Prof R. Madana Mohana

BIG DATA ANALYTICS

MapReduce Types & Formats

MapReduce Types

https://www.youtube.com/c/RASINENIMADANAMOHANA

MapReduce Types: *Outline*

MapReduce Types

The Default MapReduce Job

MapReduce Types and Formats

MapReduce Types and Formats

- MapReduce has a simple model of data processing: inputs and outputs for the map and reduce functions are key-value pairs.
- In this lecture, we looks at the MapReduce model in detail, and in particular at how data in various formats, from simple text to structured binary objects, can be used with this model.

MapReduce Types

MapReduce Types

- The **map** and **reduce** functions in **Hadoop MapReduce** have the following general form:

  `map: (K1, V1) → list(K2, V2)`

  `reduce: (K2, list(V2)) → list(K3, V3)`

- *In general*, the **map** input key and value types (**K1** and **V1**) are different from the **map** output types (**K2** and **V2**).
- However, the **reduce** input must have the same types as the **map** output, although the **reduce** output types may be different again (**K3** and **V3**).

MapReduce Types

The **Java API** mirrors this general form:

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends MapContext<KEYIN, VALUEIN,
KEYOUT, VALUEOUT> {
  // ...
  }
  protected void map(KEYIN key, VALUEIN value, Context
context) throws IOException, InterruptedException {
  // ...
  }
}
```

MapReduce Types

The **Java API** mirrors this general form:

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends ReducerContext<KEYIN,
VALUEIN, KEYOUT, VALUEOUT> {
  // ...
  }
  protected void reduce(KEYIN key, Iterable<VALUEIN>
values, Context context) throws IOException,
InterruptedException {
  // ...
  }
}
```

MapReduce Types

The context objects are used for emitting key-value pairs, and they are parameterized by the output types so that the signature of the **write()** method is:

```
public void write(KEYOUT key, VALUEOUT value)
    throws IOException, InterruptedException
```

MapReduce Types

- Since **Mapper** and **Reducer** are separate classes, the type parameters have different scopes, and the actual type argument of **KEYIN** (say) in the **Mapper** may be different from the type of the type parameter of the same name (**KEYIN**) in the **Reducer**.
- For instance, in the maximum temperature example, **KEYIN** is replaced by **Long Writable** for the **Mapper** and by **Text** for the **Reducer**.

MapReduce Types

- Similarly, even though the **map** output types and the **reduce** input types must match, this is not enforced by the Java compiler.
- The type parameters are named differently from the abstract types (**KEYIN** versus **K1**, and so on), but the form is the same.

MapReduce Types

- If a **combiner** function is used, then it has the same form as the **reduce** function (and is an implementation of **Reducer**), except its output types are the intermediate ***key*** and ***value*** types (**K2** and **V2**), so they can feed the **reduce** function:

```
map: (K1, V1) → list(K2, V2)
combiner: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

Often the **combiner** and **reduce** functions are the same, in which case **K3** is the same as **K2**, and **V3** is the same as **V2**.

MapReduce Types

- The partition function operates on the intermediate key and value types (**K2** and **V2**) and returns the partition index.
- In practice, the partition is determined solely by the key (the value is ignored):

```
partition: (K2, V2) → integer
```

MapReduce Types

In **Java**:

```
 public abstract class Partitioner<KEY, VALUE> {
   public abstract int getPartition(KEY key, VALUE
value, int numPartitions);
 }
```

MapReduce Types

The following ***Table -1*** summarizes the Configuration of **MapReduce** types in the **new API**.

Property	Job setter method	Input types		Intermediate Types		Output Types	
		K1	V1	K2	V2	K3	V3
Properties for configuring types:							
mapreduce.job.inputformat.class	setInputFormatClass()	√	√				
mapreduce.map.output.key.class	setMapOutputKeyClass()			√			
mapreduce.map.output.value.class	setMapOutputValueClass()				√		
mapreduce.job.output.key.class	setOutputKeyClass()					√	
mapreduce.job.output.value.class	setOutputValueClass()						√

MapReduce Types

The following ***Table -1*** summarizes the Configuration of **MapReduce** types in the **new API**. (Cont'd..)

Property	Job setter method	Input types		Intermediate Types		Output Types	
		K1	V1	K2	V2	K3	V3
Properties that must be consistent with the types:							
mapreduce.job.map.class	setMapperClass()	√	√	√	√		
mapreduce.job.combine.class	setCombinerClass()			√	√		
mapreduce.job.partitioner.class	setPartitionerClass()			√	√		
mapreduce.job.output.key.comparator.class	setSortComparatorClass()			√			
mapreduce.job.output.group.comparator.class	setGroupingComparatorClass()			√			
mapreduce.job.reduce.class	setReducerClass()			√	√	√	√
mapreduce.job.outputformat.class	setOutputFormatClass()					√	√

MapReduce Types

The following ***Table -2*** summarizes the Configuration of **MapReduce** types in the **old API**

Property	JobConf setter method	Input types		Intermediate Types		Output Types	
		K1	V1	K2	V2	K3	V3
Properties for configuring types:							
mapred.input.format.class	setInputFormat()	√	√				
mapred.mapoutput.key.class	setMapOutputKeyClass()			√			
mapred.mapoutput.value.class	setMapOutputValueClass()				√		
mapred.output.key.class	setOutputKeyClass()					√	
mapred.output.value.class	setOutputValueClass()						√

MapReduce Types

The following ***Table -2*** summarizes the Configuration of **MapReduce** types in the **old API** (Cont'd..)

Property	JobConf setter method	Input types		Intermediate Types		Output Types	
		K1	V1	K2	V2	K3	V3
Properties that must be consistent with the types:							
mapred.mapper.class	setMapperClass()	√	√	√	√		
mapred.map.runner.class	setMapRunnerClass()	√	√	√	√		
mapred.combiner.class	setCombinerClass()			√	√		
mapred.partitioner.class	setPartitionerClass()			√	√		
mapred.output.key.comparator.class	setOutputKeyComparatorClass()			√			
mapred.output.value.groupfn.class	setOutputValueGroupingComparator()			√			
mapred.reducer.class	setReducerClass()			√	√	√	√
mapred.output.format.class	setOutputFormat()					√	√

The Default MapReduce Job

The Default MapReduce Job

- What happens when we run **MapReduce** without setting a **mapper** or a **reducer**? Let's try it by running this minimal **MapReduce** program:

The Default MapReduce Job

```
public class MinimalMapReduce extends Configured
implements Tool {
 @Override
 public int run(String[] args) throws Exception {
  if (args.length != 2) {
   System.err.printf("Usage: %s [generic options]
<input> <output>\n", getClass().getSimpleName());
   ToolRunner.printGenericCommandUsage(System.err);
   return -1;
  }
```

The Default MapReduce Job

```
Job job = new Job(getConf());
job.setJarByClass(getClass());
FileInputFormat.addInputPath(job, new  Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
return job.waitForCompletion(true) ? 0 : 1;
}
 public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MinimalMapReduce(),
args);
    System.exit(exitCode);
 }
}
```

The Default MapReduce Job

- The only configuration that we set is an input path and an output path.
- We run it over a subset of our weather data with the following:

```
% hadoop MinimalMapReduce
"input/ncdc/all/190{1,2}.gz" output
```

The Default MapReduce Job

- We do get some output: one file named ***part-r-00000*** in the output directory.
- Here's what the first few lines look like (truncated to fit the page):

```
0→0029029070999991901010106004+64333+023450FM-
12+00059999V0202701N01591...
0→0035029070999991902010106004+64333+023450FM-
12+00059999V0201401N01181...
135→0029029070999991901010113004+64333+023450FM-
12+00059999V0202901N00821...
```

The Default MapReduce Job

- Each line is an integer followed by a tab character, followed by the original weather data record.
- Admittedly, it's not a very useful program, but understanding how it produces its output does provide some insight into the defaults that **Hadoop** uses when running **MapReduce** jobs.

The Default MapReduce Job

The default Streaming job:

- In Streaming, the default job is similar, but not identical, to the Java equivalent. The basic form is:

```
% hadoop jar
 $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
streaming-*.jar \
-input input/ncdc/sample.txt \
-output output \
-mapper /bin/cat
```

The Default MapReduce Job

The default Streaming job:

- When we specify a non-Java **mapper** and the default text mode is in effect (**-io** text), Streaming does something special.
- It doesn't pass the key to the **mapper** process; it just passes the value. (For other input formats, the same effect can be achieved by setting **stream.map.input.ignoreKey** to **true**.)

The Default MapReduce Job

The default Streaming job:

- This is actually very useful because the key is just the line offset in the file and the value is the line, which is all most applications are interested in.
- The overall effect of this job is to perform a sort of the input.

The Default MapReduce Job

The default Streaming job:

- With more of the defaults spelled out, the command looks like this (notice that Streaming uses the old **MapReduce API** classes):

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \ -input input/ncdc/sample.txt \ -output output \ -inputformat org.apache.hadoop.mapred.TextInputFormat \ -mapper /bin/cat \ -partitioner org.apache.hadoop.mapred.lib.HashPartitioner \ -numReduceTasks 1 \ -reducer org.apache.hadoop.mapred.lib.IdentityReducer \ -outputformat org.apache.hadoop.mapred.TextOutputFormat -io text
```

The Default MapReduce Job

The default Streaming job:

- The **-mapper** and **-reducer** arguments take a command or a Java class.
- A **combiner** may optionally be specified using the **-combiner** argument.

The Default MapReduce Job

Keys and values in Streaming:

- A Streaming application can control the separator that is used when a **key-value** pair is turned into a series of bytes and sent to the **map** or **reduce** process over standard input.
- The default is a tab character, but it is useful to be able to change it in the case that the keys or values themselves contain tab characters.

The Default MapReduce Job

Keys and values in Streaming:

- Similarly, when the **map** or **reduce** writes out **key-value** pairs, they may be separated by a configurable separator.
- Furthermore, the key from the output can be composed of more than the first field: it can be made up of the first ***n*** fields (defined by **stream.num.map.output.key.fields** or **stream.num.reduce.output.key.fields**), with the value being the remaining fields.

The Default MapReduce Job

Keys and values in Streaming:

- Separators may be configured independently for **maps** and **reduces**.
- The properties are listed in below Table and shown in a diagram of the data flow path in below Figure.

The Default MapReduce Job

Keys and values in Streaming:

Table: *Streaming separator properties*

Property Name	Type	Default value	Description
stream.map.input.field.separator	String	\t	The separator to use when passing the input key and value strings to the stream map process as a stream of bytes.
stream.map.output.field.separator	String	\t	The separator to use when splitting the output from the stream map process into key and value strings for the map output.
stream.num.map.output.key.fields	int	1	The number of fields separated by **stream.map.output.field.separator** to treat as the map output key.
stream.reduce.input.field.separator	String	\t	The separator to use when passing the input key and value strings to the stream reduce process as a stream of bytes.
stream.reduce.output.field.separator	String	\t	The separator to use when splitting the output from the stream reduce process into key and value strings for the final reduce output.
stream.num.reduce.output.key.fields	int	1	The number of fields separated by **stream.reduce.output.field.separator** to treat as the reduce output key.

The Default MapReduce Job

Keys and values in Streaming:

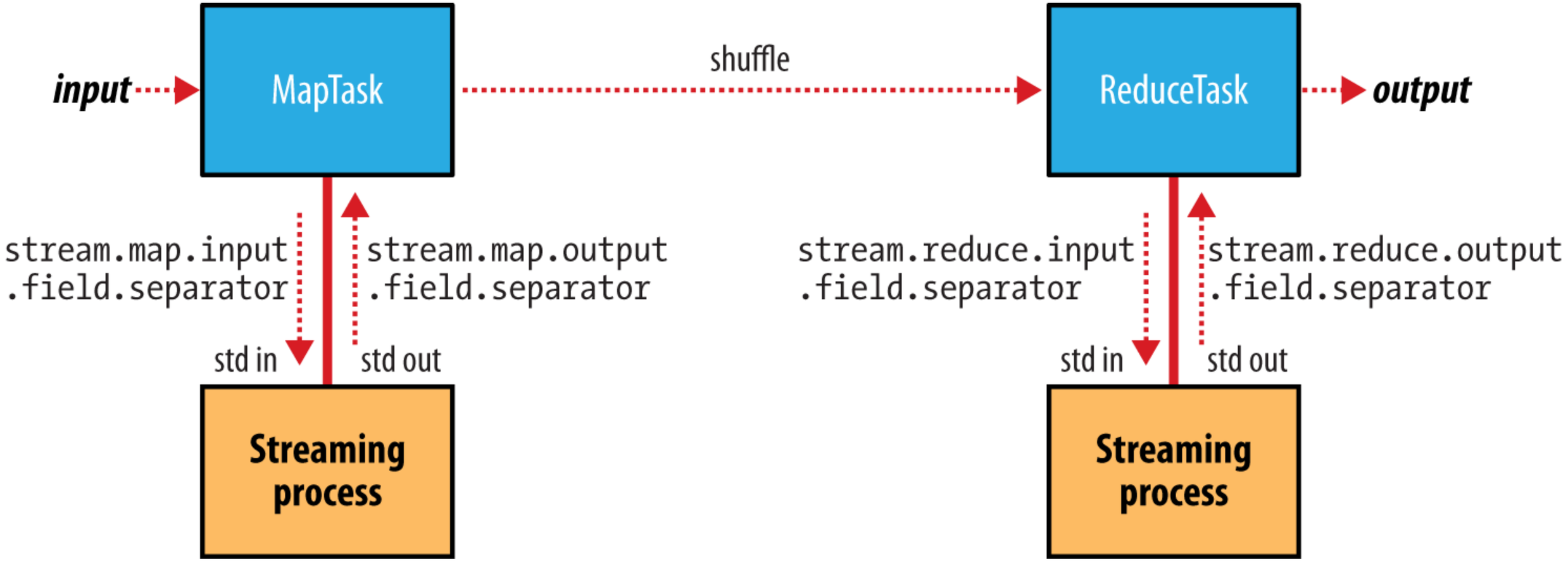

Fig: *Where separators are used in a Streaming MapReduce job*