An Empirical Performance Evaluation of Distributed SQL Query Engines

Stefan van Wouw[§†], José Viña[§], Alexandru Iosup[†], and Dick Epema[†]

[§]Azavista, the Netherlands

[†]Delft University of Technology, the Netherlands

stefanvanwouw@gmail.com, jose@azavista.com, {a.iosup,d.h.j.epema}@tudelft.nl

ABSTRACT

Distributed SQL Query Engines (DSQEs) are increasingly used in a variety of domains, but especially users in small companies with little expertise may face the challenge of selecting an appropriate engine for their specific applications. Although both industry and academia are attempting to come up with high level benchmarks, the performance of DSQEs has never been explored or compared in-depth. We propose an empirical method for evaluating the performance of DSQEs with representative metrics, datasets, and system configurations. We implement a micro-benchmarking suite of three classes of SQL queries for both a synthetic and a real world dataset and we report response time, resource utilization, and scalability. We use our micro-benchmarking suite to analyze and compare three state-of-the-art engines, viz. Shark, Impala, and Hive. We gain valuable insights for each engine and we present a comprehensive comparison of these DSQEs. We find that different query engines have widely varying performance: Hive is always being outperformed by the other engines, but whether Impala or Shark is the best performer highly depends on the query type.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*distributed systems, performance evaluation*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Experimentation; Performance

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2688053.

Keywords

Distributed SQL Query Engine; Performance Evaluation; Scalability

1. INTRODUCTION

With the decrease in cost of storage and computation of public clouds, even small and medium enterprises (SMEs) are able to process large amounts of data. This causes businesses to increase the amounts of data they collect, to sizes that are difficult for traditional database management systems to handle. This has led to Hadoop-oriented distributed query engines such as Hive [18], Impala [5], Shark [21], and more recently, Presto [7], Drill [10], and Hive-on-Tez [3]. Selecting the most suitable of these systems for a particular SME is a big challenge, because SMEs are not likely to have the expertise and the resources available to perform an in-depth study. We remove this burden from SMEs by addressing the following research question: *How well do Distributed SQL Query Engines (DSQEs) perform on SME workloads?*

Although performance studies do exist for Distributed SQL Query Engines [1, 6, 8, 9, 15, 21], many of them only use synthetic workloads or very high-level comparisons that are only based on query response time. Our work evaluates performance much more in-depth by reporting more metrics and evaluating more performance aspects. In addition to reporting query response times, we also show scalability and detailed resource utilization. The latter performance aspects are particularly important for an SME in order to choose a query engine.

In order to answer the research question we define a comprehensive performance evaluation method to assess different aspects of query engines. We compare Hive, a somewhat older but still widely used query engine, with Impala and Shark, both state-of-the-art distributed query engines. This method can be used to compare current and future query engines, despite not covering all the methodological and practical aspects of a true benchmark. The method focuses on three performance aspects: processing power, resource utilization and scalability. With the results from this study, system developers and data analysts can make informed choices related to both cluster infrastructure and query tuning.

Using both a real world and a synthetic dataset with queries representative of SME workloads, we evaluate the

Table 1: Overview of Related Work. Legend: Real World (R), Synthetic (S), Modified Workload (+)

Query Engines	Workload	Dataset Type	Largest Dataset	Cluster Size
Hive, Shark [21]	Pavlo+, other	R, S	1.55 TiB	100
Redshift, Hive, Shark, Impala, Tez [1]	Pavlo+	S	127.5 GiB	5
Impala, Tez, Shark, Presto [6]	TPC-DS+	S	13.64 TiB	20
Teradata DBMS [9]	TPC-DS+	S	186.24 GiB	8
Hive, Impala, Tez [8]	TPC-DS/H+	S	220.72 GiB	20
DBMS-X, Vertica [15]	Pavlo	S	931.32 GiB	100
Our Work	**Pavlo+, other**	**R, S**	**523.66 GiB**	**5**

query engines' performance. We find that different query engines have widely varying performance, with Hive always being outperformed by the other engines. Whether Impala or Shark is the best performer highly depends on the query type and input size.

Our main contributions are:

- We propose a method for performance evaluation of DSQEs (Section 4), which includes defining a workload representative for SMEs as well as defining the performance aspects of the query engines: processing power, resource utilization and scalability.
- We define a micro-benchmark setup for three major query engines, namely Shark, Impala and Hive (Section 5).
- We provide an in-depth performance comparison between Shark, Impala and Hive using our micro-benchmark suite (Section 6).

2. RELATED WORK

We wanted to evaluate the major Distributed SQL Query Engines currently on the market using a cluster size and dataset size that is representative for SMEs, but still comparable to similar studies. Table 1 summarizes the related previous works. Some of them run a subset or enhanced version of the TPC-DS benchmark [16] which has only recently been adopted for Big Data analytics in the form of BigBench [9]. Other studies run a variant of the Pavlo et al. micro-benchmark [15] which is widely accepted in the field.

Overall, most studies use synthetic workloads, of which some are very large. Synthetic workloads do not necessarily characterise real world datasets very well. For our work we have also taken a real world dataset in use by an SME. Besides our work, only one other study uses real world datasets [21]. However, like most of the other studies, it only reports on query response times. Our work evaluates performance much more in-depth by reporting more metrics and evaluating more performance aspects including scalability and detailed resource utilization. We argue that scalability and resource utilization are also very important when deciding which query engine will be used by an SME.

3. QUERY ENGINE SELECTION

In this study we initially attempted to evaluate 5 state-of-the-art Distributed SQL Query engines: Drill, Presto, Shark, Impala and Hive. We chose to evaluate these query engines because they are widely used and contributed to by many individuals and companies. All of the engines have more than 400 forks and more than 1,000 stars on GitHub, except for Drill, which has 188 forks and 298 stars.

We ended up discarding Drill and Presto because these systems lacked required functionality at the time of testing. Drill only had a proof of concept one node version, and Presto did not have the functionality needed to write output to disk (which is required for the kind of workloads we wanted to evaluate).

Shark [21] is a DSQE built on top of the Spark [23] execution engine, which in turn heavily relies on the concept of Resilient Distributed Datasets (RDDs) [22]. In short this means that whenever Shark receives an SQL query, it will convert it to a Spark job, execute it in Spark, and then return the results. Spark keeps all intermediate results in memory using RDDs, and only spills them to disk if no sufficient memory is available. Mid-query fault tolerance is provided by Spark. It is also possible to have the input and output dataset cached entirely in memory.

Impala [5] is a DSQE being developed by Cloudera and is heavily inspired by Google's Dremel [14]. It employs its own massively parallel processing (MPP) architecture on top of HDFS instead of using Hadoop MapReduce as execution engine (like Hive below). One large downside to this engine is that it does not provide fault tolerance. Whenever a node dies in the middle of query execution, the whole query is aborted.

Hive [18] was one of the first DSQEs, introduced by Facebook and built on top of the Hadoop platform [2]. It provides a Hive Meta Store service to put a relational database-like structure on top of the raw data stored in HDFS. Whenever a HiveQL (SQL dialect) query is submitted to Hive, Hive will convert it to a job to be run on Hadoop MapReduce. Although Hive provides mid-query fault tolerance, it relies on Hadoop MapReduce and is slowed down whenever this system stores intermediate results on disk.

4. EXPERIMENTAL METHOD

In this section we present the method of evaluating the performance of Distributed SQL Query Engines. First we define the workload as well as the aspects of the engines used for assessing this performance. Then we describe the evaluation procedure.

4.1 Workload

During the performance evaluation we use both synthetic and real world datasets with three SQL queries per dataset. We carefully selected the different types of queries and datasets to match the scale and diversity of the workloads SMEs deal with.

1) *Synthetic Dataset*: Based on the benchmark from Pavlo et al. [15], the UC Berkeley AMPLab introduced a general benchmark for DSQEs [1]. We have used an adapted version of AMPLab's Big Data benchmark where we leave out the query testing User Defined Functions (UDFs), since not all query engines support UDF in similar form. The synthethic dataset used by these 3 queries consists of 118.29 GiB of structured server logs per URL (the `uservisits` table), and 5.19 GiB of page ranks (the `rankings` table) per website, as seen in Table 2. The `uservisits` and `rankings` tables can be joined by matching on the URL field of the web page visited. No other foreign key relationships are present.

Is this dataset representative for SME data? The structure of the data closely resembles the structure of click data

Table 2: Summary of Datasets.

Table	# Columns	Description
`uservisits`	9	Structured server logs per page.
`rankings`	3	Page rank score per page.
`hotel_prices`	8	Daily hotel prices.

Table 3: Summary of SQL Queries.

Query	Input Size		Output Size		Tables
	GiB	*Records*	*GiB*	*Records*	
1	5.19	90M	5.19	90M	`rankings`
2	118.29	752M	40	254M	`uservisits`
3	123.48	842M	$< 10^{-7}$	1	`uservisits`, `rankings`
4	523.66	7900M	$< 10^{-2}$	113K	`hotel_prices`
5	20	228M	4.3	49M	`hotel_prices` subsets
6	8	94.7M	4	48M	`hotel_prices` subsets

being collected in all kinds of SMEs. The dataset size might even be slightly large for SMEs, because as pointed out by Rowstron et al. [17] analytics production clusters at *large* companies such as Microsoft and Yahoo have median job input sizes under 13.03 GiB and 90% of jobs on Facebook clusters have input sizes under 93.13 GiB.

On this dataset, we run queries 1 to 3 to test raw data processing power, aggregation and *JOIN* performance respectively. We describe each of these queries below in addition to providing query statistics in Table 3.

Query 1 performs a data scan on a relatively small dataset. It simply scans the whole `rankings` table and filters out certain records.

Query 2 computes the sum of ad revenues generated per visitor from the `uservisits` table in order to test aggregation performance.

Query 3 joins the `rankings` table with the `uservisits` table in order to test JOIN performance.

2) *Real World Dataset*: We collected price data of hotel rooms on a daily basis during a period of twelve months between November 2012 and November 2013. More than 21 million hotel room prices for more than 4 million hotels were collected on average every day. This uncompressed dataset (the `hotel_prices` table) is 523.66 GiB on disk as seen in Table 3. Since the price data was collected every day, we decided to partition the dataset in daily chunks as to be able to only use data of certain collection days, rather than having to load the entire dataset all the time.

Is this dataset representative for SME data? The queries we selected for the dataset are in use by Azavista, an SME specialized in meeting and event planning software. The real world scenarios for these queries relate to reporting price statistics per city and country.

On this dataset, we run queries 4 to 6 to also (like queries 1 to 3) test raw data processing power, aggregation and *JOIN* performance respectively. However, these queries are not interchangeable with queries 1 to 3 because they are tailored to the exact structure of the hotel price dataset, and by using different input and output sizes we test different aspects of the query engines. We describe each of the queries 4 to 6 below in addition to providing query statistics in Table 3.

Query 4 computes average prices of hotel rooms grouped by certain months.

Query 5 computes linear regression pricing curves over a timespan of data collection dates.

Query 6 computes changes in hotel room prices between two collection dates.

3) Total Workload: Combining the results from the experiments with the two datasets gives us insights in performance of the query engines on both synthetic and real world data. In particular we look at how the engines deal with data scans (Query 1 and 4), heavy aggregation (Query 2 and 5), and the JOINs (Query 3 and 6).

4.2 Performance Aspects and Metrics

In order to be able to reason about the performance differences between different query engines, the different aspects contributing to this performance need to be defined. In this study we focus on three performance aspects:

1. *Processing Power*: the ability of a query engine to process a large number of SQL queries in a set amount of time. The more SQL queries a query engine can handle in a set amount of time, the better. We measure the processing power in terms of response time, that is, the time between submitting an SQL query to the system and getting a response. In addition, we also calculate the throughput per SQL query: the number of input records divided by response time.

2. *Resource Utilization*: the ability of a query engine to efficiently use the system resources available. This is important, because especially SMEs cannot afford to waste precious system resources. We measure the resource utilization in terms of mean, maximum and total CPU, memory, disk and network usage.

3. *Scalability*: the ability of a query engine to maintain predictable performance behaviour when system resources are added or removed from the system, or when input datasets grow or shrink. Another way of defining scalability is splitting it in strong, as well as weak scalability. Strong scalability measures the query response time improvement when adding more processors to the cluster while, at the same time, keeping the *total* input size fixed. Weak scalability, on the other hand, measures the query response time when adding more processors, while at the same time increasing the input size such that the amount of data per processor stays constant.

   We perform two types of scalability. The first is horizontal scalability (a form of strong scalability), where the total input size is fixed while the number of cluster nodes increases. The second is data input size scalability, where the number of cluster nodes is fixed while the total input size increases. Ideally, the performance should improve proportionally to the amount of resources added (taking the time complexity of the query into account). The performance should only degrade inversely proportional with every unit of input data added (again taking the time complexity of the query into account). In practice this highly depends on the type of resources added as well as the overhead of parallelism introduced.

4.3 Evaluation Procedure

Our procedure for evaluating the DSQEs is as follows: we run each query 10 times on its corresponding dataset while taking snapshots of the resource utilization using the monitoring tool `collectl` [4]. After the query completes, we also store its response time. Note that we run each query in a clean system in a single-tenant environment. No side-effects of queries can affect other queries. When averaging over all the experiment iterations, we report the standard deviation as indicated with error bars in the experimental result figures. Like that, we take into account the varying performance of our cluster at different times of the day, intrinsic to the cloud [13].

The queries are submitted on the master node using the command line tools each query engine provides, and we write the output to a dedicated table which is cleared after every experiment iteration. We restart the query engine under test at the start of every experiment iteration in order to keep it comparable with other iterations.

5. EXPERIMENTAL SETUP

We define a full micro-benchmarking setup by configuring the query engines as well as tuning their data caching policies for optimal performance. We evaluate the most recent stable versions of Shark (v0.9.0), Impala (v1.2.3) and Hive (v.0.12). Many different parameters can influence the query engine's performance. In the following we define the hardware and software configuration parameters used in our experiments.

Hardware: To make a fair performance comparison between the query engines, we use the same cluster setup for each when running the experiments. The cluster consists of 5 `m2.4xlarge` worker Amazon EC2 VMs and 1 `m2.4xlarge` master VM, each having 68.4 GiB of memory, 8 virtual cores and 1.5 TiB instance storage. This cluster has sufficient storage for the real-world and synthetic data, and also has the memory required to allow query engines to benefit from in-memory caching of query inputs or outputs.

The scale of this cluster is comparable to the cluster sizes observed in SMEs and related studies (see Table 1 and [12]). Contrary to other Big Data processing systems, DSQEs (especially Impala and Shark) are tuned for nodes with large amounts of memory, which allows us to use fewer nodes than in comparable studies for batch processing systems to still get comparable (or better!) performance. An additional benefit of this specific cluster setup is the fact it is the same cluster setup used in the AMPLab benchmarks previously performed on older versions of Shark (v0.8.1), Impala (v1.2.3) and Hive (v0.12) [1]. By using the same setup, we can also compare current versions of these query engines with these older versions and see if significant performance improvements have been made.

Software: Hive uses YARN [19] as resource manager while we have used Impala's and Shark's standalone resource managers respectively, because at the time of testing the YARN compatible versions were not mature yet. All query engines under test run on top of a 64-bit Ubuntu 12.04 operating system. We use commonly known best practice configurations without system tuning. Since the queries we run compute results over large amounts of data, the configuration parameters of the distributed file system this data is stored on (HDFS) are crucial. It is therefore imperative that we keep these parameters fixed across all query engines under test. One of these parameters includes the HDFS block size, which we keep to the default of 64 MiB. The number of HDFS files used per dataset, and how these files are structured and compressed is also kept fixed. While more sophisticated file formats are available (such as RCFile [11]) we selected the Sequence file key-value pair format because unlike the more sophisticated formats this is supported by all query engines, and this format uses less disk space than the plain text format. The datasets are compressed on disk using the Snappy compression type, which aims for reasonable compression size while being very fast at decompression.

Table 4: Different ways to configure Shark with caching.

Abbreviation	OS Disk Cache	Input Cache	Output Cache
Cold	No	No	No
OC	No	No	Yes
OSC	Yes	No	No
IC	N/A	Yes	No
OSC+OC	Yes	No	Yes
IC+OC	N/A	Yes	Yes

Each worker has 68.4 GiB of memory available of which we allow a maximum of 60GiB for the query engines under test. This leaves a minimum of 8 GiB of free memory for other processes running on the same system. By doing this we ensure that all query engines under test have an equal amount of maximum memory reserved for them while still allowing the OS disk buffer cache to use more than 8 GiB when the query engine is not using a lot of memory.

Dataset Caching: Another important factor that influences query engine performance is whether the input data is cached or not. By default the operating system will cache files that were loaded from disk in an OS disk buffer cache. Because both Hive and Impala do not have any configurable caching policies available, we will simply run the queries on these two query engines both with and without the input dataset loaded into the OS disk buffer cache. To accomplish this, we perform a `SELECT` query over the relevant tables, so all the relevant data is loaded into the OS disk buffer cache. The query engines under test are restarted after every query as to prevent any other kind of caching to happen that might be unknown to us (e.g., Impala has a non-configurable internal caching system).

In contrast, Shark has more options available regarding caching. In addition to just using the OS disk buffer caching method, Shark also has the option to use an in-memory cached table as input and an in-memory cached table as output. This completely removes the (need for) disk I/O once the system has warmed up. To establish a representative configuration for Shark, we first evaluate the configurations as depicted in Table 4. OS Disk Cache means the entire input tables are first loaded through the OS disk cache by means of a `SELECT`. Input Cache means the input is first cached into in-memory Spark RDDs. Lastly, Output Cache means the result is kept in memory rather than written back to disk.

Figure 1 shows the resulting average response times for running a simple `SELECT *` query using the different possible Shark configurations. Note that no distinction is made between OS disk buffer cache being cleared or not when a cached input table is used, since in this case Shark does not read from disk at all.

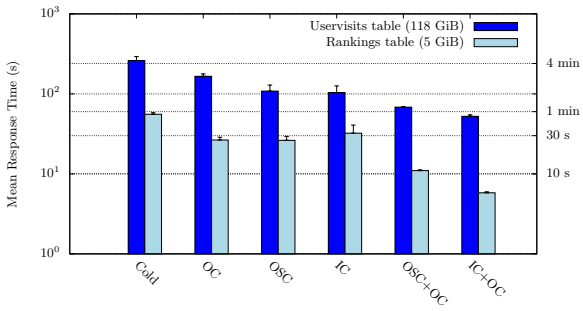

Figure 1: Response time for different Shark caching configurations. Vertical axis is in log-scale.

The configuration with both input and output cached tables enabled (IC+OC) is the fastest setup for both the small and large data set. But the IC+OC and the IC configuration can only be used when the entire input data set fits in memory, which is often not the case with data sets of multiple TBs in size. The second fastest configuration (OSC+OC) only keeps the output (which is often much smaller) in memory and still reads the input from disk. The configuration which yields the worst results is using no caching at all (as expected).

In the experiments in Section 6 we use the optimistic IC+OC configuration when the input data set fits in memory and the OSC+OC configuration when it does not, representing the best-case scenario. In addition the Cold configuration will be used to represent worst-case scenarios.

6. EXPERIMENTAL RESULTS

We evaluate the three query engines selected in Section 3 on the performance aspects described in Section 4.2 using the workloads described in Section 4.1. We evaluate processing power in Section 6.1, resource consumption in Section 6.2, and scalability in Section 6.3.

6.1 Processing Power

We have used the fixed cluster setup with a total of 5 worker nodes and 1 master node as described in Section 5 to evaluate the response time and throughput (defined as the number of input records divided by the response time) of Hive, Impala and Shark on the 6 queries in the workloads. The results of the experiments are depicted in Figure 2. All experiments have been performed 10 times except for Query 4 with Impala since it took simply too long to complete. Only 2 iterations have been performed for this particular query. We used the dataset caching configurations explained in Section 5. For Impala and Hive we used disk buffer caching and no disk buffer caching for the warm and cold situations, respectively. For Shark we used the *Cold* configuration for the cold situation. In addition we used input and output dataset caching (*IC+OC*) for the warm situation of queries 1 to 3, and disk buffer caching and output caching (*OSC+OC*) for the warm situation of queries 4 to 6, since the price input dataset does not entirely fit in memory.

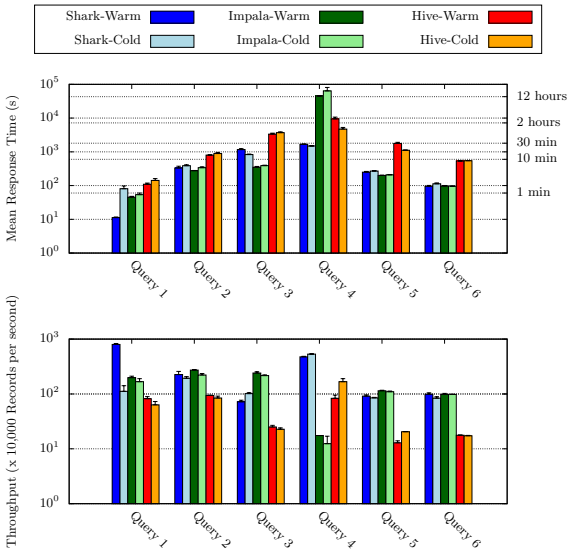

Figure 2: Query Response Time (top) and Throughput (bottom). Vertical axis is in log-scale.

Key Findings:

- Input data caching generally does not cause a significant difference in response times.
- Performance is relatively stable over different iterations.
- Impala and Shark have similar performance and Hive is the worst performer in most cases. There is no overall winner.
- Impala does not handle large input sizes very well (Query 4).

The main reason why Hive is much slower than Impala and Shark is because of the high intermediate disk I/O. Because most queries are not disk I/O bound, data input caching makes little difference in performance. We elaborate on these two findings in more detail in our technical report [20].

In the following we discuss the response times from the 6 queries in a pair-wise manner. We evaluate the data scan queries 1 and 4, the aggregation queries 2 and 5, and the JOIN performance queries 4 and 6 depicted in Figure 2.

1) Scan performance: Shark's response time for query 1 with data input and output caching enabled is significantly better than that of the other query engines (10 seconds vs. 100 seconds for Hive). This is explained by the fact that query 1 is CPU-bound for the *Shark-Warm* configuration, but disk I/O bound for all other configurations as depicted in Figure 3. Since *Shark-Warm* caches both the input and output, and the intermediate data is so small that no spilling is required, no disk I/O is performed at all for *Shark-Warm*.

Results for query 4 for Impala are particularly interesting. The response time of Impala is 12 hours, while the response time of Hive (2 hours) and Shark (30 minutes) are much lower. At the same time, resource utilization of Impala is much lower, as explained in our technical report [20]. No bottleneck can be detected in the resource utilization

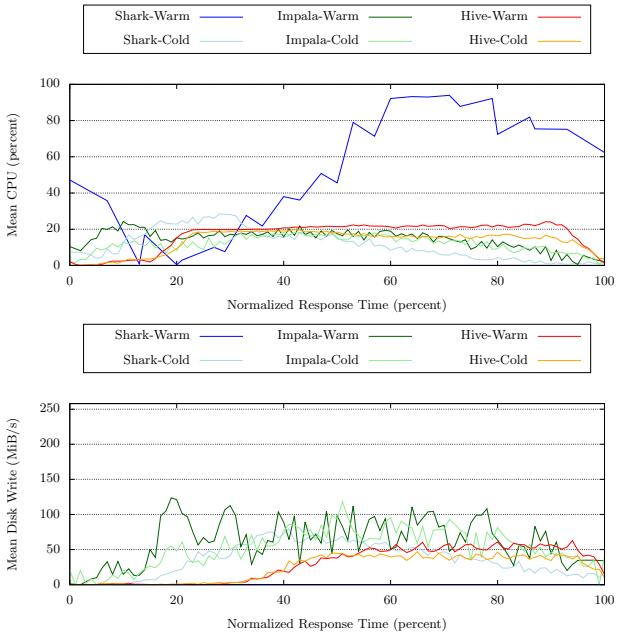

Figure 3: CPU utilization (top) and Disk Write (bottom) for query 1 over normalized response time.

logs and no errors are reported by Impala. After re-running the experiments for Impala on query 4 on a different set of Amazon EC2 instances, similar results are obtained, which makes it highly unlikely an error occured during experiment execution. A more in-depth inspection is needed to get to the cause of this problem, which is out of the scope of our work.

2) Aggregation performance: Both the aggregation query 2 and 5 are handled quite well by all the engines. The response time ranges from 5 to 10 minutes. The main reason why even though query 5 has a much smaller input dataset, the response times are close to the ones of query 2 is that this query is relatively much more compute intensive (see Figure 4).

3) JOIN performance: The query engines perform quite similar on the JOIN queries 3 and 6. A remarkable result is that the fully input and output cached configuration *Shark-Warm* starts to perform worse than its cold equivalent when dataset sizes grow. This is explained in more detail in Section 6.3.

6.2 Resource Consumption

Although the cluster consists of both a master and 5 worker nodes, we only evaluate the resource consumption on the workers, since the master is only used for coordination and remains idle the most of the time. For any of the queries the master never used more than 6 GiB of memory (<10% of total available), never exceeded more than 82 CPU Core Seconds (<0.0005% of the workers' maximum), has negligible disk I/O, and never exceeded total network I/O of 4 GiB (<0.08% of the workers' maximum).

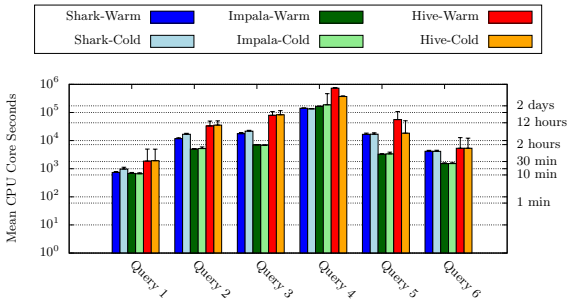

Figure 4: CPU Core Seconds. Vertical axis is in log-scale.

Key Findings:

- Impala is a winner in total CPU consumption. Even when Shark outperforms Impala in terms of response time, Impala is still more CPU efficient (Figure 4).
- All query engines tend to use the full memory assigned to them (See our technical report [20]).
- Disk I/O is as expected significantly higher for the queries without data caching vs. the queries with data caching. Impala has slightly less disk I/O than Shark. Hive ends last (Figure 5).
- Network I/O is comparable in all query engines, with the exception of Hive, which again ends last (Figure 6).

In the following we discuss the resource consumption per query averaged over 5 workers and 10 iterations (50 datapoints per average). We show the CPU Core Seconds per query in Figure 4. This shows how much total CPU a query uses during the query execution. The CPU Core Seconds metric is calculated by taking the area under the CPU utilization graphs of all workers, and then multiplying this number by the number of cores per worker (8 in our setup). For example, a hypothetical query running 2 seconds using 50% of CPU uses 1 CPU Second which is equal to 8 CPU Core Seconds in our case. The results in Figure 4 show that the total amount of time required to complete a query on a single core machine can range between 10 minutes (query 1) to more than 2 days (query 4). A query running on Impala is the most efficient in terms of total CPU utilization, followed by Shark. This is as expected since although Shark and Impala are quite close in terms of response time, Impala is written in C/C++ instead of Scala, which is slightly more efficient.

The total disk I/O per query is depicted in Figure 5. It shows that data caching does make a significant difference in the number of disk reads performed. For example, query 1 on Shark requires less than 100 MiB of disk read when caching is enabled, while it requires more than 1 GiB when the cache is cleared. However, as shown in [20], disk I/O is hardly ever the bottleneck. Although the input datasets are entirely cached in the *Shark-Warm* configuration for queries 1 to 3, disk reading still occurs. This can be explained by the fact that Shark writes shuffle output of intermediate stages to the disk buffer cache (which eventually spills to disk). For

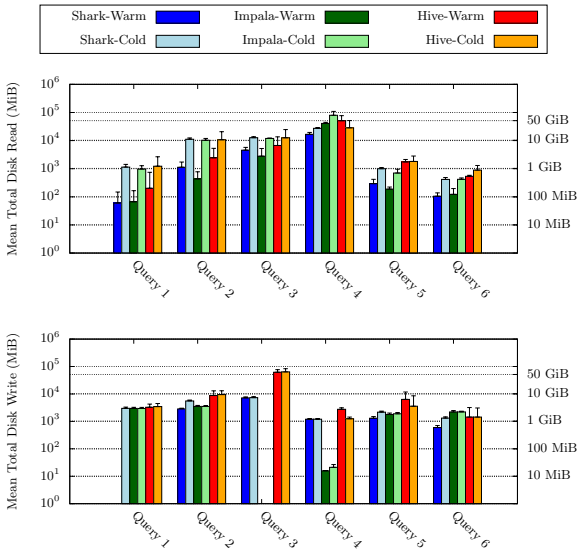

Figure 5: Total Disk Read (top) and Disk Write (bottom). Vertical axis is in log-scale.

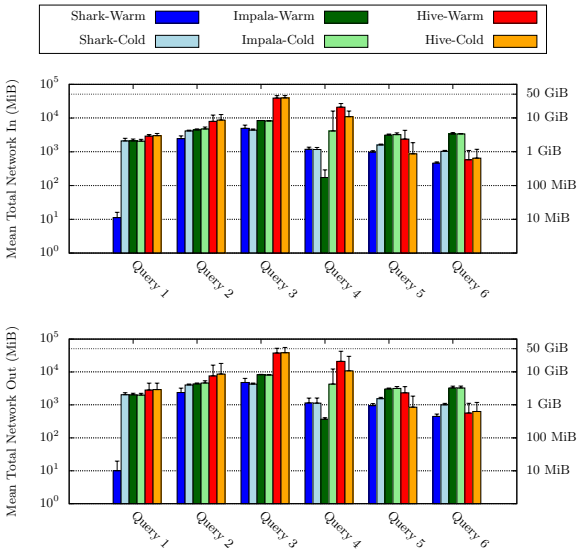

Figure 6: Total Network In (top) and Network Out (bottom). Vertical axis is in log-scale.

queries 4 to 6 less significant differences occur since *Shark-Warm* only uses the OS disk buffer cache mechanism, like Impala and Hive. Note that because the input and output are compressed (compression ratio around 10), generally no more than 10% of the datasets is read or written to disk. Query 1 and 3 have very small output datasets, which makes *Shark-Warm*'s output not visible in the figure for query 1. Similarly for query 3 Impala does not show up at all because Impala does not write intermediate shuffle data to disk.

Figure 6 shows the network I/O per query. Since most network I/O occurs between the workers, the network in and out look similar. Hive has a very variable network output total.

6.3 Scalability

In this section we analyze both the horizontal and the data size scalability of the query engines. We decided to scale horizontally down instead of up because a cluster of 5 nodes of this caliber is already quite expensive for SMEs (more than $7000 a month), and from our experimental results it shows that some queries already do no longer scale well from 4 to 5 worker nodes. We used queries 1 to 3 for data size scaling (since this dataset was already synthetic in the first place) and queries 4 to 6 for horizontal scaling.

Key Findings:

- Both Impala and Shark have good data size scalability on the scan and aggregation queries, whereas the response time has super-linear growth on the JOIN queries as the input size increases. This is as expected, since a JOIN is an operation that requires super-linear time.

- Hive has very good data scalability on all queries, but this is likely due to its large overhead, which dominates the response time at these data input sizes.

- If *Shark-Warm*'s input dataset is too large for its data storage memory, the response time will increase beyond *Shark-Cold* due to swapping.

- Shark and Impala horizontally scale reasonably well on all types of queries up to 3 nodes, whereas Hive only scales well on queries with large input sizes.

- The query engines do not benefit from having more than 3 or 4 nodes in the cluster. Impala even performs worse for query 6 at bigger cluster sizes.

1) Data Size Scalability: The data size scalability of the query engines is depicted in Figure 7. We have sampled subsets from the original dataset of the following sizes: 5%, 10%, 25%, 50%, 75%, 90%, and 100%. We display these along the horizontal axis of the figure in terms of how many times the samples are bigger compared to the 5% sample (1 equals the 5% sample, 20 equals the 100% sample). The vertical axis displays how many times the response time is worse compared to the 5% sample. The colored lines correspond to each of the query engine configurations, whereas the dashed black line depicts the situation where the response time degrades just as fast as the data input size grows. Any query engine performing above this dashed line does not scale very well with the data input size. Query engines performing close to or below the dashed line do scale very well with the input size. Engines that have a much more gentle slope in their performance compared to the dashed line, have their system overhead dominate the response time.

For query 1, all query engines have good data size scalability, but both Shark and Hive have their system overhead dominating the response time. This is because query 1 has a very small input dataset of only 5 GiB. So scalability is not easily shown. For query 2, which has much larger data input size, it shows that both Impala and Shark scale very well. Hive's response time on the other hand, is still being dominated by its system overhead. The query engines have super-linear scalability on query 3, except for Hive, which

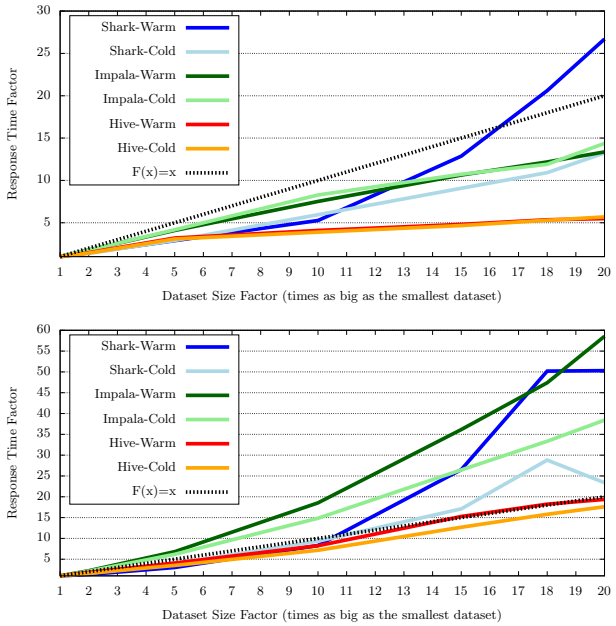

Figure 7: Response time Query 2 (Aggregation; top) and 3 (JOIN; bottom) for different data sizes. Results for Query 1 are in the technical report [20].

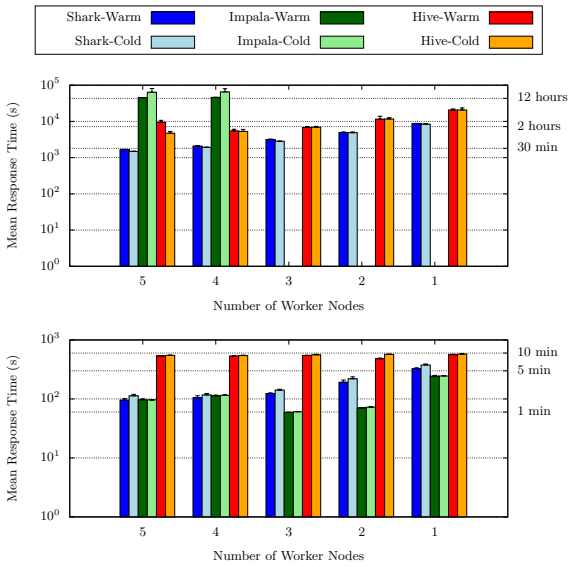

Figure 8: Response time Query 4 (top) and 6 (bottom) for different number of workers (vertical axis is in log-scale). Results for Query 5 are in the technical report [20].

shows close to linear scalability. However, since a JOIN operation takes super-linear time, the response time is expected to grow faster than linearly.

An interesting phenomenom occurs with Shark for query 2 and 3. When the data input size grows and passes 50% of the size of the original dataset (10 on the horizontal axis in the figure), *Shark-Warm* actually starts scaling worse than *Shark-Cold*. This is caused by the fact that Shark allocates only around 34 GiB of the 60 GiB it was assigned for data storage and uses the remaining amount as JVM overhead. This means that the total cluster can only store about 170 GiB of data instead of the 300 GiB it was assigned. The input dataset for query 2 and 3 are close to 120 GiB, which fills some of the worker nodes' fully at the start of query execution. When data exchange occurs between the workers, even more memory is needed for the shuffled data received from the other worker nodes, causing the node to spill some of its input data back to disk.

2) Horizontal Scalability: The horizontal scalability of the query engines is depicted in Figure 8 (note that we are scaling down instead of up). For Impala we only ran 4 and 5 nodes since it already took 12 hours to complete. Both Shark and Hive scale near linearly on the number of nodes. Hive only scales well on query 4 since Hive's Hadoop MapReduce overhead likely outweighs the computation time for query 5 and 6. This is because they have a relatively small input size. Impala actually starts to perform worse on query 6 if more than 3 nodes are added to the cluster. Similarly, both Shark and Impala no longer improve performance after more than 4 nodes are added to the cluster for query 4 and 5 [20].

This remarkable result for horizontal scaling shows that whenever a query is not CPU-bound on a cluster with some number of nodes, performance will not improve any further when adding even more nodes. In the case of network I/O bound queries like query 6, it might even be more beneficial to bind these to a smaller number of nodes so less network overhead occurs.

7. CONCLUSIONS AND FUTURE WORK

In recent years an increasing number of Distributed SQL Query Engines have become available. They all allow for large scale Big Data processing using SQL as interface language. In this work we compare three major query engines (Hive, Impala and Shark) with the requirements of SMEs in mind. SMEs have only little resources available to run their big data analytics on, and cannot afford running a query engine with large overhead.

In this work we have defined an empirical evaluation method to assess the performance of different query engines. Despite not covering all the methodological aspects of a scientific benchmark, this micro-benchmark gives practical insights for SMEs to take informed decisions when selecting a specific tool. Moreover, it can be used to compare current and future engines. The method focuses on three performance aspects: processing power, resource utilization, and scalability.

Using both a real world and a synthetic dataset with representative queries, we evaluate the query engines' performance. We find that different query engines have widely varying performance. Although Hive is almost always outperformed by the other engines, it highly depends on the query type and input size whether Impala or whether Shark is the best performer. Shark can also perform well on queries

with over 500 GiB in input size in our cluster setup, while Impala starts to perform worse for these queries. Overall Impala is the most CPU efficient, and all query engines have comparable resource consumption for memory, disk and network. A remarkable result found is that query response time does not always improve when adding more nodes to the cluster. Remaining key findings can be found at the top of every experiment in Section 6.

This work has been an attempt to get insights in DSQE performance in order to make life easier for SMEs picking the query engine that best suits their needs. Query engine performance in a multi-tenant environment has not been evaluated, and is part of future work.

ACKNOWLEDGEMENTS

This research was supported by the Dutch national program COMMIT.

8. REFERENCES

[1] AMPLab's Big Data Benchmark. `https://amplab.cs.berkeley.edu/benchmark/`. [Online; Last accessed 1st of September 2014].

[2] Apache Hadoop. `http://hadoop.apache.org`. [Online; Last accessed 1st of September 2014].

[3] Apache Tez. `http://tez.apache.org`. [Online; Last accessed 1st of September 2014].

[4] Collectl Resource Monitoring. `http://collectl.sourceforge.net`. [Online; Last accessed 1st of September 2014].

[5] Impala. `http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/`. [Online; Last accessed 1st of September 2014].

[6] Impala Benchmark. `http://blog.cloudera.com/blog/2014/05/new-sql-choices-in-the-apache-hadoop-ecosystem-why-impala-continues-to-lead/`. [Online; Last accessed 1st of September 2014].

[7] Presto. `http://www.prestodb.io`. [Online; Last accessed 1st of September 2014].

[8] A. Floratou, U. F. Minhas, and F. Ozcan. Sql-on-hadoop: Full circle back to shared-nothing database architectures. *Proceedings of the VLDB Endowment*, 7(12), 2014.

[9] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 international conference on Management of data*, pages 1197–1208, 2013.

[10] M. Hausenblas and J. Nadeau. Apache Drill: Interactive Ad-Hoc Analysis at Scale. *Big Data*, 2013.

[11] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1199–1208, 2011.

[12] T. Hegeman, B. Ghit, M. Capota, J. Hidders, D. Epema, and A. Iosup. The BTWorld Use Case for Big Data Analytics: Description, Mapreduce Logical Workflow, and Empirical Evaluation. In *Big Data, 2013 IEEE International Conference on*, pages 622–630. IEEE, 2013.

[13] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104–113, 2011.

[14] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

[15] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178, 2009.

[16] M. Poess, R. O. Nambiar, and D. Walrath. Why you should run TPC-DS: a workload analysis. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1138–1149. VLDB Endowment, 2007.

[17] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas. Nobody ever got fired for using Hadoop on a cluster. In *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing*, page 2. ACM, 2012.

[18] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[19] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[20] S. v. Wouw, J. Viña, D. Epema, and A. Iosup. An Empirical Performance Evaluation of Distributed SQL Query Engines: Extended Report. *Delft University of Technology, Tech Rep.*, PDS-2014-002, 2014. `http://www.pds.ewi.tudelft.nl/research-publications/technical-reports/2014/`.

[21] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM International Conference on Management of Data*, pages 13–24, 2013.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2, 2012.

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.