UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

451/651

Part 1: MapReduce Algorithm Design
Hadoop API

Ali Abedi

MapReduce API

Mapper<K_{in},V_{in},K_{out},V_{out}>

void setup(Mapper.Context context)

Called once at the start of the task

void map(K_{in} key, V_{in} value, Mapper.Context context)

Called once for each key/value pair in the input split

void cleanup(Mapper.Context context)

Called once at the end of the task

Reducer<K_{in},V_{in},K_{out},V_{out}>/Combiner<K_{in},V_{in},K_{out},V_{out}>

void setup(Reducer.Context context)

Called once at the start of the task

void reduce(K_{in} key, Iterable<V_{in}> values, Reducer.Context context)

Called once for each key

void cleanup(Reducer.Context context)

Called once at the end of the task

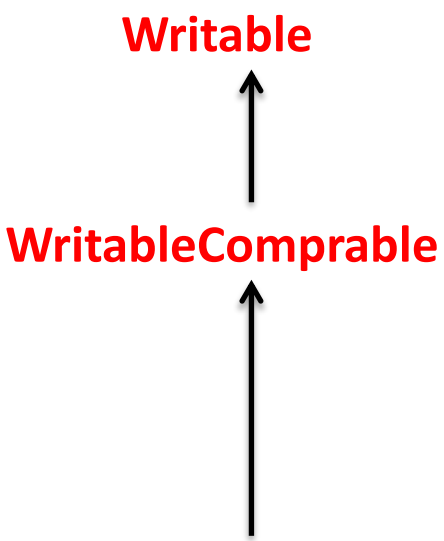
MapReduce API

Partitioner<K, V>

int getPartition(K key, V value, int numPartitions)

Returns the partition number given total number of partitions

Data Types in Hadoop: Keys and Values



Defines a de/serialization protocol.
Every data type in Hadoop is a Writable.

Defines a sort order.
All keys must be of this type (but not values).

IntWritable
LongWritable
Text
…

Concrete classes for different data types.
Note that these are container objects.

SequenceFile

Binary-encoded sequence of key/value pairs.

Word Count Mapper

```
private static final class MyMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable ONE = new IntWritable(1);
  private final static Text WORD = new Text();

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
    for (String word : Tokenizer.tokenize(value.toString())) {
      WORD.set(word);
      context.write(WORD, ONE);
    }
  }
}
```

Word Count Reducer

```
private static final class MyReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  private final static IntWritable SUM = new IntWritable();

  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
      Context context) throws IOException, InterruptedException {
    Iterator<IntWritable> iter = values.iterator();
    int sum = 0;
    while (iter.hasNext()) {
      sum += iter.next().get();
    }
    SUM.set(sum);
    context.write(key, SUM);
  }
}
```

Getting Data to Mappers and Reducers

Configuration parameters

Pass in via Job configuration object

"Side data"

DistributedCache

Mappers/Reducers can read from HDFS in setup method

Complex Data Types in Hadoop

How do you implement complex data types?

The easiest way:

Encode it as Text, e.g., (a, b) = “a:b”

Use regular expressions to parse and extract data

Works, but janky

The hard way:

Define a custom implementation of Writable(Comprable)

Must implement: readFields, write, (compareTo)

Computationally efficient, but slow for rapid prototyping

Implement WritableComparator hook for performance

Somewhere in the middle:

Bespin offers various building blocks

Input and Output

InputFormat

TextInputFormat

KeyValueTextInputFormat

SequenceFileInputFormat

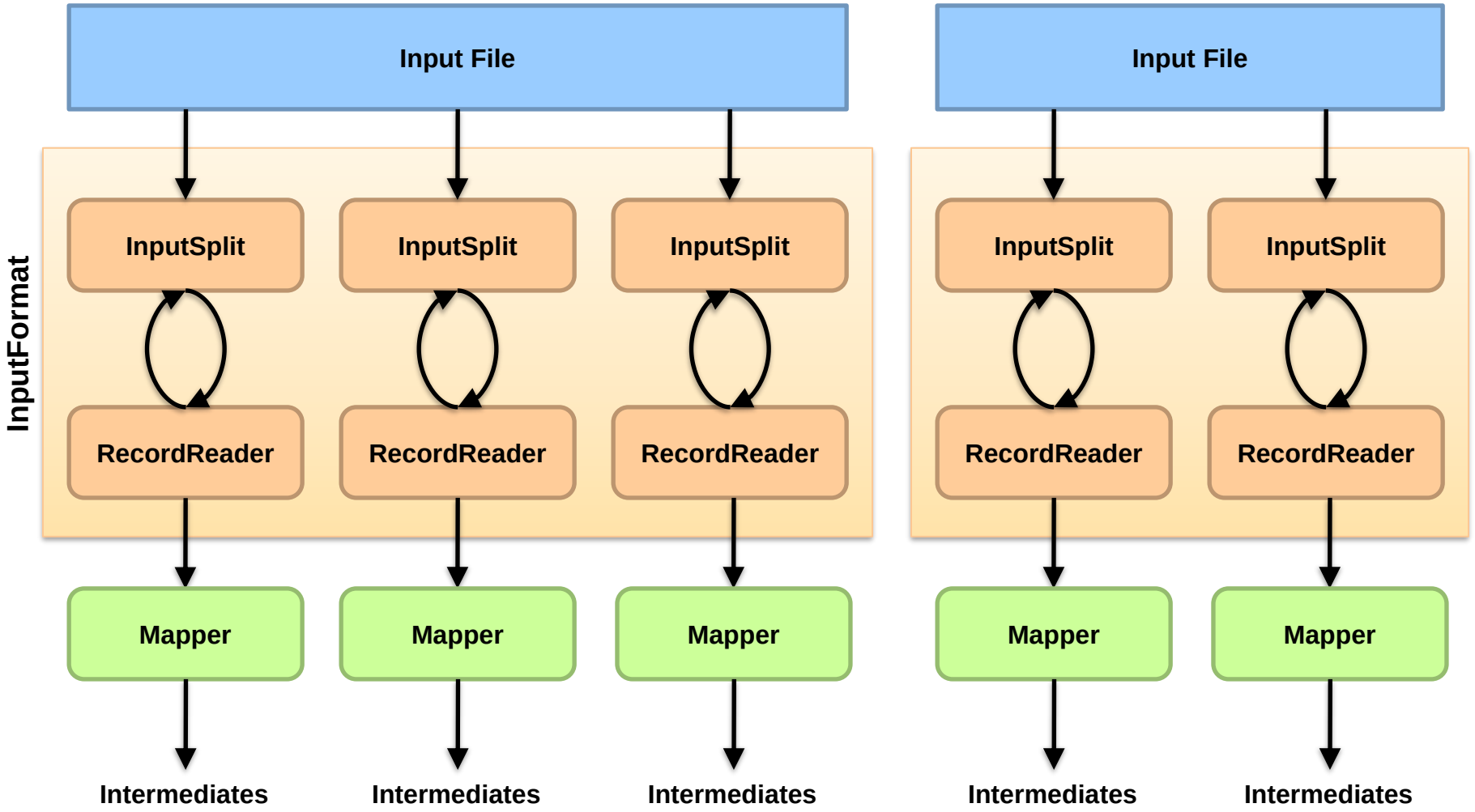
…

OutputFormat

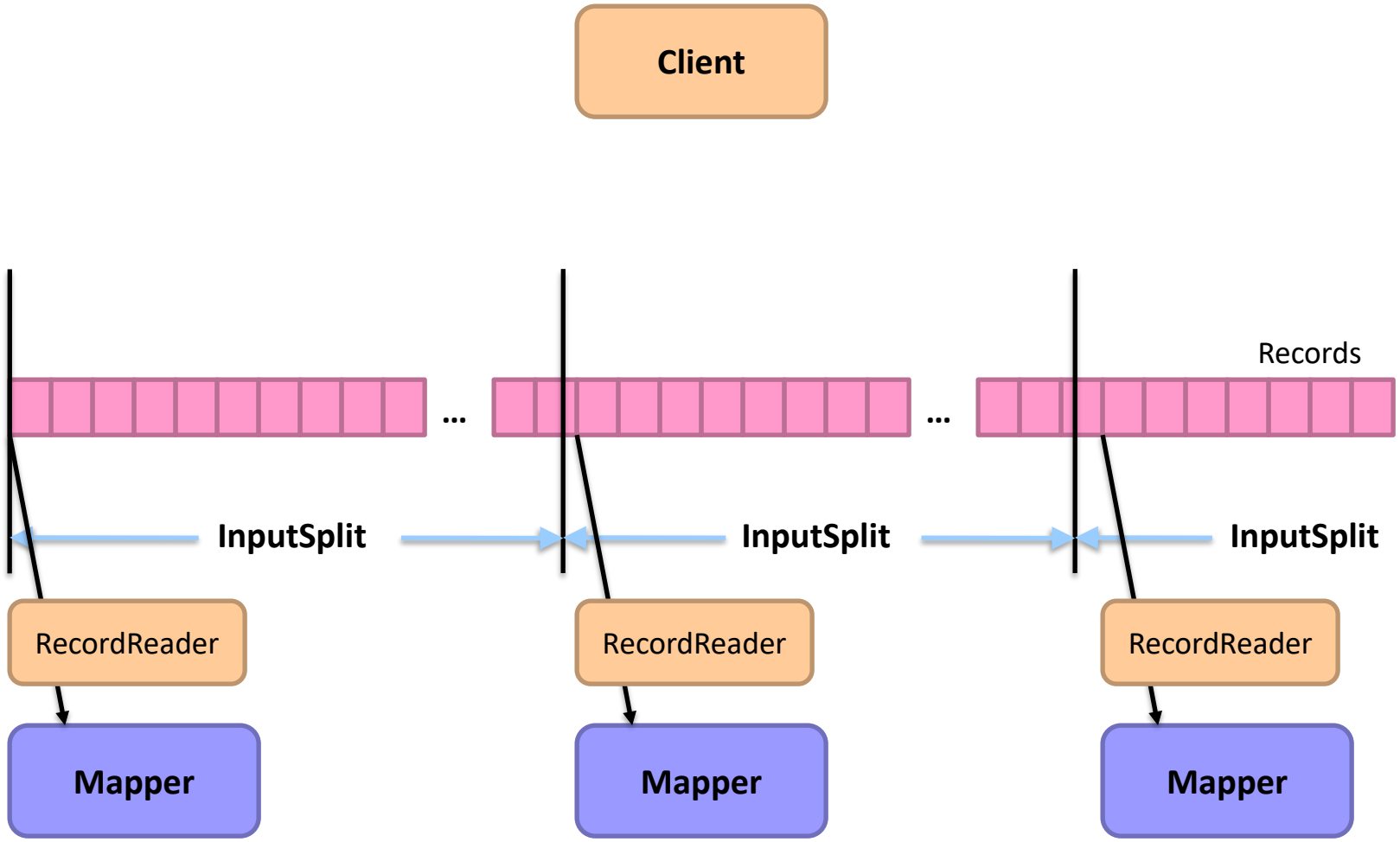
TextOutputFormat

SequenceFileOutputFormat

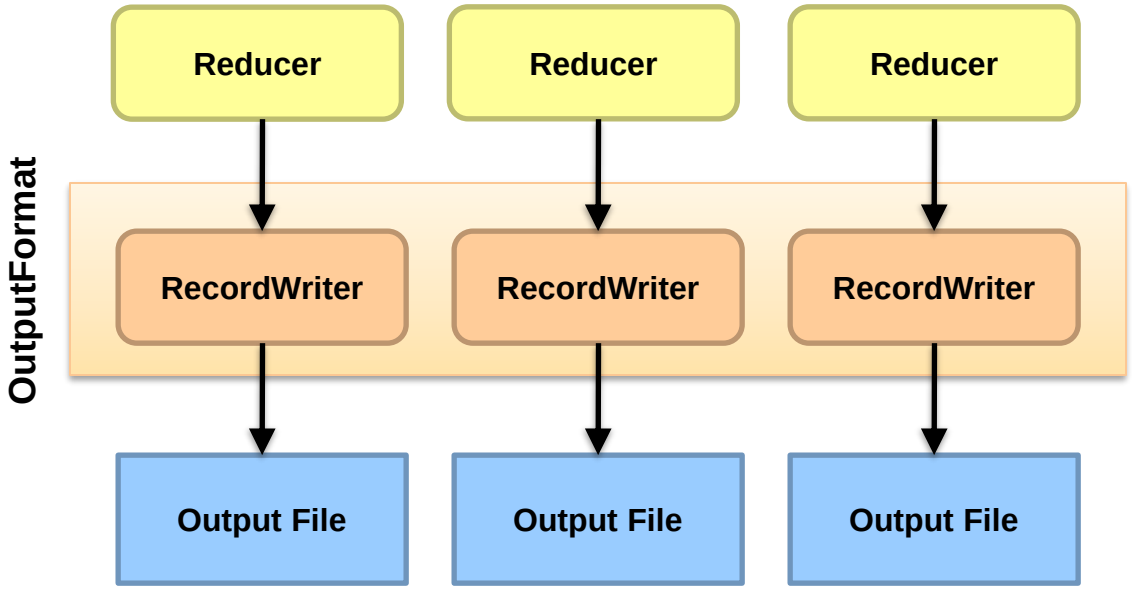
…



Source: redrawn from a slide by Cloduera, cc-licensed



Where's the data actually coming from?



Source: redrawn from a slide by Cloduera, cc-licensed

Hadoop Job

Represents a packaged Hadoop job for submission to cluster
Need to specify input and output paths
Need to specify input and output formats
Need to specify mapper, reducer, combiner, partitioner classes
Need to specify intermediate/final key/value classes
Need to specify number of reducers (but not mappers, why?)
Don’t depend on defaults!

Hadoop Workflow

You

Submit node
(datasci)

Hadoop Cluster

Getting data in?
Writing code?
Getting data out?

Where's the actual
data stored?

Debugging Hadoop

First, take a deep breath
Start small, start locally
Build incrementally

Hadoop Debugging Strategies

Good ol' System.out.println

Learn to use the webapp to access logs

Logging preferred over System.out.println

Be careful how much you log!

Fail on success

Throw RuntimeExceptions and capture state

Use Hadoop as the "glue"

Implement core functionality outside mappers and reducers

Independently test (e.g., unit testing)

Compose (tested) components in mappers and reducers