What is BigBench?
Big Data Benchmarking
About Me

• **Since 2009 research and thesis at University of Passau**
  • Parallel synthetic data generation
  • ETL benchmark development

• **Since 2013 CTO of bankmark**
  • Startup company in Passau, Germany

• **Current research**
  • Data generation
  • Big data analysis and machine learning
  • Big data benchmarking
This Talk’s Agenda

- Big data benchmarking in general
- BigBench – what it is and what it is not
- BigBench in detail
- Experiments – runtime and scaling characteristics
Big Data Benchmarking
In the beginning there was Sorting...

• Early popularity of TeraSort
  • Sortbenchmark.org
  • GraySort, MinuteSort, TeraSort, CloudSort, ...

• Pluses:
  • Simple benchmark—easy to run
  • Scalable model
  • Good for “shaking out” large hardware configurations

• Minuses:
  • (Not standardized) -> now there is TPCx-HS which is actually terasort
  • “Flat” data distribution (no skews) -> In work
  • Not application-level
  • Limited applicability

• Require more than just sorting for a big data benchmark
**Big Data Benchmarking Issues**

- **Motivation**
  - Lack of standards
  - Vendor and customer frustration
  - Opportunity to define the set of big data application “classes”, or range of scenarios

- **Which Big Data?**
  - The V’s; Volume, Velocity, Variety, (Veracity, Value, Viability, …)
  - Warehouse vs pipelines of processing; query processing vs analytics

- **Different approaches to benchmarking**
  - How does industry standard benchmarking work?
  - TPC vs SPEC model (application vs component)
  - The need for audited results
Types of Big Data Benchmarks

• Micro-benchmarks. To evaluate specific lower-level, system operations
  • E.g., A Micro-benchmark Suite for Evaluating HDFS Operations on Modern Clusters, Panda et al, OSU

• Functional benchmarks. Specific high-level function.
  • E.g. Sorting: Terasort
  • E.g. Basic SQL: Individual SQL operations, e.g. Select, Project, Join, Order-By, ...

• Genre-specific benchmarks. Benchmarks related to type of data
  • E.g. Graph500. Breadth-first graph traversals

• Application-level benchmarks
  • Measure system performance (hardware and software) for a given application scenario—with given data and workload
BigBench – what it is and what it is not
The BigBench Proposal

• End-to-end, application level benchmark
• Focused on Parallel DBMS and MR engines
  • Framework agnostic
  • SW based reference implementation

• History
  • Launched at 1st WBDB, San Jose, 2012
  • Published at SIGMOD 2013
  • Full kit at WBDB 2014
  • TPC BigBench Working Group in 2015

• Collaboration with Industry & Academia
  • First: Teradata, University of Toronto, Oracle, InfoSizing
  • Now: bankmark, CLDS, Cisco, Cloudera, Hortonworks, Infosizing, Intel, Microsoft, Oracle, Pivotal, SAP, IBM, UoFT, …
BigBench - 2013

- Collaborative industry effort
  - Sigmod 2013
  - Address 3V’s of big data
  - Very first concept for a big data benchmark specification
  - Wide industry support

- Use case sampling
  - Retail use case example
  - End to end and component

- Framework agnostic
  - Well defined specification
  - SW based reference implementation
Benchmark Process

• Adapted to batch systems
• No trickle update
• Measured processes
  • Loading
  • Power Test (single user run)
  • Throughput Test I (multi user run)
  • Data Maintenance
  • Throughput Test II (multi user run)
• Result
  • Additive Metric
Workload

• Business functions (adapted from McKinsey report)
  • Marketing
    • Cross-selling, customer micro-segmentation, sentiment analysis, enhancing multichannel consumer experiences
  • Merchandising
    • Assortment optimization, pricing optimization
  • Operations
    • Performance transparency, product return analysis
  • Supply chain
    • Inventory management
  • Reporting (customers and products)
BigBench – What its not

• Not your first benchmark to run on a new cluster!
• Not your first benchmark to run if you are new to big data
• Not a 100% utilization benchmark (100% CPU, 100% I/O...)
• Why?
  • Every query requires its own custom mixture of functionality from your big data
    stack. Every phase of each query has its own workload pattern in terms of CPU/ I/O
  • A lot of parts of your big data stack have to work together and have to be configured
    appropriately
  • Hard to find the culprit if something is slow or does not work -> requires that you
    are already familiar with your cluster
  • If you tune for Query X, Query Y will run slow/fail (to get a good overall
    configuration, not tune for a single special case)
BigBench – What it is

• Test your cluster with multiple relevant high level use cases
• On a very large scale
• Find corner cases and issues early (e.g. over/under tuning in a specific direction) before your production team encounters them
• Simulate effects of concurrent access to a cluster and the impact on runtime and resource utilization
• You can
  • Test/tune your cluster setup
  • Relate to your own business through one or more of the high level queries
  • Do some scale up/scale out/throughput tests of you existing setup
  • Compare different Hadoop stacks
  • Compare different cluster configurations
  • Compare different hardware configurations
  • ....
Great! Lets run it

- **Perquisites**
  - Get a nice Hadoop cluster with Hadoop MR2, Hive, Mahout, Python and Java 1.7
  - Clone the Github repository (and follow the readme from the repository)
    - https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench
  - Add you cluster specific settings in BigBench/conf/userSettings.conf
  - Choose a scaling factor, e.g: SF1 ~ 1GB (you want to start low at your first runs!)
  - Figure out number of yarn containers/map tasks you want to run the DataGen stage with
    - DataGen Map tasks == number of partitions per table! Will Influence load stage performance.
  - Run it! ./bin/bigBench runBenchmark -m 4 -f 1 -s 2
    - Check out logs and results in folder : BigBench/logs/
  - Add the –b option for a verbose output to see Hadoop/hive stdout!
  - run it again, but maybe only a specific query : ./bin/bigBench runQuery 12 or skip certain benchmark phases to debug/tune your cluster (see built in help of BigBench driver and the documentation on github)
BigBench in Detail
Derived from TPC-DS

- Multiple snowflake schemas with shared dimensions
- 24 tables with an average of 18 columns
- 99 distinct SQL ‘99 queries with random substitutions
- Representative skewed database content
- Sub-linear scaling of non-fact tables
- Ad-hoc, reporting, iterative and extraction queries
- ETL-like data maintenance
BigBench Data Model

- **Structured:** TPC-DS + market prices
- **Semi-structured:** website click-stream
- **Unstructured:** customers' reviews
Data Model – 3 Vs

- **Variety**
  - Different schema parts

- **Volume**
  - Based on scale factor
  - Similar to TPC-DS scaling, but continuous
  - Weblogs & product reviews also scaled

- **Velocity**
  - Refresh for all data with different velocities
Scaling

- **Continuous scaling model**
  - Realistic

- **SF 1 ~ 1 GB**

- **Different scaling speeds**
  - Adapted from TPC-DS
    - Static
    - Square root
    - Logarithmic
    - Linear (LF)

---

\[
LF = SF + (SF - (\log_3(SF) \times \sqrt{SF})) = 2SF - \log_3(SF) \times \sqrt{SF}
\]

---

<table>
<thead>
<tr>
<th>Table Name</th>
<th># Rows SF 1</th>
<th>Bytes/Row</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>109573</td>
<td>141</td>
<td>static</td>
</tr>
<tr>
<td>time</td>
<td>86400</td>
<td>75</td>
<td>static</td>
</tr>
<tr>
<td>ship_mode</td>
<td>20</td>
<td>60</td>
<td>static</td>
</tr>
<tr>
<td>household_demographics</td>
<td>7200</td>
<td>22</td>
<td>static</td>
</tr>
<tr>
<td>customer_demographics</td>
<td>1920800</td>
<td>40</td>
<td>static</td>
</tr>
<tr>
<td>customer</td>
<td>100000</td>
<td>138</td>
<td>square root</td>
</tr>
<tr>
<td>customer_address</td>
<td>50000</td>
<td>107</td>
<td>square root</td>
</tr>
<tr>
<td>store</td>
<td>12</td>
<td>261</td>
<td>square root</td>
</tr>
<tr>
<td>warehouse</td>
<td>5</td>
<td>107</td>
<td>logarithmic</td>
</tr>
<tr>
<td>promotion</td>
<td>300</td>
<td>132</td>
<td>logarithmic</td>
</tr>
<tr>
<td>web_page</td>
<td>60</td>
<td>134</td>
<td>logarithmic</td>
</tr>
<tr>
<td>item</td>
<td>18000</td>
<td>308</td>
<td>square root</td>
</tr>
<tr>
<td>item_marketprice</td>
<td>90000</td>
<td>43</td>
<td>square root</td>
</tr>
<tr>
<td>inventory</td>
<td>23490000</td>
<td>19</td>
<td>square root * logarithmic</td>
</tr>
<tr>
<td>store_sales</td>
<td>810000</td>
<td>143</td>
<td>linear</td>
</tr>
<tr>
<td>store_returns</td>
<td>40500</td>
<td>125</td>
<td>linear</td>
</tr>
<tr>
<td>web_sales</td>
<td>810000</td>
<td>207</td>
<td>linear</td>
</tr>
<tr>
<td>web_returns</td>
<td>40500</td>
<td>154</td>
<td>linear</td>
</tr>
<tr>
<td>web_clickstreams</td>
<td>6930000</td>
<td>27</td>
<td>linear</td>
</tr>
<tr>
<td>product_reviews</td>
<td>98100</td>
<td>670</td>
<td>linear</td>
</tr>
</tbody>
</table>
Generating Big Data

- Data generation based on PDGF (Parallel Data Generation Framework)
  - Repeatable computation
  - Enables independent generation of every value in the data set
  - Enables independent re-generation of every value for references
  - High quality data – distributed, in parallel (CPU/Nodes), in the correct format
  - Large data – terabytes, petabytes
  - Can write directly into HDFS

- Based on xml configuration
  - Schema – data model
  - Format – CSV, SQL statements, ...
  - Distribution – multi-core, multi-node, partially

- PDGF generates
  - High quality data – distributed, in parallel, in the correct format
  - Large data – terabytes, petabytes

- You want to contribute and modify the BB dataset yourself?
  - Just ask bankmark for an evaluation license of PDGF!
Workload

- **Workload - queries**
  - 30 “queries” or use-cases
  - No required syntax – implement it on your own big data stack!
    - first implementation was in Aster SQL MR
  - Current kit implemented in Hive + HadoopMR2 + Mahout + OpenNLP
  - Specified in English (sort of)
Query 1

Find products that are sold together frequently in given stores. Only products in certain categories sold in specific stores are considered and "sold together frequently" means at least 50 customers bought these products together in a transaction.
HiveQL Query 1

```sql
SELECT pid1, pid2, COUNT (*) AS cnt
FROM ( 
    FROM ( 
        SELECT s.ss_ticket_number AS oid, s.ss_item_sk AS pid
        FROM store_sales s
        INNER JOIN item i ON s.ss_item_sk = i.i_item_sk
        WHERE i.i_category_id in (1, 2, 3) and s.ss_store_sk in (10, 20, 33, 40, 50)
        CLUSTER BY oid
    ) q01_map_output
    REDUCE q01_map_output.oid, q01_map_output.pid
    USING 'java -cp bigbenchqueriesmr.jar:hive-contrib.jar de.bankmark.bigbench.queries.q01.Red'
    AS (pid1 BIGINT, pid2 BIGINT)
) q01_temp_basket
GROUP BY pid1, pid2
HAVING COUNT (pid1) >= 50
ORDER BY pid1, cnt, pid2;
```
Run a Different Engine than Hive

• Execute benchmark with different engine:
  • a) Big-Bench/conf/userSettings.conf
    • export BIG_BENCH_DEFAULT_ENGINE="hive"
  • b) Start BigBench driver with –e *engine* option

• Add a new engine:
  • Engine name == folder name in BigBench/engines/
  • Copy one of the existing engine folders (e.g. hive) and start modifying/extending it.
  • Make sure to change the paths in *engine*/conf/engineSettings.conf
## Generic Characteristics

<table>
<thead>
<tr>
<th>Data Sources</th>
<th>#Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td>18</td>
<td>60%</td>
</tr>
<tr>
<td>Semi-structured</td>
<td>7</td>
<td>23%</td>
</tr>
<tr>
<td>Un-structured</td>
<td>5</td>
<td>17%</td>
</tr>
</tbody>
</table>

## Hive Implementation Characteristics

<table>
<thead>
<tr>
<th>Query Types</th>
<th>#Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure HiveQL</td>
<td>14</td>
<td>46%</td>
</tr>
<tr>
<td>Mahout</td>
<td>5</td>
<td>17%</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>5</td>
<td>17%</td>
</tr>
<tr>
<td>Custom MR</td>
<td>6</td>
<td>20%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>Input Datatype</th>
<th>Processing Model</th>
<th>Query</th>
<th>Input Datatype</th>
<th>Processing Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Structured</td>
<td>Java MR</td>
<td>#16</td>
<td>Structured</td>
<td>Java MR (OpenNLP)</td>
</tr>
<tr>
<td>#2</td>
<td>Semi-Structured</td>
<td>Java MR</td>
<td>#17</td>
<td>Structured</td>
<td>HiveQL</td>
</tr>
<tr>
<td>#3</td>
<td>Semi-Structured</td>
<td>Python Streaming MR</td>
<td>#18</td>
<td>Unstructured</td>
<td>Java MR (OpenNLP)</td>
</tr>
<tr>
<td>#4</td>
<td>Semi-Structured</td>
<td>Python Streaming MR</td>
<td>#19</td>
<td>Structured</td>
<td>Java MR (OpenNLP)</td>
</tr>
<tr>
<td>#5</td>
<td>Semi-Structured</td>
<td>HiveQL</td>
<td>#20</td>
<td>Structured</td>
<td>Java MR (Mahout)</td>
</tr>
<tr>
<td>#6</td>
<td>Structured</td>
<td>HiveQL</td>
<td>#21</td>
<td>Structured</td>
<td>HiveQL</td>
</tr>
<tr>
<td>#7</td>
<td>Structured</td>
<td>HiveQL</td>
<td>#22</td>
<td>Structured</td>
<td>HiveQL</td>
</tr>
<tr>
<td>#8</td>
<td>Semi-Structured</td>
<td>HiveQL</td>
<td>#23</td>
<td>Structured</td>
<td>HiveQL</td>
</tr>
<tr>
<td>#9</td>
<td>Structured</td>
<td>HiveQL</td>
<td>#24</td>
<td>Structured</td>
<td>HiveQL</td>
</tr>
<tr>
<td>#10</td>
<td>Unstructured</td>
<td>Java MR (OpenNLP)</td>
<td>#25</td>
<td>Structured</td>
<td>Java MR (Mahout)</td>
</tr>
<tr>
<td>#11</td>
<td>Unstructured</td>
<td>HiveQL</td>
<td>#26</td>
<td>Structured</td>
<td>Java MR (Mahout)</td>
</tr>
<tr>
<td>#12</td>
<td>Semi-Structured</td>
<td>HiveQL</td>
<td>#27</td>
<td>Unstructured</td>
<td>Java MR (OpenNLP)</td>
</tr>
<tr>
<td>#13</td>
<td>Structured</td>
<td>HiveQL</td>
<td>#28</td>
<td>Unstructured</td>
<td>Java MR (Mahout)</td>
</tr>
<tr>
<td>#14</td>
<td>Structured</td>
<td>HiveQL</td>
<td>#29</td>
<td>Structured</td>
<td>Python Streaming MR</td>
</tr>
<tr>
<td>#15</td>
<td>Structured</td>
<td>Java MR (Mahout)</td>
<td>#30</td>
<td>Semi-Structured</td>
<td>Python Streaming MR</td>
</tr>
</tbody>
</table>
Metric

• **Throughput metric**
  • BigBench queries per hour

• **Number of queries run**
  • \(30 \times (2S + 1)\)

• **Measured times**
  \(T_L\): Execution time of the loading process;
  \(T_P\): Execution time of the power test;
  \(T_{TT_1}\): Execution time of the first throughput test;
  \(T_{DM}\): Execution time of the data maintenance task.
  \(T_{TT_2}\): Execution time of the second throughput test;

• **Metric**
  \[
  BBQP_H = \frac{30 \times 3 \times 3600}{T_L + T_P + \frac{T_{TT_1}}{S} + T_{DM} + \frac{T_{TT_2}}{S}}
  \]

  \[
  BBQP_H = \frac{30 \times 3 \times S \times 3600}{S \times T_L + S \times T_P + T_{TT_1} + S \times T_{DM} + T_{TT_2}}
  \]

Source: www.wikipedia.de

6/16/2015
BigBench Driver BigBench/bin/bigBench

- Automation of the benchmark process via simple command line
- Run different engines and benchmark configurations
- Modularized
- Orchestrate the throughput test by resource management (simulate multiuser access on single user)
- Various customization and configuration options (like loading benchmark specific engine settings)
- Ability to run only specific benchmark phases or specific queries (or even parts of queries to debug them)
- Collect and log statistics and errors
- Collect stdout & stderr logs for each query and each phase -> zip results
- Clean up (temporary and long live data)
- **NEW**: Basic result validation!
Experiments – Runtime and Scaling Characteristics
BigBench Experiments

• Tests on
  • Cloudera CDH 5.0/5.3, Pivotal GPHD-3.0.1.0, IBM InfoSphere BigInsights

• In progress: Spark, Stinger, ...

• 3 Clusters (+)
  • 1 node: 2x Xeon E5-2450 0 @ 2.10GHz, 64GB RAM, 2 x 2TB HDD
  • 6 nodes: 2 x Xeon E5-2680 v2 @2.80GHz, 128GB RAM, 12 x 2TB HDD
  • 546 nodes: 2 x Xeon X5670 @2.93GHz, 48GB RAM, 12 x 2TB HDD
BigBench Experiments

Powertest

3x Data => ~1.7x Time
BigBench Experiments

• **Power Test**
  - Single stream of all 30 queries
  - Runs sequentially

• **Throughput Test**
  - 2 parallel streams of all 30 queries
  - 2 runs + refresh between
  - Throughput test runs 2*2*30=120 queries

4x Queries => ~3.15x Time

![Graph showing POWER_TEST and THROUGHPUT_TEST for 1TB and 3TB datasets]
BigBench Reference Implementation ‘14

- **Hadoop Map-Reduce and Hive**
  - Hadoop Map-Reduce 2.0
  - HIVE, Mahout
  - Java 1.7

- **Reference Kit Queries**
  - All 30 queries are implemented.
  - Represents structured, semi-structured, un-structured data types.

- **Complete runnable kit**
  - Data generator, queries, benchmark driver
  - Tested on various Hadoop implementation
  - Easy to configure and run, detailed setup instructions
  - [https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench](https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench)

- **Bring some time**
  - Full BigBench run on Hive with reasonable large scaling factor takes 2 days+
  - Will verify if your cluster is setup correctly
  - Will find bugs and shortcomings in less mature systems
Thank you!

• Get involved

• BigBench GitHub Repository and GoogleGroups
  • https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench
  • https://groups.google.com/forum/#!forum/#!forum/big-bench

• Contact
  • michael.frank@bankmark.de
  • www.bankmark.de