BigBench on Hadoop

Tilmann Rabl, Badrul Chowdhury, Pooya Saadatpanah, Jiang Du, Hans-Arno Jacobsen
University of Toronto
Full Credits for V0.1

- **Teradata:**
  - Ahmad Ghazal, Minqing Hu, Alain Crolotte

- **University of Toronto:**
  - Tilmann Rabl, Hans-Arno Jacobsen

- **InfoSizing:**
  - Francois Raab

- **Oracle:**
  - Meikel Poess

- **WBDB - Big Data Benchmarking Subcommittee**
Overview

- End to end benchmark
  - Application level
- Based on a product retailer (TPC-DS)
- Focus on
  - Parallel DBMS
  - MR engines
- History
  - Launched at 1st WBDB, San Jose
  - Published at SIGMOD 2013
  - Full spec at WBDB proceedings 2012
Data Model 1

- **Structured Data**
  - Marketprice
  - Item
  - Sales
  - Customer

- **Unstructured Data**
  - Reviews

- **Semi-Structured Data**
  - Web Log

- **Structured**: TPC-DS + market prices
- **Semi-structured**: website click-stream
- **Unstructured**: customers’ reviews
Review Generation

Offline Preprocessing

- Categorization
- Tokenization
- Generalization
- Markov Chain Input

Online Data Generation

- Product Customization
- Text Generation
- Parameter Generation (PDGF)

Real Reviews

Generated Reviews
Data Model – 3 Vs

- **Variety**
  - Different schema parts

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

- **Velocity**
  - Periodic refreshes for all data
  - Different velocity for different areas
    - $V_{\text{structured}} < V_{\text{unstructured}} < V_{\text{semistructured}}$
  - Queries run with refresh
Workload

- **Workload Queries**
  - 30 queries
  - Specified in English (sort of)
  - No required syntax

- **Business functions** (Adapted from McKinsey+)
  - **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

<table>
<thead>
<tr>
<th>Data Sources</th>
<th>Number of 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>

<table>
<thead>
<tr>
<th>Analytic techniques</th>
<th>Number of Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics analysis</td>
<td>6</td>
<td>20%</td>
</tr>
<tr>
<td>Data mining</td>
<td>17</td>
<td>57%</td>
</tr>
<tr>
<td>Reporting</td>
<td>8</td>
<td>27%</td>
</tr>
</tbody>
</table>
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_sk
      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;
Evaluation

- Hardware RAID 1 with 8 2.5" disks
- Proof of concept
- No runaway queries
Implementation on HIVE

- Proof of concept system
  - Open-source
  - Publicly available

- Hadoop 0.20.2
- OpenNLP 1.5.3
- NLTK 2.0
- Hive 0.8.1
- Mahout 0.6
Current Condition

- All queries are translated

<table>
<thead>
<tr>
<th>Query Types</th>
<th>Number of Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Hive</td>
<td>14</td>
<td>47%</td>
</tr>
<tr>
<td>Mahout</td>
<td>5</td>
<td>17%</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>4</td>
<td>13%</td>
</tr>
<tr>
<td>Custom MR</td>
<td>7</td>
<td>23%</td>
</tr>
</tbody>
</table>

- Try it:
  - msrg.org/project/BigBench
  - msrg.org/datasets/BigBenchQueries
ADD FILE q1_mapper.py;
ADD FILE q1_reducer.py;

-- Find the most frequent ones
SELECT pid1, pid2, COUNT(*) AS cnt
FROM (  
  -- Make items basket
  FROM (    
    -- Joining two tables
    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 (1,4,6) and s.ss_store_sk in (10, 20, 33, 40, 50)       
    ) temp_join MAP temp_join.oid, temp_join.pid         
    USING 'python q1_mapper.py'                             
    AS oid, pid                                                       
  ) map_output REDUCE map_output.oid, map_output.pid         
    USING 'python q1_reducer.py'                             
    AS (pid1 BIGINT, pid2 BIGINT)                           
  ) temp_basket GROUP BY pid1, pid2
HAVING COUNT(pid1) > 49
ORDER BY pid1, cnt, pid2;

(c) Tilmann Rabl - msrg.org

Mapper
import sys

if __name__ == "__main__":
  for line in sys.stdin:
    key, val = line.strip().split("\t")
    print "%s\t%s" % (key, val)

Reducer
import sys

def print_permutations(vals):
  l = len(vals)
  if l <= 1 or l*(l-1)/2 > 500:
    return
  vals.sort()
  for i in range(l-1):
    for j in range(i+1,l):
      print "%s\t%s" % (vals[i], vals[j])

if __name__ == "__main__":
  current_key = "
  vals = []
  for line in sys.stdin:
    key, val = line.strip().split("\t")
    if current_key == "":
      current_key = key
      vals.append(val)
    elif current_key == key:
      vals.append(val)
    elif current_key != key:
      print_permutations(vals)
      vals = []
      current_key = key
      vals.append(val)
  print_permutations(vals)
Next Steps I: Data Generation

- DSDGen is not flexible
  - Scale factors are hardcoded
  - Configuration is limited
- Parallel Data Generation Framework to the rescue
  - Flexible, generic data generation framework
  - Used in TPC-DI
  - Commercialized by bankmark (www.bankmark.de)

- In this process
  - Improve the schema (simplify, remodel)
Next Steps II: Metric

- TPC Performance, Price/Performance, Energy/Performance
  - All nice, but not necessarily fair

- Per node performance?
  - Atom vs Xeon, disk vs flash?

- (Per node) efficiency!
  - BigBench performance / hardware performance
  - E.g., BigBench / Linpack(successor) * FIO
Get involved

- **BigBench website:**
  - [msrg.org/project/BigBench](msrg.org/project/BigBench)
  - [msrg.org/datasets/BigBenchQueries](msrg.org/datasets/BigBenchQueries)

- **BigBench papers:**
  - Ghazhal et al., *BigBench: Towards an Industry Standard Benchmark for Big Data Analytics*. Sigmod 2013
  - Rabl et al., *BigBench Specification V0.1*. WBDB 2012 Proceedings, LNCS 8163, to appear

- **PDGF:**
  - Demo version & papers online: [www.paralleldatageneration.org](www.paralleldatageneration.org)
  - Pro-version coming soon: [www.bankmark.de](www.bankmark.de)
Thank You!

Questions?

Contact

Tilmann Rabl

tilmann.rabl@utoronto.ca

http://msrg.org/profiles/tilmann