `http://www.tpc.org/tpcds/`
4. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen, H.-A.: BigBench: Towards an Industry Standard Benchmark for Big Data Analytics (2013)
5. GridMix program. Available in Hadoop source distribution: src/benchmarks/gridmix
6. Gruenheid, A., Omiecinski, E., Mark, L.: Query optimization using column statistics in hive. In: Proceedings of the 15th Symposium on International Database Engineering & Applications, pp. 97–105. ACM (2011)
7. HadoopTeraSort program. Available in Hadoop source distribution since 0.19 version: src/examples/org/apache/hadoop/examples/terasort
8. Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F.B., Babu, S.: Starfish: A Self-tuning System for Big Data Analytics. In: CIDR, vol. 11, pp. 261–272 (2011)

9. Hive Performance Benchmark, https://issues.apache.org/jira/browse/hive-396
10. Hortonworks HDP 1.3.2, http://hortonworks.com/products/hdp/hdp-1-3/#overview
11. Hortonworks Stinger Initiative, http://hortonworks.com/labs/stinger/
12. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW), pp. 41–51. IEEE (2010)
13. Nambiar, R.O., Poess, M.: The making of TPC-DS. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 1049–1058. VLDB Endowment (2006)
14. Pansare, N., Cai, Z.: Using Hive to perform medium-scale data analysis (2010)
15. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 165–178. ACM (2009)
16. Running the TPC-H benchmark on Hive, https://issues.apache.org/jira/secure/attachment/12416257/TPC-H_on_Hive_2009-08-11.pdf
17. Sort program. Available in Hadoop source distribution: src/examples/org/apache/hadoop/examples/sort
18. Shi, Y., Meng, X., Zhao, J., Hu, X., Liu, B., Wang, H.: Benchmarking cloud-based data management systems. In: Proceedings of the Second International Workshop on Cloud Data Management, pp. 47–54. ACM (2010)
19. TPC-DS benchmarking standard, http://www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf
20. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: A warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2(2), 1626–1629 (2009)
21. White, T.: Hadoop: The definitive guide. O'Reilly (2012)