CoreBigBench: Benchmarking Big Data Core Operations

Todor Ivanov[1], Ahmad Ghazal[2], Alain Crolotte[3], Pekka Kostamaa[3], Yoseph Ghazal[4]

1. Frankfurt Big Data Lab, Goethe University, Germany
2. Facebook Corporation, Seattle, WA, USA
3. Teradata Corporation, El Segundo, CA, USA
4. University of California, Irvine, CA, USA

Outline

- Motivation
- Background
- CoreBigBench Specification
  - Data Model
  - Workload
  - Proof of Concept
- Conclusion

Motivation

- Growing number of emerging Big Data systems

--> high number of new Big Data benchmarks

- ***Micro-benchmarks*** that focus on testing specific functionality or single operations:
  - WordCount [W1], Pi [P1], Terasort [T1], TestDFSIO [D1]
  - HiveBench [A2010], HiBench [H1], AMP Lab Benchmark [A1], HiveRunner [H2]
  - SparkBench [S1], Spark-sql-perf [S2]

- ***End-to-end application benchmarks*** focus on a business problem and simulate a real world application with a data model and workload:
  - BigBench [G2013] and BigBench V2 [G2017]

End-to-End Application Benchmarks

BigBench/TPCx-BB [G2013]

- Technology agnostic, analytics, application-level Big Data benchmark.
- On top of TPC-DS (decision support on retail business)
- Adding semi-structured and unstructured data.
- ***Focus on***: Parallel DBMS and MR engines (Hadoop, Hive, etc.).
- ***Workload***: 30 queries
  - Based on big data retail analytics research
  - 11 queries from TPC-DS
- Adopted by TPC as TPCx-BB
- Implementation in HiveQL and Spark MLlib.

BigBench V2 [G2017]

- a major rework of BigBench
- separate from TPC-DS and takes care of ***late binding***.
- ***New simplified data model*** and late binding requirements.
- Custom made ***scale factor-based data generator*** for all components.
- ***Workload***:
  - All 11 TPC-DS queries are ***replaced*** with new queries in BigBench V2.
  - New queries with similar business questions - ***focus on analytics on the semi-structured web-logs.***

What is not covered by micro and application benchmarks?

- Both micro-benchmarks and application benchmarks can be tuned for the specific application they are testing

- There is a need for Big Data ***White box (or core engine operations) benchmarking***
- Examples of core operations
  - Table scans, two way joins, aggregations and window functions
  - Common User Defined Functions (UDFs) like sessioinze, path, ..
- Core operators benchmarking also helps with performance regression of big data system
- Not replacement for application level benchmarking
  - Complements them

- Similar problem for DBMS was addressed by Crolotte & Ghazal [C&G2010] covering: scans, aggregations, joins and other core relational operators

CoreBigBench Data Model
inspired by BigBench V2 [G2017]

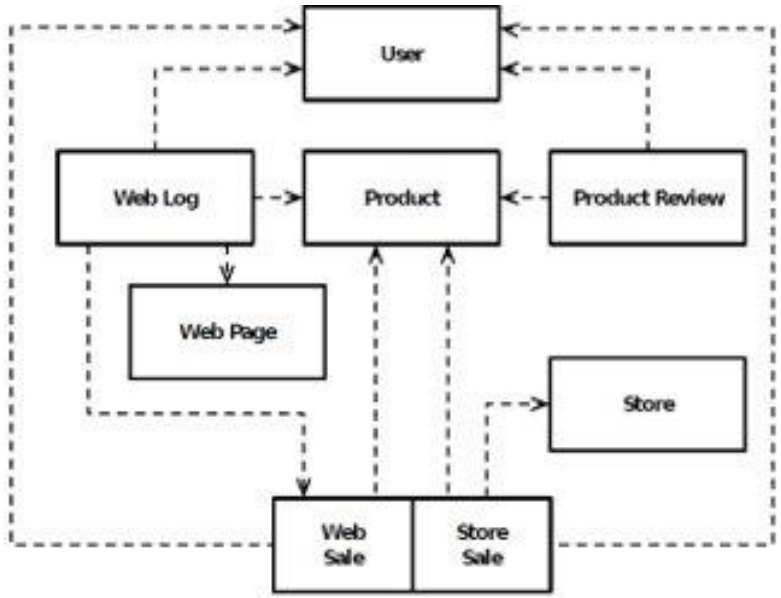
- **New simplified (star-schema) data model**
  - Structured part consisting of 6 tables
- ***Semi-structured part (JSON)***
  - Key-value pairs representing user clicks
  - Keys corresponding to structured part and random keys and values
  - Example :

    <user,user1> <time,t1> <webpage,w1>

    <product,p1>

    <key1,value1> <key2,value2> ...

    <key100,value100>



- 1 – many relationship : ----➔
- **Semi-structured : key-value WebLog**
- Un-structured: Product Reviews

- ***Unstructured part (text)***: Product reviews similar to the one in BigBench
- Custom made ***scale factor-based data generator*** for all components.

Summary of Workload Queries

- **Variety of core operations on structured, semi structured and unstructured data**
- **Scans**
  - Q**1 -** Q**5** cover ***variations of scans*** with different selectivity's on structured and semi-structured data
- **Aggregations**
  - Q**6 -** Q**12** cover ***different aggregations*** on structured and semi-structured data
- **Window functions**
  - Q**13 -** Q**16** cover variations of ***window functions*** with different data partitioning
- **Joins**
  - Q**17 -** Q**18** cover ***binary joins with partitioning*** variations on structured and unstructured data
- **Common Big Data functions**
  - Q**19 -** Q**22** cover ***four UDFs*** (sessionize, path, sentiment analysis and K-means) on structured, semi-structured and unstructured data

Queries	Text Descriptions
Q1	List all store sold products (items) together with their quantity. This query does a full table scan of the store data.
Q2	List all products (items) sold together in stores with their quantity sold between 2013-04-21 and 2013-07-03. This query tests scans with low selectivity 10% filter.
Q3	List all products (items) together with their quantity sold between 2013-01-21 and 2014-11-10. Similar to $Q2$ but with high selectivity (90%).
Q4	List names of all visited web pages. This query tests parsing the semi-structured web logs and scanning the parsed results. The query requires only one key from the web logs.
Q5	Similar to $Q4$ above but returning a bigger set of keys. This variation measures the ability of the underlying system for producing a bigger schema out of the web logs.
Q6	Find total number of all stores sales. This query covers basic aggregations with no grouping. The query involves scanning store sales and to get the net cost of aggregations we deduct the cost of $Q1$ from this query run time.
Q7	Find total number of visited web pages. This query requires parsing and scanning the web logs and therefore it is adjusted by subtracting $Q4$ from its run time.
Q8	Find total number of store sales per product (item). This query is adjusted similar to $Q6$.
Q9	Find number of clicks per product (item). This query also requires parsing the web logs and can be adjusted similar to $Q7$.
Q10	Find a list of aggregations from store sales by customer. Aggregations include number of transactions, maximum and minimum quantities purchased in an order. This query also finds correlations between stores and products (items) purchased by a a customer. The purpose of this query is to test cases of a big set of aggregations.
Q11	This query has a simple objective like $Q10$ but applied to web logs. Again, the query need to be adjusted by removing the parsing and scan cost represented by $Q4$.

Queries	Text Descriptions
Q12	$Q12$ is the same as $Q8$ but on store sales partitioned by customer (different than the group key). The shuffle cost is computed as run-time of $Q12$ minus run-time of $Q8$.
Q13	Find row numbers of store sales records order by store id.
Q14	Find row numbers of web log records ordered by timestamp of clicks.
Q15	Find row numbers of store sales records order by store id for each customer. This query is similar to $Q13$ but computes the row numbers for each customer individually.
Q16	Same as $Q14$ where row numbers are computed per customer.
Q17	Find all store sales with products that were reviewed. This query is a join between the stores sales and product reviews both partitioned on item ID.
Q18	Same as $Q17$ with different partitioning. (Table store sales is partitioned on customer ID and no partitioning on table product reviews.)
Q19	List all customers that spend more than 10 minutes on the retailer web site. This query involves finding all sessions of all users and filtering them to those which are 10 minutes of less.
Q20	Find the 5 most popular web page paths that lead to a purchase. This query is based on finding paths in clicks that lead to purchases, aggregating the results and finding the top 5.
Q21	For all products, extract sentences from its product reviews that contain Positive or Negative sentiment and display the sentiment polarity of the extracted sentences.
Q22	Cluster customers into book buddies/club groups based on their in-store book purchasing histories. After model of separation is build, report for the analyzed customers to which "group" they were assigned.

Proof Of Concept

- **Objective --> show the feasibility of CoreBigBench (no serious tuning effort)**
- **Setup**
  - 4 node cluster (Ubuntu Server)
  - Cloudera CDH 5.16.2 + Hive 1.10
  - Data Generation with Scale Factor = 10
  - Late binding on the JSON file

```
CREATE EXTERNAL TABLE IF NOT EXISTS
web_logs (line string)
ROW FORMAT DELIMITED LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION 'hdfsPath/web_logs/clicks.json';
```

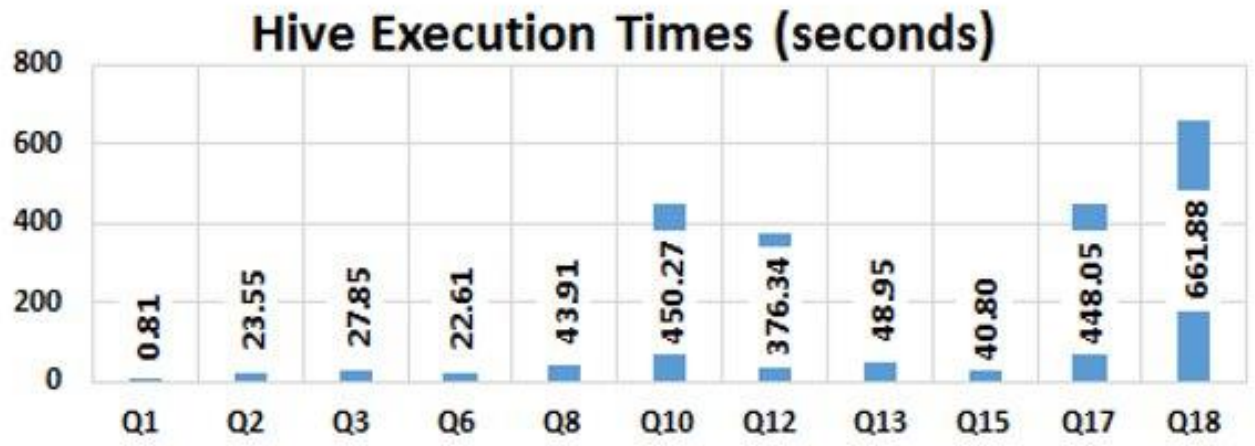
	Scale Factor 1		Scale Factor 10	
Table Name	**Data Size**	**Loading (sec.)**	**Data Size**	**Loading (sec.)**
user	256 KB	16.63	456 KB	19.19
product	39 KB	13.16	46 KB	17.33
product review	6.9 MB	15.20	13 MB	13.40
web log	22 GB	0.02	40 GB	0.04
web page	687 B	13.05	687 B	14.80
web sale	9.7 MB	16.10	18 MB	14.63
store sale	10 MB	15.03	19 MB	15.20
store	6.6 KB	13.88	6.2 KB	13.67
Total:	22.03 GB	103.08	40.05 GB	108.2

- Query implementation in Hive is available in github: https://github.com/t-ivanov/CoreBigBench

Queries on Structured Data

- $Q2$: List all products (items) sold together in stores with their quantity sold between 2013-04-21 and 2013-07-03. This query tests scans with low selectivity 10% filter.

```
SELECT ss_item_id, ss_quantity FROM store_sales
WHERE to_date(ss_ts) >= '2013-04-21'
AND to_date(ss_ts) < '2013-07-03';
```

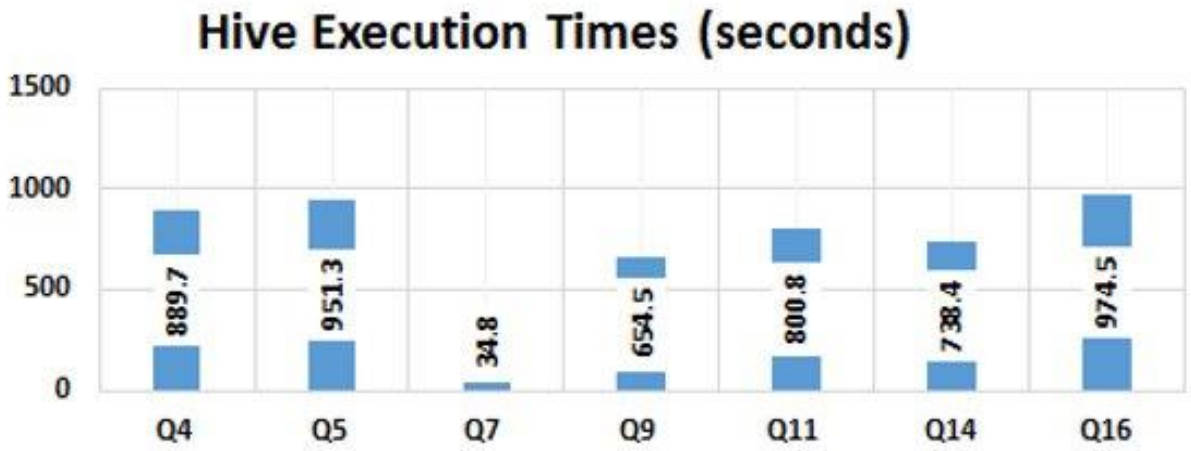


- $Q1$ performs a full table scan of the store data.
- We deduct the $Q1$ operation time for queries $Q6$ to $Q15$ operating on the structured data.
- The geometric mean of all query times in this group is 62.07 seconds.

Queries on Semi-structured Data

- $Q4$: List names of all visited web pages. This query tests parsing the semi-structured web logs and scanning the parsed results. The query requires only one key from the web logs.

```
SELECT wl_webpage_name
FROM web_logs
lateral view json_tuple(
web_logs.line,'wl_webpage_name'
)logs as wl_webpage_name
WHERE wl_webpage_name IS NULL;
```



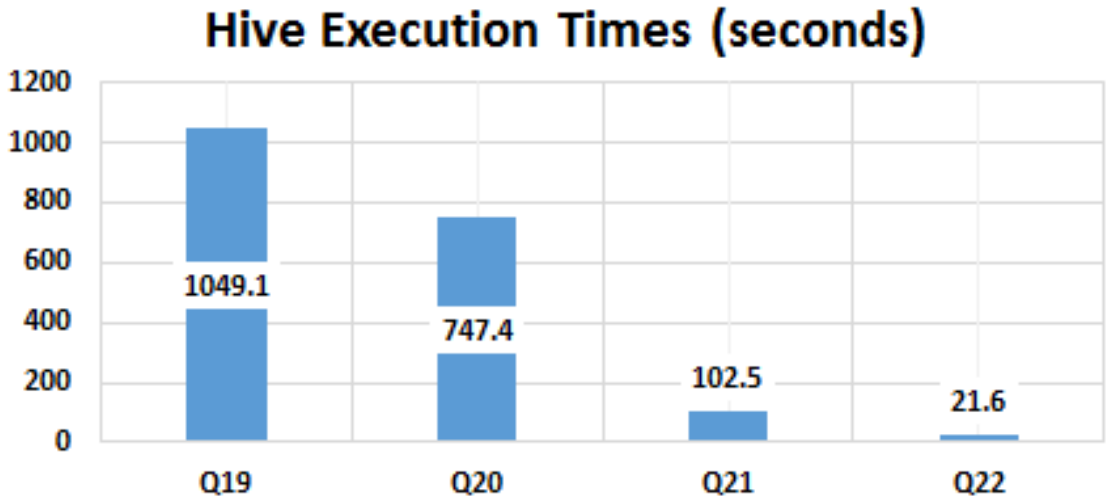
- $Q4$ performs a simple scan operation that involves parsing all the JSON records on the fly and extracting only the necessary attributes.
- We deduct $Q4$ operation time from all other queries in this group.
- The geometric mean of all query times in this group is 525.88 seconds.

Queries with UDF Functions

- $Q22$: Cluster customers into book buddies/club groups based on their in-store book purchasing histories. After model of separation is build, report for the analysed customers to which "group" they where assigned.

```
set cluster_centers=8;
set clustering_iterations=20;

SELECT kmeans(
collect_list(array(id1, id3, id5, id7, id9,
id11, id13, id15, id2, id4, id6, id8, id10,
id14, id16)),
${hiveconf:cluster_centers},
${hiveconf:clustering_iterations}) AS out
FROM q22_prep_data;
```



- $Q19$ and $Q20$ operate on the semi-structured key-value data and we deduct the basic key-value scan $Q4$ operation time.
- $Q21$ and $Q22$ operate on the structured and unstructured data and we deduct the simple table scan $Q1$ operation time.
- The geometric mean of all query times in this group is 204.15 seconds.

Conclusion

- CoreBigBench
  - is a benchmark assessing the performance of core (basic) operations of big data engines like scans, two way joins, UDF functions;
  - consists of 22 queries applied on sales data, key-value web logs and unstructured product reviews (inspired by BigBench V2);
  - queries have textual definitions and reference implementation in Hive.
- CoreBigBench can be used for
  - complimentary to end-to-end benchmarks like BigBench;
  - regression testing of commercial Big Data engines.
- In future the CoreBigBench can be extended to include ETL, which is very basic functionality for Big Data engines.

Thank you for your attention!

- **Acknowledgments.** This work has been partially funded by the European Commission H2020 project DataBench - Evidence Based Big Data Benchmarking to Improve Business Performance, under project No. 780966. This work expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this work. The authors thank all the participants in the project for discussions and common work.

www.databench.eu

DataBench

References (1)

- [C&G2010] Alain Crolotte and Ahmad Ghazal. 2010. Benchmarking Using Basic DBMS Operations. In 2nd TPC Technology Conference, TPCTC 2010, Singapore, September 13-17, 2010
- [G2013] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards An Industry Standard Benchmark for Big Data Analytics. In SIGMOD 2013. 1197–1208.
- [G2017] Ahmad Ghazal, Todor Ivanov, Pekka Kostamaa, Alain Crolotte, Ryan Voong, Mohammed Al-Kateb, Waleed Ghazal, and Roberto V. Zicari. 2017. BigBench V2: The New and Improved BigBench. In ICDE 2017, San Diego, CA, USA, April 19-22.
- [W1] WordCount. https://cwiki.apache.org/confluence/display/HADOOP2/WordCount
- [T1] TeraSort. http://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html
- [P1] Package hadoop.examples.pi. http://hadoop.apache.org/docs/r0.23.11/api/org/apache/hadoop/examples/pi/package-summary.html
- [D1] DFSIO benchmark. http://svn.apache.org/repos/asf/hadoop/common/tags/release-0.13.0/src/test/org/apache/hadoop/fs/TestDFSIO.java

References (2)

- [A2010] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A comparison of approaches to large-scale data analysis. In Proc. of the ACM SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009. ACM, 165–178
- [A1] AMP Lab Big Data Benchmark. https://amplab.cs.berkeley.edu/benchmark/
- [S1] SparkBench. https://bitbucket.org/lm0926/sparkbench
- [S2] Spark-SQL-perf. https://github.com/databricks/spark-sql-perf
- [H1] HiBench Suite. https://github.com/intel-hadoop/HiBench
- [H2] HiveRunner. https://github.com/klarna/HiveRunner