BigBench Overview

Towards a Comprehensive End-to-End Benchmark for Big Data

Tilmann Rabl - bankmark UG (haftungsbeschränkt)

02/04/2015 @ SPEC RG Big Data

bankmark

The BigBench Proposal

- **End to end benchmark**
  - Application level
- **Based on a product retailer (TPC-DS)**
- **Focused on Parallel DBMS and MR engines**
- **History**
  - Launched at 1st WBDB, San Jose
  - Published at SIGMOD 2013
  - Spec at WBDB proceedings 2012 (queries & data set)
  - Full kit at WBDB 2014
- **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, …

Before BigBench

- **Micro-Benchmarks**
  - System level measurement
  - Illustrative not informative
- **Functional Benchmarks**
  - Better than micro-benchmarks
  - Simplified approach – limited representation in e2e
- **Benchmark suites**
  - Collection of micro and functional
  - Standardization problems

Specs and Standards

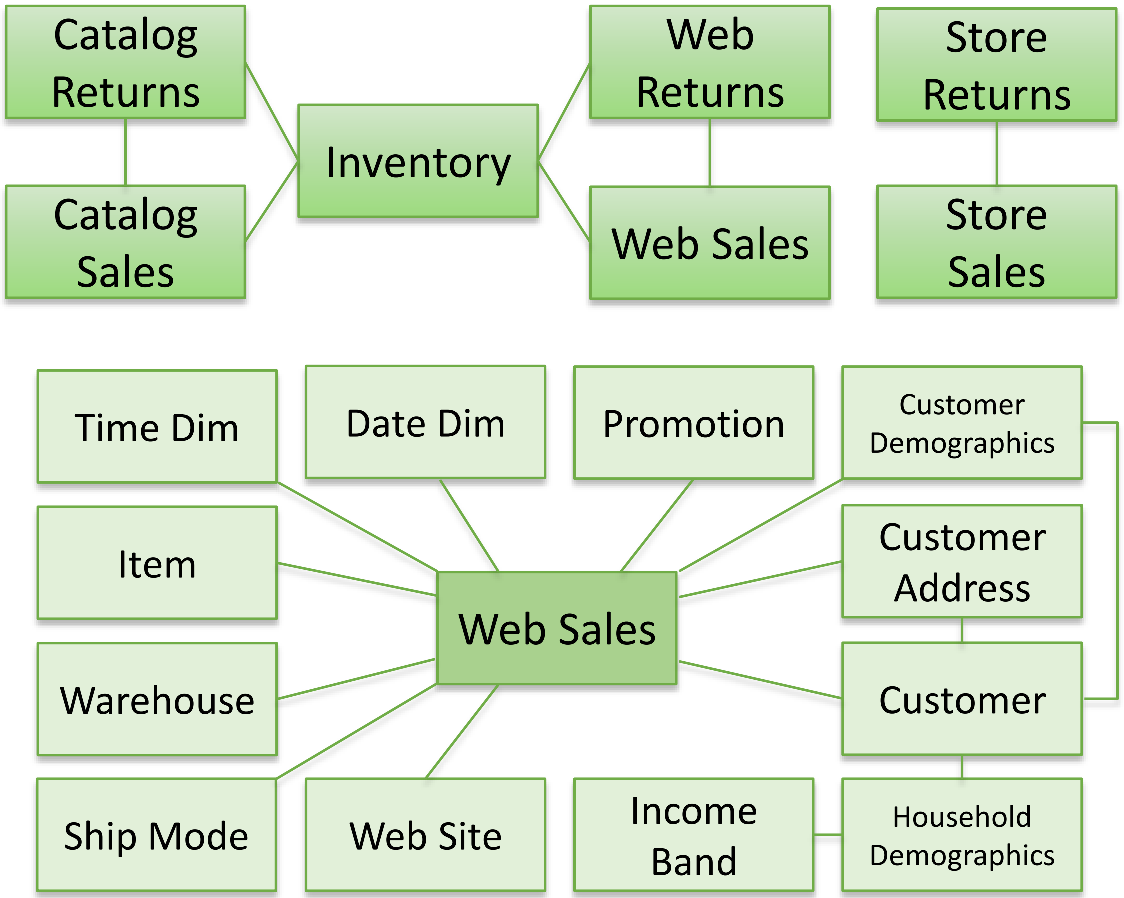
- **TPC xHS**
  - Very first Industry standard
  - Detailed metrics and run rules
  - Model framework
- **TPC-DS BigData**
  - Derive as-is from TPC-DS
  - Query based for SQL on Hadoop
- **BigBench**
  - Based on new specification with some reuse
  - Complex batch analytics
  - Long term bridge

BigBench - 2013

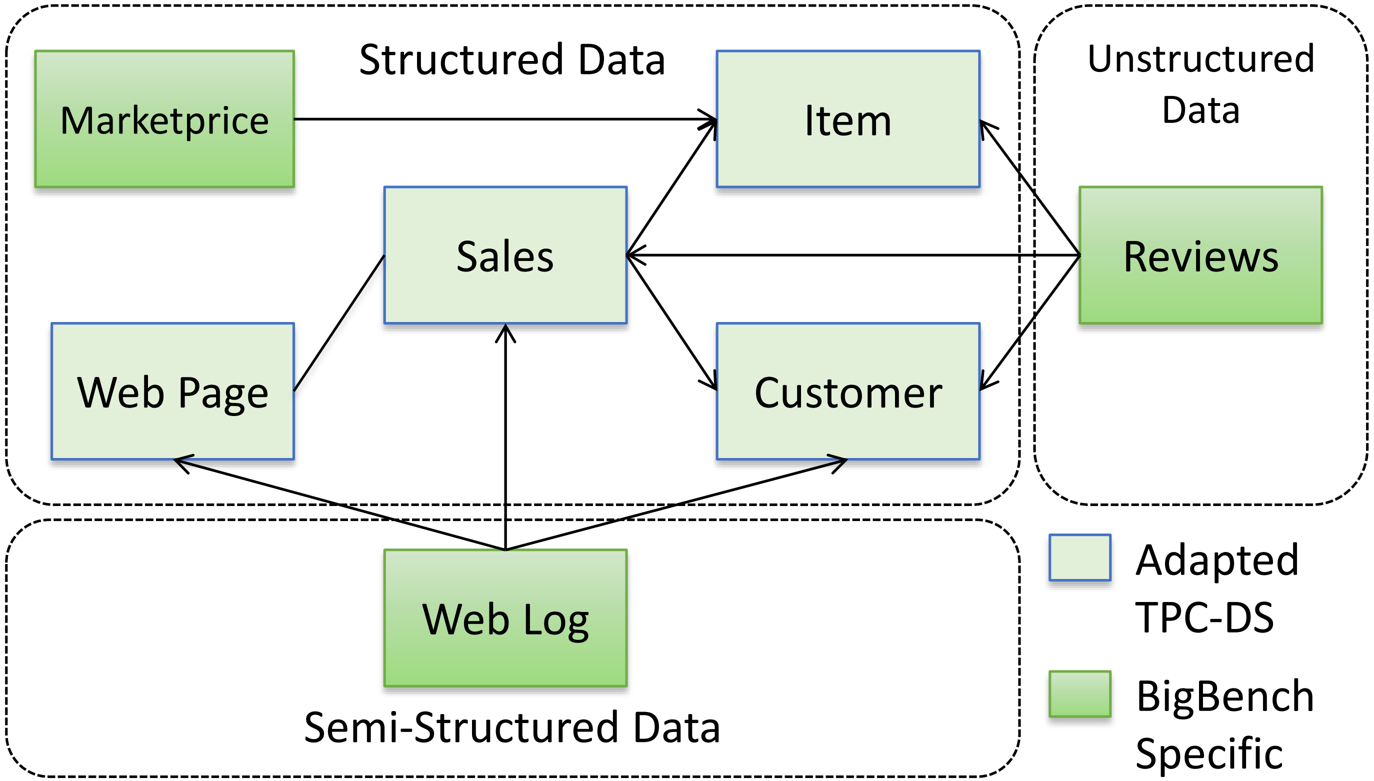
- **Collaborative industry effort**
  - Sigmod 2013
  - Address 3V's of big data
  - Very first concept for a big data benchmark specification
  - Wide industry support
- **Use case sampling**
  - Retail use case example
  - End to end and component
- **Framework Agnostic**
  - Well defined specification
  - SW based reference implementation

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



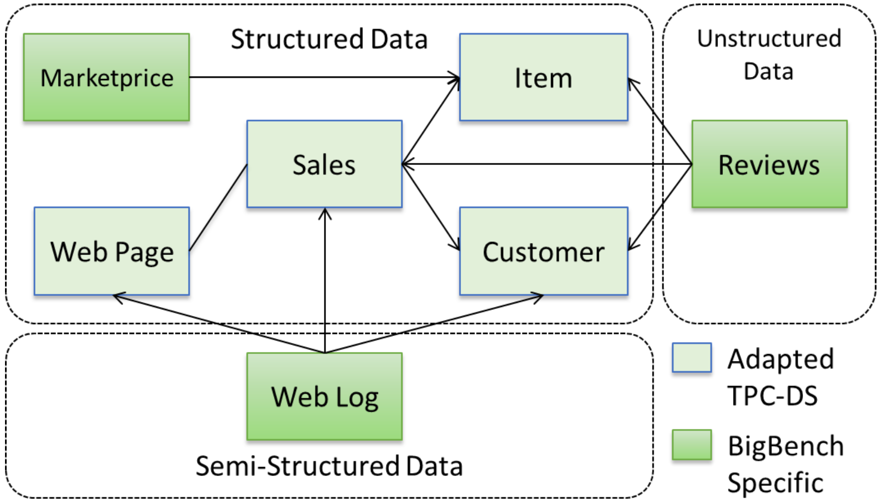
BigBench Data Model



- 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

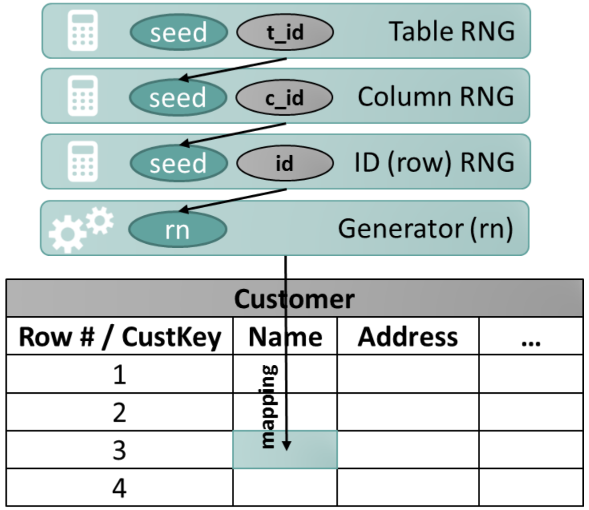
$$LF = SF + (SF - (\log_5(SF) * \sqrt{SF})) = 2SF - \log_5(SF) * \sqrt{SF}$$

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

bankmark

Generating Big Data

- **Repeatable computation**
  - Based on XORSHIFT random number generators
- **Hierarchical seeding strategy**
  - Enables independent generation of *every* value in the data set
  - Enables independent *re-generation* of every value for references
- **User specifies**
  - 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




```
<property name="METERS_DAY" type="double">1000000</property>
<table name="smartgrid">
  <size>${METERS_DAY}</size>
  <field name="id" size="" type="NUMERIC"
        primary="true" unique="true">
    <gen_IdGenerator/>
  </field>
  <field name="territory" type="NUMERIC">
    <gen_LongGenerator>
      <min>100000000</min>
      <max>999999999</max>
    </gen_LongGenerator>
  </field>
  <field name="regionnumber" type="NUMERIC">
    <gen_SequentialGenerator concatenate="true">
      <gen_OtherFieldValueGenerator>
        <reference field="territoryhierarchy" />
      </gen_OtherFieldValueGenerator>
      <gen_LongGenerator>
        <min>1000000000000.0</min>
        <max>9999999999999.0</max>
      </gen_LongGenerator>
    </gen_SequentialGenerator>
  </field>
[..]
```

Workload

- **Workload Queries**
  - 30 “queries”
  - Specified in English (sort of)
  - No required syntax (first implementation in Aster SQL MR)
  - Kit implemented in Hive, HadoopMR, Mahout, OpenNLP
- **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)**

Workload - Technical Aspects

Generic Characteristics

Data Sources	#Queries	Percentage
Structured	18	60%
Semi-structured	7	23%
Un-structured	5	17%

Hive Implementation Characteristics

Query Types	#Queries	Percentage
Pure HiveQL	14	46%
Mahout	5	17%
OpenNLP	5	17%
Custom MR	6	20%

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

SQL-MR Query 1

```
SELECT category_cd1 AS category1_cd,
       category_cd2 AS category2_cd , COUNT (*) AS cnt
FROM basket_generator (
      ON
              ( SELECT i.i_category_id AS category_cd ,
                       s.ws_bill_customer_sk AS customer_id
                FROM web_sales s INNER JOIN item i
                ON s.ws_item_sk = i.item_sk )
       PARTITION BY customer_id
       BASKET_ITEM (‘category_cd')
       ITEM_SET_MAX (500)
)
GROUP BY 1,2
ORDER BY 1, 3, 2;
```

HiveQL Query 1

```
SELECT     pid1, pid2, COUNT (*) AS cnt
FROM (
           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)
                      ) q01_temp_join
                      MAP q01_temp_join.oid, q01_temp_join.pid
                      USING 'cat'
                      AS oid, pid
                      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) > 49
ORDER BY pid1, cnt, pid2;
```

Benchmark Process

Adapted to batch systems

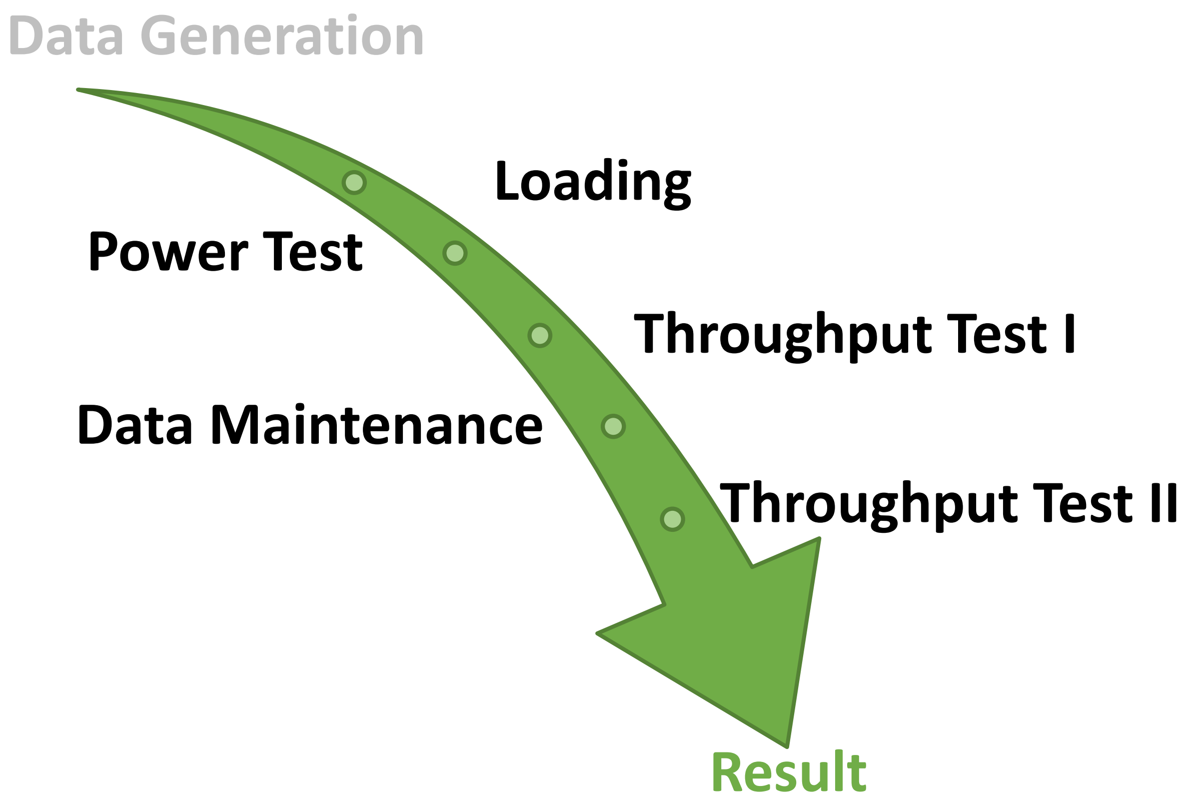
- No trickle update

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



Metric

- **Throughput metric**
  - BigBench queries per hour
- **Number of queries run**
  - 30*(2*S+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**

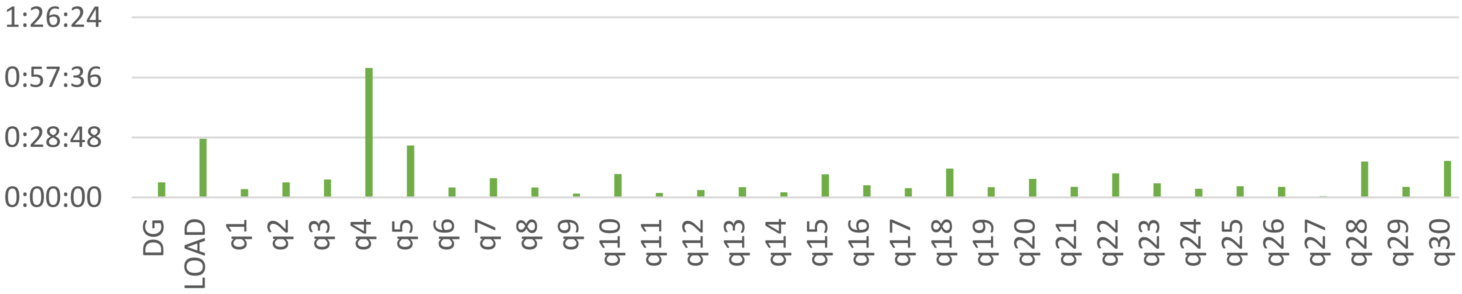
$$BBQpH = \frac{30 * 3 * 3600}{T_L + T_P + \frac{T_{TT1}}{S} + T_{DM} + \frac{T_{TT2}}{S}}$$

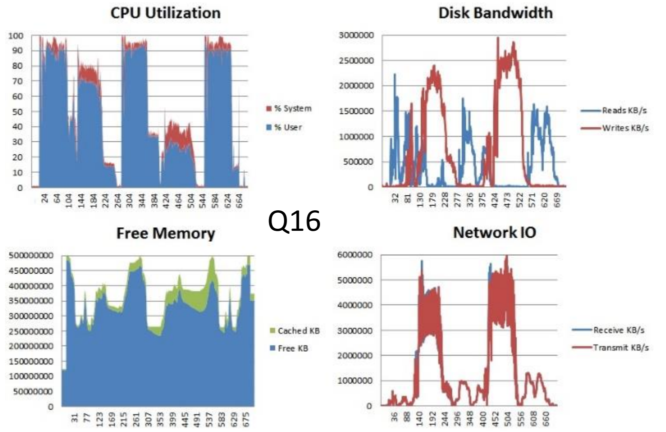
$$BBQpH = \frac{30 * 3 * S * 3600}{S * T_L + S * T_P + T_{TT1} + S * T_{DM} + T_{TT2}}$$

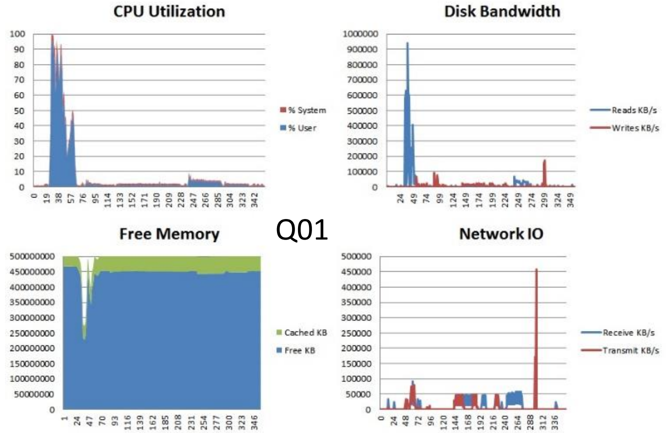
Source: www.wikipedia.de

BigBench Experiments

- **Tests on**
  - Cloudera CDH 5.0, Pivotal GPHD-3.0.1.0, IBM InfoSphere BigInsights
- **In progress: Spark, Impala, 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







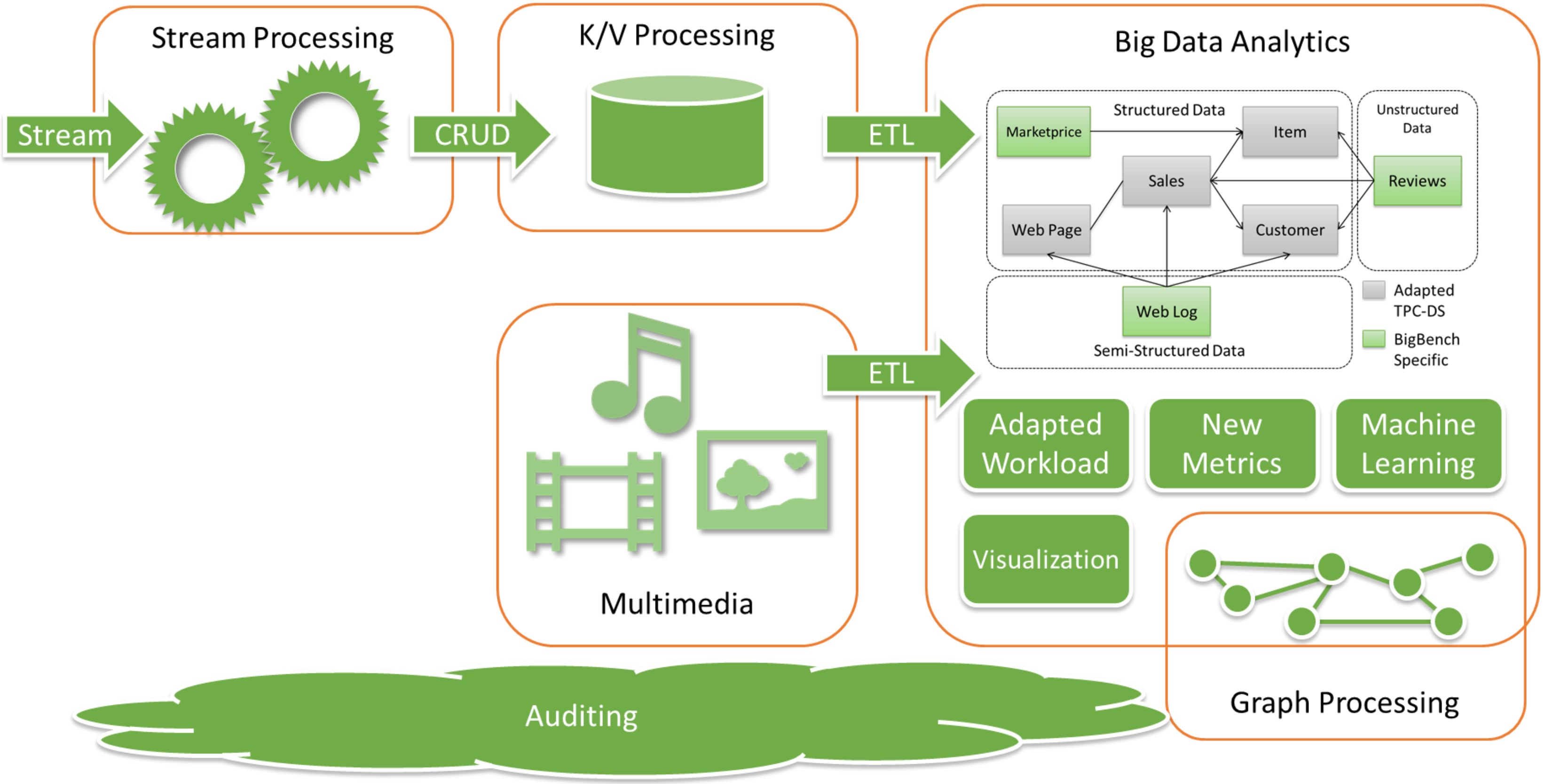
BigBench reference implementation - 2014

- **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-Structrured, 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-Bench
- **Bring some time**
  - Full BigBench run on Hive takes 2 days+
  - Will verify if your cluster is setup correctly

bankmark

BigBench 2015+ - Towards a Big Data Pipeline

- **Extensive Procedural Coverage**
  - Large input Datasets
  - Web-based procedural
  - Functional component level
- **Machine Learning**
  - Dedicated machine learning workloads
  - Algorithm coverage
  - Segmented
- **Others**
  - Key-value pair representation
  - Cover graph
  - Streaming applications
  - Multimedia



BigBench 2015 – Metrics

Metric

- **Current metric based on geometric mean**
- **Represent performance / $ - scaling factor**
- **Segmented**
  - Structural
  - Unstructured/procedural
  - Machine learning
  - Other components
  - All components = E2E
- **Machine learning accuracy**
- **Fault tolerant performance**

Contact

Tilmann Rabl

tilmann.rabl@bankmark.de

www.bankmark.de