CHAPTER 40

Google Cloud Dataflow

Google Cloud Dataflow provides a serverless, parallel, and distributed infrastructure for running jobs for batch and stream data processing. One of the core strengths of Dataflow is its ability to almost seamlessly handle the switch from processing of batch historical data to streaming datasets while elegantly taking into consideration the perks of streaming processing such as windowing. Dataflow is a major component of the data/ML pipeline on GCP. Typically, Dataflow is used to transform humongous datasets from a variety of sources such as Cloud Pub/Sub or Apache Kafka to a sink such as BigQuery or Google Cloud Storage.

Critical to Dataflow is the use of the Apache Beam programming model for building the parallel data processing pipelines for batch and stream operations. The data processing pipelines built with the Beam SDKs can be executed on various processing backends such as Apache Apex, Apache Spark, Apache Flink, and of course Google Cloud Dataflow. In this section, we will build data transformation pipelines using the Beam Python SDK. As of this time of writing, Beam also supports building data pipelines using Java, Go, and Scala languages.

Beam Programming

Apache Beam provides a set of broad concepts to simplify the process of building a transformation pipeline for distributed batch and stream jobs. We'll go through these concepts providing simple code samples:

- A Pipeline: A Pipeline object wraps the entire operation and prescribes the transformation process by defining the input data source to the pipeline, how that data will be transformed, and where the data will be written. Also, the Pipeline object indicates the distributed processing backend to execute on. Indeed, a Pipeline

© Ekaba Bisong 2019
E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_40

is the central component of a Beam execution. Code for creating a pipeline is as shown in the following:

```
import apache_beam as beam
from apache_beam.options.pipeline_options import PipelineOptions

p = beam.Pipeline(options=PipelineOptions())
```

In the preceding code snippet, the Pipeline object is configured using 'PipelineOptions' to set the required fields. This can be done both programmatically and from the command line.

- A PCollection: A PCollection is used to define a data source. The data source can either be *bounded* or *unbounded.* A bounded data source refers to batch or historical data, whereas an unbounded data source refers to streaming data. Beam uses a technique called *windowing* to partition unbounded PCollections into finite logical segments using some attribute of the data such as a timestamp. PCollections can also be created from in-memory data where PCollections are both the inputs and outputs for a particular step in the pipeline. Let's see an example of reading a csv data from an external source:

  ```
  lines = p | 'ReadMyFile' >> beam.io.ReadFromText('gs://gcs_bucket/
  my_data.csv')
  ```

  The pipe operator '|' in the preceding code is also called the apply method and is used to apply the PCollection to the pipeline instantiated as 'p'.

- A PTransform: A PTransform refers to a particular transformation task carried out on one or more PCollections in the pipeline. PTransforms can be applied to PCollections as follows.

  ```
  [Output PCollection] = [Input PCollection] | [Transform]
  ```

  Note that while a PTransform creates a new PCollection, it does not modify or alter the input collection. A number of core Beam transforms include

  - ParDo: For parallel processing
  - GroupByKey: For processing collections of key/value pairs

    - CoGroupByKey: For a relational join of two or more key/value PCollections with the same key type
    - Combine: For combining collections of elements or values in your data
    - Flatten: For merging multiple PCollection objects
    - Partition: Splits a single PCollection into smaller collections
- I/O transforms: These are PTransforms that read or write data to different external storage systems. Some of the currently available I/O transforms working with Beam Python SDK include
    - avroio: For reading from and writing to an Avro file
    - textio: For reading from and writing to text files

For a simple linear pipeline with sequential transformation, the processing graph looks like what is shown in Figure 40-1.

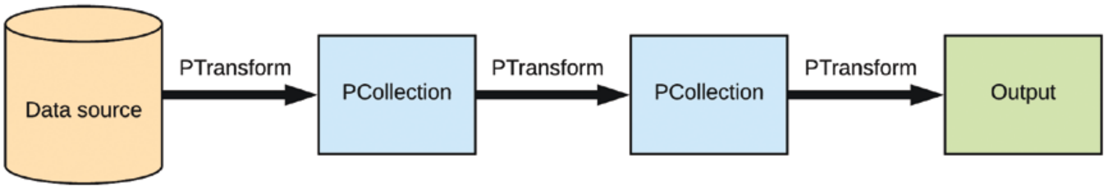

Figure 40-1. *A simple linear Pipeline with sequential transforms*

Building a Simple Data Processing Pipeline

In this simple Beam application, we will build a Dataflow pipeline to preprocess a CSV file from a GCS bucket and write the output back to GCS. This example selects certain features and rows that are of interest to the downstream modeling task. Here, we considered the ‘crypto-markets.csv’ dataset. In the data preprocessing pipeline, we removed data attributes that may not be relevant for analytics/model building and we