A Study of SQL-on-Hadoop Systems

Yueguo Chen[1,2(✉)], Xiongpai Qin[1,2], Haoqiong Bian[1,2], Jun Chen[1,2], Zhaoan Dong[1,2], Xiaoyong Du[1,2], Yanjie Gao[1,2], Dehai Liu[1,2], Jiaheng Lu[1,2], and Huijie Zhang[1,2]

[1] Key Laboratory of Data Engineering and Knowledge Engineering, MOE, Beijing, China
[2] School of Information, Renmin University of China, Beijing 100872, China
chenyueguo@gmail.com

Abstract. Hadoop is now the de facto standard for storing and processing big data, not only for unstructured data but also for some structured data. As a result, providing SQL analysis functionality to the big data resided in HDFS becomes more and more important. Hive is a pioneer system that support SQL-like analysis to the data in HDFS. However, the performance of Hive is not satisfactory for many applications. This leads to the quick emergence of dozens of SQL-on-Hadoop systems that try to support interactive SQL query processing to the data stored in HDFS. This paper firstly gives a brief technical review on recent efforts of SQL-on-Hadoop systems. Then we test and compare the performance of five representative SQL-on-Hadoop systems, based on some queries selected or derived from the TPC-DS benchmark. According to the results, we show that such systems can benefit more from the applications of many parallel query processing techniques that have been widely studied in the traditional MPP analytical databases.

Keywords: Big data · SQL-on-Hadoop · Interactive query · Benchmark

1 Introduction

Since introduced by Google in 2004, MapReduce [12] has become a mainstream technology for big data processing. Hadoop is an open-source implementation of MR (MapReduce). It has been used in various data analytic scenarios such as web data search, reporting and OLAP, machine learning, data mining, information retrieval, and social network analysis [19, 23]. Researchers from both industry and academia have made much effort to improve the performance of the MR computing paradigm in many aspects, such as optimization and indexing support of the storage layout, extension to streaming processing and iterative style processing, optimization of join and deep analysis algorithms, scheduling strategies for multi-core CPU/GPU, easy-to-use interfaces and declarative languages support, energy saving and security guarantee etc. As a result, Hadoop becomes more and more mature.

Hadoop is basically a batch-oriented tool for processing a large volume of un-structured data. However, as the underlying storage model is ignored by the Hadoop framework, when some structured layout is applied to the HDFS (Hadoop Distributed File System) data blocks, Hadoop can also handle structured data as well [17]. Apache Hive and its HiveQL language have become a SQL interface for Hadoop since introduced by Facebook in 2007.

Some researchers have compared Hadoop against RDBMSs [22], and they concluded that Hadoop is much inferior in terms of structured data processing. However, the situation has been changing recently. Traditional database vendors, startups, as well as some researchers are trying to transplant SQL functionalities onto the Hadoop platform, and providing interactive SQL query capability with a response time of seconds or even sub-seconds. If the goal is accomplished, Hadoop will be not only a batch-oriented tool for exploratory analysis and deep analysis, but also a tool for interactive ad-hoc SQL analysis of big data.

The paper firstly reviews various SQL-on-Hadoop systems from a technical point of view. Then we test and compare the performance of five representative SQL-on-Hadoop systems, based on some selected workloads from the TPC-DS benchmark. By comparing the results, strengths and limitations of the systems are analyzed. We try to identify some important factors and challenges in implementing a high performance SQL-on-Hadoop system, which could guide the efforts to improve current systems.

2 SQL-on-Hadoop Systems

2.1 Why Transplant SQL onto Hadoop

There are so many RDBMS systems in the market that support data analysis with SQL and provide real time response time. Why bother to transplant SQL onto Hadoop to provide the same function? The first reason is the cost to sale. Hadoop can run on large clusters of commodity hardware to support big data processing. SQL-on-Hadoop systems are more cost efficient than MPP options such as TeraData, Vertica, and Netezza, which need to run on expensive high end servers and don't scale out to thousands of nodes.

The second reason is the I/O bottlenecks. When the volume of data is really big, only some portion of data can be loaded into main memory, the remaining data has to be stored on disks. Spreading I/Os to a large cluster is one of merits of the MapReduce framework, which also justifies SQL-on-Hadoop systems.

The third reason is that beyond the SQL query functionality, we also need more complex analytics. SQL-on-Hadoop systems not only provide SQL query capability, but also provide machine learning and data mining functionalities, which are directly executed on the data, just like what has been done in BDAS (Berkeley Data Analytics Stack) [16]. Although RDBMSs also provide some form of in-database analytics, Hadoop-based systems however, can offer more functions, such as graph data analysis. Hadoop systems can be not only a complementary tool to RDBMSs, in some cases they can replace RDBMS systems.

The fourth reason is that, people are getting more and more interested in analysis of multi-structured data together in one place for insightful information. Hadoop has been the standard tool for unstructured data processing. If structured data processing techniques are implanted onto Hadoop, all data could be in one place. There is no need to move big data around across different tools. SQL layer will empower people who are familiar with SQL and have a big volume of data to analyze.

2.2 An Overview of SQL-on-Hadoop Systems

Systems coming from open source communities and startups include Hive, Stinger, Impala, Hadapt, Platfora, Jethro Data, HAWQ, CitusDB, Rainstor, MapR and Apache Drill, etc. Apache Hive and its HiveQL language have become the standard SQL interface for Hadoop since introduced by Facebook in 2007. Some works [18,20] have been done on translating SQL into MR jobs with some optimizations. Stinger [8] is an initiative of HortonWorks to make Hive much faster, so that people can run ad-hoc queries on Hadoop interactively. Impala [3] uses its own processing framework to execute queries, bypassing the inefficient MR computing model. Hadapt is the commercialized version of the HadoopDB project [9], by combining PostgreSQL and Hadoop together, it tries to retain high scalability and fault tolerance of MR while leveraging high performance of RDBMS when processing both structured and un-structured data. Platfora maintains scale-out in memory aggregates layer that roll up raw data of Hadoop. It is also a fast in memory query engine. Jethro Data [5] uses indexes to avoid full scan of the entire dataset, leading to dramatic reduction in query response time. EMC Greenplum's HAWQ [11] use various techniques to improve performance of SQL query on Hadoop, including query optimization, in memory data transferring, data placement optimization etc. Citus Datas CitusDB [2] extends HDFS in the Hadoop system by running a PostgreSQL instance on each data node, which could be accessed through a wrapper. CitusDB achieve the performance boost by leveraging structured data processing capability of PostgreSQL databases. Rainstor [7] provides compression techniques instead of a fully functional SQL-on-Hadoop system. Compression can reduce the data space used by 50X, which leads to a rapid response time. Apache Drill [4] has been established as an Apache Incubator Project, and MapR is the most involved startup in the development of Drill. Columnar storage and optimized query execution engine help to improve their query performance.

Systems from traditional database vendors include Microsoft PolyBase, TeraData SQL-H, Oracles SQL Connector for Hadoop. PolyBase [13] uses a cost based optimizer to decide whether offloading some data processing tasks onto Hadoop to achieve higher performance. TeraData SQL-H [10] and Oracles SQL Connector for Hadoop [1] enable users to run standard SQL queries on the data stored within Hadoop through the RDBMS, without moving the data into RDBMS. Systems from academia include Spark/Shark [24] and Hadoop++/HAIL [14]. Shark [24] is a large-scale data warehouse system built on top of Spark. By using in memory data processing it achieves higher performance

than Hive. Hadoop++ and HAIL [14] improve Hadoop performance by optimizing Hadoop query plan, creating index, and co-locating data that will join together later during data loading.

2.3 A Closer Look at Our Benchmarked Systems

We choose five representative systems of above for benchmarking study.

Hive. Apache Hive is a data warehouse software that facilitates querying and managing big datasets in Hadoop. Hive applies structure to Hadoop data and enables querying the data using a SQL-like language named HiveQL. Custom mappers and reducers could be plugged in HiveQL to express complex data processing logic. Hive supports various file formats such as char delimited text, Sequence Files, RCFile (Row Columnar) [17], ORC (Optimized Row Columnar), and custom SerDe (Serialization and De-Serialization). Some query optimization strategies are applied in Hive. For example, Hive can select from Shuffle join, Map Join, Sort Merge Join according to the data characteristics. The performance of Hive is limited by the fact that HiveQL is translated into MR jobs to be executed on Hadoop cluster. Some operations such as join are translated into multiple stages of MR tasks that are executed round by round. Each task reads inputs from disk and writes intermediate outputs back to the disk. In our benchmark, Hive is used as the baseline to see the performance boost by the other systems.

Stinger. HortonWorks implements Stinger in a few steps. Firstly, it tries to make Hive as a more suitable tool for people to perform decision support queries by adding new features to the language and making the Hive system more like the standard SQL model. Secondly, a cost-based query optimizer is investigated for better query plans, and a vectorized query execution engine is applied. Thirdly, a new columnar file format is designed for higher performance of analytic tasks. Finally, a new runtime framework, named Tez, is introduced to reduce the Hive's latency and throughput constraints as much as possible.

Cloudera Impala. Cloudera believes that Hadoop is the core of future generation of data warehouse. It involves in the competition of SQL-on-Hadoop battle with its open sourced system Impala [3]. Impala uses its own processing framework to execute queries, bypassing the inefficient MR computing model. It disperses query plans instead of fitting them into a pipeline of map and reduce jobs, thus enables parallelizing multiple stages of a query to avoid the overhead of sort and shuffle if these operations are unnecessary. Moreover, (1) Impala does not materialize intermediate results to disks, similar to MPP databases, it uses in memory data transfers. (2) It avoids MR startup time by running as a service. (3) The execution engine tries to take full advantage of the modern hardware. Its uses the latest set of SSE (SSE4.2) instructions which can offer tremendous speedups in some cases. (4) Impala uses LLVM (Low Level Virtual

Machine) to generate assembly code for the running queries. (5) It is aware of the disk location of blocks and is able to schedule the order to process blocks to keep all disks busy. (6) Impala is designed with performance as the top concern. Various optimization techniques are used when possible, including tight inner loops, in-lined function calls, minimal branching, better use of cache, and minimal memory usage. (7) Impala supports new columnar storages of Parquet for higher performance of query intensive workloads. According to Cloudera's benchmarking results, for purely I/O bound queries, they typically see performance gains in the range of 3-4X. For queries that require multiple MR phases or reduce-side joins in Hive, they see a higher speedup than simple single-table aggregation queries. For queries with at least one join, they have seen performance gains of 7-45X. If the data accessed by the query is resident in the cache, the speedup can be as more as 20X-90X over Hive even for simple aggregation queries [3].

Spark and Shark. Shark [24] is a large-scale data warehouse system built on top of Spark [25], designed to be compatible with Apache Hive. Spark provides the fine granular lineage based fault tolerance required by Shark. Shark supports Hive's query language, meta store, serialization formats, and user-defined functions. Shark can answer HiveQL queries much faster than Hive without modification to the existing data or queries. It leverages several optimization techniques, including in memory column-oriented storage layout, dynamic mid query re-planning of execution plan. These techniques allow Shark to run SQL queries up to 100X faster than Apache Hive, matching the speedups which are reported for MPP analytic databases over MR. Spark can be treated as a replacement of MR, and Shark can be treated as a replacement of Hive. The success of the Spark and Shark projects shows that by leveraging in memory data processing techniques and using careful data layouts, the Hadoop framework can achieve a fast response time to support interactive analysis.

Presto. Presto [6] is an interactive distributed SQL query engine that runs fast on a Hadoop Cluster. It is developed by Facebook for data analysis on petabyte-sized data warehouses. Presto is optimized for ad-hoc analysis at interactive speed by avoiding the MR. It employs a custom query and execution engine with operators designed to support SQL semantics. It uses its own query processing model. The client sends SQL to the Presto coordinator. The coordinator parses, analyzes, and plans the query execution. The scheduler wires together the execution pipeline, assigns work to nodes closest to the data, and monitors the progress. The client pulls data from output stage, which in turn pulls data from underlying stages. Presto compiles parts of the query on the fly and does all of its processing in memory. The pipelined execution model runs multiple stages at once, and streams data from one stage to the next as it becomes available. Since all processing is in memory and pipelined across the network between stages, the associated latency overhead of unnecessary I/Os is avoided. This significantly reduces the latency for many types of queries. Presto also dynamically

compiles certain portions of the query plan down to byte code which lets the JVM optimize and generate native machine code. Presto can do many of the tasks that standard ANSI SQL engines can, including complex queries, aggregations, joins, left/right outer joins, sub-queries, window functions, and most of the common aggregate and scalar functions, including approximate distinct counts and approximate percentiles. The first version still lacks the ability to write results back to tables and cannot create table joins beyond a certain size. Presto supports various file formats such as Text, Sequence File, RCFile, and ORC. It scales up to a cluster of 1,000 nodes. It is 10X better than Hive/MR in terms of CPU efficiency and latency for most queries according to Facebook.

3 Experimental Evaluation

We select five representative SQL-on-Hadoop systems for benchmarking: Hive, Stinger, Impala, Presto and Shark. We test their performance on some selected or modified queries from the TPC-DS benchmark [21]. This section reports the results of our benchmarking tests, as well as some analysis and comparison of the performance of these systems.

3.1 Hardware and Software Configuration

Our experiments run on a cloud comprised of 50 physical nodes (as a part of Renda Xing Cloud[1]). Each node has a memory of 48 GB, 2×6 cores Intel Xeon E5645 CPU, and a disk storage of 6 TB configured with RAID 5. By using OpenStack, we are able to generate clusters of 25, 50 and 100 nodes respectively. The memory size of each virtual nodes ranges from 10 GB, 20 GB to 40 GB. A Gigabit ethernet is deployed in the clusters of our experiments.

The versions of the tested systems are listed in Table 1. The default parameters are typically applied to each benchmarked system, with some of the important parameters manually optimized for better performance.

3.2 Workloads

Considering that most queries of the TPC-DS benchmark are complex SQL analysis queries designed for data warehousing applications, they are not practical for many existing systems due to the computational complexity. For example, in some other studies of big data analysis systems [15,22], only simple SQL queries are executed serially to test the systems. In our study, to benchmark the systems for workloads of different complexity, we modified some queries from TPC-DS, and derive 11 queries (which are also executed serially) used in our test. Among them, some are simple single-table queries, some are the original and complex SQL queries of the TPC-DS benchmark. The applied queries include reports, ad-hoc queries, star-join queries, and complex SQL analytic queries as

[1] http://deke.ruc.edu.cn/yunyuyue.php

Table 1. Versions of tested systems

System	Version
Apache Hive	0.10
Hortonworks Stinger	Hive 0.12
Berkeley Shark	0.7.0
Cloudera Impala	1.0.1
Facebook Presto	0.54

well. Here, we show two examples of them: q5Ao and q6Cgo. When naming the queries in our benchmark, 'A' indicates a single table query, 'B' indicates a join of two tables, 'C' involves 3 tables or more, 'o' indicates an 'order by' operator, and 'g' means a 'group by' operator.

```
q5Ao: select ss_store_sk as store_sk, ss_sold_date_sk as date_sk
     ss_ext_sales_price as sales_price, ss_net_profit as profit
from store_sales
where ss_ext_sales_price>20
order by  profit
limit 100;

q6Cgo: select  a.ca_state state, count(*) cnt
     from customer_address   a
     join  customer c on(a.customer_address.ca_address_sk
         = c.c_current_addr_sk)
     join  store_sales s  on(c.c_customer_sk = s.ss_customer_sk)
     join date_dim d  on(s.ss_sold_date_sk = d.d_date_sk)
     join item i  on(s.ss_item_sk = i.i_item_sk)
group by a.ca_state
having count(*) >= 10
order by cnt
limit 100;
```

We generate two datasets from TPC-DS for benchmarking the systems. One has a size of 1 TB (having a scale factor of 1,000), and the other is in 3 TB (having a scale factor of 3,000). We refer to the other benchmarks [15,22] when choosing the dataset sizes of benchmark. The data model of TPC-DS benchmark follows a snow flake schema, with tables *store_sales* and *store_returns* as two fact tables.

3.3 Results

We conduct 4 groups of tests, with each having a different setting in terms of the number of virtual nodes in the cluster and the sizes of dataset. The details are listed in Table 2. In the followings, we report and compare the results from the query, system, and workload's point of views respectively[2].

[2] Due to the page limit, more details about the experimental settings, results, and result analysis are available at http://deke.ruc.edu.cn/sqlonhadoop.

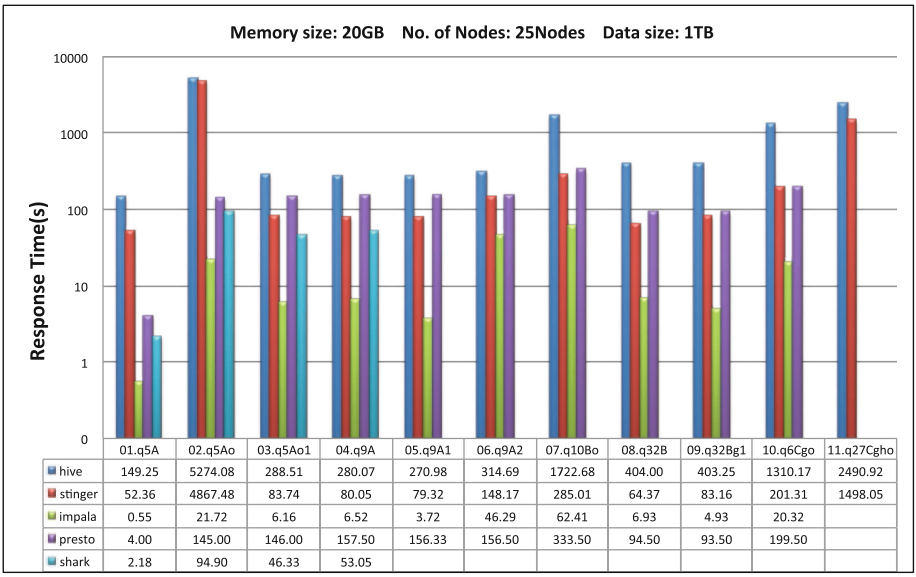
Table 2. Experimental settings for 4 test groups

Test group	No. of nodes	Data size	Results	Relative workloads per node
1	25	1 TB	Fig. 1	Heavy (40 GB/node)
2	50	1 TB	Fig. 2	Normal (20 GB/node)
3	100	1 TB	Fig. 3	Light (10 GB/node)
4	100	3 TB	Fig. 4	Heavy (30 GB/node)

Queries. Among those 11 queries used in our benchmark, there are 6 simple queries (query 1 to query 6) that are conducted over a large fact table. Three simple join queries (query 7 to query 9) include a join operator between a dimension table and a fact table. Query 10 is a complex join query (includes both a star join and a chain join) over 5 tables. Query 11 is a star join query over multiple tables. According to the results reported in the figures, we can find that, simple queries (without join) basically perform better than those complex queries with join operations. Simple join queries basically perform better than those complex queries (query 10 and query 11). However, there are two exceptions (query 2 and query 7) which both have an 'order by' operator. A further study over these two queries show that the intermediate results (before the 'order by' operation) are very huge, which incur large cost for the 'order by' operation. Query 3 is modified from query 2 by simply applying an adjustment of the filtering condition to cause its intermediate results much less than those of query 2. As a result, we can see that query 3 performs much faster than the query 2. We should also note that queries q9* perform slightly slower than queries q5* because there are aggregation operators in queries q9*.

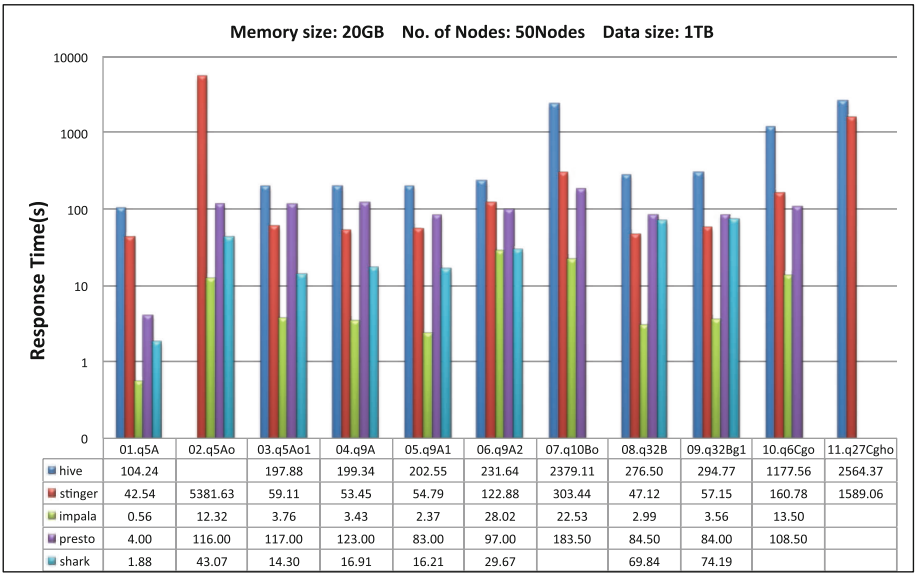
Systems. When comparing the performance of systems in different test groups, we find that impala performs much better than the others in most test cases. On the other hand, Hive often performs the worst among all the systems. This is reasonable because the other systems treat Hive as a baseline, and they try to improve their performance over Hive. Comparatively, the performance of Stinger and Presto is similar in many test cases. For Shark, we find that it performs well for light workloads, especially for simple queries over single table. However, when the workloads are heavy or when the queries are complex, Shark often fails to evaluate the queries due to many reasons such as out of memory. Note that a blank cell in the results of Figs. 1, 2, 3 and 4 represents that a system fails to execute a corresponding query.

Workloads. Queries are executed serially for all tested systems. We adjust the workloads of each node by varying the number of nodes and the size of dataset. For Figs. 1, 2, and 3, we actually reduce the workloads of each node by increasing the number of nodes from 25 to 100. By comparing the results of the



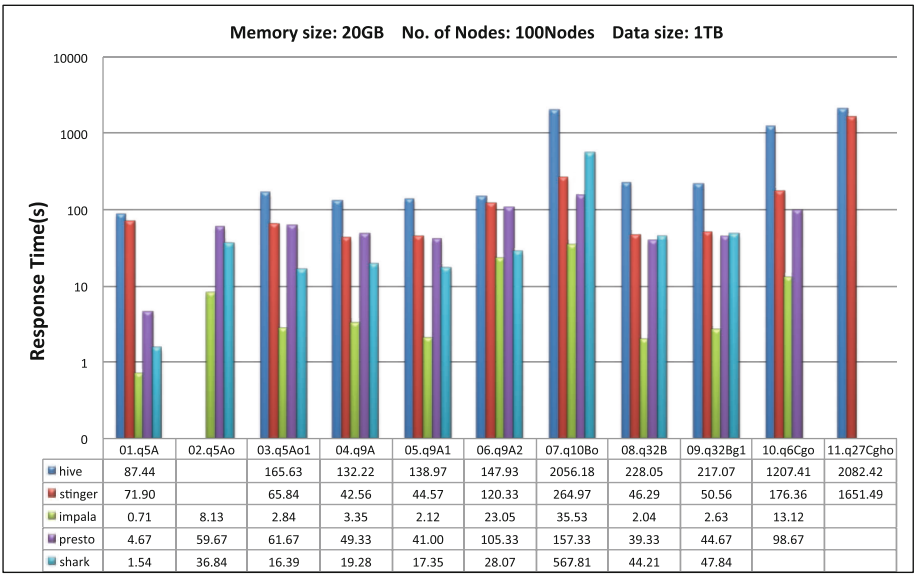
	01.q5A	02.q5Ao	03.q5Ao1	04.q9A	05.q9A1	06.q9A2	07.q10Bo	08.q32B	09.q32Bg1	10.q6Cgo	11.q27Cgho
hive	149.25	5274.08	288.51	280.07	270.98	314.69	1722.68	404.00	403.25	1310.17	2490.92
stinger	52.36	4867.48	83.74	80.05	79.32	148.17	285.01	64.37	83.16	201.31	1498.05
impala	0.55	21.72	6.16	6.52	3.72	46.29	62.41	6.93	4.93	20.32	
presto	4.00	145.00	146.00	157.50	156.33	156.50	333.50	94.50	93.50	199.50	
shark	2.18	94.90	46.33	53.05							

Fig. 1. Results on a cluster of 25 nodes for 1 TB data



	01.q5A	02.q5Ao	03.q5Ao1	04.q9A	05.q9A1	06.q9A2	07.q10Bo	08.q32B	09.q32Bg1	10.q6Cgo	11.q27Cgho
hive	104.24		197.88	199.34	202.55	231.64	2379.11	276.50	294.77	1177.56	2564.37
stinger	42.54	5381.63	59.11	53.45	54.79	122.88	303.44	47.12	57.15	160.78	1589.06
impala	0.56	12.32	3.76	3.43	2.37	28.02	22.53	2.99	3.56	13.50	
presto	4.00	116.00	117.00	123.00	83.00	97.00	183.50	84.50	84.00	108.50	
shark	1.88	43.07	14.30	16.91	16.21	29.67		69.84	74.19		

Fig. 2. Results on a cluster of 50 nodes for 1 TB data



	01.q5A	02.q5Ao	03.q5Ao1	04.q9A	05.q9A1	06.q9A2	07.q10Bo	08.q32B	09.q32Bg1	10.q6Cgo	11.q27Cgho
hive	87.44		165.63	132.22	138.97	147.93	2056.18	228.05	217.07	1207.41	2082.42
stinger	71.90		65.84	42.56	44.57	120.33	264.97	46.29	50.56	176.36	1651.49
impala	0.71	8.13	2.84	3.35	2.12	23.05	35.53	2.04	2.63	13.12	
presto	4.67	59.67	61.67	49.33	41.00	105.33	157.33	39.33	44.67	98.67	
shark	1.54	36.84	16.39	19.28	17.35	28.07	567.81	44.21	47.84		

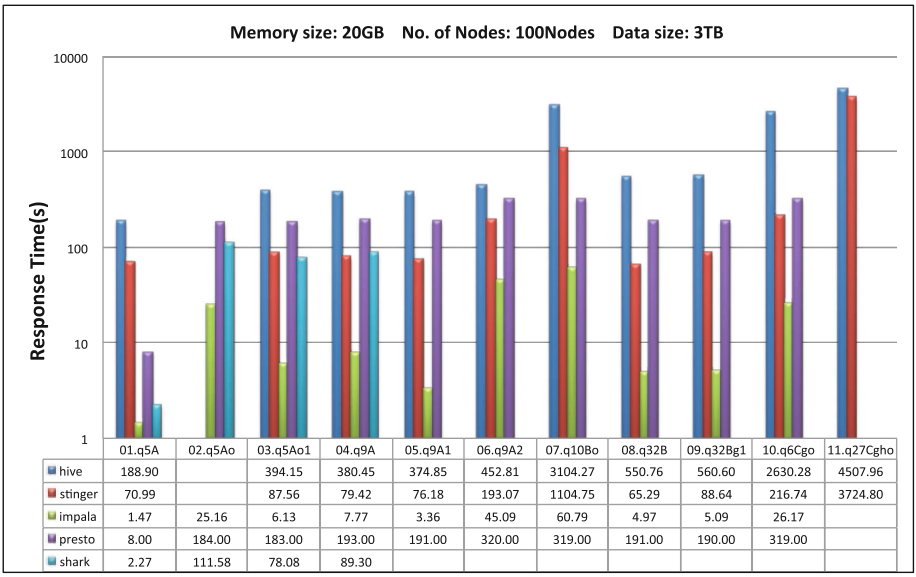
Fig. 3. Results on a cluster of 100 nodes for 1 TB data

same system in Figs. 1, 2 and 3, we can find that the enlargement of the cluster size does help to improve the performance of query evaluation. This is especially obvious for the Shark, where in Fig. 1 (with 25 nodes), only 4 queries can be successfully conducted. The number increases to 9 in Fig. 3. By comparing the corresponding numbers of the same query for the same system, we find that the speedup of query processing can hardly keep up with the rate of the cluster size enlargement. This is also reasonable because the communication cost will be increased when enlarging the number of nodes in a cluster. By keeping the number of node unchanged, and increasing the size of data (from Fig. 3 to Fig. 4), we actually increase the workloads of each node. As a result, the performance of each system drops accordingly. When the workload of each node is heavy enough (in Fig. 4), shark fails to evaluate many queries. In the meanwhile, some systems fails to evaluate query 2 and query 11.

3.4 Analysis

By analyzing the results of our benchmarking and referring to the system implementation, we have the following observations:

- Columnar storage is important for performance improvement, especially when the table has many columns and the query only need to access a small part of them. This is verified by Stinger (with ORCFile) over Hive, and Impala (with Parquet format and the Textfile format, whose results are not shown out).
- By discarding the MR model, the performance benefits from saving the cost of persisting the intermediate results of query processing. Impala, Shark and



	01.q5A	02.q5Ao	03.q5Ao1	04.q9A	05.q9A1	06.q9A2	07.q10Bo	08.q32B	09.q32Bg1	10.q6Cgo	11.q27Cgho
hive	188.90		394.15	380.45	374.85	452.81	3104.27	550.76	560.60	2630.28	4507.96
stinger	70.99		87.56	79.42	76.18	193.07	1104.75	65.29	88.64	216.74	3724.80
impala	1.47	25.16	6.13	7.77	3.36	45.09	60.79	4.97	5.09	26.17	
presto	8.00	184.00	183.00	193.00	191.00	320.00	319.00	191.00	190.00	319.00	
shark	2.27	111.58	78.08	89.30							

Fig. 4. Results on a cluster of 100 nodes for 3 TB data

Presto perform better than Hive and Stinger. They all benefit from the discarding of MR. However, the superiority decreases when the queries are complex, where the other cost will be significant.

- Techniques (e.g. distributed join processing and optimization) from MPP databases do help a lot. This is verified by the Impala system, which performs much better for join queries over two or more tables.
- Performance benefits more from the usage of large memory. Shark and Impala perform much better for small datasets. However, when the workloads increase, the available memory for each node may not be enough for implementing some join operations. This leads to many problems for memory-reliant systems such as Shark.
- Data skewness significantly affects the query performance. Systems such as Hive, Stinger and Shark are sensitive to the data skewness. Query 2 and query 7 shows that when the 'order by' operator falls in one reduce node, it will be a bottleneck of such systems.

4 Conclusion

In this paper, we briefly review the recent efforts of SQL-on-Hadoop systems. We test five representative systems using part of the TPC-DS benchmark. The results shows that these systems are still not efficient enough (for the interactive query processing purpose) for complex analytic queries. Even though, we find that the existing SQL-on-Hadoop systems have benefited a lot from the application of many state-of-the-art parallel query processing techniques (such

as columnar storage, MPP architecture, join optimization) that have been extensively studied for many years in database community. It is expected that with more advanced parallel database techniques applied, the performance of SQL-on-Hadoop systems can be further improved. The merit of providing high performance SQL analysis functionality to the data stored in HDFS will be very attractive to many companies surfing the big data wave.

Acknowledgements. This work is partially supported by National 863 High-tech Program (Grant No. 2012AA011001), the National Science Foundation of China under grant No. 61472426 and No. 61170013, the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China No. 14XNLQ06, the Chinese National "111" Project "Attracting International Talents in Data Engineering and Knowledge Engineering Research", and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

References

1. http://docs.oracle.com/cd/E37231_01/doc.20/e36961/sqlch.htm (2013)
2. Citusdata (2013). http://citusdata.com/docs/SQL-on-Hadoop
3. Cloudera impala (2013). http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/
4. Drill proposal (2013). http://wiki.apache.org/incubator/DrillProposal/
5. Jethro data (2013). http://jethrodata.com/product/
6. Presto (2013). http://prestodb.io
7. Rainstor (2013). http://rainstor.com/products/rainstor-database/
8. Stinger (2013). http://hortonworks.com/stinger/
9. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D.J., Rasin, A., Silberschatz, A.: Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. PVLDB **2**(1), 922–933 (2009)
10. Argyros, T.: The enterprise approach to interactive sql on hadoop data: teradata sql-h (2013). http://www.asterdata.com/blog/2013/04/the-enterprise-approach-to-interactive-SQL-on-Hadoop-data-teradata-sql-h/
11. Chang, L., Wang, Z., Ma, T., Jian, L., Ma, L., Goldshuv, A., Lonergan, L., Cohen, J., Welton, C., Sherry, G., Bhandarkar, M.: Hawq: a massively parallel processing sql engine in hadoop. In: SIGMOD Conference, pp. 1223–1234 (2014)
12. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
13. DeWitt, D.J., Halverson, A., Nehme, R.V., Shankar, S., Aguilar-Saborit, J., Avanes, A., Flasza, M., Gramling, J.: Split query processing in polybase. In: SIGMOD Conference, pp. 1255–1266 (2013)
14. Dittrich, J., Quiané-Ruiz, J.-A., Jindal, A., Kargin, Y., Setty, V., Schad, J.: Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). PVLDB **3**(1), 518–529 (2010)
15. Floratou, A., Teletia, N., DeWitt, D.J., Patel, J.M., Zhang, D.: Can the elephants handle the nosql onslaught? PVLDB **5**(12), 1712–1723 (2012)
16. Franklin, M.J.: Making sense of big data with the berkeley data analytics stack. In: SSDBM, p. 1 (2013)

17. He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., Xu, Z.: Rcfile: a fast and space-efficient data placement structure in mapreduce-based warehouse systems. In: ICDE, pp. 1199–1208 (2011)
18. Iu, M.-Y., Zwaenepoel, W.: Hadooptosql: a mapreduce query optimizer. In: EuroSys, pp. 251–264 (2010)
19. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with mapreduce: a survey. SIGMOD Rec. **40**(4), 11–20 (2011)
20. Lee, R., Luo, T., Huai, Y., Wang, F., He, Y., Zhang, X.:. Ysmart: yet another sql-to-mapreduce translator. In: ICDCS, pp. 25–36 (2011)
21. Nambiar, R.O., Poess, M.: The making of tpc-ds. In: VLDB, pp. 1049–1058 (2006)
22. 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: SIGMOD Conference, pp. 165–178 (2009)
23. Sakr, S., Liu, A., Fayoumi, A.G.: The family of mapreduce and large-scale data processing systems. ACM Comput. Surv. **46**(1), 11 (2013)
24. Xin, R.S., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., Stoica, I.: Shark: Sql and rich analytics at scale. In: SIGMOD Conference, pp. 13–24 (2013)
25. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI, pp. 15–28 (2012)