Hands on with Beam and Dataflow

Eric Anderson, Google
@ericmander

With slide contributions from Eugene Kirpichov, Frances Perry, and Tyler Akidau
Hands on with Beam and Dataflow

Apologies: Predicting Hacker News with Beam and Tensorflow

Hands on with Beam and Dataflow

- Evolution of distributed processing - context for Beam
- What’s still hard about distributed processing - motivation for Beam/Dataflow
  - Stragglers and Scaling
- Beam demo: Local python runner

Details on Beam and event time processing -> Eugene Kirpochov’s session
History of distributed data processing

- MapReduce
- Google
  - Colossus
  - BigTable
  - PubSub
  - Dremel
  - Spanner
  - Megastore
  - Millwheel
  - Flume
- Apache
  - Apache Hadoop
  - Apache Drill
  - Apache Oozie
  - Apache Drill
  - Apache Crunch
  - Apache Spark
History of distributed data processing

Google
- Colossus
- BigTable
- PubSub
- Dremel
- Spanner
- Megastore
- Millwheel
- Flume

Google Cloud Dataflow

Apache Beam

MapReduce

Apache
- Apache HBase
- Crunch
- Spark
- Drill
- Oozie
- Tez
What problems are left to solve?

Well, lots, but to name a few...
- Event time processing
- Changes to production streaming pipelines
- Straggling workers
- Flexible/autoscaled resources
Streaming Ops

How to update/upgrade a production streaming pipeline?

Without...

- losing data in pipeline
- missing new data during downtime
- comprising de-dupe guarantees
- impacting write pattern
Streaming Ops

Common approaches

Parallel swap
- Deploy a 2nd pipeline
- Swap traffic over
- Take down the 1st

Stop & Restart
- Stop pulling from sources
- Drain pipeline; then stop it
- Start new pipeline
- Resume sources

<table>
<thead>
<tr>
<th>Feature</th>
<th>Parallel swap</th>
<th>Stop &amp; restart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve data in pipeline</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>Avoid missing new data</td>
<td>Green</td>
<td>Yellow</td>
</tr>
<tr>
<td>De-dupe guarantees</td>
<td>Red</td>
<td>Yellow</td>
</tr>
<tr>
<td>Preserve write pattern</td>
<td>Red</td>
<td>Yellow</td>
</tr>
</tbody>
</table>
Streaming Ops

Ideal approach

Update in-place

● Submit a new job with the same name and --update flag

Parallel swap | Stop & restart | Update in-place
---|---|---
Preserve data in pipeline | Green | Green | Green
Avoid missing new data | Green | Yellow | Green
De-dupe guarantees | Red | Yellow | Green
Preserve write pattern | Red | Yellow | Green

Status

● It works!
● ...for minor pipeline changes
● Included in Dataflow
Straggling workers

A system is only as fast as its slowest worker

Workers lag because:

- Variable machines
- Unequal work
  - Poor distribution of elements
  - Variable work per element

Fixes?

- Filter bad machines?
- Backup / speculative execution?
Dynamic Work Rebalancing

Identify where work can be split
Dynamically rebalance it onto another worker
Make sure everything is processed exactly once
Dynamic Work Rebalancing

Status:

- Works great - see before and after below
- Requires signals from source (Beam IO API)
- Included in Dataflow
Scaling clusters & jobs

Traditional model: Static set of machines
- Allocate a data cluster
- Deploy jobs to the cluster
- Compute coupled to storage

New model: Cloud is infinite and ephemeral
- Separate compute from storage?
- Dynamic allocation to cluster?
- Dynamic allocation to job?

What signals to scale on?
What work do you give new workers?
Scaling clusters & jobs

Common approach
- Framework scales job
- Cloud provider scales cluster
- What signals to use?
- How to allocate work to new machines?

Ideal approach
- Job, cluster 1:1 (Job = cluster)
- Machines spin up/down fast
- Data processing specific signals
- Can split work on the fly
Scaling clusters & jobs

Status

- Signals provided by Beam SDK:
  - How much work there is to do (backlog)
  - How parallelizable is it
- Works great in batch (see right)
- Streaming progressing well
- Only think in terms of jobs
Motivation for Beam and Dataflow

Dataflow contains Google’s latest approaches to streaming ops, straggler mitigation and autoscaling.

Beam SDK provides an IO API that plumbs signals required for next generation data processing.

Can be extended to new sources or runners (execution framework).
Demo

https://github.com/silviulica/WorkflowExamples/blob/master/notebooks/DataflowIntroduction.ipynb