Towards Selecting Best Combination of SQL-on-Hadoop Systems and JVMs

Tatsuhiro Chiba, Takeshi Yoshimura, Michihiro Horie and Hiroshi Horii

IBM Research
Agenda

- **Motivation, Problems and Challenges**
- **Backgrounds**
  - Backend engines: Spark and Tez
  - Backend runtimes: OpenJDK and J9
- **Empirical Study**
  - Performance evaluation
  - Performance analysis
- **ML model**
  - training classification model
  - evaluating classification model
- **Summary**
Distributed Processing Framework for Big Data

- **Hadoop Eco-Systems**
  - HDFS: the center of data store
  - utilizing data between different frameworks
    - Spark, Tez, Flink, YARN, MR, Hive, Pig, Hbase, etc...

- **Big Data Workload**
  - ETL
  - SQL
  - ML / DL / Streaming

- **JVM as a Hadoop Runtime**
  - disk-oriented → in-memory oriented
  - I/O intensive → CPU-intensive
Motivation and Problem – Many choices of the systems

- **Rapid Development Cycle**
  - Fast open sources releases
  - marge new feature frequently
  - query performance is also improved

- **Too many SQL-on-Hadoop Systems**
  - Which one is best? (SparkSQL or Hive or Impara or Presto or …)
  - Should we switch a system to another one?
  - No single SQL-on-Hadoop engine is best for ALL queries
  - No single JVM is best for ALL queries as well
Motivation and Problem – Selecting a system adaptively

- Requirements of Query Execution on Cloud
  - query users: do not care about backend system as long as it returns a result fast
  - cloud providers: wants to minimize resources by using fast processing backend

- Related work: workload translation
  - generate suitable code for a best system
  - Musketeer [Eurosys ’16], Weld [CIDR ’17]

- Related work: Multi Store / Hybrid Engines
  - MISO [SIGMOD ’14], MuSQLLE [BigData ’16]
  - using multiple engines/stores based on cost model / heuristics / etc.

No JVM awareness
need to update cost model / heuristics frequently
Questions and Challenges

1. What about potential gains?
   - Observation

2. What features make the differences?
   - Reasoning

3. What data to help building a model?
   - Training

4. How accurately can the model predict?
   - Predicting

GOAL

Empirical Study

Run SQL on best engine/runtime

Meta Scheduler

ML Model

Choices of Engine and Runtime

- Spark
- TEZ
- OpenJDK
- OpenJ9
Agenda

- Motivation, Problems and Challenges

- Backgrounds
  - Backend engines: Spark and Tez
  - Backend runtimes: OpenJDK and J9

- Empirical Study
  - Performance evaluation
  - Performance analysis

- ML model
  - training classification model
  - evaluating classification model

- Summary
Spark/Spark SQL

- **Spark**
  - DAG-based distributed framework
  - execute stage by stage

- **Spark SQL**
  - Catalyst – Query Optimizer
  - Parquet Columnar Format
  - code generation (SIMD, loop unrolling)

引用: Michael et al., Spark SQL: Relational Data Processing in Spark. SIGMOD’15

引用: https://spark.apache.org/docs/latest/cluster-overview.html
Tez/Hive

- **Tez**
  - Generalized Map Reduce
  - DAG-based distributed framework

- **Hive/LLAP**
  - focus on interactive query
  - Vectorization / Pipeline
  - In-Memory Columnar Cache (off-heap)
  - ORC Columnar Format

Ref: Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications, SIGMOD’15

Ref: https://www.slideshare.net/Hadoop_Summit/llap-longlived-execution-in-hive, Hadoop Summit 2015
JVM – OpenJDK & IBM J9

- **JVM**
  - OpenJDK / J9 (Eclipse OMR based)
  - internal optimization / implementation are different

- **JIT**
  - Tiered Compilation Level
  - Intrinsics
  - Inlining Heuristics
  - Vectorization Code

- **Memory Management**
  - GC Algorithm (G1GC / Generational / CMS / Parallel / Copying etc.)
  - Memory Fence

- **Thread**
  - Lock Reservation
Agenda

- Motivation, Problems and Challenges
- Backgrounds
  - Backend engines: Spark and Tez
  - Backend runtimes: OpenJDK and J9
- Empirical Study
  - Performance evaluation
  - Performance analysis
- ML model
  - training classification model
  - evaluating classification model
- Summary
Questions and Challenges

1. What about potential gains?
   - Observation

2. What features make the differences?
   - Reasoning

3. What data to help building a model?
   - Training

4. How accurately can the model predict?
   - Predicting

GOAL

- Empirical Study
- Meta Scheduler
- Run SQL on best engine/runtime
- ML Model

Choices of Engine and Runtime

IEEE CLOUD 2018 / Towards Selecting Best Combination of SQL-on-Hadoop Systems and JVMs
Environment – HW/SW Spec & Benchmark

- **Machine**
  - evaluated on a single POWER8 node
  - Use Flash storage for HDFS

- **TPC-DS Benchmark**
  - hive-testbench (*1)
  - 68 queries

- **data set**
  - Scale Factor 500 (500GB)
  - prepared two columnar dataset; Parquet & ORC

<table>
<thead>
<tr>
<th>Machine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>POWER8 3.3 GHz * 2</td>
</tr>
<tr>
<td># Cores</td>
<td>24 cores (2 Sockets * 12 Cores)</td>
</tr>
<tr>
<td>SMT</td>
<td>8</td>
</tr>
<tr>
<td>Memory</td>
<td>1TB</td>
</tr>
<tr>
<td>Disk</td>
<td>Flash System (9.3TB)</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 16.04 (kernel 4.4.0-31)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>2.1.0</td>
</tr>
<tr>
<td>Hadoop (HDFS)</td>
<td>2.7.2</td>
</tr>
<tr>
<td>Tez</td>
<td>0.9.0</td>
</tr>
<tr>
<td>Hive</td>
<td>2.2.0</td>
</tr>
<tr>
<td>OpenJDK</td>
<td>1.8.0_u121</td>
</tr>
<tr>
<td>IBM J9 JVM</td>
<td>1.8.0 SR4FP2</td>
</tr>
</tbody>
</table>

(*1) [https://github.com/hortonworks/hive-testbench](https://github.com/hortonworks/hive-testbench)
Environment - Others

- Configurations of Spark & Tez

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Spark / Spark SQL</th>
<th>Tez / Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executor JVM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Worker Threads</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>I/O Threads</td>
<td>-</td>
<td>12</td>
</tr>
<tr>
<td>On Heap Size</td>
<td>192 GB</td>
<td>96 GB</td>
</tr>
<tr>
<td>Off Heap Size</td>
<td>-</td>
<td>96 GB</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Daemon (Thrift Server)</td>
<td>LLAP Daemon</td>
</tr>
<tr>
<td>Columnar Format</td>
<td>Parquet</td>
<td>ORC</td>
</tr>
<tr>
<td>Compression Format</td>
<td>gzip (zlib)</td>
<td>gzip (zlib)</td>
</tr>
<tr>
<td>Other JVM Options (Common)</td>
<td>GC Threads = 12, -agentpath:libjvmti_oprofile.so</td>
<td></td>
</tr>
</tbody>
</table>

- Evaluation Methodology
  - used Thrift Server
  - picked up fastest result in 5-times test per query
  - reset buffer cache (echo 3 > /proc/sys/vm/drop_caches)
Performance Comparison of TPC-DS on Spark - Which JVM is better for Spark?

- **Performance Comparison Result**
  - OpenJDK achieved faster than J9 in 35 queries (35/62 = 56.5%)
  - J9 achieved faster than OpenJDK in 27 queries (27/62 = 43.5%)
  - leads up to 3x drawback

**Lower is Better**

IBM J9          OpenJDK          ratio

OpenJDK is faster

IBM J9 is faster
Performance Comparison of TPC-DS on Tez - Which JVM is better for Tez?

- **Performance Comparison Result**
  - OpenJDK achieved faster than J9 in 35 queries (35/65 = 53.8%)
  - J9 achieved faster than OpenJDK in 30 queries (30/65 = 46.1%)
  - *leads up to 2x drawback*
Performance Comparison of TPC-DS with OpenJDK - Which query engine is better with OpenJDK?

- Performance Comparison Result
  - Tez is faster in two-thirds queries than Spark
  - leads up to 17x drawback

Lower is Better (Using OpenJDK)

![Chart showing performance comparison between Tez and Spark](chart.png)
Summary of Motivational Evaluation

- **Result**
  - 60 queries are successfully run
  - picked up a best combination for all queries

- **Tendency**
  - Tez is better than Spark
  - J9 is better than OpenJDK
  - Combination of Tez & J9 is good at in many cases
Comparison of picked up queries

- **System**
  - Spark wins Tez: Q51
  - Tez wins Spark: Q50, Q58, Q82

- **Runtime**
  - J9 wins OpenJDK: Q51, Q58
  - OpenJDK wins J9: Q50, Q82

- **analysis**
  - query plan (DAG) / middleware execution stats
  - hot method profiling (oprofile) / system utilization
  - Java method stack trace / GC Log / JIT Log
Gain comes from JVM difference – Spark case

- **Q51**
  - J9 wins
  - many stages
  - less shuffle data
  - gets 2.6x gain in shuffle stage

- **Q82**
  - OpenJDK wins
  - few stages
  - much shuffle data
  - gets 1.4x gain in map stage

<table>
<thead>
<tr>
<th>Query</th>
<th># Map Stages</th>
<th># Reduce Stages</th>
<th>Input Read</th>
<th>Shuffle Output</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q51</td>
<td>2</td>
<td>6</td>
<td>6.0 GB</td>
<td>1.0 GB</td>
<td>Shuffle J9: 11s, OpenJDK: 29 s</td>
</tr>
<tr>
<td>Q82</td>
<td>2</td>
<td>2</td>
<td>2.5 GB</td>
<td>5.6 GB</td>
<td>Map J9: 66s, OpenJDK: 47s</td>
</tr>
</tbody>
</table>

IEEE CLOUD 2018 / Towards Selecting Best Combination of SQL-on-Hadoop Systems and JVMs
Gain comes from JVM difference – Spark case

- **Method Profiling**
  - J9 is good at Intrinsic for `Sun.misc.Unsafe.copyMemory` (JNI overhead)
  - OpenJDK is good at serialization and sort in data shuffling

**J9 Advantage: Many Stages, less Shuffling Data**

**OpenJDK Advantage: Few Stages, much Shuffling Data**
Gain comes from JVM difference – Tez case

- **Q50**
  - OpenJDK wins
  - gets 1.7x gain in reduce vertex

- **Q51**
  - J9 wins
  - gets 3x gain in map vertex

<table>
<thead>
<tr>
<th>Query</th>
<th># Map Stages</th>
<th># Reduce Stages</th>
<th>Input Records / GB</th>
<th>Shuffle Records / GB</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q50</td>
<td>5</td>
<td>7</td>
<td>1.3 * 10^9 (5.4 GB)</td>
<td>5.0 * 10^7 (1.9 GB)</td>
<td>Reduce J9: 106s, OpenJDK: 60s</td>
</tr>
<tr>
<td>Q51</td>
<td>4</td>
<td>5</td>
<td>3.5 * 10^8 (1.3 GB)</td>
<td>3.5 * 10^8 (3.5 GB)</td>
<td>Map J9: 7s, OpenJDK: 21s</td>
</tr>
</tbody>
</table>
Gain comes from JVM difference – Tez case

- **Q51**
  - J9 achieved 3x gain in map vertex
  - writing intermediate data (including in-mem agg. & SerDe) is time-consuming

**J9 Advantage: Few Vertices, Much Shuffling Data**

<table>
<thead>
<tr>
<th>PipelinedSorter (Reduce Vertex)</th>
<th>In memory ORC (LLAP) Read</th>
<th>Serialization/Deserialization</th>
<th>JOIN / Aggregation</th>
<th>java.io.DataOutputStream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialize + Spill</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Q50</th>
<th></th>
<th>Q51</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J9</td>
<td>OpenJDK</td>
<td>J9</td>
<td>OpenJDK</td>
</tr>
<tr>
<td>kallsyms</td>
<td>18.4%</td>
<td>6.5%</td>
<td>4.1%</td>
<td>3.3%</td>
</tr>
<tr>
<td>jvm</td>
<td>9.1%</td>
<td>25.9%</td>
<td>11.2%</td>
<td>7.0%</td>
</tr>
<tr>
<td>java</td>
<td>68.6%</td>
<td>61.0%</td>
<td>81.5%</td>
<td>87.0%</td>
</tr>
<tr>
<td>tez</td>
<td>3.1%</td>
<td>2.6%</td>
<td>28.9%</td>
<td>12.0%</td>
</tr>
<tr>
<td>orc</td>
<td>11.9%</td>
<td>17.0%</td>
<td>0.8%</td>
<td>0.3%</td>
</tr>
<tr>
<td>java.io.DataOutputStream</td>
<td>1.0%</td>
<td>4.0%</td>
<td>6.0%</td>
<td>18.3%</td>
</tr>
<tr>
<td>hive.q1</td>
<td>38.0%</td>
<td>30.7%</td>
<td>16.0%</td>
<td>9.3%</td>
</tr>
<tr>
<td>serDe</td>
<td>2.3%</td>
<td>1.3%</td>
<td>12.4%</td>
<td>29.0%</td>
</tr>
</tbody>
</table>
Gain comes from JVM difference – Tez case

- **Q50**
  - OpenJDK achieved **1.7x gain** in reduce vertex
  - many shuffle threads / many vertices
  - huge context switch overhead

OpenJDK Advantage: Many Vertices, Less Shuffling Data

<table>
<thead>
<tr>
<th></th>
<th>Q50 J9</th>
<th>Q50 OpenJDK</th>
<th>Q51 J9</th>
<th>Q51 OpenJDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>kallsyms</td>
<td>18.4%</td>
<td>6.5%</td>
<td>4.1%</td>
<td>3.3%</td>
</tr>
<tr>
<td>jvm</td>
<td>9.1%</td>
<td>25.9%</td>
<td>11.2%</td>
<td>7.0%</td>
</tr>
<tr>
<td>java</td>
<td>68.6%</td>
<td>61.0%</td>
<td>81.5%</td>
<td>87.0%</td>
</tr>
<tr>
<td>tez</td>
<td>3.1%</td>
<td>2.6%</td>
<td>28.9%</td>
<td>12.0%</td>
</tr>
<tr>
<td>orc</td>
<td>11.9%</td>
<td>17.0%</td>
<td>0.8%</td>
<td>0.3%</td>
</tr>
<tr>
<td>java.io</td>
<td>1.0%</td>
<td>4.0%</td>
<td>6.0%</td>
<td>18.3%</td>
</tr>
<tr>
<td>hive.ql</td>
<td>38.0%</td>
<td>30.7%</td>
<td>16.0%</td>
<td>9.3%</td>
</tr>
<tr>
<td>serDe</td>
<td>2.3%</td>
<td>1.3%</td>
<td>12.4%</td>
<td>29.0%</td>
</tr>
</tbody>
</table>
Gain comes from query engine difference – Spark or Tez

- **Spark Advantage (Q51)**
  - reduce shuffling data by better filtering rule
  - Spark: 366 rows \(\rightarrow\) shuffling 687MB
  - Tez: 8,116 rows \(\rightarrow\) shuffling 3.6GB

- **Tez Advantage (Q50, Q58, Q82)**
  - reduce shuffling data by Bloom Filter

---

*Good query optimizer (Cost Based Optimizer) helps to reduce shuffling data*
Empirical Study Summary - What features affect the performance

- **Query Engine**
  - DAG
    - # of Vertices / Stages (Map or Reduce)
    - amount of shuffling data (intermediate data)
    - input data size (tables)
  - CBO
    - filtering rule

- **JVM**
  - # of threads
  - Intrinsic
  - SerDe performance
  - I/O performance
Agenda

- Motivation, Problems and Challenges
- Backgrounds
  - Backend engines: Spark and Tez
  - Backend runtimes: OpenJDK and J9
- Empirical Study
  - Performance evaluation
  - Performance analysis
- ML model
  - training classification model
  - evaluating classification model
- Summary
Questions and Challenges

1. What about potential gains? *observation*
   - Empirical Study

2. What features make the differences? *reasoning*
   - Meta Scheduler

3. What data to help building a model? *training*
   - ML Model

4. How accurately can the model predict? *predicting*
   - Run SQL on best engine/runtime

GOAL

IEEE CLOUD 2018 / Towards Selecting Best Combination of SQL-on-Hadoop Systems and JVMs
Proposed Classifier Overview - Training and Prediction

- **Key points**
  - Making classifier model based on the features that come from DAG
  - Selecting a combination of the system based on the model before query execution
Training Classifier

- Why extract features from DAG? Why not SQL?
  - contains much more info including table stats/actual stages than SQL

- What features are used
  - # of stages, # of joins, join types, used tables, etc.
  - 69 features in total
Predicting best combination using classifier

- **Extract features without actual query run**
  - sql explain generates DAG (compiling it in 2-5 sec)
- **Predict best system for the query**
  - decide a combination based on the classifier

```
expressed

© 2016 IBM Corporation
```
Evaluation of classifier

- **Training and testing four ML algorithms**
  - kNN, Decision Tree, SVM, Random Forest
  - k-fold cross-validation (split data into 80:20)

- **Models**
  - binary class: Spark or Tez
  - multi class: Spark/OpenJDK or Spark/J9 or Tez/OpenJDK or Tez/J9

<table>
<thead>
<tr>
<th></th>
<th>binary classifier</th>
<th>multinomial classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>stddev (+/-)</td>
</tr>
<tr>
<td>kNN</td>
<td>0.65</td>
<td>0.08</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.69</td>
<td>0.17</td>
</tr>
<tr>
<td>SVM</td>
<td>0.72</td>
<td>0.10</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.72</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Random Forest is better than others

- # of stages makes impact to the model
- Using BF or Join types do not affect

Features Impact in Random Forest
Evaluation of classifier

- **training and testing model**
  - k-fold cross-validation except test query feature
- **Result**
  - baseline: exec time with Spark/J9 only
  - ascending order

![Graph showing accumulated execution time with different configurations and their performance improvements.]

- reduced by 35%
- reduced by 50%

**big miss prediction**
Summary and Future Works

**Summary**

- No single query engine and JVM is best for all queries
- Query engine mismatch leads up to 10x drawback
- JVM mismatch also leads up to 3x drawback
- Proposed Random Forest based classifier achieved 50% time reduction in total

**Future Works**

- Implements meta scheduler
- Applies it on Cloud/Container/Kubernetes environment
- Training data augmentation