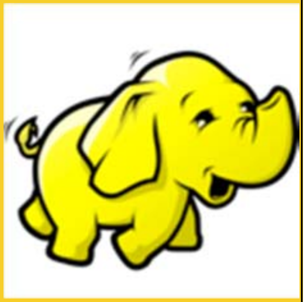
Map Reduce 2.0
Input and Output

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/



Agenda

- **MapReduce Theory**
- **Types of Keys and Values**
- **Input and Output Formats**
- **Discuss Anatomy of**
  - Mappers
  - Reducers
  - Combiners
  - Partitioners

MapReduce Theory

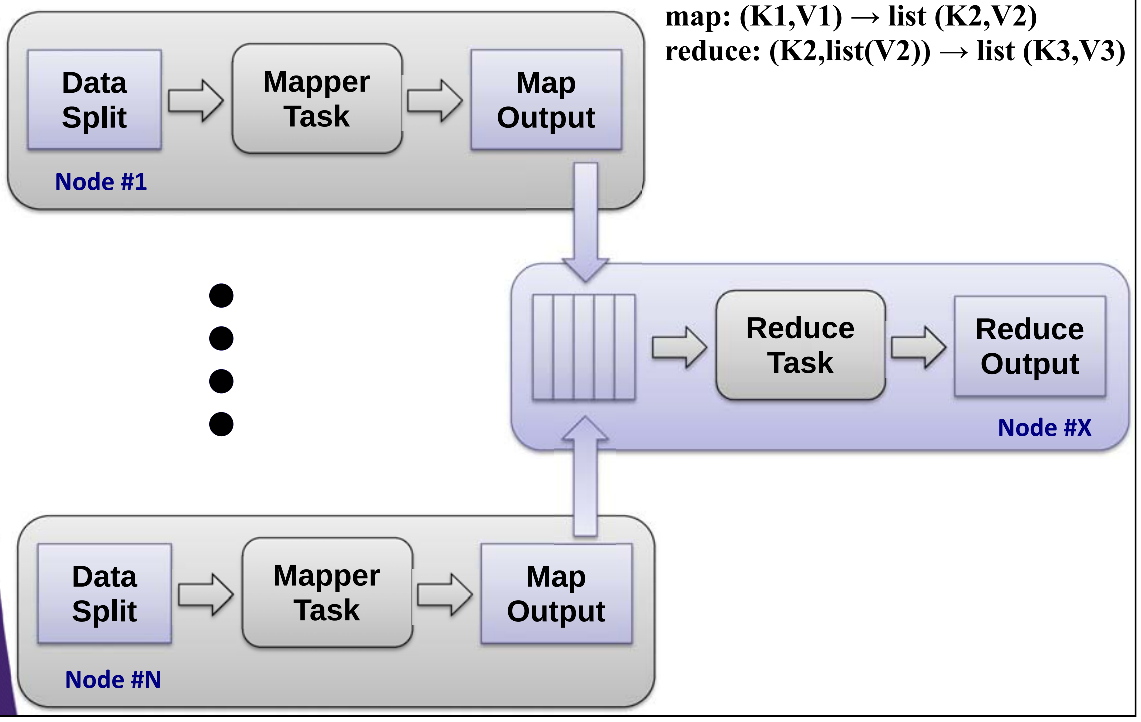
- **Map and Reduce functions produce input and output**
  - Input and output can range from Text to Complex data structures
  - Specified via Job’s configuration
  - Relatively easy to implement your own
- **Generally we can treat the flow as**

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

  - Reduce input types are the same as map output types

Map Reduce Flow of Data

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



Key and Value Types

- **Utilizes Hadoop's serialization mechanism for writing data in and out of network, database or files**
  - Optimized for network serialization
  - A set of basic types is provided
  - Easy to implement your own
- **Extends Writable interface**
  - Framework's serialization mechanisms
  - Defines how to read and write fields
  - org.apache.hadoop.io package

Key and Value Types

- **Keys must implement WritableComparable interface**
  - Extends Writable and java.lang.Comparable<T>
  - Required because keys are sorted prior reduce phase
- **Hadoop is shipped with many default implementations of WritableComparable<T>**
  - Wrappers for primitives (String, Integer, etc...)
  - Or you can implement your own

WritableComparable<T> Implementations

Hadoop's Class	Explanation
BooleanWritable	Boolean implementation
BytesWritable	Bytes implementation
DoubleWritable	Double implementation
FloatWritable	Float implementation
IntWritable	Int implementation
LongWritable	Long implementation
NullWritable	Writable with no data

Implement Custom WritableComparable<T>

- **Implement 3 methods**
  - write(DataOutput)
    - Serialize your attributes
  - readFields(DataInput)
    - De-Serialize your attributes
  - compareTo(T)
    - Identify how to order your objects
    - If your custom object is used as the key it will be sorted prior to reduce phase

BlogWritable – Implemenation of WritableComparable<T>

```
public class BlogWritable implements
                          WritableComparable<BlogWritable> {

      private String author;
      private String content;
      public BlogWritable(){}
      public BlogWritable(String author, String content) {
             this.author = author;
             this.content = content;
      }
      public String getAuthor() {
             return author;

      public String getContent() {
             return content;

...
...
...
```

BlogWritable – Implemenation of WritableComparable<T>

```
...
...
  @Override
  public void readFields(DataInput input) throws IOException {
       author = input.readUTF();
       content = input.readUTF();
  }


  @Override
  public void write(DataOutput output) throws IOException {
       output.writeUTF(author);
       output.writeUTF(content);
  }


  @Override
  public int compareTo(BlogWritable other) {
       return author.compareTo(other.author);
  }
}
```

1. How the data is read

2. How to write data

3. How to order `BlogWritables`

Mapper

- **Extend Mapper class**
  - Mapper<KeyIn, ValueIn, KeyOut, ValueOut>
- **Simple life-cycle**
  1. The framework first calls setup(Context)
  2. for each key/value pair in the split:
     - map(Key, Value, Context)
  3. Finally cleanup(Context) is called

InputSplit

- **Splits are a set of logically arranged records**
  - A set of lines in a file
  - A set of rows in a database table
- **Each instance of mapper will process a single split**
  - Map instance processes one record at a time
    - map(k,v) is called for each record
- **Splits are implemented by extending InputSplit class**

InputSplit

- **Framework provides many options for InputSplit implementations**
  - Hadoop’s FileSplit
  - HBase’s TableSplit
- **Don’t usually need to deal with splits directly**
  - InputFormat’s responsibility

InputFormat

- **Specification for reading data**
- **Creates Input Splits**
  - Breaks up work into chunks
- **Specifies how to read each split**
  - Divides splits into records
  - Provides an implementation of RecordReader

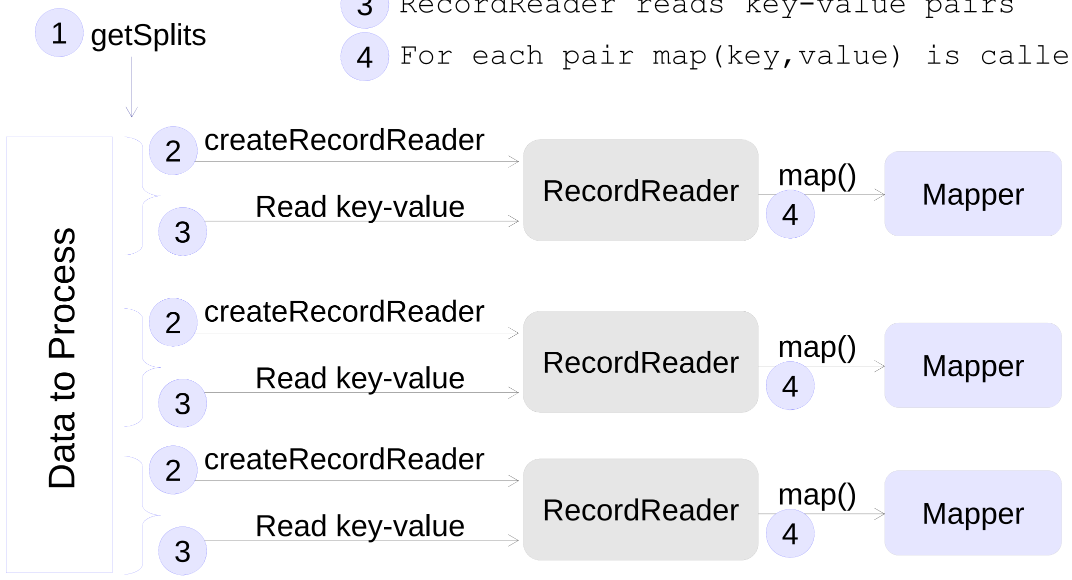
```
public abstract class InputFormat<K, V> {
  public abstract
    List<InputSplit> getSplits(JobContext context)
                          throws IOException, InterruptedException;
  public abstract
    RecordReader<K,V> createRecordReader(InputSplit split,
                   TaskAttemptContext context )
                          throws IOException, InterruptedException;
}
```

Framework's Usage of InputFormat Implementation

1. **Calculate splits by calling InputFormat.getSplits**
2. **For each split schedule a map task**
   - Distributed between the cluster
   - Each Map executes in its own JVM
3. **For each Mapper instance a reader is retrieved by InputFormat.createRecordReader**
   - Takes InputSplit instance as a parameter
4. **RecordReader generates key-value pairs**
5. **map() method is called for each key-value pair**

Framework's Usage of InputFormat Implementation

```
1 Generate splits
2 Each split gets its own RecordReader
3 RecordReader reads key-value pairs
4 For each pair map(key,value) is called
```



Hadoop's InputFormats

- **Hadoop eco-system is packaged with many InputFormats**
  - TextInputFormat
  - NLineInputFormat
  - DBInputFormat
  - TableInputFormat (HBASE)
  - StreamInputFormat
  - SequenceFileInputFormat
  - Etc...
- **Configure on a Job object**
  - job.setInputFormatClass(XXXInputFormat.class);

TextInputFormat

- **Plaint Text Input**
- **Default format**

Split:	Single HDFS block (can be configured)
Record:	Single line of text; linefeed or carriage-return used to locate end of line
Key:	LongWritable - Position in the file
Value:	Text - line of text

** Please see StartsWithCountJob for sample usage

NLineInputFormat

- **Same as TextInputFormat but splits equal to configured N lines**

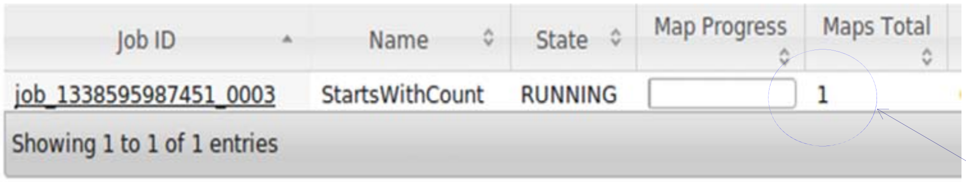
Split:	N lines; configured via *mapred.line.input.format* or *NLineInputFormat.setNumLinesPerSplit(job, 100);*
Record:	Single line of text
Key:	LongWritable - Position in the file
Value:	Text - line of text

** Please see StartsWithCountJob_NLineInput for sample usage

Running TextInputFormat vs. NLineInputFormat

- **Two separate runs StartsWithCountJob, one with TextInputFormatconfigured, next with NLineInputFormat configured**
- **Input is /training/playArea/hamlet.txt**
  - 5159 lines
  - 206.3k

```
job.setInputFormatClass(TextInputFormat.class);
```

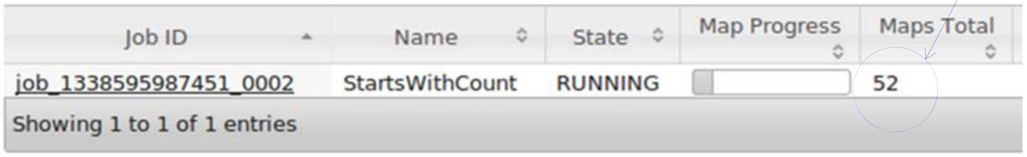


Job ID	Name	State	Map Progress	Maps Total
job_1338595987451_0003	StartsWithCount	RUNNING		1

Showing 1 to 1 of 1 entries

# of splits

```
job.setInputFormatClass(NLineInputFormat.class);
NLineInputFormat.setNumLinesPerSplit(job, 100);
```



Job ID	Name	State	Map Progress	Maps Total
job_1338595987451_0002	StartsWithCount	RUNNING		52

Showing 1 to 1 of 1 entries

TableInputFormat

- **Converts data in HTable to format consumable to MapReduce**
- **Mapper must accept proper key/values**

Split:	Rows in one HBase Region (provided Scan may narrow down the result)
Record:	Row, returned columns are controlled by a provided scan
Key:	ImmutableBytesWritable
Value:	Result (HBase class)

StartCountJob – Input from HBase

- **Let's re-write StartWithCountJob to read input from HBase table**
  - 'HBaseSamples' table, 'count:word' family-column qualifier

```
$ hbase shell
hbase(main):005:0> scan 'HBaseSamples', {COLUMNS=>'count:word'}
ROW          COLUMN+CELL
count001     column=count:word, timestamp=1338605322765, value=Elephant
count002     column=count:word, timestamp=1338605412699, value=count
count003     column=count:word, timestamp=1338605412729, value=Updating
count004     column=count:word, timestamp=1338605412757, value=all
count005     column=count:word, timestamp=1338605412780, value=regions
count006     column=count:word, timestamp=1338605412809, value=with
count007     column=count:word, timestamp=1338605412835, value=the
count008     column=count:word, timestamp=1338605412856, value=new
count009     column=count:word, timestamp=1338605412888, value=updated
count010     column=count:word, timestamp=1338605412910, value=Done
count011     column=count:word, timestamp=1338605412933, value=seconds
count012     column=count:word, timestamp=1338605414526, value=row
12 row(s) in 0.1810 seconds
```

StartCountJob – Input from HBase

1. **Re-configure Job to use HBase as input**
   - Read from table 'HBaseSamples' and column 'count:word'
   - Construct new Job 'StartWithCountJob_HBaseInput'
   - Configure Job to use new Mapper
   - New mapper now has to accept HBase Writables
     - ImmutableBytesWritable for key
     - Result for value
   - Keep reducer and combiner the same
2. **Implement a new Mapper**
   - Grab the value from Result and write-out Text and IntWritable
   - Output is the same as in the original StartWithCountJob

1: Re-configure Job to use HBase as Input

```
public class StartWithCountJob_HBaseInput
                    extends Configured implements Tool {

  protected final static String TABLE_NAME = "HBaseSamples";
  protected final static byte[] FAMILY = toBytes("count");
  protected final static byte[] COLUMN = toBytes("word");
```

Define table name, family and column to read from

```
  @Override
  public int run(String[] args) throws Exception {
       Job job = Job.getInstance(getConf(),
                    "StartsWithCount-FromHBase");
       job.setJarByClass(getClass());
  ...
  ...
  ...
```

1: Re-configure Job to use HBase as Input

Set HBase ImportFormat

```
...
...
job.setInputFormatClass(TableInputFormat.class);
job.setMapperClass(StartsWithCountMapper_HBase.class);
```

Add HBase configuration and tell framework to add HBase jars to CLASSPATH

New Mapper to handle data from HBase

```
Configuration conf = job.getConfiguration();
HBaseConfiguration.merge(conf,
                   HbaseConfiguration.create(conf));
TableMapReduceUtil.addDependencyJars(job);
```

Specify table and column to read from

```
conf.set(TableInputFormat.INPUT_TABLE, TABLE_NAME);
conf.set(TableInputFormat.SCAN_COLUMNS, "count:word");
...
...
```

1: Re-configure Job to use HBase as Input

Keep output the same – same reducer, combiner an output keys

```
...
...

     // configure mapper and reducer
      job.setCombinerClass(StartsWithCountReducer.class);
      job.setReducerClass(StartsWithCountReducer.class);

      // configure output
      TextOutputFormat.setOutputPath(job, new Path(args[0]));
      job.setOutputFormatClass(TextOutputFormat.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);

      return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
      int exitCode = ToolRunner.run(new
                    StartWithCountJob_HBaseInput(), args);
      System.exit(exitCode);
  }
}
```

2: Implement a New Mapper

```
public class StartsWithCountMapper_HBase
              extends TableMapper<Text, IntWritable> {
```

Extends TableMapper which assumes types for input key and value

```
  private final static IntWritable countOne = new IntWritable(1);
  private final Text reusableText = new Text();

  @Override
  protected void map(ImmutableBytesWritable key,
              Result value, Context context)
              throws IOException, InterruptedException {

      byte[] bytes = value.getValue(toBytes(FAMILY),
                                    toBytes(COLUMN));
      String str = Bytes.toString(bytes);


      reusableText.set(str.substring(0, 1));
      context.write(reusableText, countOne);
  }
}
```

Retrieve value from input column

Run StartWithCountJob_HBaseInput

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar
mr.wordcount.StartWithCountJob_HBaseInput /training/playArea/wordCount

$ hdfs dfs -cat /training/playArea/wordCount/part-r-00000
D       1
E       1
U       1
a       1
c       1
n       1
r       2
s       1
t       1
u       1
w       1
```

Reducer's output

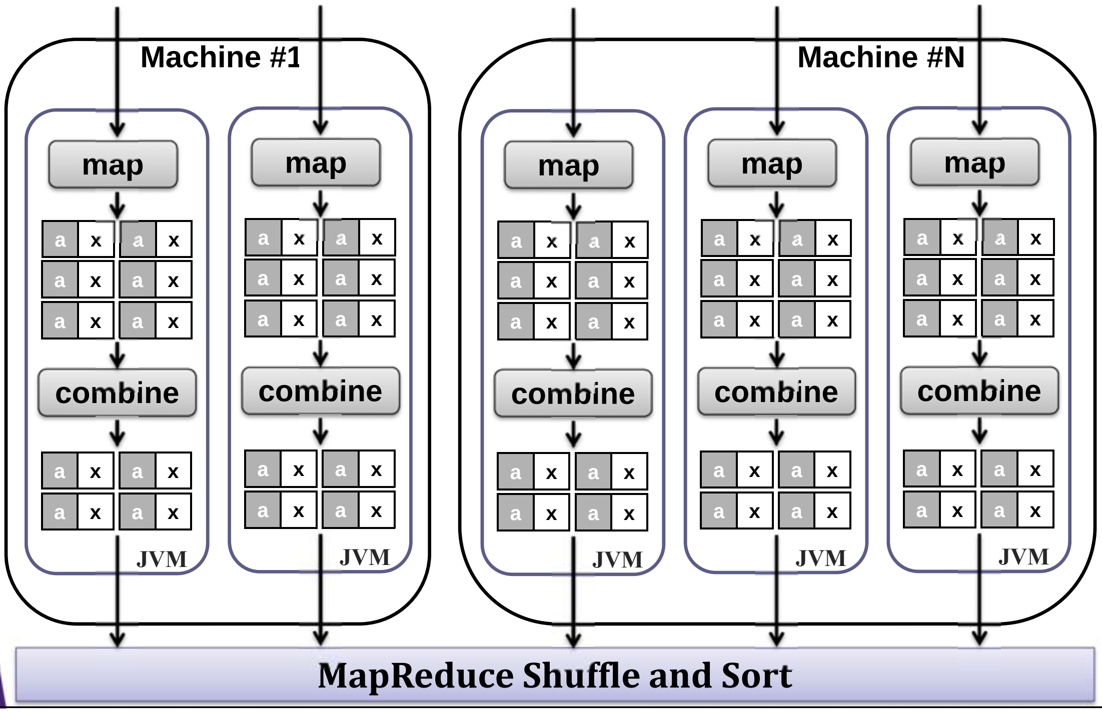
Combiner

- **Runs on output of map function**
- **Produces output for reduce function**

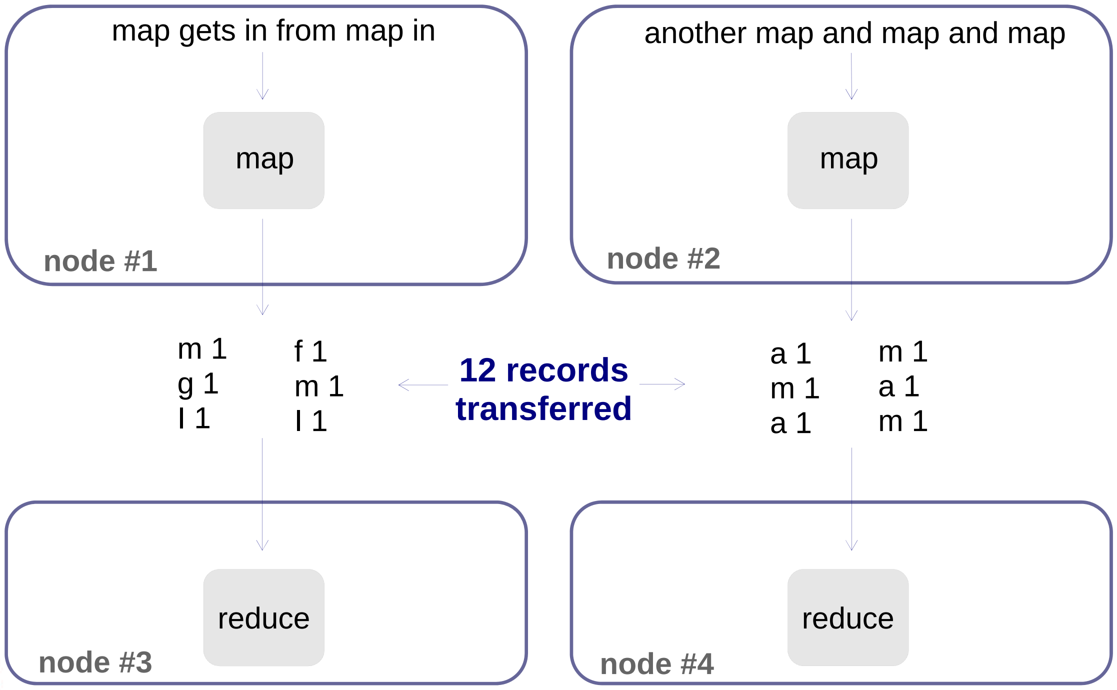
```
map: (K1,V1) → list (K2,V2)
combine: (K2,list(V2)) → list (K2,V2)
reduce: (K2,list(V2)) → list (K3,V3)
```

- **Optimization to reduce bandwidth**
  - NO guarantees on being called
  - Maybe only applied to a sub-set of map outputs
- **Often is the same class as Reducer**
- **Each combine processes output from a single split**

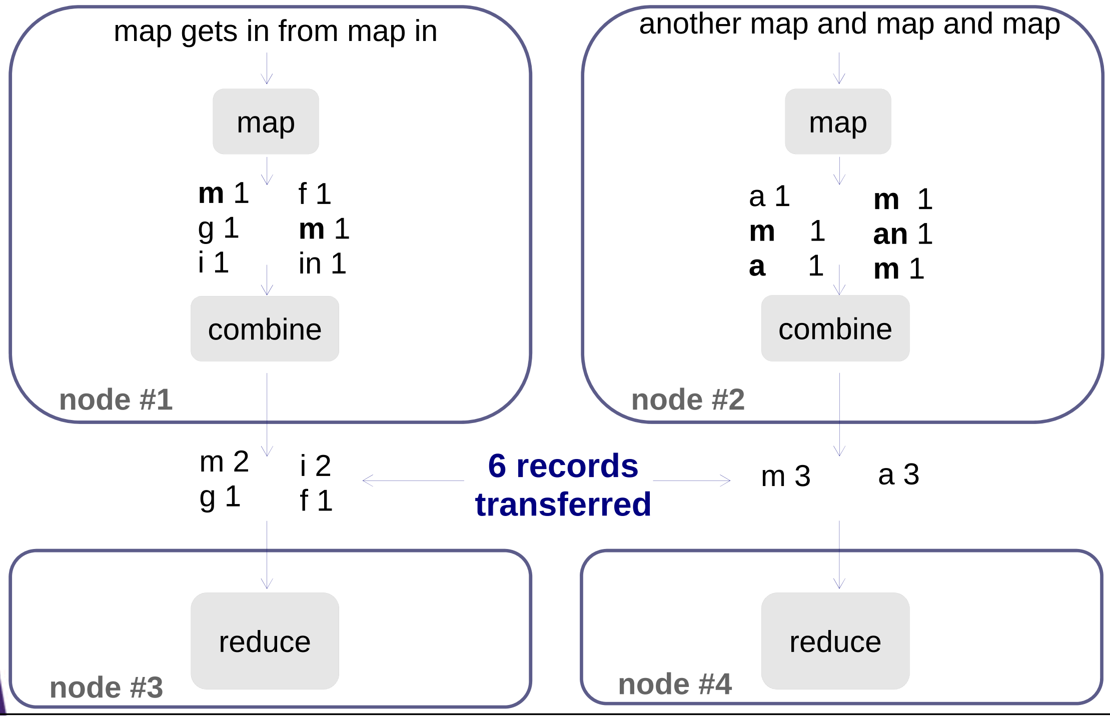
Combiner Data Flow



Sample StartsWithCountJob Run without Combiner



Sample StartsWithCountJob Run with Combiner



Specify Combiner Function

- To implement Combiner extend Reducer class
- Set combiner on Job class
  - job.setCombinerClass(StartsWithCountReducer.class);

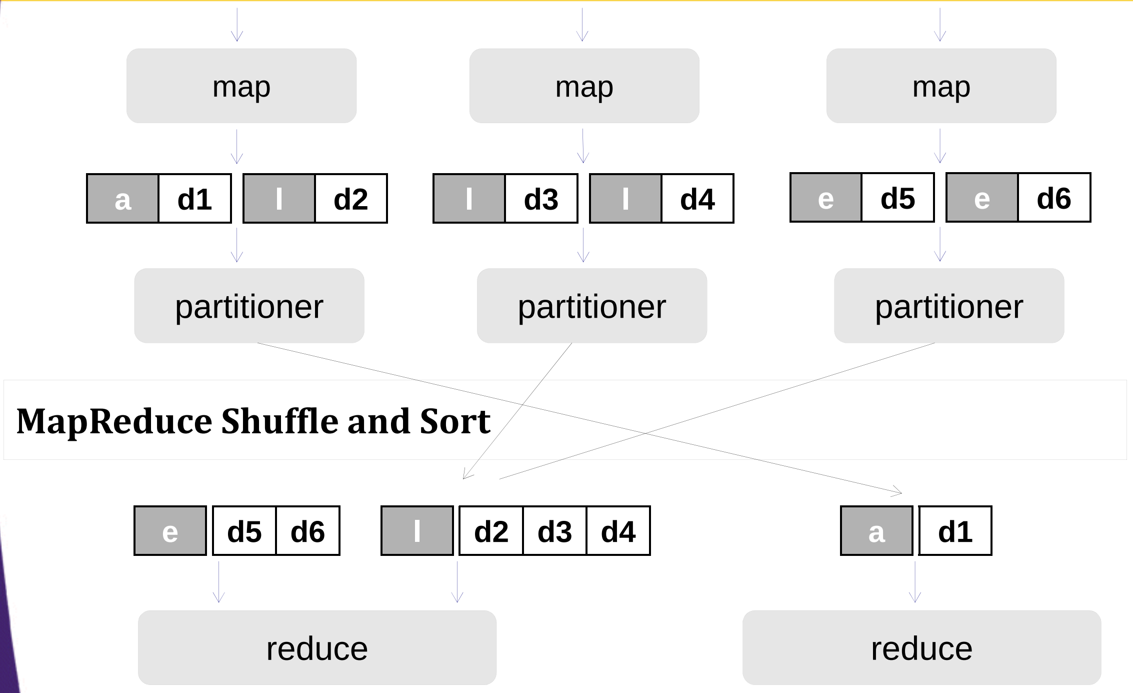
Reducer

- **Extend Reducer class**
  - Reducer<KeyIn, ValueIn, KeyOut, ValueOut>
  - KeyIn and ValueIn types must match output types of mapper
- **Receives input from mappers' output**
  - Sorted on key
  - Grouped on key of key-values produced by mappers
  - Input is directed by Partitioner implementation
- **Simple life-cycle – similar to Mapper**
  - The framework first calls setup(Context)
  - for each key → list(value) calls
    - reduce(Key, Values, Context)
  - Finally cleanup(Context) is called

Reducer

- **Can configure more than 1 reducer**
  - job.setNumReduceTasks(10);
  - mapreduce.job.reduces property
    - job.getConfiguration().setInt("mapreduce.job.reduces", 10)
- **Partitioner implementation directs key-value pairs to the proper reducer task**
  - A partition is processed by a reduce task
    - # of partitions = # or reduce tasks
  - Default strategy is to hash key to determine partition implemented by HashPartitioner<K, V>

Partitioner Data Flow



Source: Jimmy Lin, Chris Dyer. Data-Intensive Text Processing with MapReduce. Morgan & Claypool. 2010

HashPartitioner

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
  public int getPartition(K key, V value, int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```

- **Calculate Index of Partition:**
  - Convert key's hash into non-negative number
    - Logical AND with maximum integer value
  - Modulo by number of reduce tasks
- **In case of more than 1 reducer**
  - Records distributed evenly across available reduce tasks
    - Assuming a good hashCode() function
  - Records with same key will make it into the same reduce task
  - Code is independent from the # of partitions/reducers specified

Custom Partitioner

```
public class CustomPartitioner
             extends Partitioner<Text, BlogWritable>{
  @Override
  public int getPartition(Text key, BlogWritable blog,
             int numReduceTasks) {
      int positiveHash =
      blog.getAuthor().hashCode()& Integer.MAX_VALUE;
```

Use author's hash only, AND with max integer to get a positive value

```
      return positiveHash % numReduceTasks;
  }
}
```

Assign a reduce task by index

- **All blogs with the same author will end up in the same reduce task**

OutputFormat

- **Specification for writing data**
  - The other side of InputFormat
- **Implementation of OutputFormat<K,V>**
- **TextOutputFormat is the default implementation**
  - Output records as lines of text
  - Key and values are tab separated "Key /t value"
    - Can be configured via "mapreduce.output.textoutputformat.separator" property
  - Key and Value may of any type - call .toString()

OutputFormat

- **Validates output specification for that job**
  - You may have seen annoying messages that output directory already exists
- **Creates implementation of RecordWriter**
  - Responsible for actually writing data
- **Creates implementation of OutputCommitter**
  - Set-up and clean-up Job's and Tasks' artifacts (ex. Directories)
  - Commit or discard tasks output

OutputFormat Interface

```
public abstract class OutputFormat<K, V> {

  public abstract RecordWriter<K, V>
    getRecordWriter(TaskAttemptContext context
                 ) throws IOException, InterruptedException;
```

Provide implementation of RecordReader

```
  public abstract void checkOutputSpecs(JobContext context
                 ) throws IOException,InterruptedException;
```

Validates output specification for that job

Provide implementation of OutputCommitter

```
  public abstract
  OutputCommitter getOutputCommitter(TaskAttemptContext context
                 ) throws IOException, InterruptedException;

}
```

Hadoop's OutputFormats

- **Hadoop eco-system is packaged with many OutputFormats**
  - TextOutputFormat
  - DBOutputFormat
  - TableOutputFormat (HBASE)
  - MapFileOutputFormat
  - SequenceFileOutputFormat
  - NullOutputFormat
  - Etc...
- **Configure on Job object**
  - job.setOutputFormatClass(XXXOutputFormat.class);
  - job.setOutputKeyClass(XXXKey.class);
  - job.setOutputValueClass(XXXValue.class);

TextOutputFormat

- **Outputs plain text**
- **Saves key-value pairs separated by tab**
  - Configured via mapreduce.output.textoutputformat.separator property
- **Set output path**
  - TextOutputFormat.setOutputPath(job, new Path(myPath));

TableOutputFormat

- Saves data into HTable
- Reducer output key is ignored
- Reducer output value must be HBase's Put or Delete objects

TableOutputFormat

```
public class StartWithCountJob_HBase
                    extends Configured implements Tool {

  protected final static String TABLE_NAME = "HBaseSamples";
  protected final static String FAMILY = "count";
  protected final static String INPUT_COLUMN = "word";

  protected final static String RESULT_COLUMN = "result";
```

Input and output column resides
Under the same family (doesn't have to)

```
  @Override
  public int run(String[] args) throws Exception {
    Job job =
       Job.getInstance(getConf(), "StartsWithCount-HBase");
       job.setJarByClass(getClass());
...
...
```

Name the job

TableOutputFormat

```
    Scan scan = new Scan();
    scan.addColumn(toBytes(FAMILY), toBytes(INPUT_COLUMN));
    TableMapReduceUtil.initTableMapperJob(
            TABLE_NAME,
            scan,
            StartsWithCountMapper_HBase.class,
            Text.class,
            IntWritable.class,
            job);

    TableMapReduceUtil.initTableReducerJob(
            TABLE_NAME,
            StartsWithCountReducer_HBase.class,
            job);
    job.setNumReduceTasks(1);
    return job.waitForCompletion(true) ? 0 : 1;
  }
  public static void main(String[] a) throws Exception {
    int code = ToolRunner.run(new StartWithCountJob_HBase(), a);
    System.exit(code);
  }
}
```

Utilize HBase's TableMapReduceUtil to setup the job; internally delegate to job.XXX methods

TableOutputFormat

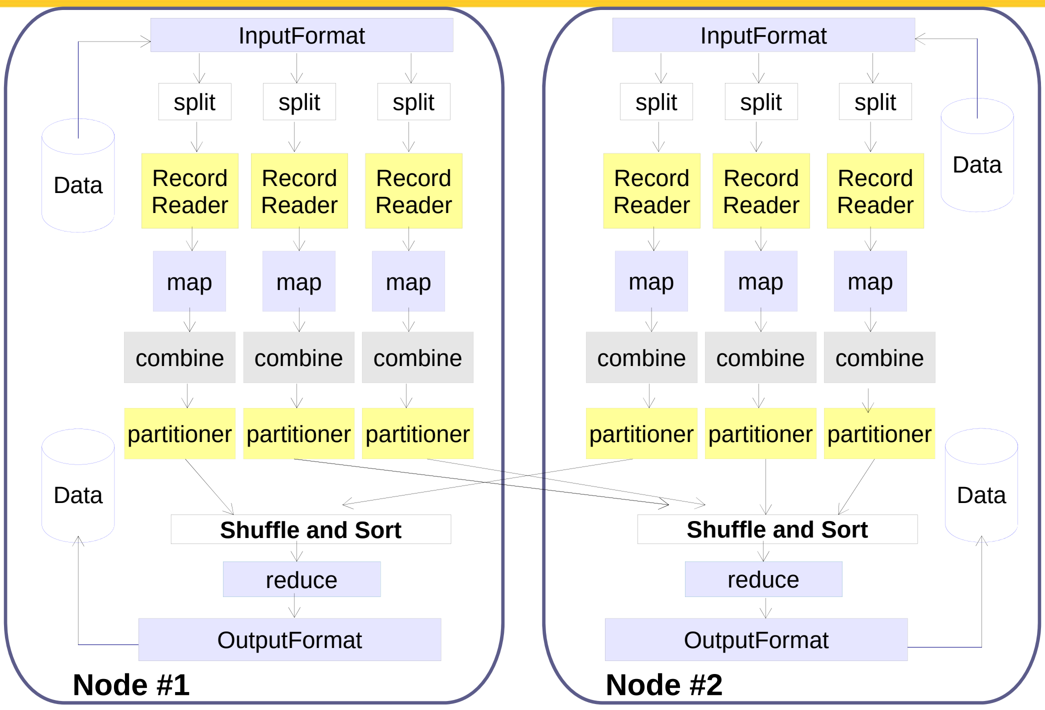
```
public class StartsWithCountReducer_HBase extends
        TableReducer<Text, IntWritable, ImmutableBytesWritable> {

  @Override
  protected void reduce(Text key, Iterable<IntWritable> counts,
        Context context)
            throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable count : counts) {
            sum+= count.get();
        }

        Put put = new Put(key.copyBytes());
        put.add(toBytes(FAMILY), toBytes(RESULT_COLUMN),
                        toBytes(Integer.toString(sum)));
        context.write(null, put);
  }
}
```

Reducer must output either Put or Delete object

TableOutputFormat

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar mr.wordcount.StartWithCountJob_HBase
$ hbase shell
hbase(main):029:0> scan 'HBaseSamples', {COLUMN=>'count:result'}
ROW                    COLUMN+CELL
 D                     column=count:result, timestamp=1338951024384, value=1
 E                     column=count:result, timestamp=1338951024384, value=1
 U                     column=count:result, timestamp=1338951024384, value=1
 a                     column=count:result, timestamp=1338951024384, value=1
 c                     column=count:result, timestamp=1338951024386, value=1
 n                     column=count:result, timestamp=1338951024386, value=1
 r                     column=count:result, timestamp=1338951024386, value=2
 s                     column=count:result, timestamp=1338951024386, value=1
 t                     column=count:result, timestamp=1338951024386, value=1
 u                     column=count:result, timestamp=1338951024386, value=1
 w                     column=count:result, timestamp=1338951024386, value=1
12 row(s) in 0.0530 seconds
```

Component Overview



Source: Yahoo! Inc. "Hadoop Tutorial from Yahoo!". 2012

Wrap-Up

Summary

- **In this lecture we learned about**
  - MapReduce Theory
  - Types of Keys and Values
  - Input and Output Formats
  - Anatomy of
    - Mappers
    - Reducers
    - Combiners
    - Partitioners

Questions?