Benchmarking Spark ML using BigBench
Motivation

Study the performance of Machine Learning use cases on large data warehouses in context of assessing
– Alternate approaches to connect from data warehouse to analytics engine
– Different machine learning frameworks

Data preparation and Modeling are the most time consuming phases in a ML cycle
High Speed Data Connectors for Spark

Highly **optimized** and parallel **data transfer between** dashDB and Spark

- **Colocation** of Spark executors and DB2 data nodes
- Optimized exchange of data

Connectors between analytics engine and database can speed up
- ETL during data preparation phase
- Reading from data store during the Model Creation phase
  - Assessing alternate models
  - Tuning the model parameters
  - During model execution
- Writing back the scoring results to the database

dashDB Spark integration Layout
Why BigBench?

Requirements for benchmarking high speed data connectors
- Representative of a realistic use case for performing ML on data warehouse
- Ability to scale to large data volumes
- Supports read and write to data source
- Invoke Machine Learning algorithms via SQL interface (Stored Procedure) or via Spark jobs (using customized RDD to connect to data source)
- Ability to execute multiple streams to test scalability and resource management in an integrated solution where Spark and database co-exist on the same cluster
- Compare efficiency and accuracy of Spark MLlib versus IBM ML algorithms

BigBench met most of our requirements
Collaborative Filtering using Matrix Factorization (MF)

Known for unique challenges
- Data Sparsity: Very few customers rate items
- Scalability: Computational complexity in filling the sparse user item association matrix grows quickly on large data sets

5,450,482 Reviews
≅ 3,130,656 Users

563,518 Items

BigBench Sparsity level = 0.00025%
Alternating Least Squares in Spark MLlib

- Step 1: Initialize with random factor
- Step 2: Hold the item factor constant and find the best value for user
- Step 3: Hold the user factor constant and find the best value for item
- Repeat Step 2 & Step 3 for convergence

Why include Matrix Factorization in BigBench?

- Unique Performance characteristics
- Trade-off between efficiency and accuracy. Accuracy improves with high number of latent factors with a corresponding drop in performance
- Facilitates creation of real time analytics scenario: Saved Matrix Factorization model can be used to predict ratings on trickling web_clickstreams data during the workload run
- Good test bench for comparing implementation and optimizations of different ML frameworks
Q05: Through the SPSS Lens

- Predict if a visitor will be interested in a given item category, based on demographics and existing users online activities (interest in items of different categories)

- Label is 1 if “Clicks in Specified category” > Average Clicks in that category

- Modeler selection & Accuracy varies depending on the specified item category
  - If CLICKS_IN column of the item category is in the input vector, models are able to predict the outcome with 100% accuracy. Models selected are Logistic Regression & models of decision tree family
  - If CLICKS_IN column of the item category is NOT in the input vector, more complex models are chosen and accuracy < 100%
Scenario #1:

- **Feature Vector**
  - \([\text{CLICKS\_IN\_1}, \text{CLICKS\_IN\_2}, \text{CLICKS\_IN\_3}, \text{CLICKS\_IN\_4}, \text{CLICKS\_IN\_5}, \text{CLICKS\_IN\_6}, \text{CLICKS\_IN\_7}, \text{COLLEGE\_EDUCATION}, \text{MALE}]\)

- Specified category = 3

---

**Graph**

<table>
<thead>
<tr>
<th>Graph</th>
<th>Model</th>
<th>Build Time (mins)</th>
<th>Max Profit</th>
<th>Max Profit Occurs in (%)</th>
<th>Lift[Top 30%]</th>
<th>Overall Accuracy (%)</th>
<th>No. Fields Used</th>
<th>Area Under Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logistic regression 1</td>
<td>15</td>
<td>1,564,925.0</td>
<td>9</td>
<td>3.333</td>
<td>100.0</td>
<td>9</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>CART Tree 1</td>
<td>15</td>
<td>1,564,925.0</td>
<td>9</td>
<td>3.333</td>
<td>100.0</td>
<td>6</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Equation For 0**

\[0.07511 \times \text{COLLEGE\_EDUCATION} +
-0.01905 \times \text{MALE} +
0.0001811 \times \text{CLICKS\_IN\_1} +
0.000004033 \times \text{CLICKS\_IN\_2} +
11.52 \times \text{CLICKS\_IN\_3} +
0.0008079 \times \text{CLICKS\_IN\_4} +
-0.00000001071 \times \text{CLICKS\_IN\_5} +
0.00000003695 \times \text{CLICKS\_IN\_6} +
0.00254 \times \text{CLICKS\_IN\_7} +
+0.67150\]

**Equation For 1**

\[0.07511 \times \text{COLLEGE\_EDUCATION} +
-0.01905 \times \text{MALE} +
0.0001811 \times \text{CLICKS\_IN\_1} +
0.000004033 \times \text{CLICKS\_IN\_2} +
11.52 \times \text{CLICKS\_IN\_3} +
0.0008079 \times \text{CLICKS\_IN\_4} +
-0.00000001071 \times \text{CLICKS\_IN\_5} +
0.00000003695 \times \text{CLICKS\_IN\_6} +
0.00254 \times \text{CLICKS\_IN\_7} +
+0.67150\]

**Tree Depth = 1**
Scenario #2:

- **Feature Vector**
  - \([\text{CLICKS\_IN\_1}, \text{CLICKS\_IN\_2}, \text{CLICKS\_IN\_3}, \text{CLICKS\_IN\_4}, \text{CLICKS\_IN\_5}, \text{CLICKS\_IN\_6}, \text{CLICKS\_IN\_7}, \text{COLLEGE\_EDUCATION}, \text{MALE}]\)

- **Specified category = 9**

Tree Depth = 25
Scenario #3:

- Feature Vector
  - [CLICKS_IN_1, CLICKS_IN_2, CLICKS_IN_3, CLICKS_IN_4, CLICKS_IN_5, CLICKS_IN_6, CLICKS_IN_7, COLLEGE_EDUCATION, MALE]

- Specified category = 3

Tree Depth = 8
Key Learning

- Not including the deterministic clicks in the input feature vector will exercise and stress the machine learning algorithms in a more realistic way. This clearly reflects in the tree depth.

- Another benefit is the ability to introduce more complex algorithms such as Neural Networks to the BigBench ML mix.
Tuning ML Pipeline

- Model Evaluation phase involves assessing alternate models or tuning the optimization parameters of an algorithm. Tuning is assessed by accuracy on test data sets using cross validation.
- Example: Tuning regularization parameter for Logistic Model/ALS, Tuning “rank” for ALS.
- Tuning can have interesting side effects on performance.

Test Environment
BigBench Scale Factor = 1TB
dashDB Local cluster, CentOS7.0-64 and Spark 1.6.2
4 nodes with the following configuration:
- 24 cores (2.6GHz Intel Xeon-Haswell)
- 512 GB memory
- 10000 Mbps full duplex N/W card
Conclusion & Next Steps

- K-Means use case in BigBench has been very effective in proving the benefits of a high speed connector between data warehouse and Spark

- Our recommendations
  - Broaden the scope of BigBench to more Machine Learning algorithms since performance characteristics of ML algorithms vary
    - Achievable via addition of new use case like Recommender and tweaking existing scenarios like Q05
  - Simulate more ML use cases
    - Real time analytics for Collaborative Filtering
    - Tuning Machine Learning pipeline

- Continued work
  - Investigate ways to incorporate data transformations in the analytic engine layer in BigBench
  - Study the performance characteristics of other ML algorithms on BigBench use case – Trees and Neural Network
Thank you!

Sweta Singh
singhSwe@us.ibm.com