Beyond Mapper and Reducer

Partitioner, Combiner,
Hadoop Parameters and more

Rozemary Scarlat

September 13, 2011

Data flow with multiple reducers

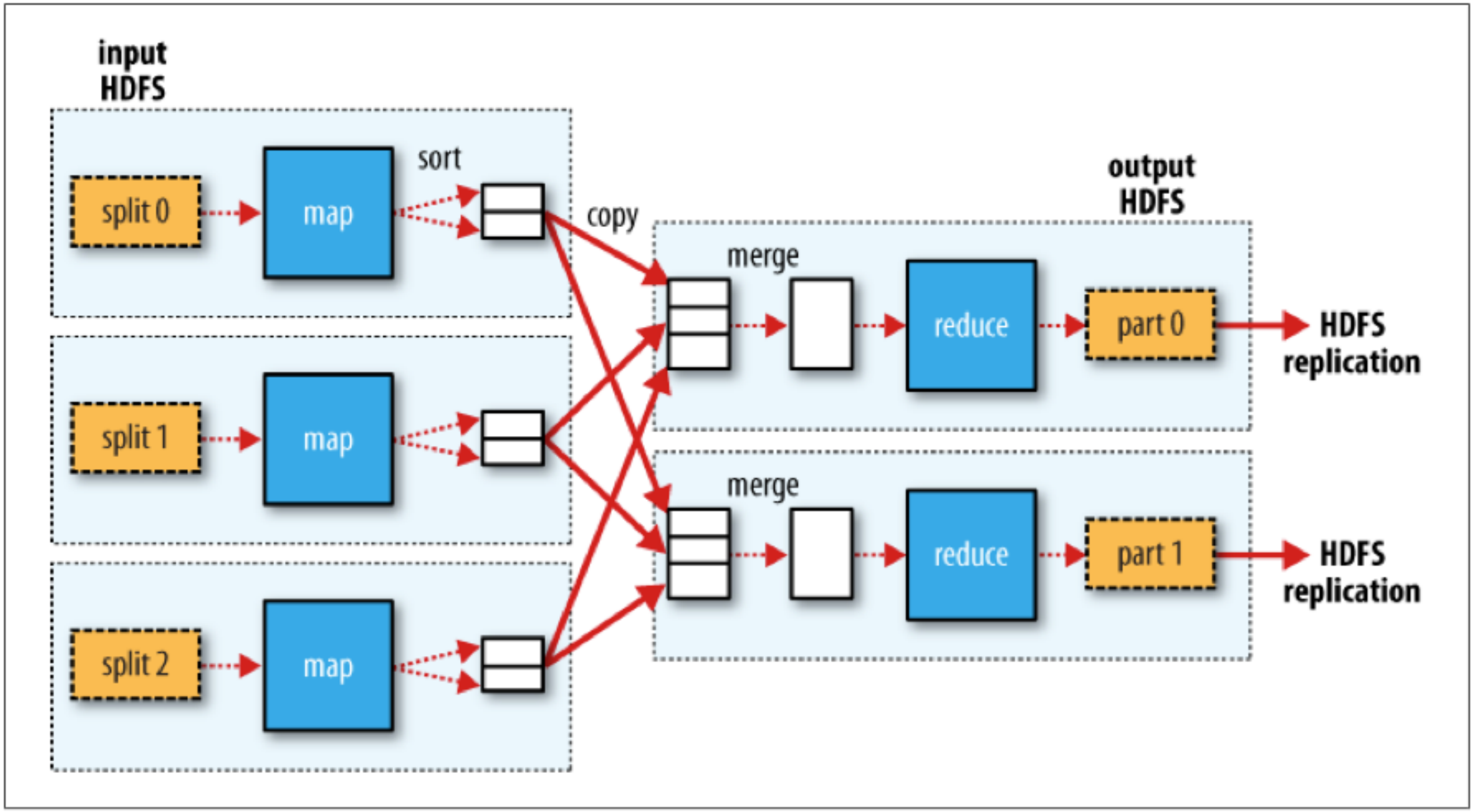
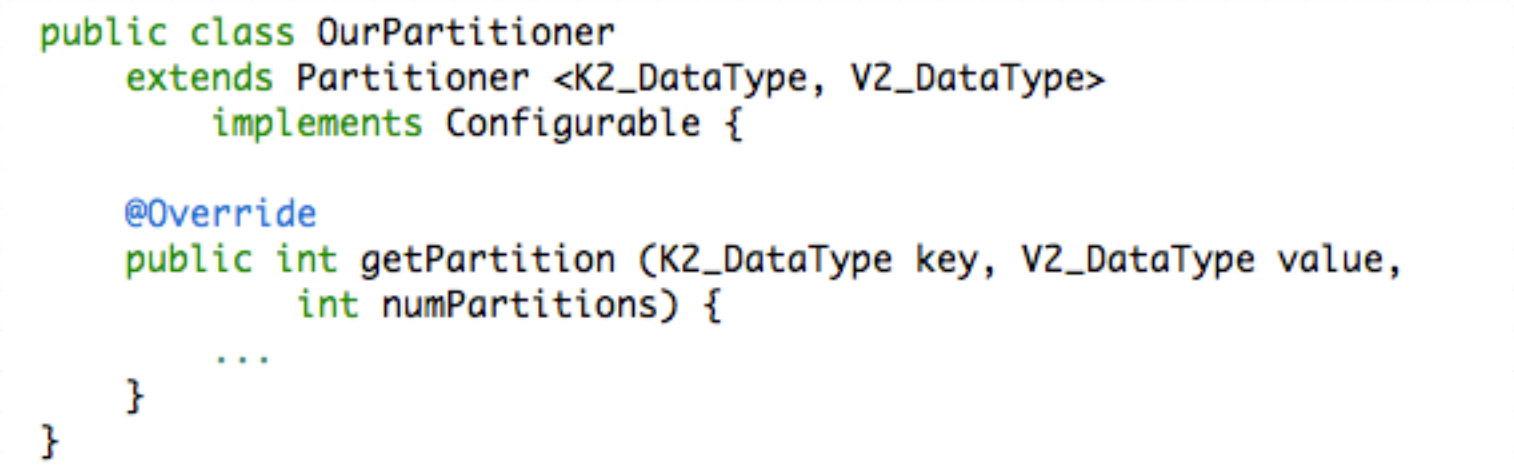


Figure 2-3. MapReduce data flow with multiple reduce tasks

Partitioner

- the map tasks *partition* their output, each creating one partition for each reduce task
- many keys per partition, but all records for a key are in a single partition
- default partitioner: *HashPartitioner* - hashes a record's key to determine which partition the record belongs in
- another partitioner: *TotalOrderPartitioner* – creates a total order by reading split points from an externally generated source

- The partitioning can be controlled by a user-defined partitioning function:



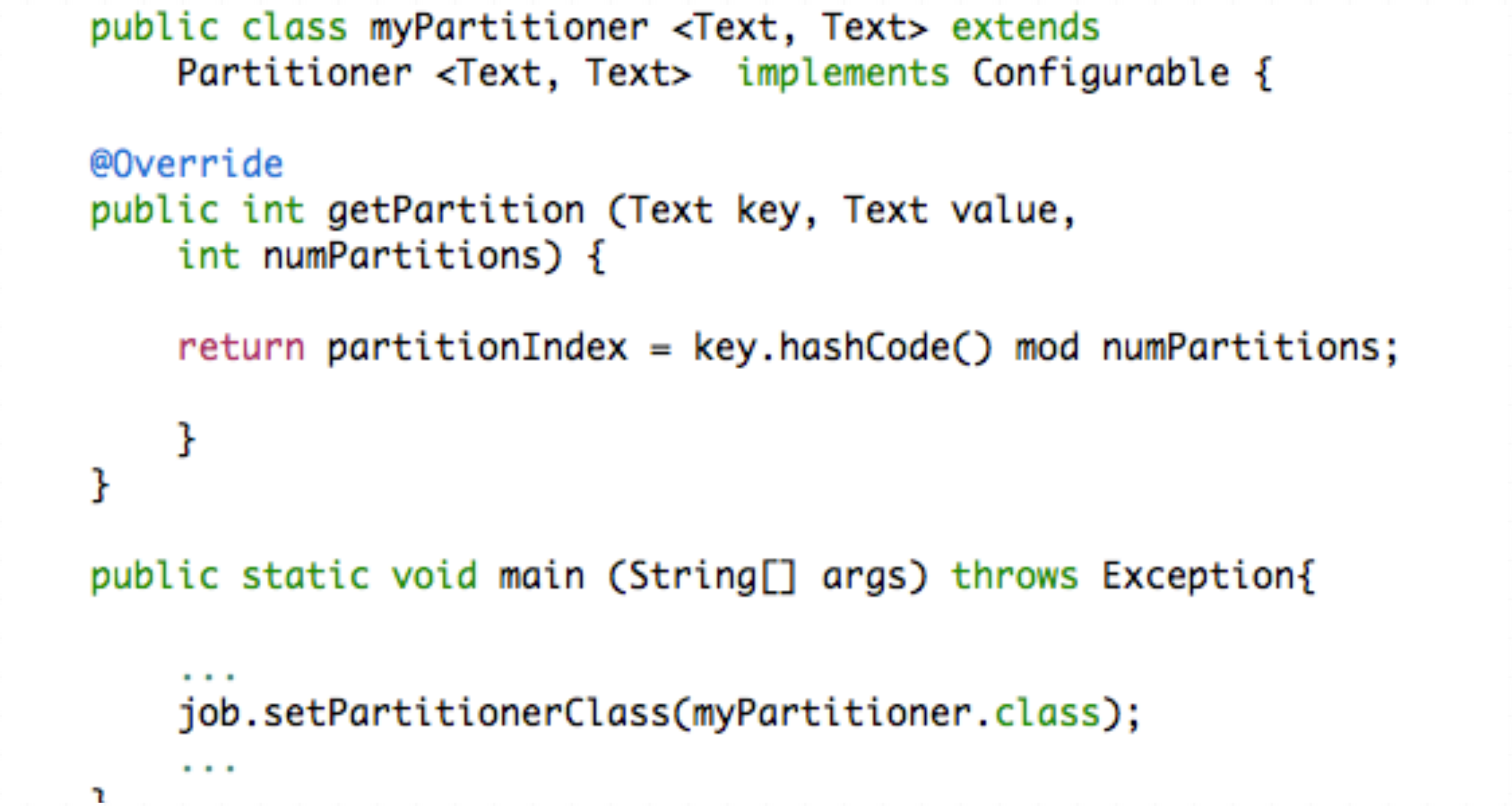
```
public class OurPartitioner
    extends Partitioner <K2_DataType, V2_DataType>
        implements Configurable {

    @Override
    public int getPartition (K2_DataType key, V2_DataType value,
            int numPartitions) {
        ...
    }
}
```

  - Don’t forget to set the partitioner class:

```
        job.setPartitionerClass(OurPartitioner.class);
```

  - Useful information about partitioners:
    - Hadoop book –*Total Sort* (pg. 237); *Multiple Outputs* (pg. 244);
    - http://chasebradford.wordpress.com/2010/12/12/reusable-total-order-sorting-in-hadoop/
    - http://philippeadjiman.com/blog/2009/12/20/hadoop-tutorial-series-issue-2-getting-started-with-customized-partitioning/ (Note: uses the old API!)

Partitioner example

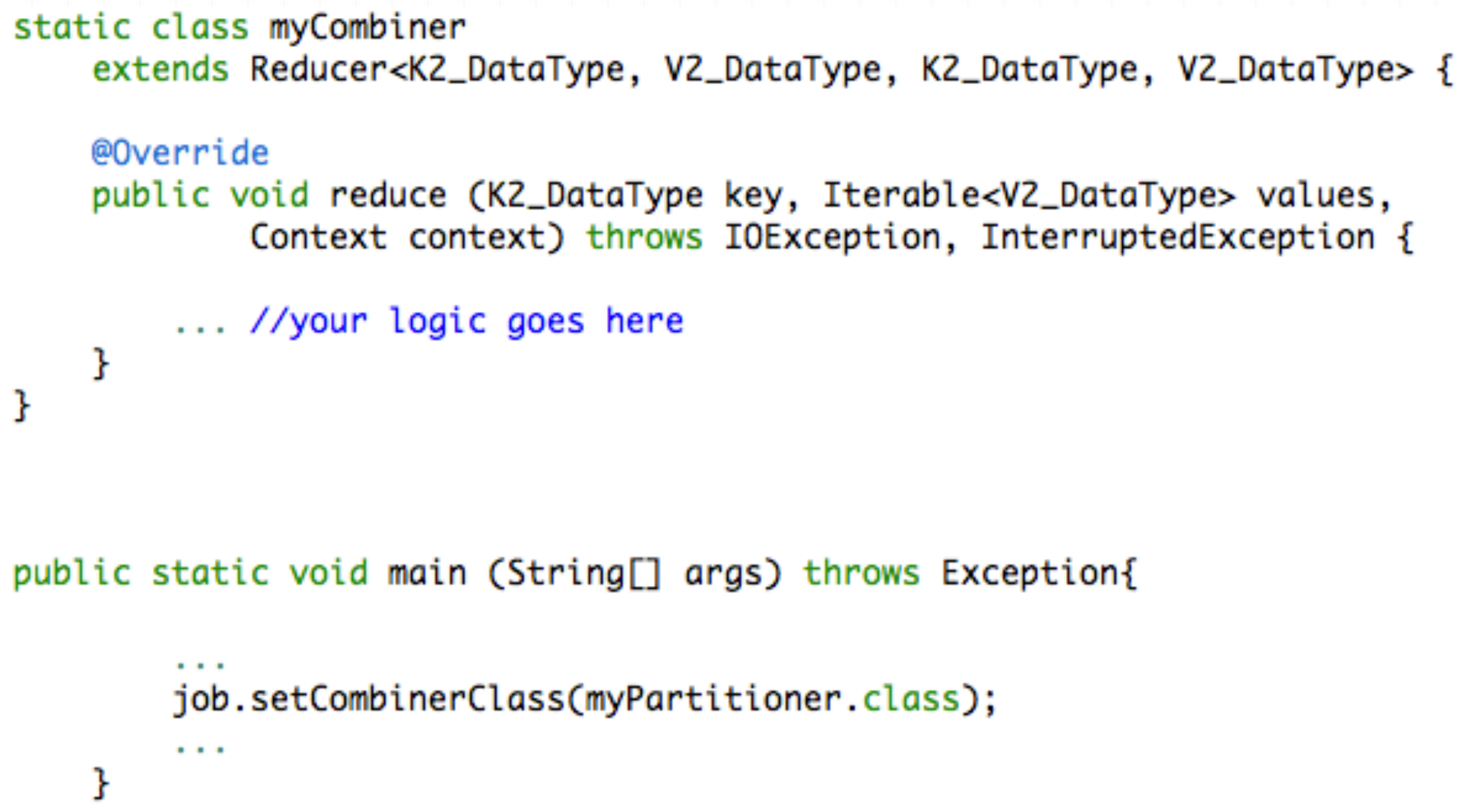


```
public class myPartitioner <Text, Text> extends
    Partitioner <Text, Text>  implements Configurable {

@Override
public int getPartition (Text key, Text value,
    int numPartitions) {

    return partitionIndex = key.hashCode() mod numPartitions;

    }
}

public static void main (String[] args) throws Exception{


    ...
    job.setPartitionerClass(myPartitioner.class);
    ...
```

Combiner

- The combiner receives as input all data emitted by the mapper instances on a given node
- Its output is sent to the Reducers (instead of the mappers' output).
- Hadoop does not guarantee how many times it will call the combiner for a particular map output record => calling the combiner for 0, 1 or many times should result in the same output of the reducer
- Generally, the combiner is called as the sort/merge result is written to disk. The combiner must:
  - be side-effect free
  - have the same input and output key types and the same input and output value types

Combiner example



```
static class myCombiner
    extends Reducer<K2_DataType, V2_DataType, K2_DataType, V2_DataType> {

    @Override
    public void reduce (K2_DataType key, Iterable<V2_DataType> values,
            Context context) throws IOException, InterruptedException {

        ... //your logic goes here
    }
}



public static void main (String[] args) throws Exception{

        ...
        job.setCombinerClass(myPartitioner.class);
        ...
    }
```

Parameters and more

- Cluster-level parameters (e.g. HDFS block size)
- Job-specific parameters (e.g. number of reducers, map output buffer size)
  - Configurable
  - Important for job performance
  - Map-side/Reduce-side/Task-environment – Tables 6-1, 6-2, 6-5 from the book
  - Full list of mapreduce paramteres with their default values: http://hadoop.apache.org/common/docs/current/mapred-default.html
- User-defined parameters
  - Used to pass information from driver (main) to mapper/reducer.
  - Help to make your mapper/reducer more generic

- Also, built-in parameters managed by Hadoop that cannot be changed, but can be read
  - For example, the path to the current input that can be used in joining datasets will be read with:

```
FileSplit split = (FileSplit)context.getInputSplit();
String inputFile = split.getPath().toString();
```

- Counters – built-in (Table 8.1 from the book) and user-defined (e.g. count the number of missing records and the distribution of temperature quality codes in the NCDC weather data set)
- MapReduceTypes – you already know some (eg. setMapOutputKeyClass()), but there are more – Table 7-1 from the book
- Identity Mapper/Reducer
  - no processing of the data (output == input)

- Why do we need map/reduce function without any logic in them?
  - Most often for sorting
  - More generally, when you only want to use the basic functionality provided by Hadoop (e.g. sorting/grouping)
  - More on sorting at page 237 from the book
- MapReduce Library Classes - for commonly used functions (e.g. InverseMapper used to swap keys and values) (Table 8-2 in the book)
- implementing Tool interface - support of generic command-line options
  - the handling of standard command-line options will be done using ToolRunner.run(Tool, String[]) and the application will only handle its custom arguments
  - most used generic command-line options:

```
-conf <configuration file>
-D <property=value>
```

- How to determine the number of splits?
  - If a file is large enough and splitable, it will be splited into multiple pieces (split size = block size)
  - If a file is non-splitable, only one split.
  - If a file is small (smaller than a block), one split for file, unless...
- CombineFileInputFormat
  - Merge multiple small files into one split, which will be processed by one mapper
  - Save mapper slots. Reduce the overhead
- Other options to handle small files?
  - hadoop fs -getmerge src dest

~