CPSC 4330: Big Data Analytics

Writing a MapReduce Program in Java

A Sample MapReduce Program: WordCount

- The program consists of three files
  - The Mapper
  - The Reducer
  - The driver code

https://hadoop.apache.org/docs/r3.4.1/api/

Keys and Values are Objects

- Keys and values in Hadoop are Java Objects
  - Not Java primitives. Cannot use things like int or float as keys or values
  - Should use the corresponding Writable subclass (e.g. IntWritable, FloatWritable) to “wrap” the values.
- Values are objects which implement *Writable*
  - The writable interface is used in Hadoop’s serialization process, allowing these values to be efficiently stored to or read from files, as well as being passed across the network.
- Keys are objects which implement *WritableComparable*

Why?
- To make it possible to sort the keys

What is Writable?

- The *Writable* interface makes serialization quick and easy for Hadoop
- Any value's type must implement the *Writable* interface
- Hadoop defines its own 'box classes' for strings, integers, and so on

  - IntWritable for ints
  - LongWritable for longs
  - FloatWritable for floats
  - DoubleWritable for doubles
  - Text for strings
  - etc.

e.g.
```
IntWritable key = new IntWritable(5);

int number = key.get();
```

What is WritableComparable?

- A WritableComparable is a Writable which is also Comparable

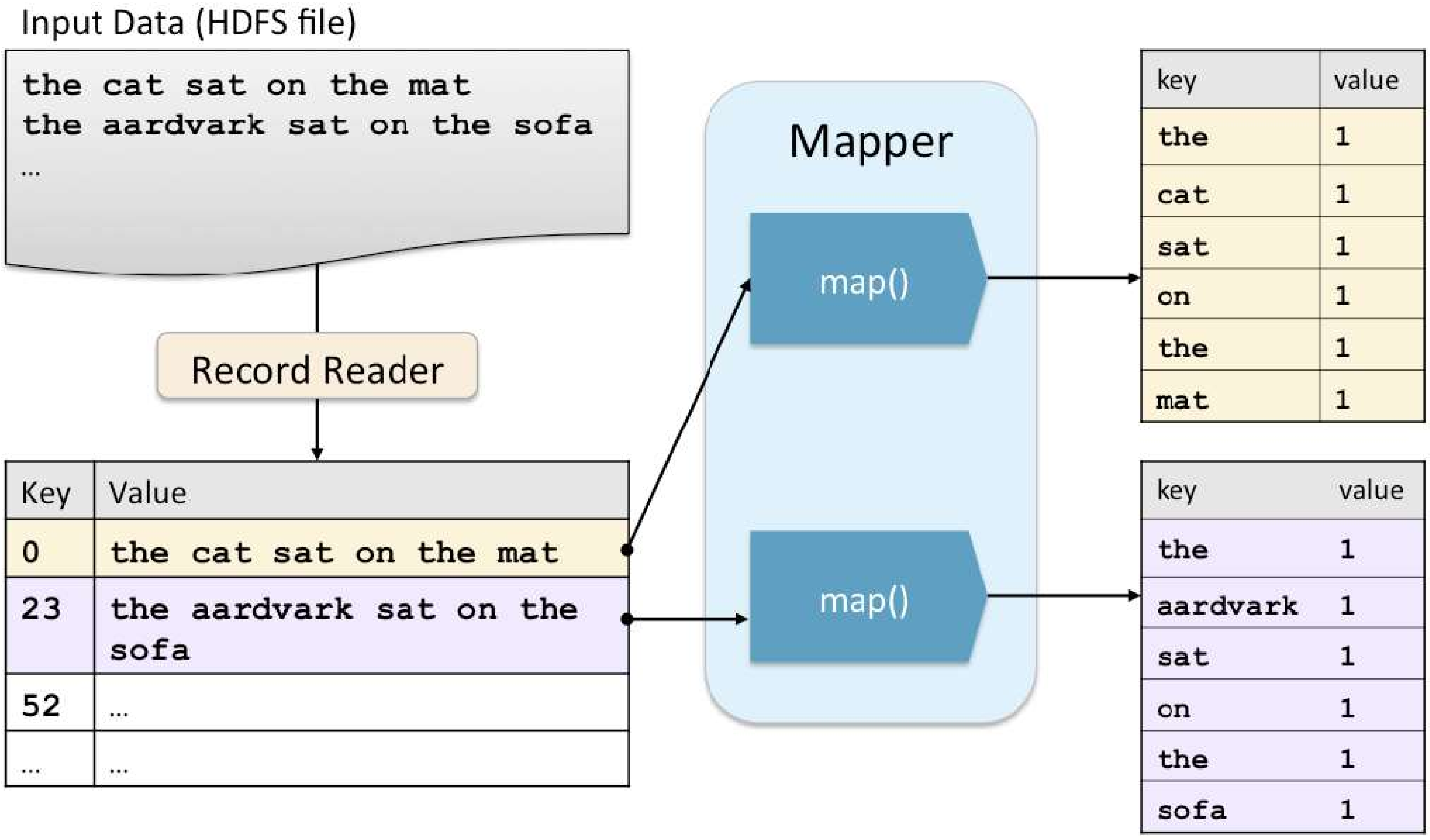
  - Two WritableComparables can be compared against each other to determine their ‘order’

  - Keys must be WritableComparables because they are passed to the Reducer in sorted order

- Note that despite their names, all Hadoop box classes implement both Writable and WritableComparable

  - For example, IntWritable is actually a WritableComparable

WordCount Program – The Mapper



```
the cat sat on the mat
the aardvark sat on the sofa
…
```

Key	Value
0	the cat sat on the mat
23	the aardvark sat on the sofa
52	…
…	…

key	value
the	1
cat	1
sat	1
on	1
the	1
mat	1

key	value
the	1
aardvark	1
sat	1
on	1
the	1
sofa	1

The Mapper: Complete Code

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      String line = value.toString();

      for (String word : line.split("\\W+")) {
          if (word.length() > 0) {
              context.write(new Text(word), new IntWritable(1));
          }
      }
    }
}
```

The Mapper: import Statements

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

You will typically import the two classes IOException and Mapper (highlighted in red), in every Mapper program you write. Other classes such as IntWritable, LongWritable, Text are imported if you need to use those WritableComparable in your program.

The Mapper: Main Code

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      String line = value.toString();

      for (String word : line.split("\\W+")) {
          if (word.length() > 0) {
              context.write(new Text(word), new IntWritable(1));
           }
       }
    }
}
```

The Mapper: Class Declaration (1)

The WordMapper class extends the base class *Mapper*

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
     throws IOException, InterruptedException {

     String line = value.toString();

     for (String word : line.split("\\W+")) {
         if (word.length() > 0) {
             context.write(new Text(word), new IntWritable(1));
          }
      }
    }
}
```

The Mapper: Class Declaration (2)

Input key and value types

Intermediate output key and value types

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
     throws IOException, InterruptedException {

     String line = value.toString();

     for (String word : line.split("\\W+")) {
         if (word.length() > 0) {
             context.write(new Text(word), new IntWritable(1));
         }
     }
   }
}
```

- Keys must be WritableComparable; Values must be Writable
- Type of Input key and value are determined by whatever InputFormat you specify. Can specify the InputFormat in Driver class (discuss later). Default is TextInputFormat.
- Programmers choose what the output key and value types are.

The Mapper: The map method

Types of input key and value specified in the class signature and the map method's signature must match

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
     throws IOException, InterruptedException {

     String line = value.toString();

     for (String word : line.split("\\W+")) {
         if (word.length() > 0) {
             context.write(new Text(word), new IntWritable(1));
         }
      }
    }
}
```

Context object is used to write intermediate data. It also contains information about the job's configuration.

The map Method: Processing the line (1)

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      String line = value.toString();

      for (String word : line.split("\\W+")) {
          if (word.length() > 0) {
              context.write(new Text(word), new IntWritable(1));
          }
      }
    }
}
```

A text object is not a string. toString() method returns the string that a text object contains.

The map Method: Processing the line (2)

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      String line = value.toString();

      for (String word : line.split("\\W+")) {
          if (word.length() > 0) {
              context.write(new Text(word), new IntWritable(1));
          }
      }
    }
}
```

- Split the string up into words using the specified delimiter and then loop through the words
- \W: Matches any non-word character.
- +: Matches the previous element one or more times.
- The backslash "\" before "\W+" is an escape character, indicates that the character that follows it either is a special character, or should be interpreted literally.

Word character:
a-z, A-Z, 0-9, _

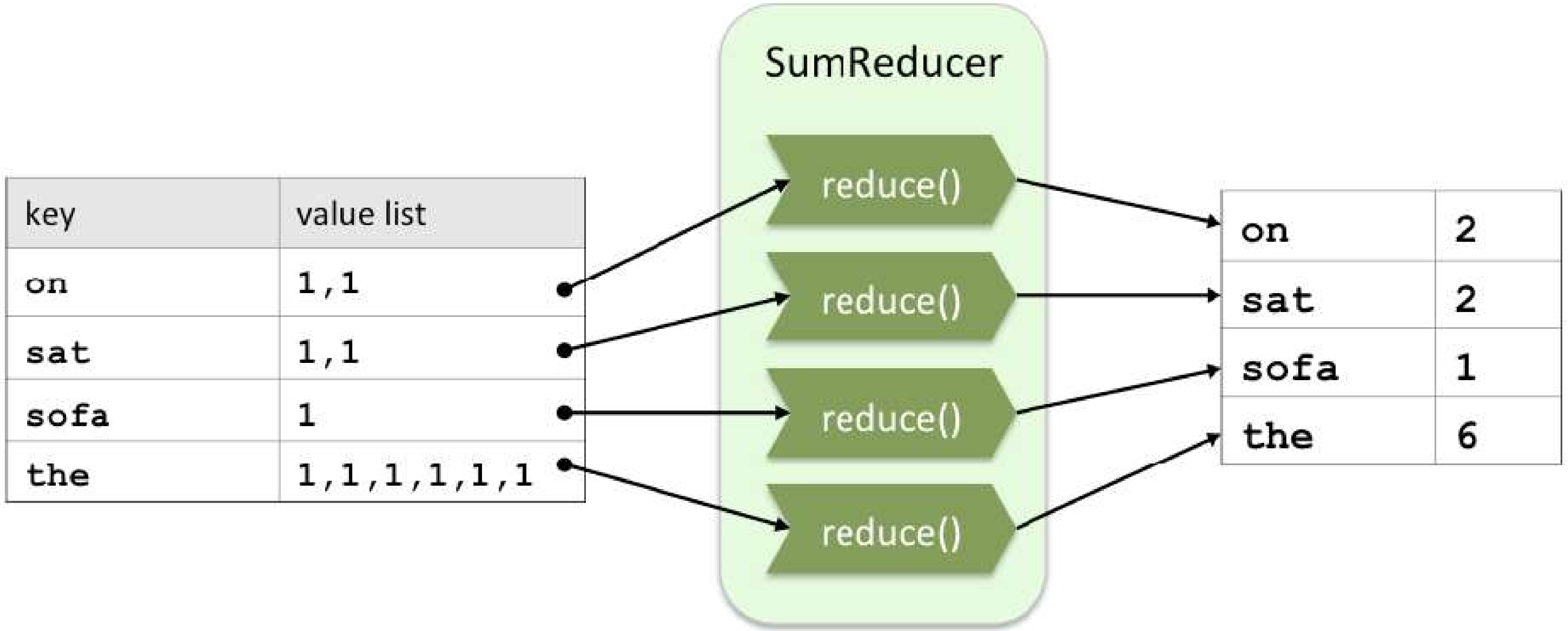
The map Method: Outputting Intermediate Data

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

      String line = value.toString();

      for (String word : line.split("\\W+")) {
           if (word.length() > 0) {
                context.write(new Text(word), new IntWritable(1));
            }
        }
      }
}
```

Type match

- Output a (key, value) pair
- Key: word
- Value: 1
- No I/O code: Hadoop takes care of I/O

WordCount Program – The Reducer



key	value list
on	1,1
sat	1,1
sofa	1
the	1,1,1,1,1,1

on	2
sat	2
sofa	1
the	6

The Reducer: Complete Code

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {

          int wordCount = 0;

          for (IntWritable value : values) {
                wordCount += value.get();
           }

           context.write(key, new IntWritable(wordCount));
     }
}
```

The Reducer: Import Statements

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

You will typically import *java.io.IOException*, and the org.apache.hadoop.mapreduce.Reducer classes shown, in every Reducer you write. Other classes like IntWritable, Text are imported if your program needs to use those WritableComparable.

The Reducer: Main Code

SumReducer class extends the *Reducer* base class

Intermediate key and value types

Final output key and value types

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
       throws IOException, InterruptedException {

       int wordCount = 0;

       for (IntWritable value : values) {
            wordCount += value.get();
        }

        context.write(key, new IntWritable(wordCount));
    }
}
```

The Reducer: reduce method

- The reduce method receives a key and an Iterable collection of objects (which are values output from the Mappers for that key)
- Types of key and value must match the types of input key value in the class signature

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {

         int wordCount = 0;

         for (IntWritable value : values) {
               wordCount += value.get();
          }

          context.write(key, new IntWritable(wordCount));
      }
}
```

The reduce Method: Processing the values

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {

          int wordCount = 0;

          for (IntWritable value : values) {
                wordCount += value.get();
           }

           context.write(key, new IntWritable(wordCount));
      }
}
```

Step through all the elements in the collection. For the wordcounting example, just simply add all the values together.

The reduce method: Writing the Final Output

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {

          int wordCount = 0;

          for (IntWritable value : values) {
                wordCount += value.get();
           }

           context.write(key, new IntWritable(wordCount));
      }
}
```

Type match

Write the final output key-value pair to HDFS

The Driver: Complete Code

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
Import org.apache.hadoop.conf.Configuration;

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

   if (args.length != 2) {
     System.out.printf("Usage: WordCount <input dir> <output dir>\n");
     System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, “wordcount”);
    job.setJarByClass(WordCount.class);
```

The Driver: Complete Code (Cont.)

```
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

   boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

The Driver: Import Statements

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
Import org.apache.hadoop.conf.Configuration;
```

You will typically import the classes that are highlighted in red, in the driver. Other classes such as IntWritable, Text are imported if you need to use those classes in your program.

The Driver: Main Code

```
public class WordCount {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.out.printf("Usage: WordCount <input dir> <output dir>\n");
    System.exit(-1);
   }

   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "wordcount");
   job.setJarByClass(WordCount.class);

   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.setMapperClass(WordMapper.class);
   job.setReducerClass(SumReducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

  boolean success = job.waitForCompletion(true);
   System.exit(success ? 0 : 1);
 }
}
```

The *main* method accepts two command-line arguments: the input and output directories

The Driver: Main Code

```
public class WordCount {
  public static void main(String[] args) throws Exception {
   if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, “wordcount”);
    job.setJarByClass(WordCount.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

   boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

Ensure two command-line arguments are provided (input dir, output dir)

The Driver: Main Code

```
public class WordCount {
  public static void main(String[] args) throws Exception {
   if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, “wordcount”);
    job.setJarByClass(WordCount.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

   boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

- Create a new Job object
- Identify the Jar which contains the Mapper and Reducer by specifying a class in that jar
- Give the job a meaningful name

Creating a new Job object

- The Job and Configuration classes allow you to set configuration options for your MapReduce job
  - The classes to be used for your Mapper and Reducer
  - The input and output directories
  - Many other options
- Any options not explicitly set in your driver code will be read from your Hadoop configuration files
- Any options not specified in your configuration files will use Hadoop’s default values
- You can also use the Job object to submit the job, control its execution, and query its state

The Driver: Main Code

```
public class WordCount {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.out.printf("Usage: WordCount <input dir> <output dir>\n");
    System.exit(-1);
   }

   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "wordcount");
   job.setJarByClass(WordCount.class);

   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.setMapperClass(WordMapper.class);
   job.setReducerClass(SumReducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

  boolean success = job.waitForCompletion(true);
   System.exit(success ? 0 : 1);
 }
}
```

- Specify the input directory from which data will be read
  - All files in the directory are read
  - Exceptions: items whose names begin with a period (.) or underscore (_)
- Specify the output directory to which final output will be written
- Can specify any number of input files or directories as input
- The output path is a single directory

Configuring the Job: Specifying the InputFormat

- The default TextInputFormat will be used unless you specify otherwise
- TextInputFormat is the input format in which the value is a line of text and the key is the byte offset at which that line began in the file.
- To use an InputFormat other than the default

  e.g.
  job.setInputFormatClass(KeyValueTextInputFormat.class)
  - Until it finds the first delimiter (by default tab) it takes everything as key and remaining line as value

Configuring the Job: Specifying Final Output with OutputFormat

- The driver can specify the format of the output data

  - default is a plain text file (TextOutputFormat)

  - Can specify other format

    e.g.

```
job.setOutputFormatClass(DBOutputFormat.class)
```

- Output to a DB table

The Driver: Main Code

```
public class WordCount {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.out.printf("Usage: WordCount <input dir> <output dir>\n");
    System.exit(-1);
   }

   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, “wordcount”);
   job.setJarByClass(WordCount.class);

   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.setMapperClass(WordMapper.class);
   job.setReducerClass(SumReducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

  boolean success = job.waitForCompletion(true);
   System.exit(success ? 0 : 1);
 }
}
```

- Give the Job object information about which class is Mapper, which class is Reducer
- Setting the Mapper and Reducer classes is optional
- If not set in the driver, Hadoop uses default
  - IdentityMapper
  - IdentityReducer

The Driver: Main Code

```
public class WordCount {
  public static void main(String[] args) throws Exception {
   if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, “wordcount”);
    job.setJarByClass(WordCount.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

   boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

- Specify the types for the intermediate output keys and values produced by the Mapper.
- Must match the types of output keys and values in the class prototype of Mapper.
- If the types of the intermediate output key and value are identical to the reducer’s output key and value (e.g. in this example), it is unnecessary to call setMapOutputKeyClass and setMapOutputValueClass methods.

The Driver: Main Code

```
public class WordCount {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.out.printf("Usage: WordCount <input dir> <output dir>\n");
    System.exit(-1);
   }

   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, “wordcount”);
   job.setJarByClass(WordCount.class);

   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.setMapperClass(WordMapper.class);
   job.setReducerClass(SumReducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

  boolean success = job.waitForCompletion(true);
   System.exit(success ? 0 : 1);
 }
}
```

- Specify the types for the Reducer’s output keys and values
- Must match the types of output keys and values in the class prototype of Reducer

The Driver: Main Code

```
public class WordCount {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
     System.out.printf("Usage: WordCount <input dir> <output dir>\n");
     System.exit(-1);
   }

   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, “wordcount”);
   job.setJarByClass(WordCount.class);

   FileInputFormat.setInputPaths(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.setMapperClass(WordMapper.class);
   job.setReducerClass(SumReducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

   boolean success = job.waitForCompletion(true);
   System.exit(success ? 0 : 1);
 }
}
```

- waitForCompletion()
  - Start the job and wait for it to complete (synchronous).
  - The parameter specifying verbosity: if true, display progress to the user.
  - The function polls the JobTracker for progress information and prints this to the console until the job is complete.
- submit()
  - does not block (driver code continues as the job is running)

In-Class Exercise

- Write a MapReduce job that reads any text input and computes the average length of all words that start with each character (case-sensitive).
- For example, for input

```
No   now   is   definitely   not   the   time
```

the output would be:

```
N      2.0
d     10.0
i      2.0
n      3.0
t      3.5
```

Discussion

- What are the key and value of input data to Mapper?

  key: byte offset within the file;   value: text of each line

- What are the key and value of output data of Mapper?

  key: first letter of the word;   value: length of the word

- What are the key and value of input data to Reducer?

  key: first letter of the word;
  value: a list of length of all words that start with the letter

- What are the key and value of output data of Reducer?

  key: first letter of the word;
  value: average length of all words that start with the letter

Now, let's write programs!