

CSE 4/587

Data Intensive Computing

Dr. Eric Mikida

epmikida@buffalo.edu

208 Capen Hall

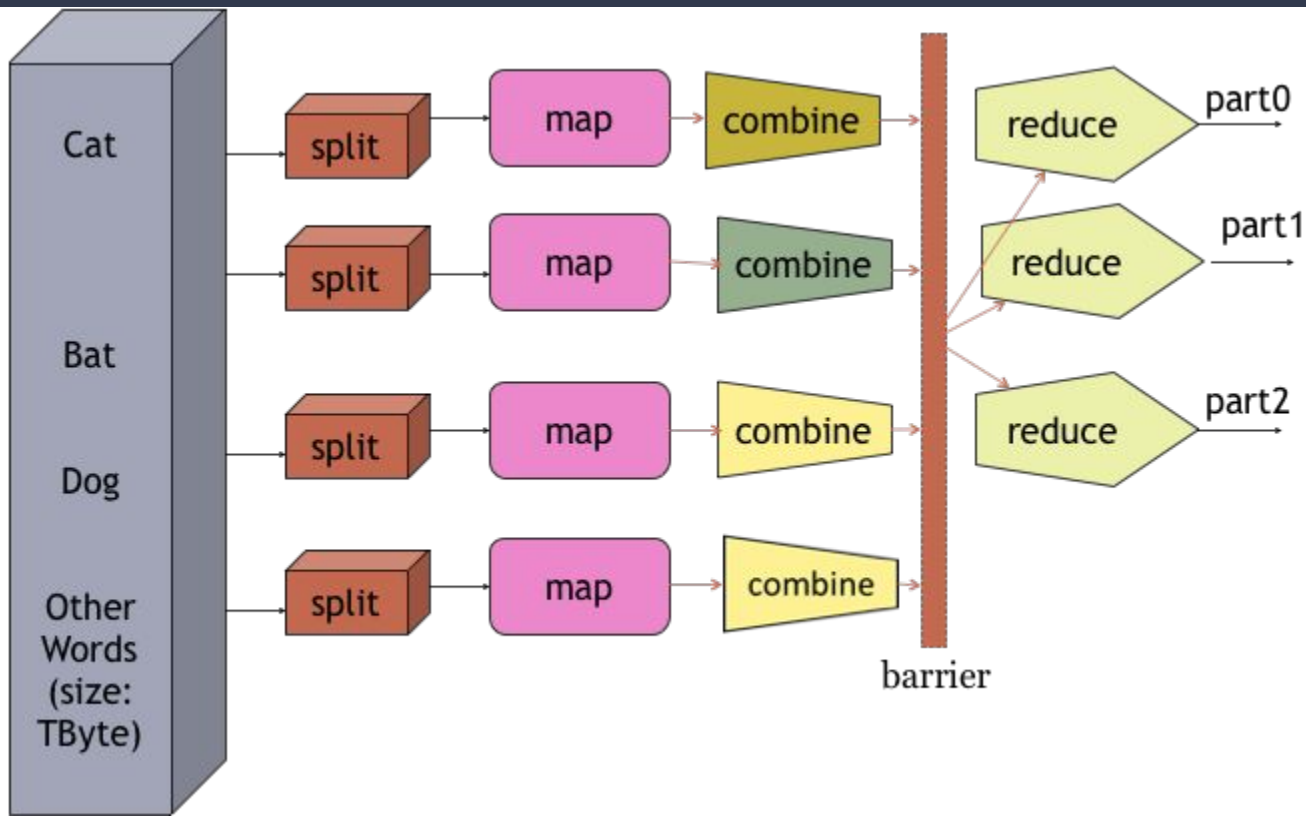
Day 09

Hands-On MapReduce

Recap from Last Class

- Introduced MapReduce as a technique for handling large amounts of data on HDFS
 - "Unreasonable effectiveness of data" – Simple algorithms with more data can be more effective than sophisticated algorithms with less data
 - Applications consist of *mappers* and *reducers*
 - Mappers take input and map it to a set of key-value pairs
 - Reducers take multiple key-value pairs and reduce them to a single key-value pair

The Big Picture



Plan for Today

Today we will do a demonstration of MapReduce in Java

1. Set up our data in a local Hadoop instance and view it in a browser
2. Write a mapper, and a reducer
3. Configure our job
4. Compile and run our MapReduce code as a job in Hadoop
5. Explore the output of our job
6. Extend our example based on our observations

Setting up Hadoop

- Today we'll be running a single node instance of Hadoop
 - For installation I used Homebrew (brew install hadoop)
 - Directions for setup can be found at [1]
- Once we've done all the setup we'll need to create our directories and add our files to the instance
- We can also look at our instance via web browser

[1] <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>

Some Useful Commands

```
hadoop fs -mkdir
```

```
hadoop fs -touch
```

```
hadoop fs -ls
```

```
hadoop fs -put
```

```
hadoop fs -mv
```

```
hadoop fs -cp
```

```
hadoop fs -rm
```

```
hadoop fs -cat
```

```
hadoop fs -mkdir /user/epmikida
```

```
hadoop fs -mkdir input
```

```
hadoop fs -put <path>/pride_and_prejudice.txt input/
```

```
hadoop fs -cat input/pride_and_prejudice.txt
```

Some Useful Commands

```
hadoop fs -mkdir
```

```
hadoop fs -touch
```

```
hadoop fs -ls
```

```
hadoop fs -put
```

```
hadoop fs -mv
```

```
hadoop fs -cp
```

```
hadoop fs -rm
```

```
hadoop fs -cat
```

```
hadoop fs -mkdir /user/epmikida
```

←Create our home directory

```
hadoop fs -mkdir input
```

```
hadoop fs -put <path>/pride_and_prejudice.txt input/
```

```
hadoop fs -cat input/pride_and_prejudice.txt
```

Some Useful Commands

```
hadoop fs -mkdir
```

```
hadoop fs -touch
```

```
hadoop fs -ls
```

```
hadoop fs -put
```

```
hadoop fs -mv
```

```
hadoop fs -cp
```

```
hadoop fs -rm
```

```
hadoop fs -cat
```

```
hadoop fs -mkdir /user/epmikida
```

←Create our home directory

```
hadoop fs -mkdir input
```

←Create a directory for input

```
hadoop fs -put <path>/pride_and_prejudice.txt input/
```

```
hadoop fs -cat input/pride_and_prejudice.txt
```


Some Useful Commands

```
hadoop fs -mkdir
```

```
hadoop fs -touch
```

```
hadoop fs -ls
```

```
hadoop fs -put
```

```
hadoop fs -mv
```

```
hadoop fs -cp
```

```
hadoop fs -rm
```

```
hadoop fs -cat
```

```
hadoop fs -mkdir /user/epmikida
```

←Create our home directory

```
hadoop fs -mkdir input
```

←Create a directory for input

```
hadoop fs -put <path>/pride_and_prejudice.txt input/
```

←Add a local file to HDFS

```
hadoop fs -cat input/pride_and_prejudice.txt
```

Some Useful Commands

```
hadoop fs -mkdir
```

```
hadoop fs -touch
```

```
hadoop fs -ls
```

```
hadoop fs -put
```

```
hadoop fs -mv
```

```
hadoop fs -cp
```

```
hadoop fs -rm
```

```
hadoop fs -cat
```

```
hadoop fs -mkdir /user/epmikida
```

←Create our home directory

```
hadoop fs -mkdir input
```

←Create a directory for input

```
hadoop fs -put <path>/pride_and_prejudice.txt input/
```

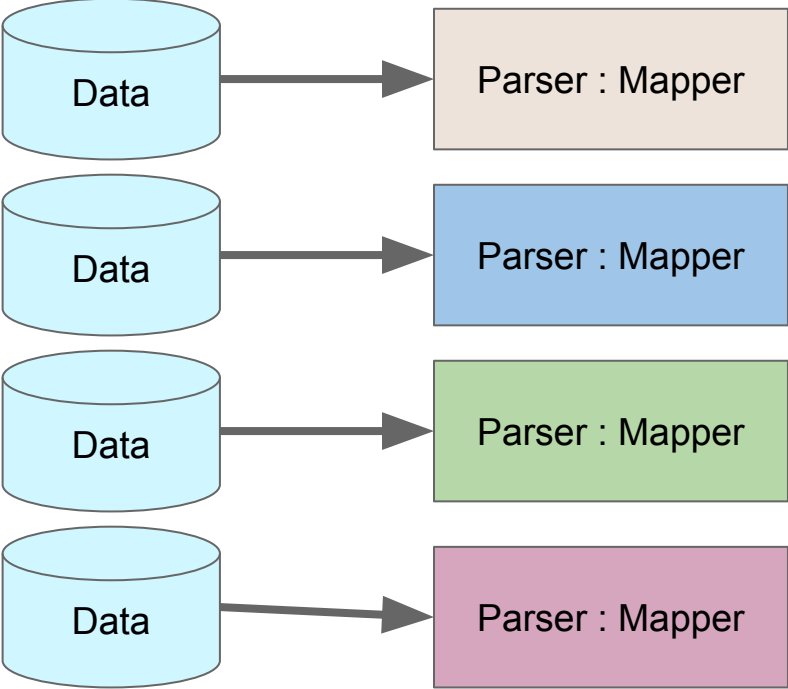
←Add a local file to HDFS

```
hadoop fs -cat input/pride_and_prejudice.txt
```

←Check contents of the file

HDFS Demo

Map Operation



The mapped data is shown as a grid of colored boxes. Dashed lines connect the boxes to the corresponding "Parser : Mapper" boxes from the previous diagram. The colors of the boxes correspond to the colors of the "Parser : Mapper" boxes: light brown, light blue, light green, and light pink.

web	1		
weed	1	color	1
green	1	green	1
			1
flower	1		
weed	1		
green	1		
sun	1		
flower	1		
land	1		
grow	1		
...	...		
		web	1
		web	1
		order	1
		dollar	1
		moon	1
		land	1
		moon	1

Mapper Java Code

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3
4     private final static IntWritable one = new IntWritable(1);
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context)
8         throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

Mapper Java Code

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3
4     private Input Key type and Input Value type new IntWritable(1);
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context)
8         throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

Mapper Java Code

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3
4     private final static Int Output Key type and Output Value type;
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context)
8         throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

Mapper Java Code

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3
4     private final static IntWritable one = new IntWritable(1);
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context)
8         throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

This is our input text that we are parsing

Mapper Java Code

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3
4     private final static IntWritable one = new IntWritable(1);
5     private Text word = new Text();
6
7     public void map(Object key, Text value, Context context)
8         throws IOException, InterruptedException {
9         StringTokenizer itr = new StringTokenizer(value.toString());
10        while (itr.hasMoreTokens()) {
11            word.set(itr.nextToken());
12            context.write(word, one);
13        }
14    }
15 }
```

Write out a key-value pair for the word we found, with a count of one.

Mapper Notes

- The MapReduce framework will spawn one Mapper per input split
 - Usually this is the number of blocks
 - Mappers are expensive to startup – choose granularity wisely
- Key types and value types must be serializable (Writable interface)
- Output will be grouped by key by the system
 - Optionally can be combined if a **Combiner** is also defined

Reducer Java Code

```
1 public static class IntSumReducer
2     extends Reducer<Text,IntWritable,Text,IntWritable> {
3
4     private IntWritable result = new IntWritable();
5
6     public void reduce(Text key, Iterable<IntWritable> values, Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10             sum += val.get();
11         }
12         result.set(sum);
13         context.write(key, result);
14     }
15 }
```

Reducer Java Code

```
1 public static class IntSumReducer
2     extends Reducer<Text,IntWritable,Text,IntWritable> {
3
4     private InputFormat<Text,IntWritable> inputFormat;
5
6     public void reduce(Text key, Iterable<IntWritable> values, Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10             sum += val.get();
11         }
12         result.set(sum);
13         context.write(key, result);
14     }
15 }
```

Reducer Java Code

```
1 public static class IntSumReducer
2     extends Reducer<Text,IntWritable,Text,IntWritable> {
3
4     private IntWritable result = new Output Key type and Output Value type
5
6     public void reduce(Text key, Iterable<IntWritable> values, Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10            sum += val.get();
11        }
12        result.set(sum);
13        context.write(key, result);
14    }
15 }
```

Reducer Java Code

```
1 public static class IntSumReducer
2     extends Reducer<Text,IntWritable,Text,IntWritable> {
3
4     private IntWritable result =
5
6     public void reduce(Text key, Iterable<IntWritable> values, Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10             sum += val.get();
11         }
12         result.set(sum);
13         context.write(key, result);
14     }
15 }
```

A single input key, and a list of values to be reduced

Reducer Java Code

```
1 public static class IntSumReducer
2     extends Reducer<Text,IntWritable,Text,IntWritable> {
3
4     private IntWritable result = new IntWritable();
5
6     public void reduce(Text key, Iterable<IntWritable> values, Context context)
7         throws IOException, InterruptedException {
8         int sum = 0;
9         for (IntWritable val : values) {
10             sum += val.get();
11         }
12         result.set(sum);
13         context.write(key, result);
14     }
15 }
```

Write out our reduced key-value pair

Reducer Notes

- The number of reducers is set in job configuration
 - Rule of thumb is either 0.95 or 1.75...but this can vary
 - Overhead vs Load balance is an important trade-off
- Output of mapping is sorted before reducing begins
- The framework passes each key and its values to a reducer
 - Which keys go to which reducer can be controlled with a partitioner

Job Configuration

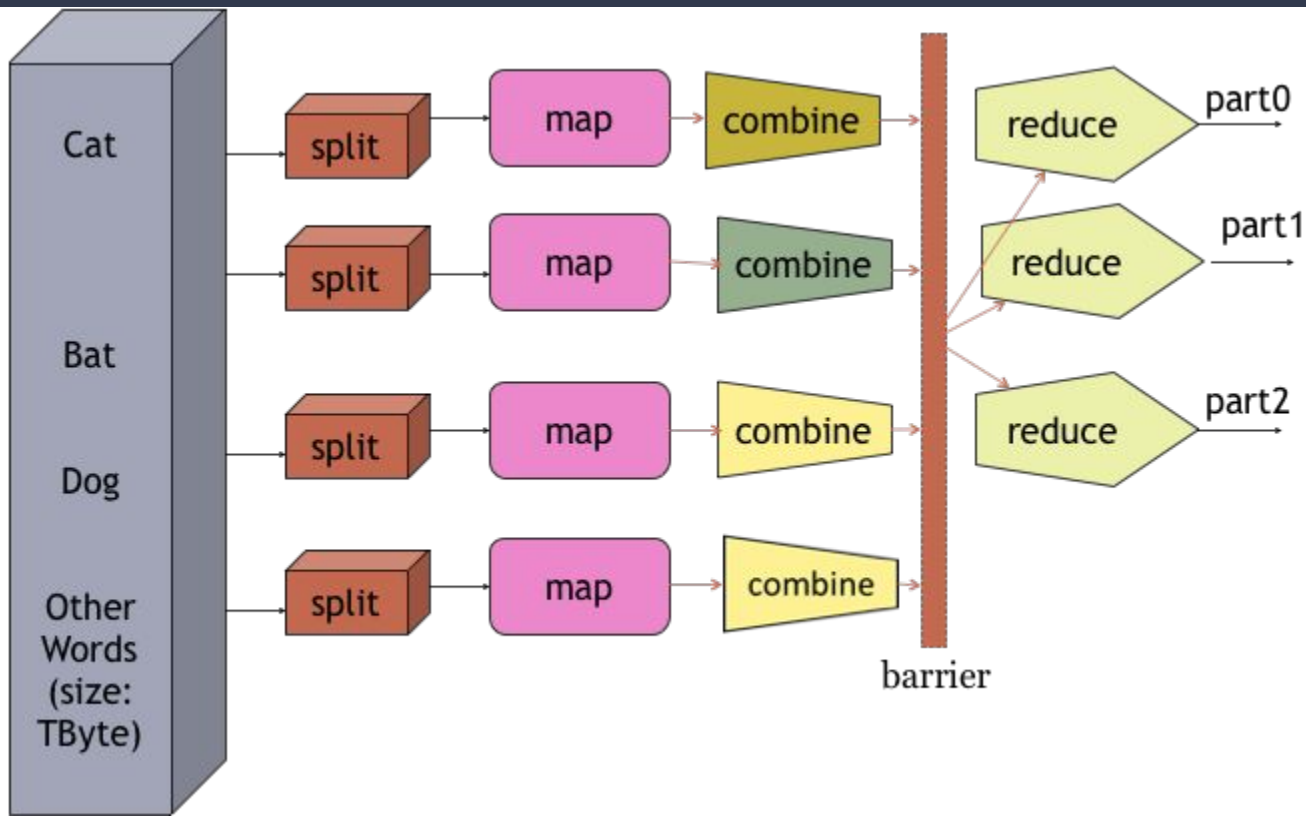
```
1 public static void main(String[] args) throws Exception {
2     Configuration conf = new Configuration();
3     Job job = Job.getInstance(conf, "word count");
4     job.setJarByClass(WordCount.class);
5     job.setMapperClass(TokenizerMapper.class);
6     job.setCombinerClass(IntSumReducer.class);
7     job.setReducerClass(IntSumReducer.class);
8     job.setOutputKeyClass(Text.class);
9     job.setOutputValueClass(IntWritable.class);
10    FileInputFormat.addInputPath(job, new Path(args[0]));
11    FileOutputFormat.setOutputPath(job, new Path(args[1]));
12    System.exit(job.waitForCompletion(true) ? 0 : 1);
13 }
```

Job Configuration

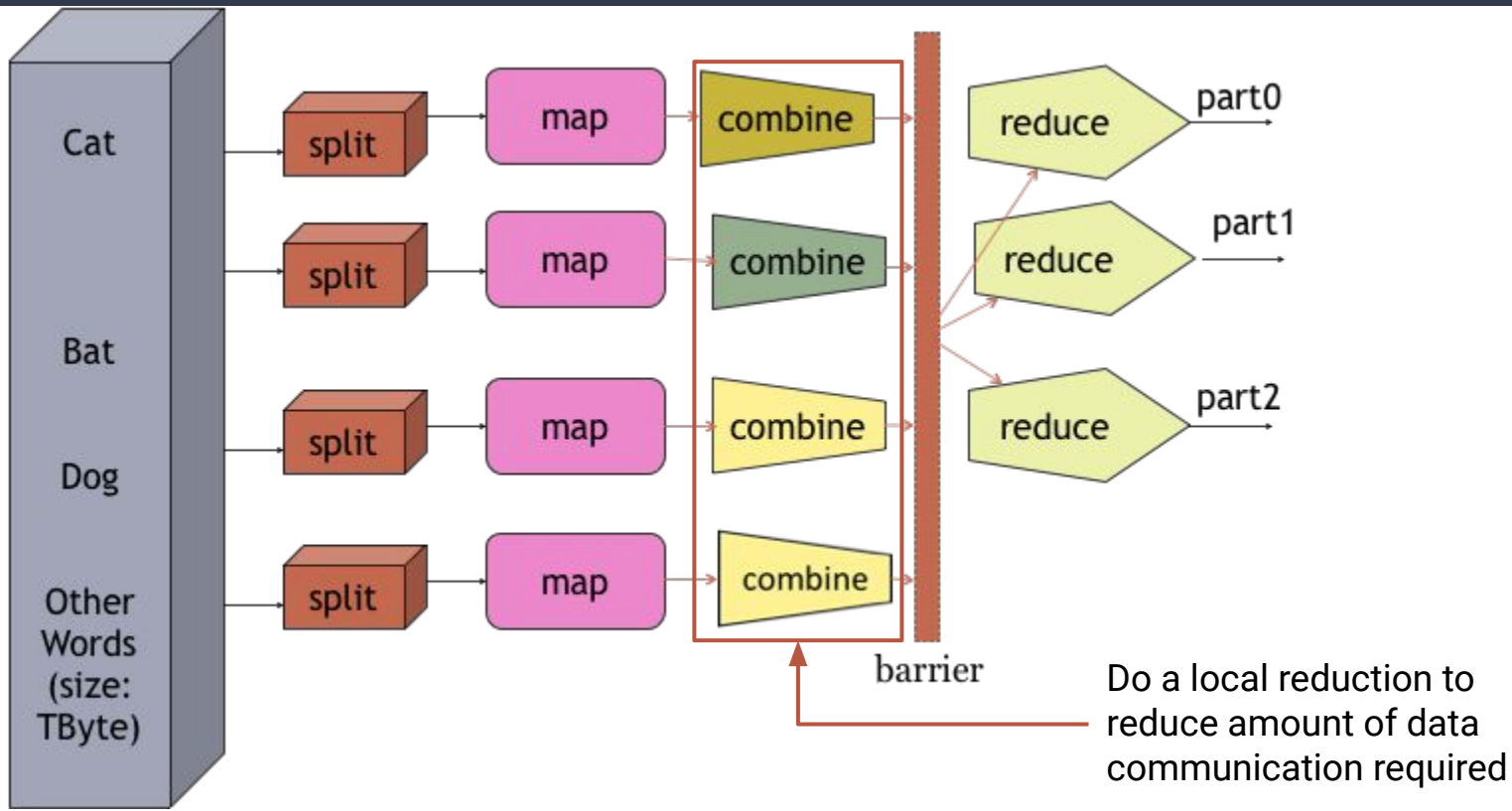
```
1 public static void main(String[] args) throws Exception {
2     Configuration conf = new Configuration();
3     Job job = Job.getInstance(conf, "word count");
4     job.setJarByClass(WordCount.class);
5     job.setMapperClass(TokenizerMapper.class);
6     job.setCombinerClass(IntSumReducer.class);
7     job.setReducerClass(IntSumReducer.class);
8     job.setOutputKeyClass(Text.class);
9     job.setOutputValueClass(IntWritable.class);
10    FileInputFormat.addInputPath(job, new Path(args[0]));
11    FileOutputFormat.setOutputPath(job, new Path(args[1]));
12    System.exit(job.waitForCompletion(true) ? 0 : 1);
13 }
```

Set types of Mappers and Reducers
(and Combiners)

What's A Combiner?



What's A Combiner?



MapReduce Demo

Other Useful Components

- Argument Parsing - Get command line arguments and set global configuration parameters
- Distributed Cache - Allows for data sharing between components
- Counter - Allows for status information to be reported
- Partitioner - Allows for control of how keys are distributed to reducers

Generic Argument Parsing

```
1 Configuration conf = new Configuration();
2 GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
3 String[] remainingArgs = optionParser.getRemainingArgs();
4 if ((remainingArgs.length!=2) && (remainingArgs.length!=4)) { System.exit(2); }
5
6 List<String> otherArgs = new ArrayList<String>();
7 for (int i=0; i < remainingArgs.length; ++i) {
8     if ("-skip".equals(remainingArgs[i])) {
9         job.addCacheFile(new Path(remainingArgs[++i]).toUri());
10        job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
11    } else {
12        otherArgs.add(remainingArgs[i]);
13    }
14 }
```

Generic Argument Parsing

```
1 Configuration conf = new Configuration();
2 GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
3 String[] remainingArgs = optionParser.getRemainingArgs();
4 if ((remainingArgs.length!=2) && (remainingArgs.length!=1)) {
5     // Get generic arguments and set them in our job configuration
6     List<String> otherArgs = new ArrayList<String>();
7     for (int i=0; i < remainingArgs.length; ++i) {
8         if ("-skip".equals(remainingArgs[i])) {
9             job.addCacheFile(new Path(remainingArgs[++i]).toUri());
10            job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
11        } else {
12            otherArgs.add(remainingArgs[i]);
13        }
14    }
```


Generic Argument Parsing

```
1 Configuration conf = new Configuration();
2 GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
3 String[] remainingArgs = optionParser.getRemainingArgs();
4 if ((remainingArgs.length!=2) && (remainingArgs.length!=4)) { System.exit(2); }
5
6 List<String> otherArgs = new ArrayList<String>();
7 for (int i=0; i < remainingArgs.length; ++i) {
8     if ("-skip".equals(remainingArgs[i])) {
9         job.addCacheFile(new Path(remainingArgs[++i]).toUri());
10        job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
11    } else {
12        otherArgs.add(remainingArgs[i]);
13    }
14 }
```

Check for non-generic arguments

Generic Argument Parsing

```
1 Configuration conf = new Configuration();
2 GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
3 String[] remainingArgs = optionParser.getRemainingArgs();
4 if ((remainingArgs.length!=2) && (remainingArgs.length!=4)) { System.exit(2); }
5
6 List<String> otherArgs = new ArrayList<String>();
7 for (int i=0; i < remainingArgs.length; i++) {
8     if ("-skip".equals(remainingArgs[i])) {
9         job.addCacheFile(new Path(remainingArgs[++i]).toUri());
10        job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
11    } else {
12        otherArgs.add(remainingArgs[i]);
13    }
14 }
```

Load a skip file into the distributed cache and set a configuration parameter

Generic Argument Parsing

- Generic arguments can be passed to the MapReduce job with -D
 - ie: `-Dwordcount.case.sensitive=false`
- Mappers/Reducers can access configuration values and cache files via the Context object
 - They can also define a `setup` method which is called before mapping/reducing begins

Augmenting Our Mapper

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3     ...
4     public void setup(Context context) throws IOException, InterruptedException {
5         conf = context.getConfiguration();
6         caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
7         if (conf.getBoolean("wordcount.skip.patterns", false)) {
8             URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
9             for (URI patternsURI : patternsURIs) {
10                Path patternsPath = new Path(patternsURI.getPath());
11                String patternsFileName = patternsPath.getName().toString();
12                parseSkipFile(patternsFileName);
13            }
14        }
15    }
16
```

Augmenting Our Mapper

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3     ...
4     public void setup(Context context) throws IOException, InterruptedException {
5         conf = context.getConfiguration();
6         caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
7         if (conf.getBoolean("wordcount.skip.patterns", false)) {
8             URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
9             for (URI patternsURI : patternsURIs) {
10                Path patternsPath = new Path(patternsURI.getPath());
11                String patternsFileName = patternsPath.getName().toString();
12                parseSkipFile(patternsFileName);
13            }
14        }
15    }
16 }
```

Setup function allows our mapper to do some initial setup based on global configuration

Augmenting Our Mapper

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3     ...
4     public void setup(Context context) throws IOException, InterruptedException {
5         conf = context.getConfiguration();
6         caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
7         if (conf.getBoolean("wordcount.skip.patterns", false)) {
8             URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
9             for (URI patternsURI : patternsURIs) {
10                Path patternsPath = new Path(patternsURI);
11                String patternsFileName = patternsPath.getName(0).toString();
12                parseSkipFile(patternsFileName);
13            }
14        }
15    }
16
```

Read in configuration values

Augmenting Our Mapper

```
1 public static class TokenizerMapper
2     extends Mapper<Object, Text, Text, IntWritable> {
3     ...
4     public void setup(Context context) throws IOException, InterruptedException {
5         conf = context.getConfiguration();
6         caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
7         if (conf.getBoolean("wordcount.skip.patterns", false)) {
8             URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
9             for (URI patternsURI : patternsURIs) {
10                Path patternsPath = new Path(patternsURI.getPath());
11                String patternsFileName = patternsPath.getName().toString();
12                parseSkipFile(patternsFileName);
13            }
14        }
15    }
16
```

Access files from the distributed cache

Augmenting our Mapper

We can now use this information inside our map function

Augmenting Our map Function

```
1 static enum CountersEnum { INPUT_WORDS }
2 public void map(Object key, Text value, Context context)
3     throws IOException, InterruptedException {
4     String line=(caseSensitive)?value.toString():value.toString().toLowerCase();
5     for (String pattern : patternsToSkip) { line = line.replaceAll(pattern, ""); }
6
7     StringTokenizer itr = new StringTokenizer(line);
8     while (itr.hasMoreTokens()) {
9         word.set(itr.nextToken());
10        context.write(word, one);
11        Counter counter = context.getCounter(CountersEnum.class.getName(),
12            CountersEnum.INPUT_WORDS.toString());
13        counter.increment(1);
14    }
15 }
```

Augmenting Our map Function

```
1 static enum CountersEnum { INPUT_WORDS }
2 public void map(Object key, Text value, Context context)
3     throws IOException, InterruptedException {
4     String line=(caseSensitive)?value.toString():value.toString().toLowerCase();
5     for (String pattern : patternsToSkip) { line = line.replaceAll(pattern, ""); }
6
7     StringTokenizer itr = new String
8     while (itr.hasMoreTokens()) {
9         word.set(itr.nextToken());
10        context.write(word, one);
11        Counter counter = context.getCounter(CountersEnum.class.getName(),
12            CountersEnum.INPUT_WORDS.toString());
13        counter.increment(1);
14    }
15 }
```

Use our configuration from setup

Augmenting Our map Function

```
1 static enum CountersEnum { INPUT_WORDS }
2 public void map(Object key, Text value, Context context)
3     throws IOException, InterruptedException {
4     String line=(caseSensitive?value.toString().value.toString().toLowerCase());
5     for (String pattern : patternsToSkip) { line = line.replaceAll(pattern, ""); }
6
7     StringTokenizer itr = new StringTokenizer(line);
8     while (itr.hasMoreTokens()) {
9         word.set(itr.nextToken());
10        context.write(word, one);
11        Counter counter = context.getCounter(CountersEnum.class.getName(),
12            CountersEnum.INPUT_WORDS.toString());
13        counter.increment(1);
14    }
15 }
```

Define a Counter (as an Enum)

Augmenting Our map Function

```
1 static enum CountersEnum { INPUT_WORDS }
2 public void map(Object key, Text value, Context context)
3     throws IOException, InterruptedException {
4     String line=(caseSensitive)?value.toString():value.toString().toLowerCase();
5     for (String pattern : patternsToSkip) { line = line.replaceAll(pattern, ""); }
6
7     StringTokenizer itr = new StringTokenizer(line);
8     while (itr.hasMoreTokens()) {
9         word.set(itr.nextToken()); Use our Counter to count the number of words we parsed
10        context.write(word, one);
11        Counter counter = context.getCounter(CountersEnum.class.getName(),
12            CountersEnum.INPUT_WORDS.toString());
13        counter.increment(1);
14    }
15 }
```

Improved MapReduce Demo

References

Setting up Hadoop

<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>

MapReduce Tutorial

<https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>