Monday, April 20, 2020

Class Note

1. The same PCollection can be written to multiple data sinks including BigQuery and Bigtable
2. Within the `DoFn` always use a try-catch block around activities like parsing data. In the exception block, rather than just log the issue, send the raw data out as a `SideOutput` into a storage medium such as BigQuery or Cloud Bigtable using a String column to store the raw unprocessed data.
3. Many Cloud Dataflow jobs, especially those in batch mode, are triggered by real-world events such as a file landing in Google Cloud Storage or serve as the next step in a sequence of data pipeline transformations. One common way to implement this approach is to package the Cloud Dataflow SDK and create an executable file that launches the job. But a better option is to use a simple REST endpoint to trigger the Cloud Dataflow pipeline.
4. Create a composite key made up of both properties. Use the composite key as input to the `KV.of()` factory method to create a KV object that consists of the composite key `(K)` and pertaining data element from which the composite key was derived (V). Use a `GroupByKey` transform to get your desired groupings (e.g., a resulting `PCollection>>`).
5. If possible, use `SideInputs` for any activity where one of the join tables is actually small — around 100MB in stream mode or under a 1GB in batch mode. This will perform much better than the join operation described here. This join is shuffle heavy and is best used when both collections being joined are larger than those guidelines. `SideInputs` are not precisely equivalent, however, as they're only read when data shows up on the main input. In contrast, a `CoGroupByKey` triggers if data shows up on either side. Think of `SideInputs` as an inner-loop join — with nested loops, the inner loop only runs if the outer loop runs first.