TPC-DS and Apache Beam - the time has come!

Alexey Romanenko (@AlexRomDev)
Principal Software Engineer, Talend
Apache Beam PMC Member

Ismaël Mejía (@iemejia)
Principal Software Engineer, Talend
Cloud Advocate, Microsoft
Apache Beam PMC Member
WARNING #1

This is a Work In Progress (WIP) presentation

Lots of early progress but also still lots of things to be done
Motivation: Beam Overhead Performance FUD or Reality?
User Reports of Performance Issues

Beam / BEAM-9440
Performance Issues with Beam Runners compared with Native Systems

Details
- Type: Bug
- Priority: P3
- Affects Version/s: None
- Component/s: runner-apex, runner-flink, runner-spark
- Labels: None
- Status: OPEN
- Resolution: Unresolved
- Fix Version/s: None

Description
While doing a performance evaluation of Apache Beam with Spark Runner - I found that even for a simple word count problem on a text file – Beam with Spark runner was slower by a factor of 5 times as compared to Spark for a dataset as small as 14 GB.
Our employer (product team comments)

“Previewing the result of pipelines takes too much time (~30s for tiny data)”
Anonymous Software Engineer #1

“Running jobs takes too long”
Anonymous QA Engineer #1
Quantitative Impact Evaluation of an Abstraction Layer for Data Stream Processing Systems

Guenter Hesse*, Christoph Matthies*, Kelvin Glass†, Johannes Huegle* and Matthias Uflacker*
*Hasso Plattner Institute
University of Potsdam
Email: firstname.lastname@hpi.de
†Department of Mathematics and Computer Science
Freie Universität Berlin
Email: kelvin.glass@fu-berlin.de

Abstract—With the demand to process ever-growing data volumes, a variety of new data stream processing frameworks have been developed. Moving an implementation from one such system to another, e.g., for performance reasons, requires adapting existing applications to new interfaces. Apache Beam addresses these high substitution costs by providing an abstraction layer that enables executing programs on any of the supported streaming frameworks. In this paper, we present a novel benchmark architecture for comparing the performance impact of using Apache Beam on three streaming frameworks: Apache Spark Streaming, Apache Flink, and Apache Apex. We find significant performance penalties when using Apache Beam for application development in the surveyed systems. Overall, usage of Apache Beam for the examined streaming applications caused a high variance of query execution times with a slowdown of up to a factor of 58 compared to queries developed without the abstraction layer. All developed benchmark artifacts are publicly Apache Beam [2], which provides a unified programming model for describing both batch and streaming data-parallel processing pipelines. Pipelines are described using a single Software Development Kit (SDK) and can then be executed by a variety of different frameworks, without developers needing detailed knowledge of the employed implementations. Thus, execution frameworks can be exchanged without the need to adapt code. As an additional benefit, Apache Beam enables to benchmark multiple systems with a single implementation. Conceptually, this idea can be compared to object-relational mapping (ORM), where data stored in database tables is encapsulated in objects. Data can be queried and manipulated just by using these objects instead of writing SQL [3].

A question concerning abstraction layers is if their usage has
Quantitative Impact Evaluation of an Abstraction Layer for Data Stream Processing Systems

Abstract—With the demand to process ever-growing data volumes, a variety of new data stream processing frameworks have been developed. Moving an implementation from one such system to another, e.g., for performance reasons, requires adapting existing applications to new interfaces. Apache Beam addresses these high substitution costs by providing an abstraction layer that enables executing programs on any of the supported streaming frameworks. In this paper, we present a novel benchmark architecture for comparing the performance impact of using Apache Beam on three streaming frameworks: Apache Spark Streaming, Apache Flink, and Apache Apex. We find significant performance penalties when using Apache Beam for application development in the surveyed systems. Overall, usage of Apache Beam for the examined streaming applications caused a high variance of query execution times with a slowdown of up to a factor of 58 compared to queries developed without the abstraction layer. All developed benchmark artifacts are publicly available to ensure reproducible results.

* This is worse case but paper highlights an average 3-7X overhead
The eternal question:

What is the overhead of Beam?

Let’s run a benchmark to find out…

“If you can’t measure it, you can’t improve it.”
WARNING #2

Performance results can be HEAVILY biased

**Benchmarketing.** Convenient narrative (my project as the winner)
Results not reproducible or worse backed by any data

Is Raw Performance the only thing that matters?
- Correctness
- Reliability
- Price
**Nexmark** (current Beam benchmark)

Benchmark for queries over data streams

Online Auction System

Example:
Query 4: What is the average selling price for each auction category?
Nexmark

8 (+5) benchmark queries of a continuous processing system

- Continuous queries a good match for the Beam Model
- Run regularly on Beam and helped find MANY issues + regressions

but
- Not ran at scale (*mea culpa* - Ismaël)
- Unpublished research paper (not Industry standard)
- You cannot compare results with other systems
What is TPC-DS?
TPC-DS Benchmark

TPC-DS is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance.

- Industry standard benchmark (OLAP/Data Warehouse)
  - [http://www.tpc.org/tpcds/](http://www.tpc.org/tpcds/)

- Implemented for many analytical processing systems
  - RDBMS, Apache Spark, Apache Flink, etc

- Wide range of different queries (SQL)

- Existing tools to generate input data of different sizes
TPC-DS Tables
TPC-DS Input Data

Data source:

- Input files are generated with CLI tool (CSV)
- The tool constraints the minimum amount of data to be generated to 1GB.
- TPC-DS dsdgen tool for text (CSV) generation.

Generated datasets:

- Total sizes: 1GB / 10GB / 100GB / 1000GB
TPC-DS Queries

- **99 distinct** SQL-99 queries (including OLAP extensions)
- Each query answers a **business question**, which illustrates the business context in which the query could be used
- All queries are “*templated*” with random **input parameters**.
- Used to **compare SQL implementation** of completeness and performance
TPC-DS via Beam SQL
TPC-DS extension in Beam

- **Goals:**
  - Compare the performance of Beam SQL for different runners and their different versions
  - Run Beam SQL on different environments
  - Detect missing Bean SQL features / incompatibilities
  - Find Performance issues in Beam
TPC-DS extension in Beam

- Initially contributed by Yuwei Fu as a part of GSoC 2020 project [BEAM-9891]
  - Supported only Dataflow runner
  - Text files (CSV) as an input source
  - 3 (of 99) queries passing

- Later adjusted
  - + support of Spark Runner
  - + support of Parquet input (on the way). Why? Let’s talk a bit later....
  - + 25 (of 103) queries passing
TPC-DS extension in Beam

- 103 SQL queries (99 + 4) to run
  - 25 passed
  - 78 failed
- The most common issues:
  - “java.lang.UnsupportedOperationException: Non equi-join is not supported”
  - “java.lang.UnsupportedOperationException: CROSS JOIN, JOIN ON FALSE is not supported!”
  - “java.lang.UnsupportedOperationException: ORDER BY without a LIMIT is not supported!”
  - org.apache.calcite.plan.RelOptPlanner$CannotPlanException: There are not enough rules to produce a node with desired properties: convention=BEAM_LOGICAL. All the inputs have relevant nodes, however the cost is still infinite.”
Different implementations of TPC-DS queries in Beam
**TPC-DS Query 3**

*Query3* is a good example that contains all main data processing primitives (filtering, aggregation, sorting, selecting, etc) and implemented in different ways as Beam and Spark pipelines.

```sql
SELECT dt.d_year, item.i_brand_id brand_id, item.i_brand brand, SUM(ss_ext_sales_price) sum_agg
FROM date_dim dt, store_sales, item
WHERE dt.d_date_sk = store_sales.ss_sold_date_sk
  AND store_sales.ss_item_sk = item.i_item_sk
  AND item.i_manufact_id = 128
  AND dt.d_moy=11
GROUP BY dt.d_year, item.i_brand, item.i_brand_id
ORDER BY dt.d_year, sum_agg desc, brand_id
LIMIT 100
```
TPC-DS Query 3, Beam SQL, CSV

```java
@DefaultSchema(JavaFieldSchema.class)
private static class DateDim {
  public final int d_date_sk;
  public final int d_year;
  public final int d_moy;
}
```
Is CSV the best format to benchmark?

- Works with the TPC-DS generated data via `dsdgen`
- Nice to compare with other benchmarks running on raw TPC-DS
- CSV-like format is not good enough for SQL data optimizations
  - column projection, filter predicates, etc
- More realistic Big Data use case (Datalake)

Parquet to the rescue!

*Databricks* TPC-DS Kit to generate Parquet files (re-uses `dsdgen`)
TPC-DS Query 3, Beam SQL, Parquet

```java
Schema schemaDateDim = Utils.getAvroSchema("date_dim");
Schema schemaDateDimProjected =
    getProjectedSchema(new String[] {"d_date_sk", "d_year", "d_moy"}, schemaDateDim);

PCollection<GenericRecord> recordsDateDim = pipeline.apply(
    ParquetIO.read(schemaDateDim)
    ... .withProjection(schemaDateDimProjected);
...

PCollection<GenericRecord> recordsStoreSales = ...;
PCollection<GenericRecord> recordsItem = ...;

PCollectionTuple tuple = PCollectionTuple.of(
    new TupleTag<>("date_dim"), recordsDateDim)
    .and(new TupleTag<>("store_sales"), recordsStoreSales)
    .and(new TupleTag<>("item"), recordsItem);
```
TPC-DS Query 3, Beam SQL, Parquet

ParquetIO .read("date_dim")
ParquetIO .read("store_sales")
ParquetIO .read("item")

PCollection<GenericRecord>
PCollection<GenericRecord>
PCollection<GenericRecord>

PCollectionTuple.of(…)

SqlTransform .query("SELECT...")
RowToCsv(csvFormat)

PCollection<Row>
PCollection<String>

TextIO.write()
Other missing SQL features

**BEAM-12315** Support PARTITIONED BY on Beam's SQL DDL
Databricks TPC-DS Parquet generation tool partitions date columns as paths

**BEAM-7929** ParquetTable support for column projection and filter predicate
We completed Column Projection, Filter Predicate (pending PR)

**BEAM-12134** Add Table statistics / Row estimation for ParquetTable
(Cost-Based Optimization)
- E.g. Query3 joins 2 small tables with a big one (star-like) so it could benefit of a Map-Side based Join strategy
TPC-DS Query 3, Beam SDK, CSV

1. **TextIO.read()** from("/path/to/date_dim") → PCollection<String>
   - `WHERE dt.d_moy=11` → FilterByFieldValue() + SelectFieldsFn()

2. **TextIO.read()** from("/path/to/store_sales") → PCollection<String>
   - `WHERE dt.d_date_sk = store_sales.ss_sold_date_sk` → FilterByFieldValue() + CoGroupByKey
   - `GROUP BY (dt.d_year, item.i_brand, item.i_brand_id)` → StringToKVDofFn() + CoGroupByKey
   - `SUM(ss_ext_sales_price)` → GroupByKey.of()
   - `ORDER BY dt.d_year, sum_agg desc, brand_id` → Top.of()
   - `LIMIT 100` → ParDo.of()

3. **TextIO.read()** from("/path/to/item") → PCollection<String>
   - `WHERE item.i_manufact_id = 128` → FilterByFieldValue() + SelectFieldsFn()

4. **TextIO.read()** from("/path/to/item") → PCollection<String>
   - `SELECT dt.d_year, item.i_brand_id, item.i_brand, sum_agg` → ParDo.of()

5. **TextIO.read()** from("/path/to/store_sales") → PCollection<String>
   - `WHERE store_sales.ss_item_sk = item.i_item_sk` → StringToKVDofFn() + CoGroupByKey

6. **TextIO.read()** from("/path/to/store_sales") → PCollection<String>
   - `WHERE store_sales.ss_item_sk = item.i_item_sk` → StringToKVDofFn() + CoGroupByKey

7. **TextIO.write()**
TPC-DS Query 3, Beam SDK, Parquet

**ParquetIO**
- `.read("date_dim")` .withProjection(…)
- `.read("store_sales")` .withProjection(…)
- `.read("item")` .withProjection(…)

**PCollection<GenericRecord>**
- ParquetIO
- `ParquetIO.read("date_dim") .withProjection(…)`
- `ParquetIO.read("store_sales") .withProjection(…)`
- `ParquetIO.read("item") .withProjection(…)`

**Filter.by()**
- WHERE `dt.d_moy=11`
- WHERE `store_sales.ss_item_sk = item.i_item_sk`
- WHERE `item.i_manufact_id = 128`

**GroupByKey**
- GROUP BY `(dt.d_year, item.i_brand, item.i_brand_id)`

**GenericRecordToKVDoFn**
- `+ CoGroupByKey` + `+ KVToGenericRecordDoFn`

**ParDo.of()**
- SELECT `dt.d_year, item.i_brand_id as brand_id, item.i_brand as brand, sum_agg` into CSV format

**TextIO.write()**
- format results into CSV format

**ParDo.of()**
- ORDER BY `dt.d_year, sum_agg desc, brand_id LIMIT 100`
Local benchmark runs

**Configuration:**

Dependencies:
- Beam 2.28.0
- Spark 2.4.7

4 workers
- local[4] or parallelism=4

1Gb input data set
- Parquet / CSV, local files
- Macbook Pro 2017, 2,9 GHz Intel Core i7, RAM 16 GB
TPC-DS Query 3, 1Gb dataset, Spark
Distributed Execution Time!
WARNING #3

Fair Benchmarking is HARD

- Instance Variability (CPU/RAM speed)
- Cloud Networking Performance Variability
- Bad Default Configurations
- Other silly configuration issues
Cluster Setup – Amazon EMR

**yarn-cluster**

m4.xlarge (4 CPUS / 16GB RAM)
- 1 YARN master
- 5 YARN workers (1 master + 4 workers)

**AWS EMR 5.32.0 (us-east-1)**
- Hadoop 2.10.1
- Spark 2.4.7
- Flink 1.11.2

Input dataset and results in **AWS S3**

**Goal:** Test default configurations. Only change for similarity between systems purposes for example same parallelism.
Cluster runs – Amazon EMR

Implementations (5)
- Beam SDK Parquet
- Beam SQL Parquet
- Beam SQL CSV
- Spark SQL Parquet
- Spark SQL CSV

Datasets (4)
- 1GB
- 10GB
- 100GB
- 1000GB*

Runs (3)

* Beam SQL CSV version not working on this size on Spark Runner (yet)
<table>
<thead>
<tr>
<th>1GB</th>
<th>10GB</th>
<th>100GB</th>
<th>1000GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>❌</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Runtime System Issues

- **BEAM-11958** Jackson MethodNotFoundException on EMR ✔
  - AWS SDK for Java available by default in the EMR classpath and it uses a Jackson dependency older than Beam’s.

- **BEAM-10430** Jackson JaxbAnnotationModule breaks Flink Runner on EMR 🚧
<table>
<thead>
<tr>
<th></th>
<th>1GB</th>
<th>10GB</th>
<th>100GB</th>
<th>1000GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✅</td>
<td>❌</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IO and File System surprises

**BEAM-12070** Make ParquetIO splittable by default ✓
ParquetIO Read default implementation was NOT Splittable so it OOM-ed on workers

```java
ParquetIO
  .read(schema)
  .from(path)
  .withSplit()
```
IO and File System surprises

**BEAM-11972** ParquetIO should close all opened channels/readers ✔
AWS S3 cancels reads when connections are kept open.

**BEAM-12329** S3 logs warnings about non-drained InputStreams ✔
<table>
<thead>
<tr>
<th></th>
<th>1GB</th>
<th>10GB</th>
<th>100GB</th>
<th>1000GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
Cluster runs - Google Dataflow

Constrained cluster 4 x e2-standard-4 workers + Data stored on Google Cloud Storage (GCS)
Cluster runs – Google Dataflow

Implementations (3)
- Beam SDK Parquet
- Beam SQL Parquet
- Beam SQL CSV

Datasets (4)
- 1GB
- 10GB
- 100GB
- 1000GB

Runs (3x2)
- 4 workers
- Unlimited workers
Credit where credit is due: Everything ran smoothly on Dataflow
Runner Translation Issues

Reports of read performance regressions on the Mailing List (ML)

25-30% Read performance degradation on Spark Runner when using the SplittableDoFn based Read translation

BEAM-10670 Use non-SDF translation for Read by default on Spark Runner ✔

No performance difference on Dataflow 😐
And finally the benchmark results...
Cluster runs – EMR – Query 3 – CSV - 1TB
Cluster runs – EMR – Query 3 – CSV – 1-100 GB

![Query 3 - CSV - 1/10/100 GB](image)
Cluster runs – EMR – Query 3 - Parquet - 1TB
Cluster runs – EMR – Query 3 – Parquet – 1-100GB
Cluster runs – Google Dataflow – Unlimited Workers

* Unlimited workers give us results closer to raw Spark
Now the question became:

Where is the overhead of Beam?
Issues affecting TPC-DS on Beam performance

- SQL features and optimizations
- IO Connector
- Runtime
- Runner translation
- Implementation issues
- Beam Model overhead
# Finding implementation issues - Profiling Beam

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Live Bytes (%)</th>
<th>Live Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Arrays.copyOf(char[], int)</td>
<td>954.17%</td>
<td>1,703,599</td>
</tr>
<tr>
<td>java.lang.AbstractStringBuilder.ensureCapacityInternal(int)</td>
<td>953.54%</td>
<td>952,200</td>
</tr>
<tr>
<td>java.lang.StringBuilder.append(String)</td>
<td>952.20%</td>
<td>952,200</td>
</tr>
<tr>
<td>org.apache.beam.sdk.schemas.utils.AvroUtils.checkTypeName</td>
<td>945.46%</td>
<td>945,466</td>
</tr>
<tr>
<td>sun.net.www.protocol.jar.Handler.parseURL</td>
<td>1,664 B</td>
<td>1,664 B</td>
</tr>
<tr>
<td>java.net.URLStreamHandler.parseURL</td>
<td>1,504 B</td>
<td>1,504 B</td>
</tr>
<tr>
<td>java.io.UnixFileSystem.resolve(String, String)</td>
<td>1,216 B</td>
<td>1,216 B</td>
</tr>
<tr>
<td>com.sun.jmx.remote.internal.ServerCommunicatorAdmin.logtime</td>
<td>968 B</td>
<td>968 B</td>
</tr>
<tr>
<td>java.lang.management.ThreadInfo.initialize</td>
<td>760 B</td>
<td>760 B</td>
</tr>
<tr>
<td>org.slf4j.impl.SimpleLogger.log</td>
<td>216 B</td>
<td>216 B</td>
</tr>
<tr>
<td>java.lang.AbstractStringBuilder.append(char[], int, int)</td>
<td>680 B</td>
<td>680 B</td>
</tr>
<tr>
<td>java.lang.AbstractStringBuilder.append(char)</td>
<td>636 B</td>
<td>636 B</td>
</tr>
<tr>
<td>java.lang.String.concat(String)</td>
<td>584 B</td>
<td>584 B</td>
</tr>
<tr>
<td>java.lang.String.&lt;init&gt;(char[])</td>
<td>48 B</td>
<td>48 B</td>
</tr>
<tr>
<td>java.lang.String.&lt;init&gt;</td>
<td>438,944 B</td>
<td>438,944 B</td>
</tr>
<tr>
<td>java.lang.StringBuilder.toString()</td>
<td>435,200 B</td>
<td>435,200 B</td>
</tr>
<tr>
<td>org.apache.beam.sdk.schemas.utils.AvroUtils.checkTypeName</td>
<td>415,966 B</td>
<td>415,966 B</td>
</tr>
</tbody>
</table>

**BEAM-12210** Performance issue in AvroUtils.checkTypeName

**BEAM-12247** Reduce memory allocations in InMemoryTimerInternals

**BEAM-12248** Reduce ArrayList allocation in Row/RowUtils

*Thanks Dan Kulp for the profiling analysis/fixes*
Beam Model Overhead

“Every element on Beam has an associated event timestamp”

WindowedValue is the internal representation of a value (~13 bytes overhead)

\{value, timestamp, [windows], paneInfo\}

- More memory required per-element + extra GC
- Bigger shuffle size

Shuffle overhead can be improved by smarter encoding
Beam Model Overhead

**GroupByKey** in Beam also groups by Window, and...

- Merge windows if possible
- Adjust timestamps if multiple inputs
- Drop data from expired windows
- Emit results based on triggers

Extra CPU use and more GC
And there is also the Timers/State overhead

[BEAM-12135](https://issues.apache.org/jira/browse/BEAM-12135) Batch optimized translation for Spark Runner
Row conversion and Coder improvements (2-3%)

**BEAM-12135** Use ParamWindowedValueCoder for Bounded PCollections 🚧

**BEAM-11571** Avoid conversion if input and output types are equal on Convert transform ✔

**BEAM-12328** Conversion from Avro GenericRecords to Beam Rows takes too much time 🚧
Conclusions
Contributions

8  Issues required to run TPC-DS found (8 fixed)
10 Nice to have issues reported (5 fixed)
11 Performance Improvement Issues found (3 fixed / 3 pending)
Lessons Learned

- Defaults matter (A LOT!)
- You need real life scale runs to find real life issues
- Execution in different platforms/clouds matters for a project like Beam
- Measuring Big Data pipelines performance in the cloud is hard
- It is hard to compare specific execution systems:
  - Spark has been optimized for the batch data-lake use case for at least 6 years
- Price of abstractions
Future work

Still LOTS of things to do (wanna join?)

- Make more queries pass and other SQL improvements
- Automate daily runs on big size datasets
- Benchmark open source runners on Google Dataproc / Kubernetes
- Continue performance improvements runners translation
- Run also via Portability: Python / Go queries

Open Questions
- Are native system optimizations blocked by model translation?
- Can we have a schema-based translation of Beam pipelines?
Questions ?
Extra Slides
SELECT
    i_item_id,
i_item_desc,
s_store_id,
s_store_name,
sum(ss_quantity) AS store_sales_quantity,
sum(sr_return_quantity) AS store_returns_quantity,
sum(cs_quantity) AS catalog_sales_quantity,
FROM store_sales,
store_returns,
catalog_sales,
date_dim d1,
date_dim d2,
date_dim d3,
store,
item
WHERE
    d1.d_moy            = 9
AND
    d1.d_year           = 1999
AND
    d1.d_date_sk        = ss_sold_date_sk
AND
    i_item_sk           = ss_item_sk
AND
    s_store_sk          = ss_store_sk
AND
    ss_customer_sk      = sr_customer_sk
AND
    ss_item_sk          = sr_item_sk
AND
    ss_ticket_number    = sr_ticket_number
AND
    sr_returned_date_sk = d2.d_date_sk
AND
    d2.d_moy BETWEEN 9 AND 9 + 3
AND
    d2.d_year           = 1999
AND
    sr_customer_sk      = cs_bill_customer_sk
AND
    sr_item_sk          = cs_item_sk
AND
    cs_sold_date_sk     = d3.d_date_sk
AND
    d3.d_year IN (1999,1999+1,1999+2)
GROUP BY
    i_item_id,
i_item_desc,
s_store_id,
s_store_name
ORDER BY
    i_item_id,
i_item_desc,
s_store_id,
s_store_name
LIMIT 100
TPC-DS Query 29

Types of implementations:

Beam SQL
- ParquetIO
- Parquet Table Provider

Beam SDK (Java)
- Avro GenericRecord
- Beam Schema & Row
TPC-DS Query 29, 1Gb dataset
User Reports of Performance Issues

Extremely Slow DirectRunner

Hi all,

I'm experiencing very slow performance and startup delay when testing a pipeline locally. I'm reading data from a Google Pub/Sub subscription as the data source, and before each pipeline execution I ensure that data is present in the subscription (readable from GCP console).

I'm seeing startup delay on the order of minutes with DirectRunner (5-10 min). Is that expected? I did find a Jira ticket[1] that at first seemed related, but I think it has more to do with BQ than DirectRunner.

I've run the pipeline with a debugger connected and confirmed that it's minutes before the first DoFn in my pipeline receives any data. Is there a way I can profile the direct runner to see what it's churning on?