Introduction to performance testing in Apache Beam

Alexey Romanenko
Principal Software Engineer, Talend
Apache Beam PMC Member

BEAM
SUMMIT

Intro

Performance testing vs Benchmarking

***Performance testing** is in general a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload.*

It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

https://en.wikipedia.org/wiki/Software_performance_testing

***Benchmark** is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it.*

https://en.wikipedia.org/wiki/Benchmark_(computing)

Why do we need performance testing in Beam?

- Measure a **runner performance** and detect performance degradation (if any)
  - e.g. between two Beam releases or periodically
- Test how Beam pipelines run **under the load**
- Compare the performance for **different runners** and **SDK** in Beam
  - Same test suite, same datasets, same environment (well, we do our best…)
- Compare the performance between Beam **runners** and **native engines**
  - Sensible topic =)

Performance testing in Beam

IO transform integration tests

Core Beam Operations tests

Nexmark suites

TPC-DS suites

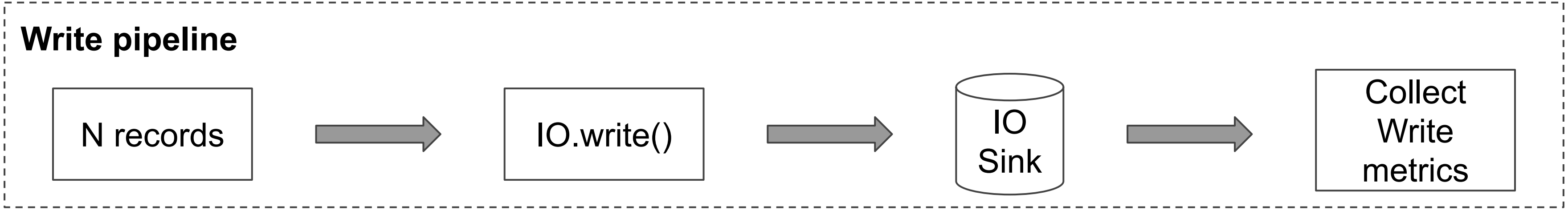
IOIT

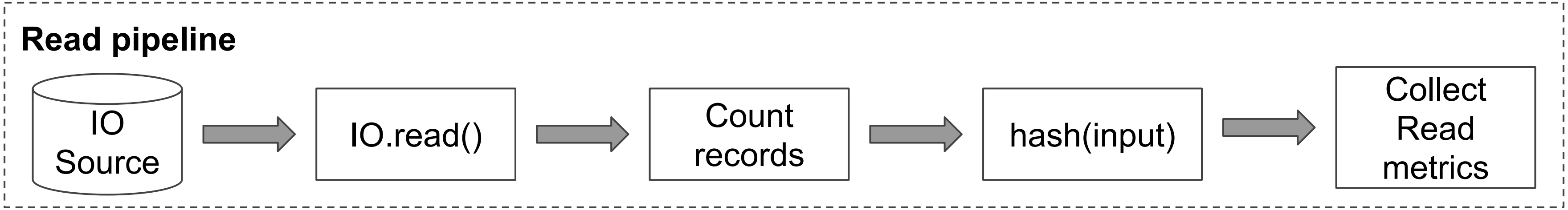
BEAM SUMMIT

IOIT (IO Integration Tests)

- “**2-in-1**”: integration and performance tests (depending on input data size)
- Intended to be implemented for **every IO** connector
  - Some IOs are still missing
- Only **batch** mode
  - *BoundedSource* has to be used for streaming pipelines
- For now, implemented only for **Java SDK**
- Supported runners:
  - Any runner that supports Java SDK
- Run manually / on Jenkins
- Grafana dashboard integration

IOIT: Common scenario

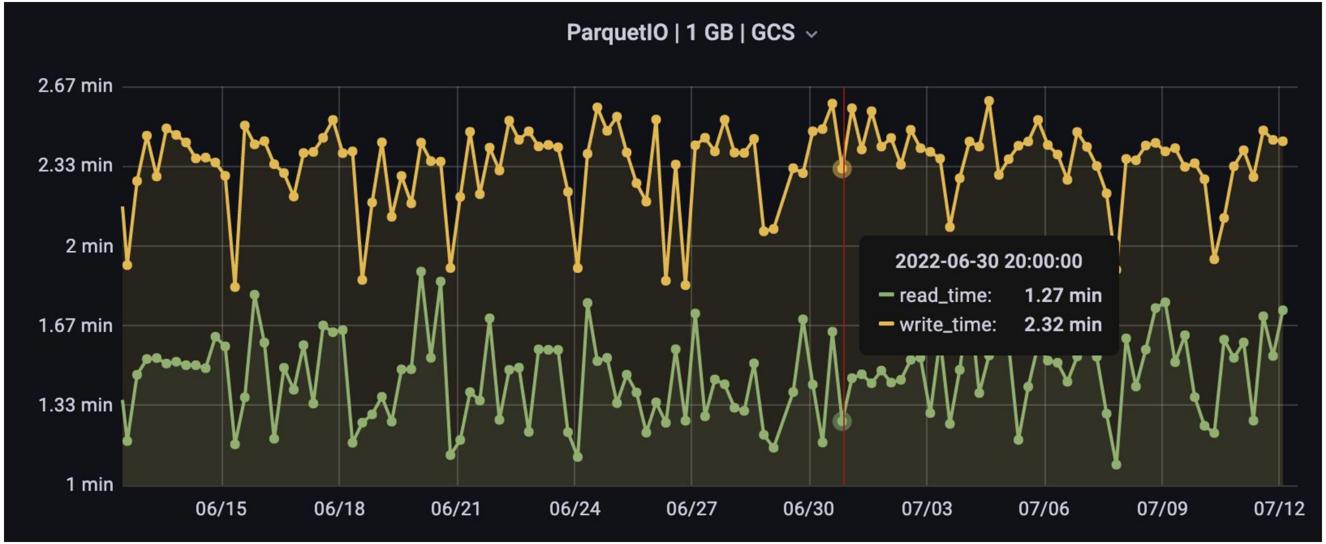




IOIT (IO Integration Tests)

Collected metrics:

- Read time
- Write time



IOIT: Pros/Cons

Pros:

- Leverage the **same code** as for ITs
  - Most Java IOs already has them
- **Easy** to implement for new IO
- Runs against **real** (or *k8s*) data backends

Cons:

- Only **Java** SDK and Batch mode
- Very **few** metrics
- **Limited** number of predefined input records (N)

Core Beam Operations

BEAM SUMMIT

Core Beam Operations Load Tests

- Test performance of the core beam operations from Apache Beam model on different runners:
    - ParDo, ParDo with SideInput, GroupByKey, CoGroupByKey, Combine
- Uses *Synthetic Source* and *Synthetic Step*
- Supports **Batch** and **Streaming**
- SDK supported:
    - Java SDK, Python SDK and Go SDK
- **Runners** supported:
    - Dataflow, Flink, Spark (Dataset)
- Runs on **Jenkins**
- **Grafana** dashboard integration

Synthetic Source & Step

Synthetic Source is a highly parameterizable Source that provides deterministic data (*KV<byte[], byte[]>*).

Provided options:

- Seed
- Key and value size
- Hot keys
- Delay between consequent data emissions
- Number of generated records
- … and others

Synthetic Step is a highly parameterizable DoFn that consumes *KV<byte[], byte[]>* and emits *KV<byte[], byte[]>*.

Provided options:

- Actions between data emissions
- Delay per bundle
- Upper throughput limit
- … and others

+ iterations
+ fanout

Core Beam Operations Load Tests

Gathered metrics:

- Run time
- Consumed bytes
- Memory usage
- Split/bundle count
- Throughput / lag (for streaming scenarios)

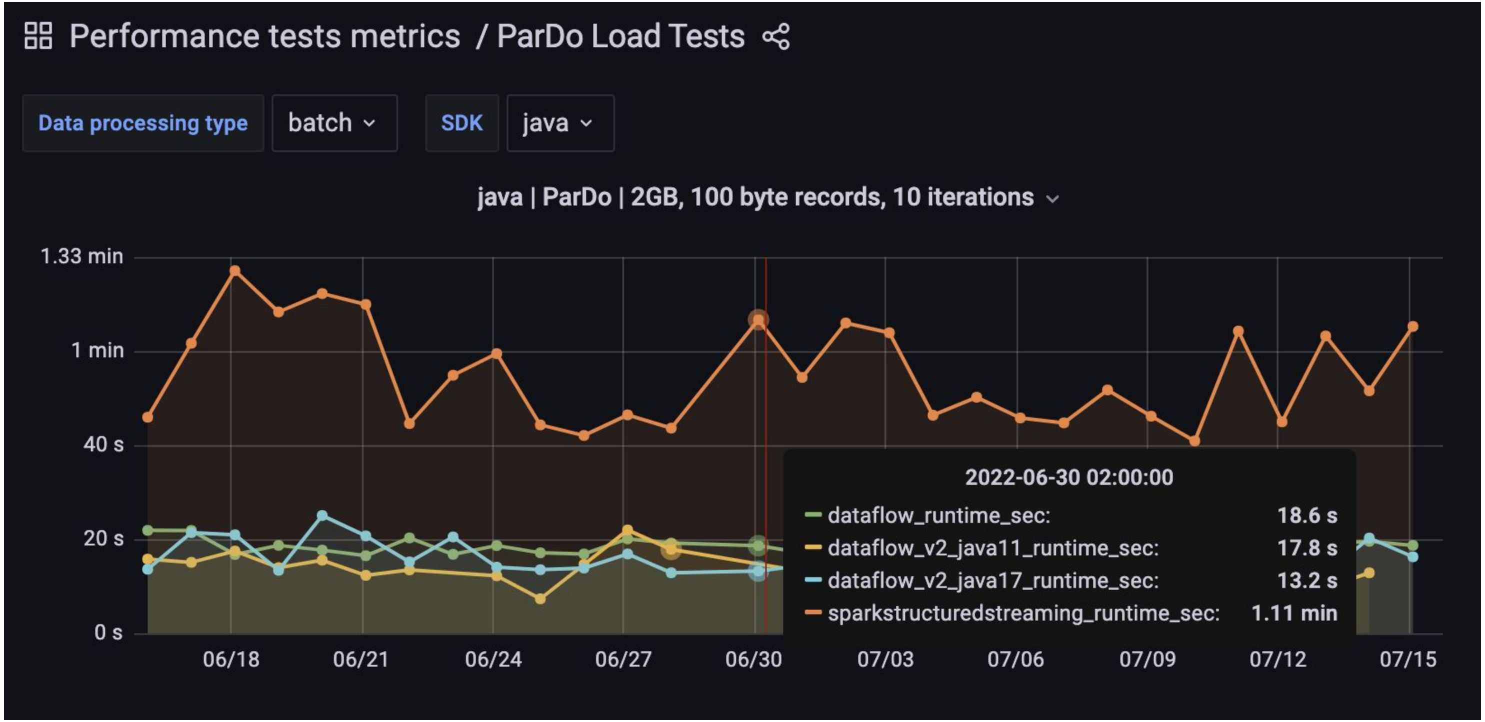
Example: ParDoLoadTest

```
PCollection<KV<byte[], byte[]>> input =
   pipeline
       .apply("Read input", readFromSource(sourceOptions)) // Synthetic Source
       .apply(ParDo.of(runtimeMonitor))
       .apply(ParDo.of(new ByteMonitor(METRICS_NAMESPACE, "totalBytes.count")));

for (int i = 0; i < options.getIterations(); i++) {
 input =
     input.apply(
         String.format("Step: %d", i),
         ParDo.of(
             new CounterOperation<>(
                 options.getNumberOfCounters(), options.getNumberOfCounterOperations())));
}

input.apply(ParDo.of(runtimeMonitor));
```

Example: ParDoLoadTest



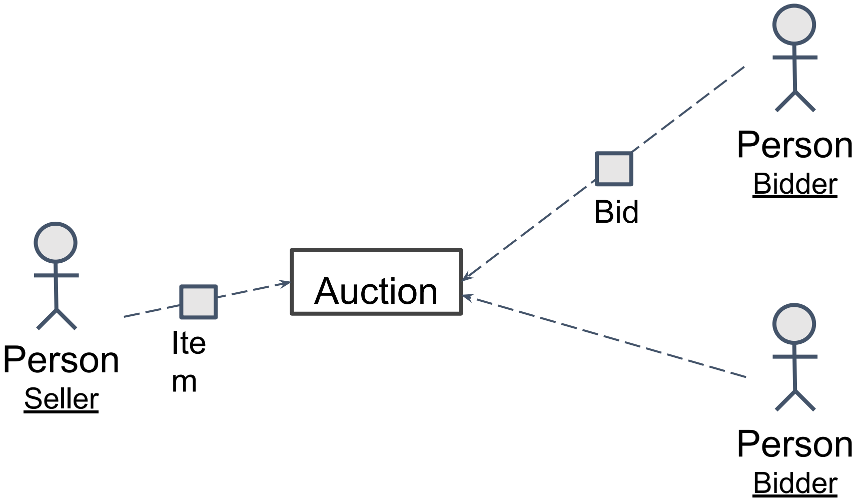
Nexmark

Nexmark benchmark suite

Nexmark is a suite of pipelines inspired by the '*continuous data stream*' queries in Nexmark research paper

These are multiple queries over a three entities model representing on online auction system:

- **Person** represents a person submitting an item for auction and/or making a bid on an auction.
- **Auction** represents an item under auction.
- **Bid** represents a bid for an item under auction.



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

Nexmark

9 (+6) benchmark queries of a continuous processing system

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

but

- Not running at big scale
- Not Industry standard
- We can’t compare results with other systems (only inside Beam)

Nexmark in Beam

- Supports **batch** and **streaming** pipelines
- Implemented only for **Java SDK**
  - non-SQL
  - SQL
- Running on:
  - Dataflow runner
  - Spark (RDD and Dataset) runner
  - Flink runner
- Used to detect **performance regression** for Beam releases

Nexmark: Default configuration

Events generation

- 100 000 events generated
- 100 generator threads
- Event rate in SIN curve
- Initial event rate of 10 000
- Event rate step of 10 000
- 100 concurrent auctions
- 1000 concurrent persons bidding / creating auctions

Windows

- size 10s
- sliding period 5s
- watermark hold for 0s

Events Proportions

- Hot Auctions = ½
- Hot Bidders =¼
- Hot Sellers=¼

Technical

- Artificial CPU load
- Artificial IO load

Nexmark: Output

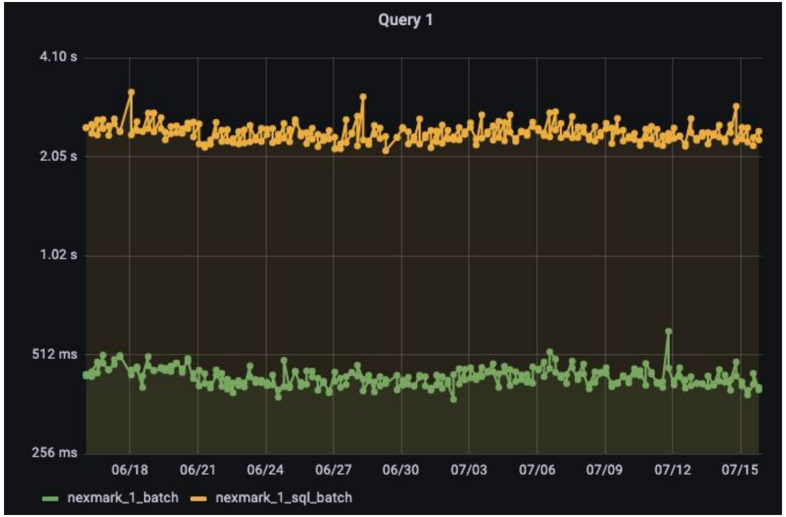
Performance:

Conf	Runtime(sec)	Events(/sec)	Results
0000	5,5	18138,9	100000
0001	4,2	23657,4	92000
0002	2,2	45683,0	351
0003	3,9	25348,5	444
0004	1,6	6207,3	40
0005	5,0	20173,5	12
0006	0,9	11376,6	401
0007	121,4	823,5	1
0008	2,5	40273,9	6000
0009	0,9	10695,2	298
0010	4,0	25025,0	1
0011	4,4	22655,2	1919
0012	3,5	28208,7	1919

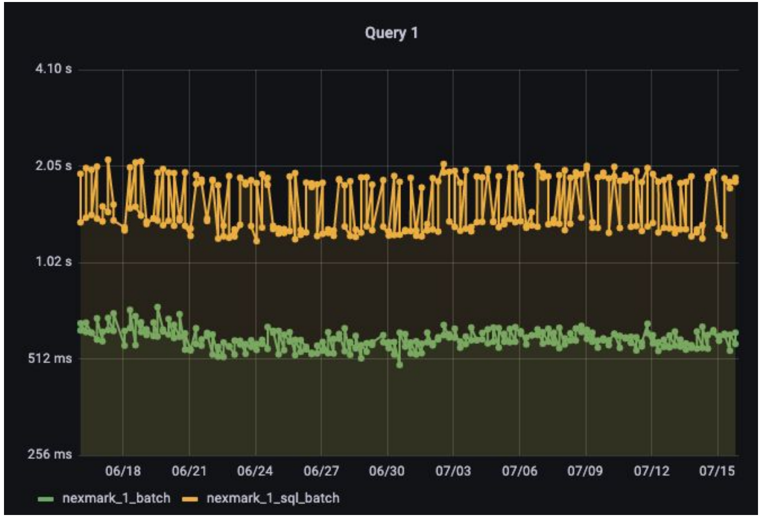
Nexmark: Dashboards

Query1 or CURRENCY_CONVERSION:
What are the bid values in Euro’s? Illustrates a simple map.

SparkRunner (RDD)



SparkRunner (Dataset)



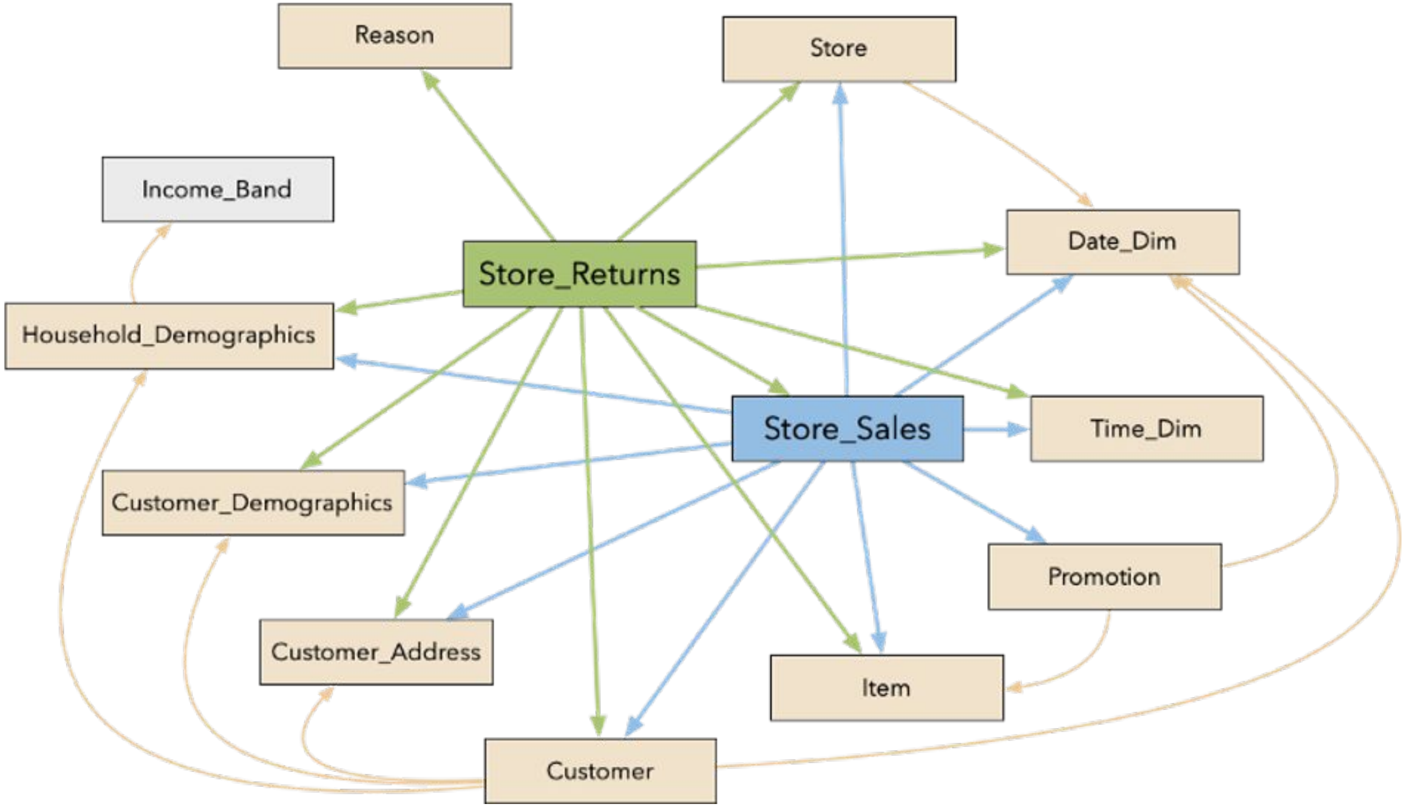
TPC-DS

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/
- 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: Basic tables schema



TPC-DS: Input Data

Data source:

- Input files are generated with CLI tool (CSV)
- The tool constrains the minimum amount of data to be generated to 1GB.
- TPC-DS *dsdgen* tool for text (CSV) generation.
  - 3rd-party tools to generate input in different formats (Parquet)

Generated datasets:

- Data size scale factors:
  - 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: Query example

Query3 is a good example that contains all main data processing primitives (filtering, aggregation, sorting, selecting, etc)

Report the total extended sales price per item brand of a specific manufacturer for all sales in a specific month of the year.

```
   SELECT dt.d_year, item.i_brand_id brand_id, item.i_brand  brand,
          SUM(<AGGC=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 = <MANUFACT=128>
      AND dt.d_moy=<MONTH.01=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 extension in Beam

- It can be **used** to:
    - Compare the performance of Beam SQL for **different runners** and their different versions
    - Run Beam SQL on **different environments**
    - Detect **missing** Beam SQL **features** / **incompatibilities**
    - Find **performance issues** in Beam

- **Data sources** supported:
    - *CSV* and *Parquet*

- **Runners** supported:
    - *Dataflow, Spark (RDD and Dataset), Flink*

- **25** (of 103) **queries** are passing
    - Many queries are not supported by Beam SQL

TPC-DS: Pros/Cons

Pros:

- **Industry** standard benchmark
- Helped to **find** a bunch of Beam **issues** while running on scale
  - See a talk:
    "TPC-DS and Apache Beam - the time has come!"
    (Ismael Mejía/Alexey Romanenko)

    https://2021.beamsummit.org/sessions/tpc-ds-and-apache-beam/

Cons:

- Still **under development**
  - Requires more attention from Beam community
- Many SQL queries are **not supported** by Beam SQL
  - Can't run the whole benchmark
- Only **batch** mode is supported

Infra

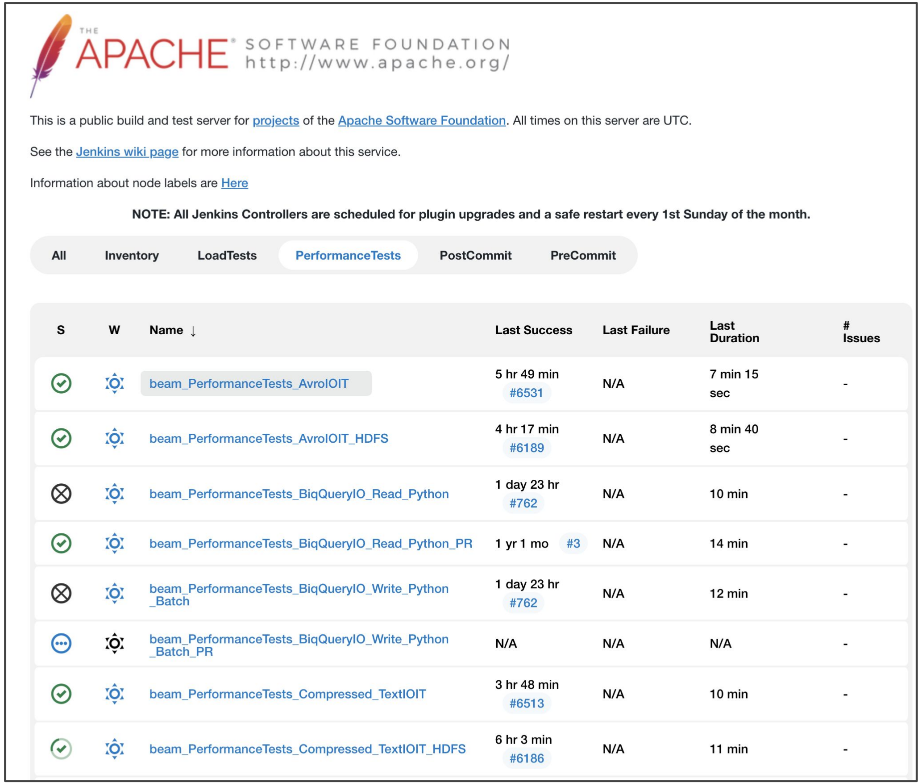
Collect runtime metrics

- Collect metrics
  - Use Metrics API
    - *TimeMonitor* (Java), *MetricsReader* (Python)
  - Custom collector
    - Nexmark, TPC-DS
- Store metrics
  - *BigQuery, InfluxDB*
- Visualisation
  - *PerfKit* (past), *Grafana*

Automation: Jenkins

```
[
  name                : 'beam_PerformanceTests_AvroIOIT',
  description         : 'Runs performance tests for AvroIOIT',
  test                : 'org.apache.beam.sdk.io.avro.AvroIOIT',
  githubTitle         : 'Java AvroIO Performance Test',
  githubTriggerPhrase: 'Run Java AvroIO Performance Test',
  pipelineOptions     : [
    numberOfRecords      : '225000000',
    expectedHash         : '2f9f5ca33ea464b25109c0297eb6aecb',
    datasetSize          : '1089730000',
    bigQueryDataset      : 'beam_performance',
    bigQueryTable        : 'avroioit_results',
    influxMeasurement    : 'avroioit_results',
    numWorkers           : '5',
    autoscalingAlgorithm: 'NONE'
  ]
],
```

https://ci-beam.apache.org/



THE APACHE SOFTWARE FOUNDATION http://www.apache.org/

This is a public build and test server for projects of the Apache Software Foundation. All times on this server are UTC.

See the Jenkins wiki page for more information about this service.

Information about node labels are Here

NOTE: All Jenkins Controllers are scheduled for plugin upgrades and a safe restart every 1st Sunday of the month.

All | Inventory | LoadTests | PerformanceTests | PostCommit | PreCommit

S	W	Name ↓	Last Success	Last Failure	Last Duration	# Issues
		beam_PerformanceTests_AvroIOIT	5 hr 49 min #6531	N/A	7 min 15 sec	-
		beam_PerformanceTests_AvroIOIT_HDFS	4 hr 17 min #6189	N/A	8 min 40 sec	-
		beam_PerformanceTests_BiqQueryIO_Read_Python	1 day 23 hr #762	N/A	10 min	-
		beam_PerformanceTests_BiqQueryIO_Read_Python_PR	1 yr 1 mo #3	N/A	14 min	-
		beam_PerformanceTests_BiqQueryIO_Write_Python_Batch	1 day 23 hr #762	N/A	12 min	-
		beam_PerformanceTests_BiqQueryIO_Write_Python_Batch_PR	N/A	N/A	N/A	-
		beam_PerformanceTests_Compressed_TextIOIT	3 hr 48 min #6513	N/A	10 min	-
		beam_PerformanceTests_Compressed_TextIOIT_HDFS	6 hr 3 min #6186	N/A	11 min	-

Dashboards: Grafana

http://metrics.beam.apache.org/

Beam Metrics Report

Beam Metrics Report

Color legend:

>= 20	>= 10	<= -10	<= -20

dev@beam.apache.org

Possible regression

Measurement	Metric	Runner	Mean previous week	Mean last week	Diff %	Dashboard
go_batch_cogbk_1	dataflow_runtime	-	243.78	274.01	12.4	[1]
go_batch_combine_1	dataflow_runtime	-	406.0	683.05	68.24	[1]
go_batch_gbk_7	dataflow_runtime	-	107.65	133.68	24.18	[1]
java_batch_cogbk_1	dataflow_runtime_sec	-	38.54	45.26	17.45	[1]
java_batch_gbk_7	dataflow_v2_java11_runtime_sec	-	40.36	53.18	31.76	[1]
java_batch_pardo_1	dataflow_v2_java11_runtime_sec	-	14.63	16.85	15.21	[1]
java_streaming_gbk_3	dataflow_v2_java17_runtime_sec	-	66.34	85.2	28.44	[1]
java_streaming_gbk_5	dataflow_v2_java11_runtime_sec	-	594.44	663.12	11.55	[1]
java_streaming_gbk_6	dataflow_v2_java11_runtime_sec	-	143.59	168.9	17.63	[1]
java_streaming_gbk_6	dataflow_v2_java17_runtime_sec	-	157.25	189.98	20.81	[1]
java_streaming_gbk_7	dataflow_v2_java17_runtime_sec	-	200.71	252.81	25.96	[1]
java_streaming_pardo_1	dataflow_runtime_sec	-	18.16	20.29	11.73	[1]
java_streaming_pardo_1	dataflow_v2_java11_runtime_sec	-	29.1	34.64	19.06	[1]
java_streaming_pardo_2	dataflow_v2_java17_runtime_sec	-	146.98	169.42	15.27	[1]
java_streaming_pardo_3	dataflow_v2_java11_runtime_sec	-	24.17	26.76	10.69	[1]
java_streaming_pardo_4	dataflow_v2_java17_runtime_sec	-	44.09	49.9	13.16	[1]
python_streaming_pardo_2	python_dataflow_streaming_pardo_2_runtime	-	2921.57	3254.8	11.41	[1]
tfrecordioit_results	read_time	-	15.5	18.2	17.42	[1]
xmlioit_results	write_time	-	24.96	28.79	15.37	[1]
nexmark_11_batch	RuntimeMs	FlinkRunner	287.89	319.59	11.01	[1] [2] [3] [4]
nexmark_13_sql_streaming	RuntimeMs	DataflowRunner	36914.86	42544.73	15.25	[1] [2]
nexmark_14_batch	RuntimeMs	FlinkRunner	538.61	617.91	14.72	[1] [2] [3] [4]
nexmark_14_streaming	RuntimeMs	DataflowRunner	109147.82	124664.5	14.22	[1] [2]
nexmark_15_batch	RuntimeMs	FlinkRunner	474.89	563.91	18.74	[1] [2] [3] [4]
nexmark_5_streaming	RuntimeMs	DataflowRunner	203490.09	236892.9	16.41	[1] [2]
nexmark_9_batch	RuntimeMs	FlinkRunner	303.61	347.77	14.55	[1] [2] [3] [4]

Possible improvement

Measurement	Metric	Runner	Mean previous week	Mean last week	Diff %	Dashboard
go_batch_sideinput_3	dataflow_runtime	-	2.93	2.44	-16.68	[1]
java_batch_cogbk_3	dataflow_v2_java11_runtime_sec	-	16.07	13.47	-16.18	[1]
java_batch_cogbk_3	dataflow_v2_java17_runtime_sec	-	15.92	13.11	-17.65	[1]
java_batch_gbk_3	dataflow_v2_java17_runtime_sec	-	17.1	14.73	-13.88	[1]
java_batch_gbk_4	dataflow_v2_java17_runtime_sec	-	26.6	23.23	-12.7	[1]
java_batch_gbk_5	dataflow_v2_java17_runtime_sec	-	20.92	17.87	-14.55	[1]
java_batch_gbk_6	dataflow_runtime_sec	-	42.51	34.2	-19.55	[1]
java_batch_pardo_3	dataflow_runtime_sec	-	13.72	11.58	-15.59	[1]
java_batch_pardo_3	dataflow_v2_java11_runtime_sec	-	16.6	14.2	-14.44	[1]
java_streaming_gbk_1	dataflow_v2_java11_runtime_sec	-	3936.29	1461.72	-62.87	[1]
java_streaming_gbk_1	dataflow_v2_java17_runtime_sec	-	3338.54	1258.08	-62.32	[1]
python_batch_combine_5	python_dataflow_batch_combine_5_runtime	-	53.43	36.4	-31.87	[1]
python_batch_gbk_3	python_dataflow_batch_gbk_3_runtime	-	27.71	24.0	-13.4	[1]
python_batch_sideinput_5	python_dataflow_batch_sideinput_5_runtime	-	46.14	40.8	-11.58	[1]
python_streaming_cogbk_1	python_dataflow_streaming_cogbk_1_runtime	-	5931.0	1494.2	-74.81	[1]
python_streaming_cogbk_2	python_dataflow_streaming_cogbk_2_runtime	-	1164.0	414.6	-64.38	[1]
python_streaming_cogbk_3	python_dataflow_streaming_cogbk_3_runtime	-	10235.0	544.4	-94.68	[1]
python_streaming_gbk_3	python_dataflow_streaming_gbk_3_runtime	-	147.86	89.8	-39.27	[1]
python_streaming_gbk_6	python_dataflow_streaming_gbk_6_runtime	-	2378.33	649.6	-72.69	[1]
python_streaming_gbk_7	python_dataflow_streaming_gbk_7_runtime	-	2211.67	530.5	-76.01	[1]
python_streaming_pardo_1	python_dataflow_streaming_pardo_1_runtime	-	251.43	219.0	-12.9	[1]
nexmark_12_batch	RuntimeMs	FlinkRunner	204.25	176.86	-13.41	[1] [2] [3] [4]
nexmark_16_batch	RuntimeMs	FlinkRunner	250.21	214.0	-14.47	[1] [2] [3] [4]
nexmark_6_streaming	RuntimeMs	FlinkRunner	734.86	607.18	-17.37	[1] [2]
nexmark_7_sql_batch	RuntimeMs	DirectRunner	504902.43	371615.77	-26.4	[1] [2] [3] [4]
nexmark_7_sql_streaming	RuntimeMs	DirectRunner	2184798.29	1417868.68	-35.1	[1] [2]

Some conclusions

- Performance measuring is **CRUCIAL** important!
- **Java SDK** is pretty well covered by different performance testing suites and benchmarks
- **Python SDK, Go SDK** and **Cross-Language** pipelines are missing the benchmarks
- We don’t run regularly the performance tests on **large datasets** and at **real scale**
    - It helps to find the specific issues
- Beam is in a good shape on this but…

Want to contribute?

Examples of things to do:

- Add perf tests / benchmarks for *Python* and *Go* SDKs
- Add **more runners** to run regularly
  - Portable runners including!
- **Automate** perf regressions with “*git bisect*”
  - Grafana alerts
  - Add to release testing routine
- Make *TPC-DS* in Beam more **mature** and part of release testing
- Add benchmarks/tests of **your choice**
- … etc

References

Nexmark:

- Main doc: https://datalab.cs.pdx.edu/niagara/NEXMark/
- Beam: https://beam.apache.org/documentation/sdks/java/testing/nexmark/
- Wiki: https://cwiki.apache.org/confluence/display/BEAM/Nexmark

TPC-DS:

- Website: https://www.tpc.org/tpcds/default5.asp
- Beam: https://beam.apache.org/documentation/sdks/java/testing/tpcds/

Thanks!

BEAM SUMMIT