UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Big Data Management (BDMA & MIRI Masters)
Session: MapReduce (Training)

Alberto Abelló & Sergi Nadal

The purpose of this document is that you get familiar with the environment and tools required for the lab session. This is an optional task, thus no delivery is expected and no grade will be assigned. In this session we will explore Apache MapReduce. If you encounter any problems during the preparation of the training, please contact lecturer to clarify any issues before the lab session.

Instructions

In this session, as MapReduce jobs will run locally, you do not need to start any service. In the lab's computers, if you work on Linux, you can locally execute MapReduce jobs in eclipse and dismiss the virtual machine. In the *MainLocal.java* class you can see that different arguments are expected, and will drive the behaviour of the program. The simplest way to define such arguments is by creating a *Run Configuration* of type *Java Application* from the *Run* menu. In the tab menu called *Arguments* you will be able to edit such arguments, in each exercise you will be guided with the required arguments.

MapReduce basics

In this exercise, you will get familiar with MapReduce by executing and analyzing the provided jobs. To help you, we provide the `MainLocal.java` method, where you will be able to run MapReduce jobs locally in eclipse. For simplicity, we provide you with a local `adult.1000.sf` file in the resources folder of the project.

Projection

The `Projection.java` job selects a subset of the attributes in the dataset. Precisely, we are projecting the attributes `age`, `relationship` and `native_country`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT age, relationship, native_country FROM adult
```

Configuration First, take a look at the method `configureJob`. In this method we first define the mapper class and its output types for the key and the value. Next, as the projection does not require a reduce step, we already provide the same information for the output of the job. The next lines specify the format of the input (SequenceFile for all the cases) and the input/output paths. Finally, we pass parameters to the mapper. In this case, we send the list of `projection` attributes (i.e., `age`, `relationship` and `native_country`). Note that a projection does not require an aggregation phase, thus the combiner or reducer are not configured.

Map task Take a look at the `ProjectionMapper` class, which extends `Mapper` and its consistently typed with the configuration (i.e., its input key and value types are `Text`, as well as for the map output). The `map` method starts fetching the projection parameters from the context and splits the comma-separated input. The `Utils.getAttribute` method returns the projection for a specific attribute in the splitted line. To this end, we iterate on as many projection attributes as requested by appending the output in `newValue`.

Now, let's execute the MapReduce job. To this end, use the following run configuration and analyze the output.

```
-projection resources/adult.1000.sf resources/projection.out
```

Aggregation sum

The `AggregationSum.java` job performs a grouping on the attribute `native_country` and aggregates using the sum the attribute `capital_gain`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT native_country, SUM(capital_gain) FROM adult GROUP BY native_country
```

Configuration The method `configureJob` is similar as before. However, now the output key-value are respectively the classes text and double. Also, note now that, oppositely as the projection, we specify the combiner and reducer classes. The last two lines of the method pass the required parameters to the map task (i.e., the `groupBy` attribute and the `aggregation` attribute).

Map task The map task starts fetching the parameters from the context, as well as splitting the input comma-separated value. Next, it extracts the key and value and writes them to the context.

Combiner and reduce task The only processing that the combiner and reduce tasks perform is to sum the list of values for the same key. This is finally written to the output.

Now, let's execute the MapReduce job. To this end, use the following run configuration and analyze the output.

```
-aggregationSum resources/adult.1000.sf resources/aggSum.out
```

Cartesian product

The `CartesianProduct.java` performs the cross product of all elements that have `native_country` with values `Italy` and `Ecuador`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT external.*, internal.*
FROM adult as internal, adult as external
WHERE external.native_country = "Italy" AND internal.native_country = "Ecuador"
```

Configuration The method `configureJob` is similar as before. Note that now we do not configure a combiner phase to maintain the semantics of the operator.

Map task The map task applies a different logic depending on whether the input value contains the `type` for external or internal. If the value corresponds to external, we generate a key with the formula $random\%N$ (with $N = 100$) and write it to the context. Otherwise, if the value corresponds to internal, we write to the context the same value for all keys in the range $1..N$.

Reduce task The reduce task defines the internal and external lists, and adds to them the elements in the list of values accordingly to the `type`. Afterwards, a double loop iterates on them which causes the generation of all combinations.

Now, let's execute the MapReduce job. To this end, use the following run configuration and analyze the output.

```
-cartesian resources/adult.1000.sf resources/cartesian.out
```

Your first MapReduce jobs

In this exercise you will implement two MapReduce jobs.

Selection

Implement the selection (i.e., filtering) of the dataset. Precisely, we want only to output those tuples where the attribute `native_country` has a value `Canada`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT * FROM adult WHERE workclass = "Private"
```

Write the required classes (i.e., map, combiner and/or reduce) in the file `Selection.java`.

Aggregation average

Implement the aggregation with an average function of the dataset. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT native_country, AVG(capital_gain) FROM adult GROUP BY native_country
```

Write the required classes (i.e., map, combiner and/or reduce) in the file `AggregationAvg.java`.

Projection from CSV

Implement the projection from a CSV file. To this end, we provide you with a CSV version of the adult file (i.e., `adult.1000.csv`). Then, implement the projection from the list of attributes passed as parameter in the configuration. Write the required classes (i.e., map, combiner and/or reduce) in the file `ProjectionCSV.java`.

Hint. Interpret the input as key-values of types `LongWritable` and `Text` respectively. In the case of an standard text file, the key is the byte offset within the file for the line being processed. To generate the output, use `Text` as both key and value types.