Nexmark: A unified benchmarking suite for data-intensive systems with Apache Beam

Ismaël Mejía
@iemejia
Who are we?

Integration Software
Big Data / Real-Time
Open Source
Enterprise
New products

We are hiring!
1. Big Data Benchmarking
   a. State of the art
   b. NEXMark: A benchmark over continuous data streams

2. Apache Beam and Nexmark
   a. Introducing Beam
   b. Advantages of using Beam for benchmarking
   c. Implementation
   d. Nexmark + Beam: a win-win story

3. Using Nexmark
   a. Neutral benchmarking: a difficult issue
   b. Example: Running Nexmark on Apache Spark

4. Current status and future work
Big Data Benchmarking
Benchmarking

Why do we benchmark?
1. Performance
2. Correctness

Benchmark suites steps:
1. Generate data
2. Compute data
3. Measure performance
4. Validate results

Types of benchmarks
- Microbenchmarks
- Functional
- Business case
- Data Mining / Machine Learning
Issues of Benchmarking Suites for Big Data

- **No de-facto** suite: Terasort, TPCx-HS (Hadoop), HiBench, ...
- No common model/API: Strongly tied to each processing engine or SQL
- Too focused on **Hadoop** infrastructure
- Mixed benchmarks for storage/processing
- Few benchmarking suites focus on **streaming** semantics
State of the art

Batch

- **Terasoft**: Sort random data
- **TPCx-HS**: Sort to measure Hadoop compatible distributions
- **TPC-DS on Spark**: TPC-DS business case with Spark SQL
- **Berkeley Big Data Benchmark**: SQL-like queries on Hive, Redshift, Impala
- **HiBench** and **BigBench**

Streaming

- **Yahoo Streaming Benchmark**

* HiBench includes also some streaming / windowing benchmarks
NEXMark

Benchmark for queries over data streams

**Online Auction System**
Research paper draft 2004
8 CQL-like queries

Example:
Query 4: What is the average selling price for each auction category?
Query 8: Who has entered the system and created an auction in the last period?
Nexmark on Google Dataflow

- Port of the queries from the NEXMark research paper
- Enriched suite with client use cases. 5 extra queries
- Used as a rich integration test scenario
Apache Beam and Nexmark
Apache Beam origin

Google

MapReduce

Google Cloud Dataflow

Apache Beam

Colossus
BigTable
PubSub
Dremel
Spanner
Megastore
Millwheel
Flume
What is Apache Beam?

Apache Beam is a unified programming model designed to provide efficient and portable data processing pipelines.
Apache Beam vision

**Batch + strEAM Unified model**

**What / Where / When / How**

1. **SDKs**: Java, Python, Go (WIP), etc
2. **DSLs & Libraries**: Scio (Scala), SQL
3. **IOs**: Data store Sources / Sinks
4. **Runners** for existing Distributed Processing Engines
Runners

Runners “translate” the code into the target runtime

* Same code, different runners & runtimes
Beam - Today (Feb 2018)

A vibrant community of contributors + companies: Google, data Artisans, PayPal, Talend, Atrato, Ali Baba

First Stable Release. 2.0.0 API stability contract (May 2017)  
Current: 2.3.0 (vote in progress, Feb 2017)

Exciting Upcoming Features:

Fn API, been able to run multiple languages on other runners  
Schema-aware PCollections and SQL improvements  
New Libraries. Perfect moment to contribute yours!
The Beam Model: **What** is Being Computed?

**Event Time:** Timestamp when the event happened

**Processing Time:** Absolute program time (wall clock)
The Beam Model: **Where** in Event Time?

- Split infinite data into finite chunks
The Beam Model: **Where** in Event Time?
The Beam Model: When in Processing Time?
Apache Beam Pipeline concepts

Data processing **Pipeline**
(exeected by a Beam runner)

- Read (Source)
- PTransform
- PTransform
- Write (Sink)

Input
PCollection

Window per min
Count

Output

* Don't think it is only a straight pipeline any directed acyclic graph (DAG) is valid.
Apache Beam - Programming Model

Element-wise

ParDo -> DoFn
MapElements
FlatMapElements
Filter

WithKeys
Keys
Values

Grouping

GroupByKey
CoGroupByKey

Combine -> Reduce
Sum
Count
Min / Max
Mean
...

Windowing/Triggers

Windows
FixedWindows
GlobalWindows
SlidingWindows
Sessions

Triggers
AfterWatermark
AfterProcessingTime
Repeatedly
Nexmark on Apache Beam

- Nexmark was ported from Dataflow to Beam 0.2.0 as an integration test case
- Refactored to the just released **stable** version of Beam **2.0.0**
- Made code generic to support all the Beam runners
- Changed some queries to use new APIs
- Validated queries in all the runners to test their support of the Beam model
- Included as part of Beam 2.2.0 (Dec. 2017)
Advantages of using Beam for benchmarking

- Rich model: all use cases that we had could be expressed using Beam API
- Can test both batch and streaming modes with exactly the same code
- Multiple runners: queries can be executed on Beam supported runners*
- Metrics

* Runners must provide the specific capabilities (features) used by the query
Implementation
Components of Nexmark

- **NexmarkLauncher:**
  Start sources to generate Events
  Run and monitor the queries (pipelines)

- **Generator:**
  Timestamped and correlated events:
  Auction, Bid, Person

- **Metrics:**
  Each query includes ParDos to update metrics:
  execution time, processing event rate, number of results,
  but also invalid auctions/bids, ...

- **Configuration*:**
  Batch: test data is finite and uses a BoundedSource
  Streaming: test data is finite but uses an UnboundedSource

* Configuration details discussed later
## Interesting Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
<th>Beam concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Who is selling in particular US states?</td>
<td>Join, State, Timer</td>
</tr>
<tr>
<td>5</td>
<td>Which auctions have seen the most bids in the last period?</td>
<td>Sliding Window, Combiners</td>
</tr>
<tr>
<td>6</td>
<td>What is the average selling price per seller for their last 10 closed auctions?</td>
<td>Global Window, Custom Combiner</td>
</tr>
<tr>
<td>7</td>
<td>What are the highest bids per period?</td>
<td>Fixed Windows, Side Input</td>
</tr>
<tr>
<td>9 *</td>
<td>What are the winning bids for each closed auction?</td>
<td>Custom Window</td>
</tr>
<tr>
<td>11 *</td>
<td>How many bids did a user make in each session he was active?</td>
<td>Session Window, Triggering</td>
</tr>
<tr>
<td>12 *</td>
<td>How many bids does a user make within a fixed processing time limit?</td>
<td>Global Window in Processing Time</td>
</tr>
</tbody>
</table>

*: not in the original NEXMark paper
Query Structure

1. Get `PCollection<Event>` as input
2. Apply `ParDo + Filter` to extract object of interest: Bids, Auction, Person
3. Apply transforms: `Filter`, `Count`, `GroupByKey`, `Window`, etc.
4. Apply `ParDo` to output the final `PCollection`: collection of `AuctionPrice`, `AuctionCount`...
Key point: **Where** in time to compute data?

- **Windows**: divide data into event-time-based finite chunks.
  - Often required when doing aggregations over unbounded data
Key point: **When** to compute data?

**Triggers**: Condition to emit the results of aggregation
Deal with producing early results or including late-arriving data

- Q11: uses a data-driven trigger fires when 20 elements were received

* Triggers can be Event-time, Processing-Time, Data-driven or Composite
Key point: How to deal with **out of order** events?

- State and Timer APIs in an incremental join (Q3):
  - Memorize person event waiting for corresponding auctions and clear at timer
  - Memorize auction events waiting for corresponding person event
Conclusion on queries

● Wide coverage of the Beam API
  ○ Most of the API
  ○ Illustrates also working in processing time

● Realistic
  ○ Real use cases, valid queries for an end user auction system

● Complex queries
  ○ Leverage all the runners capabilities
Why Nexmark on Beam? A win-win story

- Advanced streaming semantics
- A/B testing of execution engines (e.g. regression and performance comparison between 2 versions of the same engine or of the same runner, ...)
- Integration tests (SDK with runners, runners with engines, ...)
- Validate Beam runners capability matrix
Using Nexmark
Neutral Benchmarking: A difficult issue

- Different **levels of support** of capabilities of the Beam model among runners
- All execution systems have **different strengths**: we would end up comparing things that are not always comparable
  - Some runners were designed to be batch oriented, others stream oriented
  - Some are designed towards sub-second latency, others prioritize auto-scaling
- Runners / Systems can have multiple **knobs to tweak the options**
- Benchmarking on a **distributed environment** can be inconsistent. Even worse if you benchmark on the cloud (e.g. Noisy neighbors)

*Interesting Read: https://data-artisans.com/blog/curious-case-broken-benchmark-revisiting-apache-flink-vs-databricks-runtime*
Nexmark - How to run

```bash
$ mvn exec:java -Dexec.mainClass=org.apache.beam.integration.nexmark.Main -Pspark-runner
-Dexec.args="--runner=SparkRunner --suite=SMOKE --streaming=false
--manageResources=false --monitorJobs=true --flinkMaster=tbd-bench"
```

```bash
$ mvn exec:java -Dexec.mainClass=org.apache.beam.integration.nexmark.Main -Pflink-runner
-Dexec.args="--runner=FlinkRunner --suite=SMOKE --streaming=true
--manageResources=false --monitorJobs=true --sparkMaster=local"
```

```bash
$ flink run --class org.apache.beam.integration.nexmark.Main
beam-integration-java-nexmark-2.1.0-SNAPSHOT.jar --query=5 --streaming=true
--manageResources=false --monitorJobs=true --flinkMaster=tbd-bench
```
Benchmark workload configuration

Events generation
- smoke config defaults
  - 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

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

Technical
- Artificial CPU load
- Artificial IO load
# Nexmark Output - Flink Runner (Batch)

<table>
<thead>
<tr>
<th>Conf</th>
<th>Runtime(sec)</th>
<th>Events(/sec)</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1.5</td>
<td>68399.5</td>
<td>100000</td>
</tr>
<tr>
<td>0001</td>
<td>1.6</td>
<td>63291.1</td>
<td>92000</td>
</tr>
<tr>
<td>0002</td>
<td>0.9</td>
<td>108108.1</td>
<td>351</td>
</tr>
<tr>
<td>0003</td>
<td>3.0</td>
<td>33255.7</td>
<td>580</td>
</tr>
<tr>
<td>0004</td>
<td>0.9</td>
<td>11547.3</td>
<td>40</td>
</tr>
<tr>
<td>0005</td>
<td>2.6</td>
<td>38138.8</td>
<td>12</td>
</tr>
<tr>
<td>0006</td>
<td>0.7</td>
<td>13888.9</td>
<td>103</td>
</tr>
<tr>
<td>0007</td>
<td>2.5</td>
<td>39308.2</td>
<td>1</td>
</tr>
<tr>
<td>0008</td>
<td>1.4</td>
<td>69735.0</td>
<td>6000</td>
</tr>
<tr>
<td>0009</td>
<td>1.0</td>
<td>9980.0</td>
<td>298</td>
</tr>
<tr>
<td>0010</td>
<td>2.5</td>
<td>39698.3</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>2.3</td>
<td>43047.8</td>
<td>1919</td>
</tr>
<tr>
<td>0012</td>
<td>1.7</td>
<td>59701.5</td>
<td>1919</td>
</tr>
</tbody>
</table>
Comparing flink vs direct runner (local default settings)
Comparing flink vs direct runner (no enforced serialization)
Comparing different versions of the Spark engine
Current status and future work
## Execution Matrix

### Batch

<table>
<thead>
<tr>
<th>Runner</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2112</td>
<td></td>
</tr>
<tr>
<td>Flink</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apex*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1037</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Streaming

<table>
<thead>
<tr>
<th>Runner</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1035</td>
<td></td>
<td></td>
<td>2112</td>
<td></td>
</tr>
<tr>
<td>Flink</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apex*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1037</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Apex runner lacks support for metrics
  
  * We have not tested yet on Google Dataflow
Current status

- Nexmark helped discover bugs and missing features in Beam
- 10 open issues / 7 closed issues on Beam upstream. [BEAM-160](https://issues.apache.org/jira/browse/BEAM-160)
- Nexmark is used to find regressions on the release votes.
Ongoing work

- Resolve open Nexmark and Beam issues
- Add Nexmark to Beam IT/Performance tests (jenkins, k8s)
- Validate new runners: Gearpump, Samza, ...
- Add more queries to evaluate uncovered cases
- Support for Streaming SQL via Calcite
- Extract the Generator so it can be used by other projects
- Python Implementation to test the Portability effort
Contribute

You are welcome to contribute!

- Use it to test your clusters and bring us new issues, feature requests
- Multiple Jiras that need to be taken care of (label: nexmark)
- Improve documentation + more refactoring
- Help to improve CI
- New ideas, more queries, support for IOs, refactoring, etc

Not only for Nexmark, Beam is in a perfect shape to jump in.

Also if you are interested you can create an implementation of Nexmark in your own framework for comparison purposes.
Greetings

- **Mark Shields** (Google): Contributing Nexmark + answering our questions
- **Etienne Chauchot** (Talend): Co-maintainer of Nexmark
- **Thomas Groh, Kenneth Knowles** (Google): Direct runner + State/Timer API
- **Amit Sela, Aviem Zur** (Paypal): Spark Runner + Metrics
- **Aljoscha Krettek** (data Artisans), **Jinsong Lee** (Ali Baba): Flink Runner
- **Jean-Baptiste Onofre, Abbass Marouni** (Talend): comments and help to run Nexmark in our YARN cluster
- **Anton Kedin**: Integration of Streaming SQL into Nexmark.
- The rest of the **Beam** community in general for being awesome.

* The nice slides with animations were created by Tyler Akidau and Frances Perry and used with authorization.
References

Nexmark

NEXMark
Beam's Nexmark
BEAM-160

Apache Beam

https://beam.apache.org

The World Beyond Batch 101 & 102


Join the mailing lists!

user-subscribe@beam.apache.org
dev-subscribe@beam.apache.org

Follow @ApacheBeam on Twitter
Thanks