Using Google Cloud Dataflow

March, 22nd

Google Cloud Dataflow

- Provides you with a place to run Apache Beam jobs on the GCP
- Offers the ability to create jobs based on **templates**
- No need to address common aspects of running jobs on a cluster:
  - load balancing
  - scaling number of workers for a job
- These tasks are done automatically for both batch and streaming

Apache BEAM – Batch + strEAM

- Evolution of Google Dataflow that separates the dataflow logic from the programming issues (language, runners etc)
- Unified model for both batch and stream data processing
- Programs can be executed in different processing frameworks (via runners) using a set of different IOs

Apache BEAM – Batch + strEAM

Why to use BEAM instead of only Hadoop, Spark, Flink, GCP Dataflow etc?

→ The Apache Beam framework provides an abstraction between your application logic and the big data ecosystem

Apache BEAM – Batch + strEAM

In the BEAM ecosystem:

- **DataSource**: can be batches, micro-batches or streaming data
- **SDK**: Java or Python
- **Runner**: Apache Spark, Apache Flink, Google Cloud Dataflow, Apache Apex, Apache Gear pump (incubating) or Apache Samza
- Downside: BEAM only supports Python 2.7 :(
- To build a BEAM logic: Pipeline, PCollection, PTransform, ParDO and DoFn

Apache BEAM – Batch + strEAM: Components

- **Pipeline**: encapsulates the workflow of your entire data processing tasks from start to finish. Includes:
  - reading input data
  - transforming that data
  - writing output data
- All Beam driver programs must create a Pipeline
- When you create the Pipeline, you must also specify the execution options that tell the Pipeline where and how to run

Apache BEAM – Batch + strEAM: Components

- **PCollection**: distributed data set that your Beam pipeline operates on
- data may come from a fixed source like a file, or from a continuously updating source via a subscription or other mechanism

Apache BEAM – Batch + strEAM: Components

- **PTransform**: represents a data processing operation, or a step, in your pipeline
- Every PTransform takes one or more PCollection objects as input, performs a processing function that you provide on the elements of that PCollection, and produces zero or more output PCollection objects.

Apache BEAM – Batch + strEAM: Components

- **ParDo**: for generic parallel processing
- similar to the “Map” phase of a Map/Shuffle/Reduce-style algorithm
- a ParDo transform considers each element in the input PCollection, performs some processing function (your user code) on that element, and emits zero, one, or multiple elements to an output PCollection.

Apache BEAM – Batch + strEAM: Components

- **DoFn**: applies your logic in each element in the input PCollection and lets you populate the elements of an output PCollection
- to be included in your pipeline, it's wrapped in a ParDo PTransform.

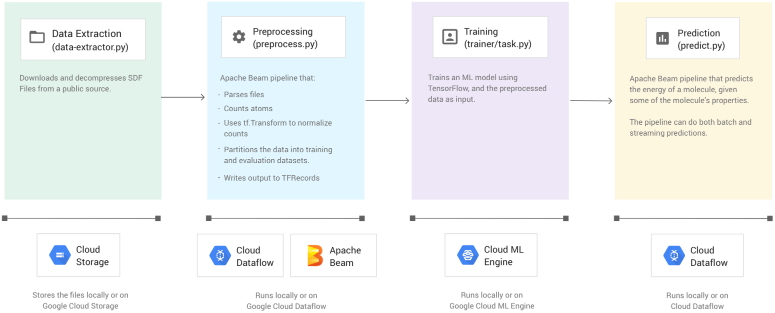
Apache BEAM – Batch + strEAM

Recommended links:

- How to use Google Cloud Dataflow with TensorFlow for batch predictive analysis
- Guide to common Cloud Dataflow use-case patterns
  - Part 1
  - Part 2

Apache BEAM – Batch + strEAM: Experiment

- Molecules (google samples) Molecules (github instructions)



Data Extraction (data-extractor.py)

Downloads and decompresses SDF Files from a public source.

Preprocessing (preprocess.py)

Apache Beam pipeline that:

- Parses files
- Counts atoms
- Uses tf.Transform to normalize counts
- Partitions the data into training and evaluation datasets.
- Writes output to TFRecords

Training (trainer/task.py)

Trains an ML model using TensorFlow, and the preprocessed data as input.

Prediction (predict.py)

Apache Beam pipeline that predicts the energy of a molecule, given some of the molecule's properties.

The pipeline can do both batch and streaming predictions.

Cloud Storage

Stores the files locally or on Google Cloud Storage

Cloud Dataflow

Apache Beam

Runs locally or on Google Cloud Dataflow

Cloud ML Engine

Runs locally or on Google Cloud ML Engine

Cloud Dataflow

Runs locally or on Cloud Dataflow

Apache BEAM – Batch + strEAM: Experiment

- This example uses:
  - Google Cloud Dataflow
  - Google Machine Learning
  - Apache Beam
  - Tensorflow transformations and Estimators

Apache BEAM molecules experiment: technical objectives

1. Understand the differences between Google Dataflow and Dataproc
2. Understand how a pipeline is created
3. Understand the learning task
4. Understand the contents of each script in the pipeline
5. Understand how to use transformations and estimators in Tensorflow
6. Run the pipeline as is locally (`run-local`) and in the cloud (`run-cloud`) (are there any differences in performance?)
7. Vary the `max-data-files` parameter with values 10, 100, 1000
8. Modify this program to include the actual ENERGY of each molecule in the predictions file
9. Modify this program to allow for cross-validation

NOTE: You may need to start with Codelab 2 in order to understand how to create a pipeline

For Wednesday 24th: Read and discuss

Polemical: Parallel and Distributed databases or MapReduce?

[Pavlo et al., SIGMOD 2009] A Comparison of Approaches to Large-Scale Data Analysis

[Dean and Ghemawat, CACM 2010] MapReduce: A Flexible Data Processing Tool

[Stonebraker et al., 2010] mapReduce and Parallel DBmss: friends or foes?