Уральский федеральный университет

Параллельные вычисления

MapReduce и Hadoop
Часть 2

Созыкин Андрей Владимирович

к.т.н.
зав.кафедрой высокопроизводительных компьютерных технологий

Литература по Hadoop

- *Tom White*. **Hadoop: The Definitive Guide**, 4th Edition
- *Alex Holmes*. **Hadoop in Practice**, Second Edition
- *Eric Sammer*. **Hadoop Operations**.
- *Jimmy Lin and Chris Dyer*. **Data-Intensive Text Processing with MapReduce**

Режимы работы Hadoop

- Локальный
  - Один процесс на одном компьютере
  - Mapper и Reducer в виде потоков
- Псевдо-распределенный
  - Все процессы Hadoop работают на одном компьютере
  - Работают демоны Hadoop (NameNode, DataNode, ResourceManager, NodeManager и др.)
  - Mapper и Reducer работают в отдельных процессах
- Распределенный
  - Процессы Hadoop работают на нескольких компьютерах
  - Работают демоны Hadoop (NameNode, DataNode, ResourceManager, NodeManager и др.)
  - Mapper и Reducer работают в отдельных процессах на разных компьютерах

Учебный кластер Hadoop

- 8 узлов
  - 1 управляющий узел
  - 7 рабочих узлов
- Управляющий узел
  - NameNode (HDFS)
  - ResourceManager (MapReduce)
- Рабочие узлы
  - DataNode (HDFS)
  - NodeManager (MapReduce)

Дистрибутивы и установка Hadoop

- Дистрибутив Apache
  - apache.hadoop.org
- Альтернативные дистрибутивы
  - Cloudera (используется на кластере, где будут выполняться домашние задания)
  - Hortonworks
  - MapR
- Локальная установка
  - Дистрибутив Cloudera (RHEL, Ubuntu, SLES): http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_qs_yarn_pseudo.html
  - Для Windows Hortonworks

Разработка и запуск программ MapReduce

- Реализовать Maper, Reducer и Driver
- Скомпилировать их и упаковать в Jar-архив
- Скопировать данные для обработки в HDFS
- Запустить MapReduce задачу
- Обработанные данные будут записаны в HDFS

WordCount: Mapper

```
public class WordCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context
        context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

WordCount: Reducer

```
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

WordCount: Driver

```
public class WordCount extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new WordCount(), args);
    System.exit(exitCode);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(super.getConf(), "WordCount");
    job.setJarByClass(getClass());
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

Компиляция и упаковка в jar

- Компиляция

```
javac -cp "`hadoop classpath`" *.java
```

- Упаковка в Jar-архив

```
jar cvf wordcount.jar *.class
```

Подготовка данных

- Демонстрационный текстовый файл

```
$ cat hadoop.txt
What Is Apache Hadoop?
The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing.
```

- Копируем файл в HDFS

```
$ hdfs dfs -put hadoop.txt input
```

Запуск задачи Hadoop

```
$ hadoop jar target/wordcount-1.jar parcourse.WordCountDriver input output
15/04/25 17:58:27 INFO client.RMProxy: Connecting to ResourceManager at
umu30.um64.imm.uran.ru/192.168.1.30:8032
15/04/25 17:58:29 INFO input.FileInputFormat: Total input paths to process : 1
15/04/25 17:58:29 INFO mapreduce.JobSubmitter: number of splits:1
15/04/25 17:58:29 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1427888786712_0016
15/04/25 17:58:30 INFO impl.YarnClientImpl: Submitted application application_1427888786712_0016
15/04/25 17:58:30 INFO mapreduce.Job: The url to track the job:
http://umu30.um64.imm.uran.ru:8088/proxy/application_1427888786712_0016/
15/04/25 17:58:30 INFO mapreduce.Job: Running job: job_1427888786712_0016
15/04/25 17:58:45 INFO mapreduce.Job: Job job_1427888786712_0016 running in uber mode : false
15/04/25 17:58:45 INFO mapreduce.Job:  map 0% reduce 0%
15/04/25 17:58:54 INFO mapreduce.Job:  map 100% reduce 0%
15/04/25 17:59:07 INFO mapreduce.Job:  map 100% reduce 15%
15/04/25 17:59:08 INFO mapreduce.Job:  map 100% reduce 23%
15/04/25 17:59:09 INFO mapreduce.Job:  map 100% reduce 85%
15/04/25 17:59:10 INFO mapreduce.Job:  map 100% reduce 100%
15/04/25 17:59:11 INFO mapreduce.Job: Job job_1427888786712_0016 completed successfully
15/04/25 17:59:11 INFO mapreduce.Job: Counters: 49
 File System Counters
                FILE: Number of bytes read=2770
                FILE: Number of bytes written=1496330
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
...
```

Результаты работы задачи

```
$hdfs dfs -ls output1
Found 14 items
-rw-r--r--   3 u1213 supergroup          0 2015-04-25 17:59 output1/_SUCCESS
-rw-r--r--   3 u1213 supergroup        181 2015-04-25 17:59 output1/part-r-00000
-rw-r--r--   3 u1213 supergroup        139 2015-04-25 17:59 output1/part-r-00001
-rw-r--r--   3 u1213 supergroup        145 2015-04-25 17:59 output1/part-r-00002
-rw-r--r--   3 u1213 supergroup        164 2015-04-25 17:59 output1/part-r-00003
-rw-r--r--   3 u1213 supergroup        151 2015-04-25 17:59 output1/part-r-00004
-rw-r--r--   3 u1213 supergroup        137 2015-04-25 17:59 output1/part-r-00005
-rw-r--r--   3 u1213 supergroup        118 2015-04-25 17:59 output1/part-r-00006
-rw-r--r--   3 u1213 supergroup        161 2015-04-25 17:59 output1/part-r-00007
-rw-r--r--   3 u1213 supergroup         52 2015-04-25 17:59 output1/part-r-00008
-rw-r--r--   3 u1213 supergroup        139 2015-04-25 17:59 output1/part-r-00009
-rw-r--r--   3 u1213 supergroup        170 2015-04-25 17:59 output1/part-r-00010
-rw-r--r--   3 u1213 supergroup        221 2015-04-25 17:59 output1/part-r-00011
-rw-r--r--   3 u1213 supergroup        140 2015-04-25 17:59 output1/part-r-00012
```

Результаты работы задачи

```
$ hdfs dfs -cat output1/part-r-00000
(HDFS): 1
Cassandra:      1
HBase:  1
computation.    1
diagnose        1
failures.       1
features        1
hardware        1
large   4
layer,  1
library 2
local   1
monitoring      1
open-source     1
scale   1
scheduling      1
top     1
```

Компиляция и упаковка в jar: Maven

- Maven – средство для автоматизации сборки Java программ
  - http://maven.apache.org/
- Описание процесса сборки
  - pom.xml
- Maven target (цели)

  `compile` - компиляция

  `test` – запуск тестов

  `package` – подготовка дистрибутива для распространения (по умолчанию jar-архив)
- Пример:

```
mvn package
```

pom.xml: Репозиторий Cloudera

```
<repositories>
    <repository>
        <id>cloudera</id>
<url>https://repository.cloudera.com/artifactory/clo
udera-repos/</url>
    </repository>
</repositories>
```

Using the CDH 5 Maven Repository
http://www.cloudera.com/content/cloudera/en/documentation/core/v5-2-x/topics/cdh_vd_cdh5_maven_repo.html

pom.xml: Зависимость Hadoop

```
<dependencies>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>2.0.0-cdh4.6.0</version>
    </dependency>
</dependencies>
```

Using the CDH 5 Maven Repository
http://www.cloudera.com/content/cloudera/en/documentation/core/v5-2-x/topics/cdh_vd_cdh5_maven_repo.html

Представление данных в Hadoop

- Пары
  - Ключ
  - Значение
- Внутренние операции с данными
  - Передача по сети (от сервера Map к серверу Reduce)
  - Запись на диск (промежуточный буфер)
- Требования к данным
  - Возможность сериализации
  - Сравнение в сериализованном виде (только для ключей)

Типы сериализации в Hadoop

- Можно ли использовать стандартную сериализацию Java?

Типы сериализации в Hadoop

- Встроенный в Hadoop интерфейс Writable
- Apache Avro
- Apache Thrift

Интерфейс Writable

- Основные методы:
  - `public void write(DataOutput out)`
  - `public void readFields(DataInput in)`
- Существующие классы
  - `IntWritable`
  - `LongWritable`
  - `DoubleWritable`
  - `NullWritable (singleton)`
  - `Text (Writable для String)`
  - `BytesWritable`
  - `ArrayWritable`

Ключи: интерфейс WritableComparable

- Ключи в Hadoop нужно не только записывать в поток, но и сравнивать между собой
  - Ключи должны реализовывать интерфейс Comparable
  - Специальный интерфейс в Hadoop: WritableComparable
- Методы WritableComparable:
  - `public void write(DataOutput out)`
  - `public void readFields(DataInput in)`
  - `public int compareTo(MyWritableComparable o)`
- Существующие классы:
  - `IntWritable, LongWritable, DoubleWritable, NullWritable, Text, BytesWritable, ArrayWritable`

Компаратор RawComparator

- Сортировка данных в бинарном виде
  - Оптимизация производительности
  - Не нужно преобразовывать в объекты и обратно для сортировки по ключам
- Класс RawComparator
  - Расширение стандартного Comparator Java
  - `public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2);`
  - `b1, b2` – массивы байтов для сравнения
  - `s1, s2` – начальные позиции для сравнения
  - `l1, l2` – количество байт для сравнения

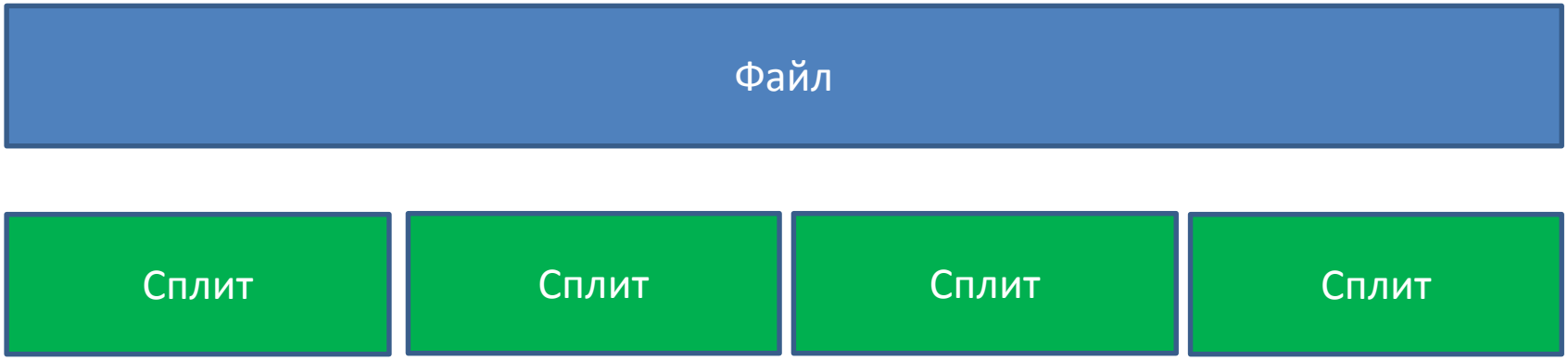
WordCount: Mapper

```
public class WordCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context
        context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

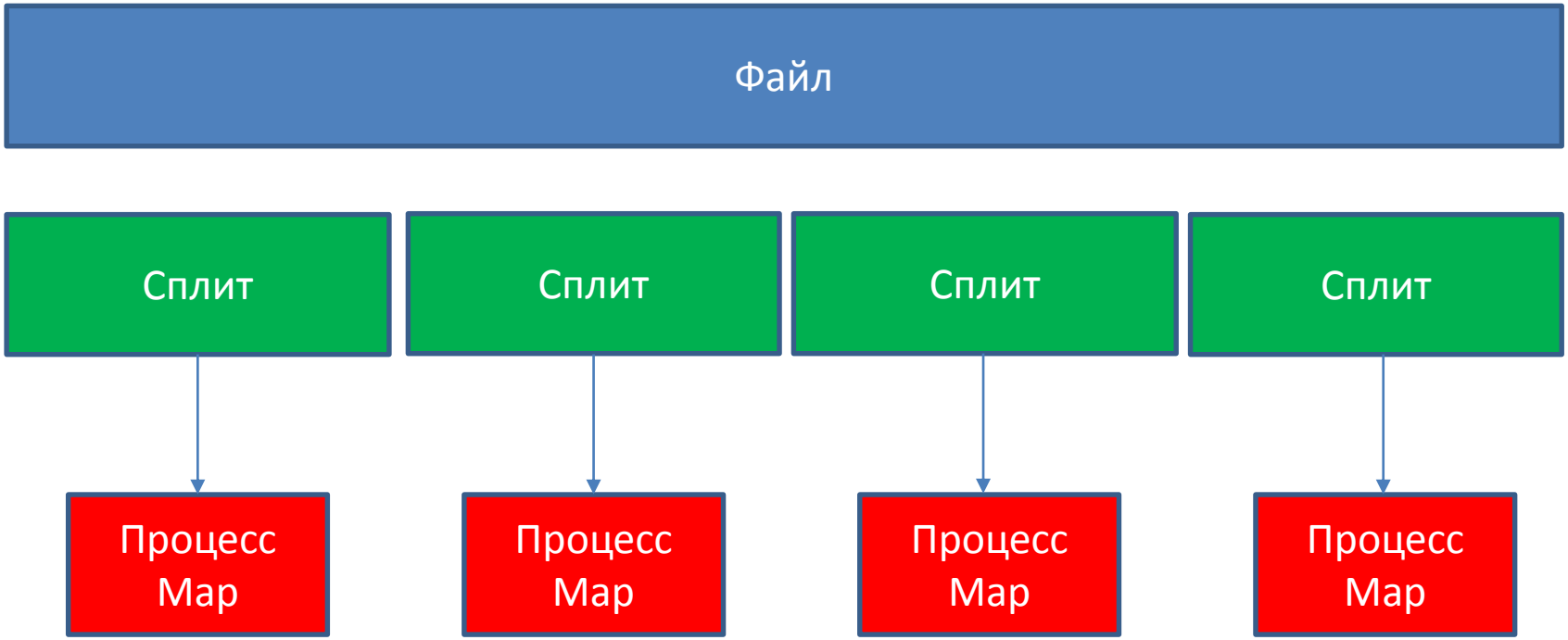
Работа с данными в Hadoop

Файл

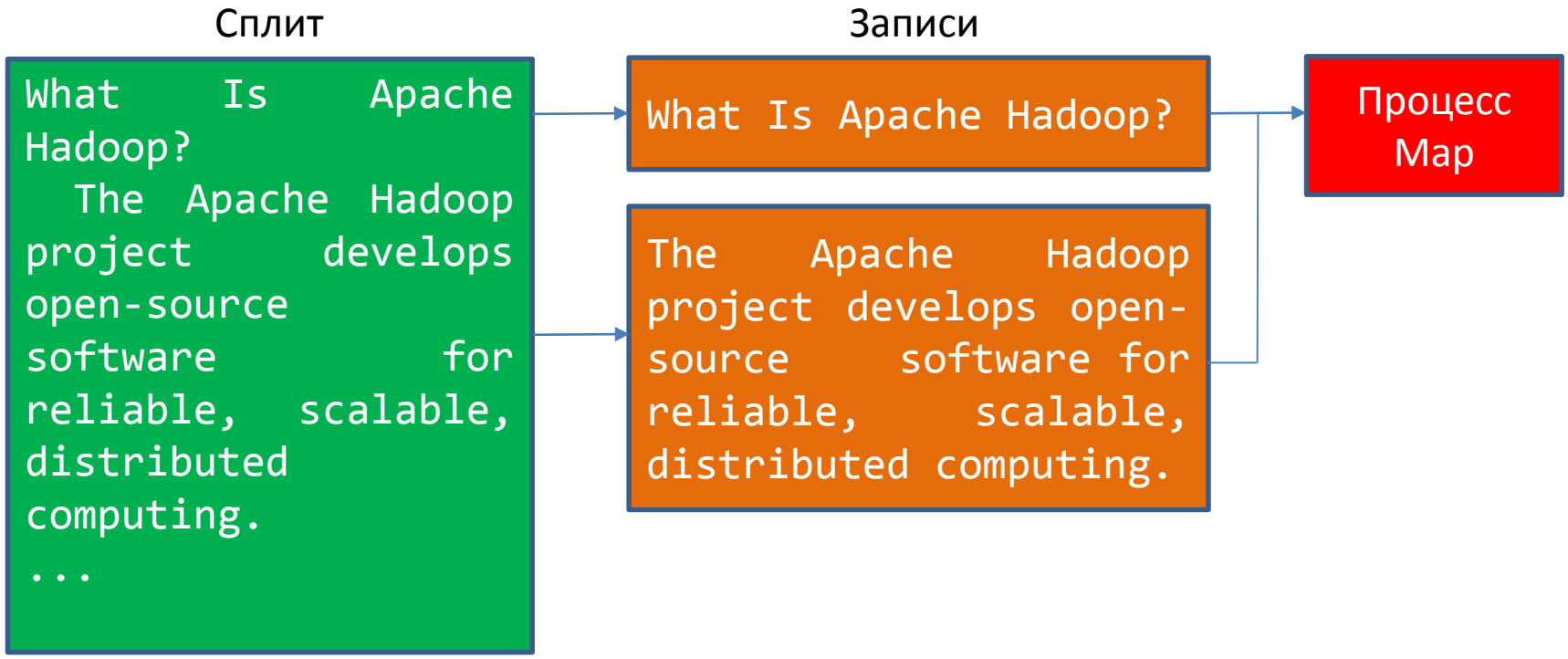
Работа с данными в Hadoop



Работа с данными в Hadoop



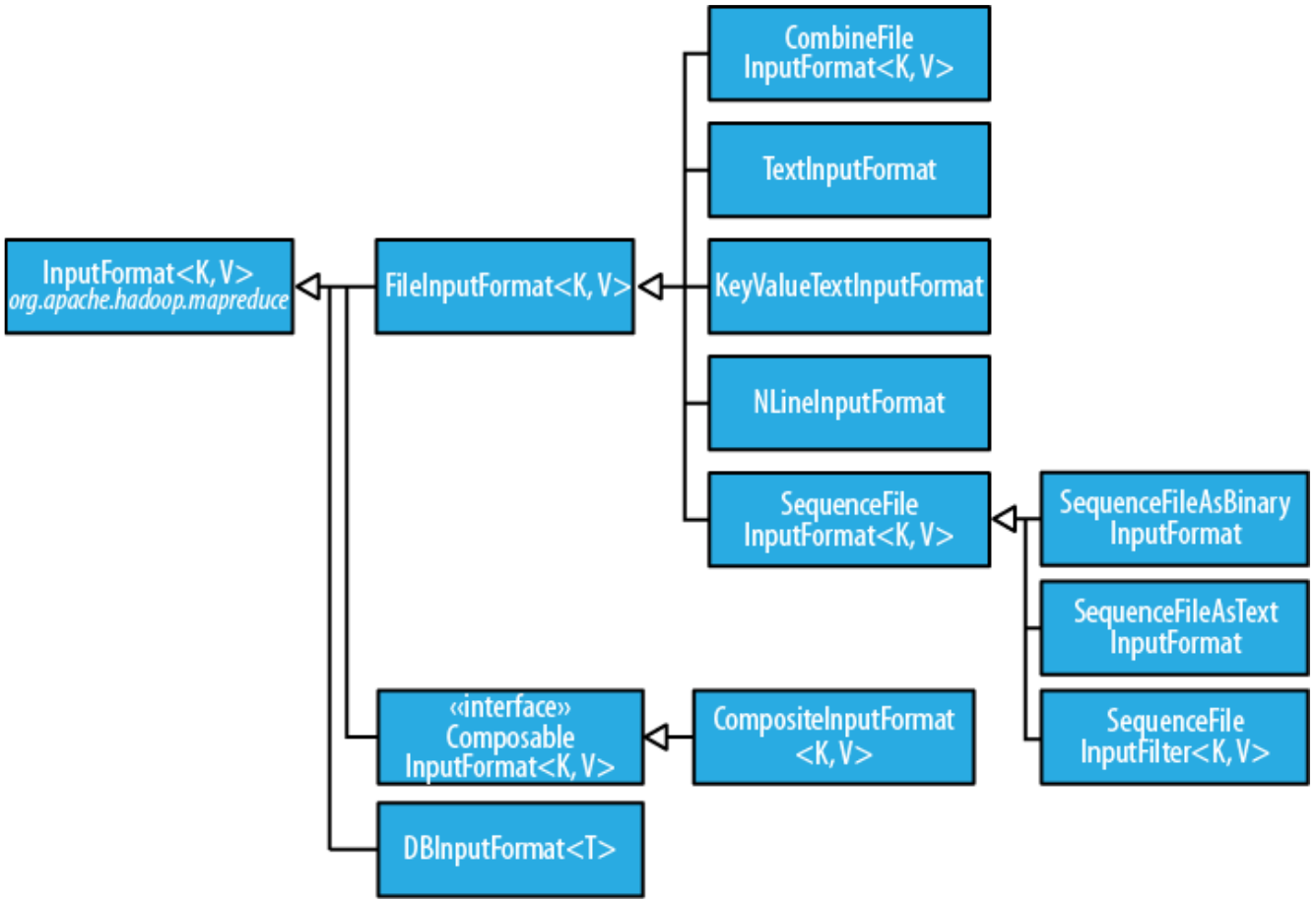
Работа с данными в Hadoop



Ввод-вывод данных в Hadoop

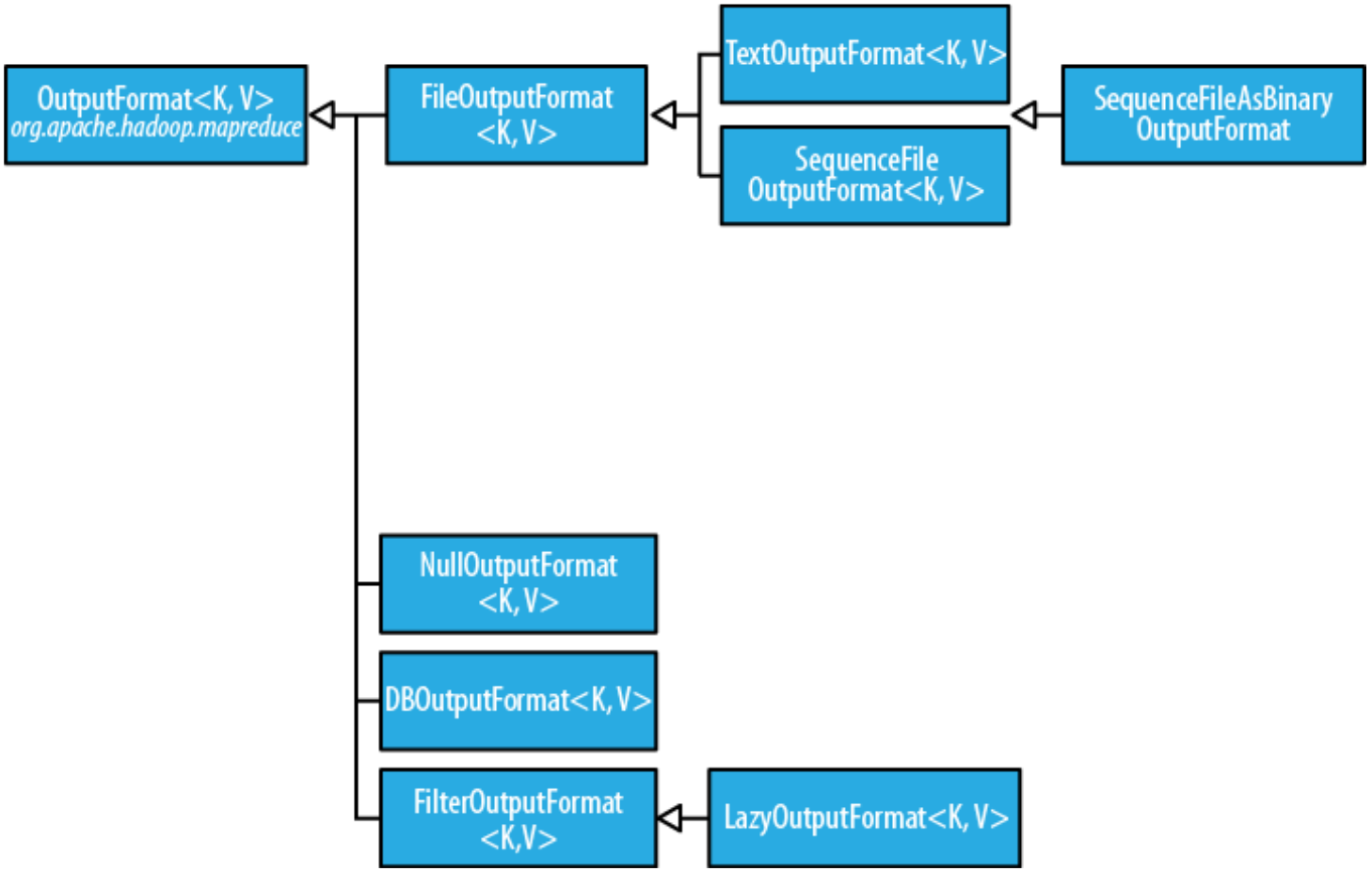
- Ввод-вывод данных в Hadoop полностью автоматизирован
- InputFormat
  - Определяет, как будут читаться данные
  - Какие типы ключ-значение будут генерироваться
  - Предоставляет RecordReader
- OutputFormat
  - Определяет, как будут записываться данные
  - Какие типы ключ-значение будут записываться
  - Предоставляет RecordWriter

Готовые InputFormat



Tom White. Hadoop: The Definitive Guide

Готовые OutputFormat



Tom White. Hadoop: The Definitive Guide

Установка I/OFormat в драйвере

```
public class WordCount extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new WordCount(), args);
    System.exit(exitCode);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(super.getConf(), "WordCount");
    job.setJarByClass(getClass());
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

Combiner

- Оптимизация производительности
  - “Локальный” Reducer
- Выходные данные от Map в wordcount:
  - <What, 1>, <Is, 1>, <Apache, 1>, <Hadoop, 1>, <The, 1>, <Apache, 1>, <Hadoop, 1>, <project, 1>, <develops, 1>…
- Выходные данные после обработки Combiner:
  - <What, 1>, <Is, 1>, <Apache, 2>, <Hadoop, 2>, <The, 1>, <project, 1>, <develops, 1>…

Требования к Combiner

- Такие же типы выходных ключей и значений, как и у Map
- Функция в Combiner:
  - Коммутативная
  - Ассоциативная
  - Может использоваться Reduce (если функция коммутативная и ассоциативная)
- Особенность работы Combiner:
  - Не гарантируется, что он будет вызван
  - Может быть вызван несколько раз (в том числе на стороне Reduce)

Установка Combiner в Driver

```
public class WordCount extends Configured implements Tool {
  ...
  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(super.getConf(), "WordCount");
    job.setJarByClass(getClass());
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setCombinerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

Combiner для вычисления среднего значения

- Можно ли сделать?

Combiner для вычисления среднего значения

- Вычисление среднего значения не ассоциативно
- Собственный тип значения
  - Частичная сумма
  - Количество элементов
  - Реализует интерфейс Writable
- Combiner
  - Отличается от Reducer
  - Считает частичную сумму и количество элементов
- Reducer
  - Вычисляет итоговое среднее значение

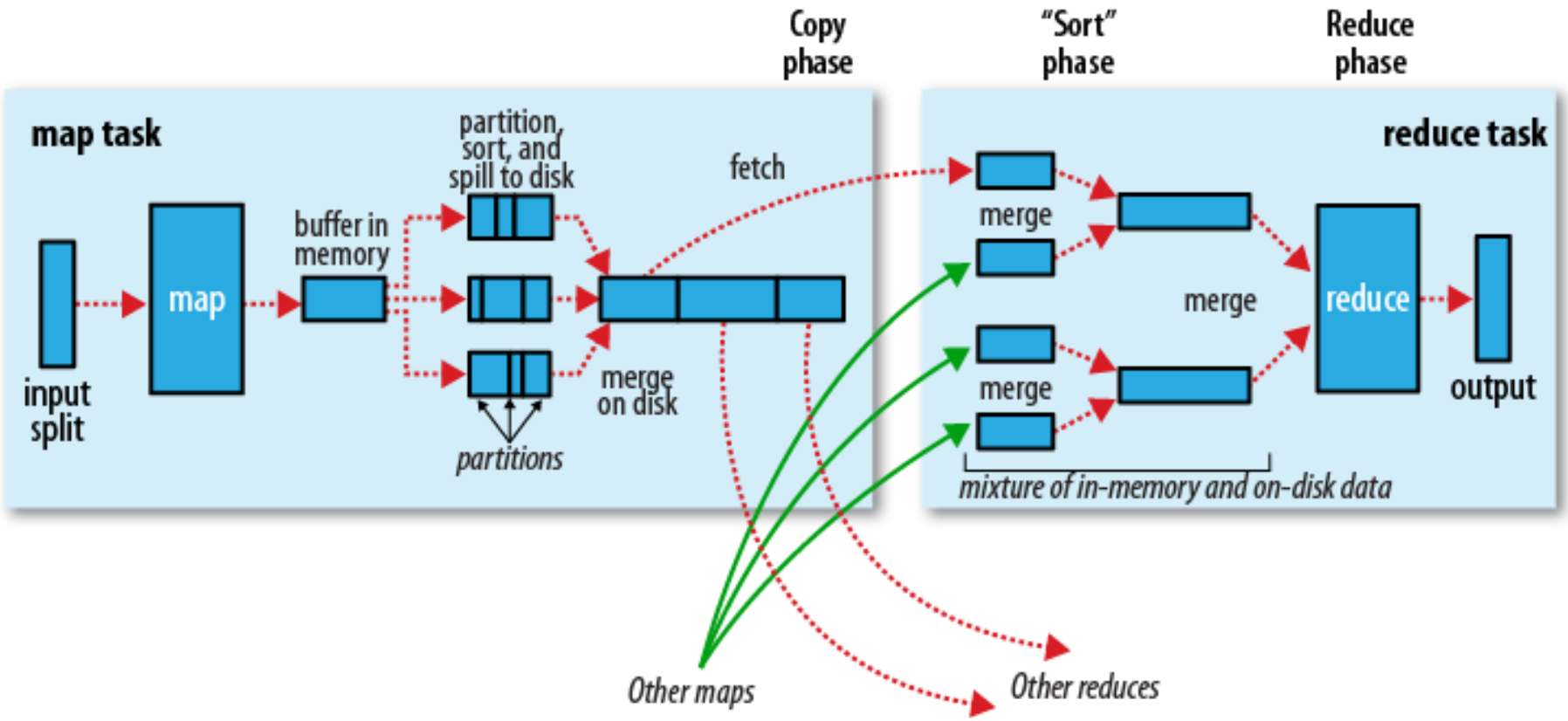
Сортировка и группировка по ключу

- После завершения функций Map данные
  - Сортируются в порядке следования ключей
  - Пары с одинаковым ключом группируются вместе и передаются в один Reducer
  - Значения данных не сортируются
- Данные на выходе из Map:
  - <What, 1>, <Is, 1>, <Apache, 1>, <Hadoop, 1>, <The, 1>, <Apache, 1>, <Hadoop, 1>, <project, 1>, <develops, 1>
- Данные на входе в Reduce:
  - <Apache, (1, 1)>, <develops, 1>, <Hadoop, (1, 1)>, <Is, 1>, <The, 1>, <project, 1>, <What, 1>

WordCount: Reducer

```
public class WordCountReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Сортировка и группировка (Shuffle and sort)



Tom White. Hadoop: The Definitive Guide

Выбор процесса Reduce

- Как определить, какому процессу Reduce нужно передавать данные с указанным ключом?

Выбор процесса Reduce

- Как определить, какому процессу Reduce нужно передавать данные с указанным ключом?
- Интерфейс Partitioner
  - По ключу значению выдает номер Reducer'а, который будет обрабатывать запись
  - `public int getPartition(K key, V value, int numReduceTasks)`
- По умолчанию используется HashPartitioner
  - `(key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;`

Установка Partitioner в Driver

```
public class WordCount extends Configured implements Tool {
  ...
  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(super.getConf(), "WordCount");
    job.setJarByClass(getClass());
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setCombinerClass(WordCountReducer.class);
    job.setPartitionerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

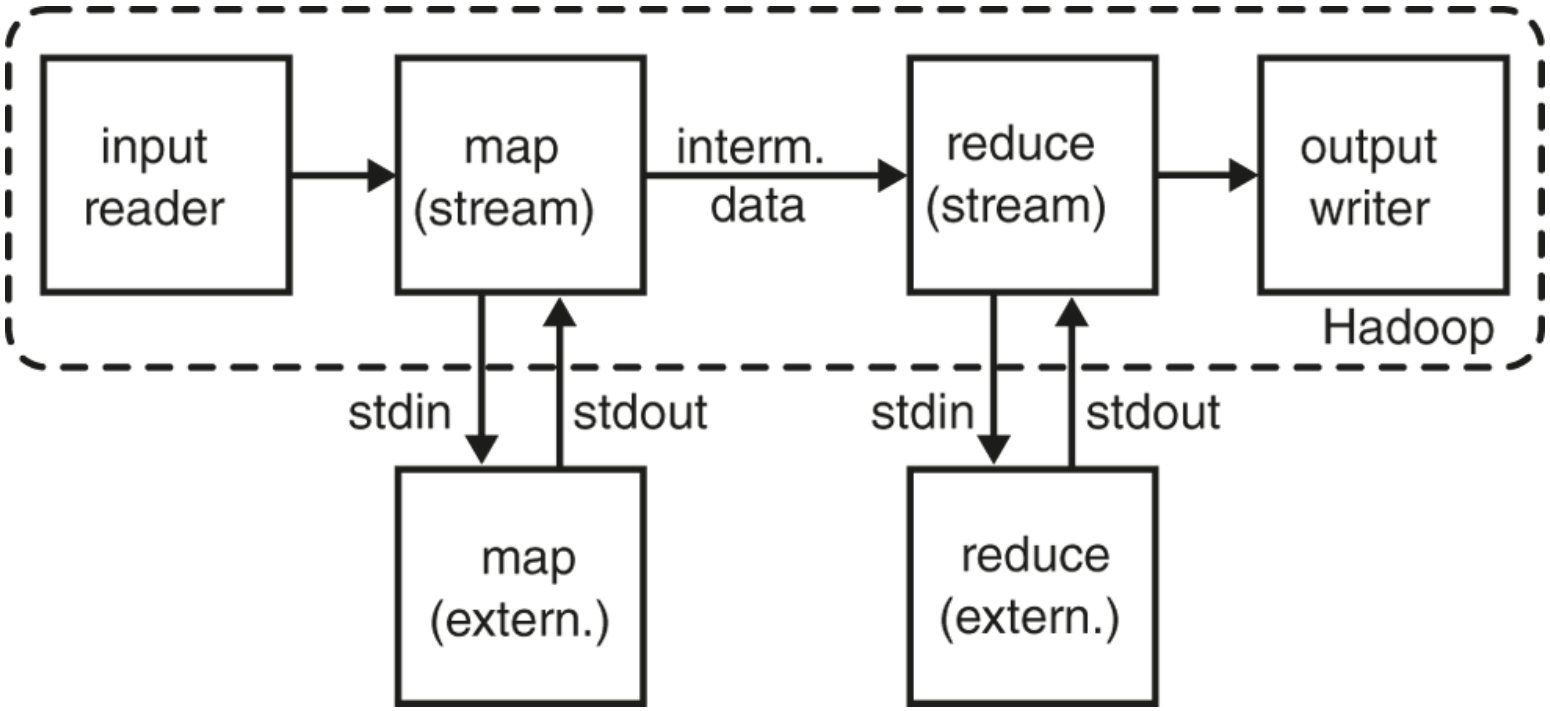
Что нужно установить в драйвере

- Обязательно:
  - Map класс
  - Reduce класс
  - Типы выходных ключей и значений
  - Пути к входным и выходным данным
  - Архив Jar с задачей
- Необязательно
  - InputFormat и OutputFormat (по-умолчанию TextInputFormat и TextOutputFormat)
  - Combiner класс
  - Partitioner класс (по-умолчанию HashPartitioner)
  - Типы ключей и значений на выходе из Map (по-умолчанию такие же, как и на выходе из Reduce)

Отладка MapReduce программы

- Установить переменную окружения:
  - `HADOOP_OPTS="-agentlib:jdwp=transport=dt_socket, server=y, suspend=y,address=5000"`
- Запустить Hadoop в локальном режиме:
  - `hadoop jar wordcount.jar WordCountDriver -D mapreduce.framework.name=local input output`
- Подключится удаленным отладчиком:
  - localhost, порт 5000 (или другой из параметра address)

Hadoop Streaming



Wordcount на Python: Map

```
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    for word in words:
        # write the results to STDOUT (standard output);
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Wordcount на Python: Reduce

```
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Проверка скриптов перед запуском

```
cat data.txt | mapper.py | sort | reducer.py
```

Запуск на кластере Hadoop

```
$ hadoop jar /opt/cloudera/parcels/CDH/jars/hadoop-streaming-
2.5.0-cdh5.2.1.jar -input input -output output -mapper
mapper.py -reducer reducer.py -file mapper.py -file
reducer.py
```

`-input` – каталог со входными данными
`-output` – каталог для выходных данных
`-mapper` – скрипт, реализующий функцию Map
`-reducer` – скрипт, реализующий функцию Reduce
`-file` – файл, который нужно скопировать на все узлы, где будет выполняться MapReduce задача

Рекомендации по разработке

1. Напишите программу и опробуйте в локальном режиме на небольшом объеме данных
2. Протестируйте программу на кластере на небольшом объёме данных
3. Запускайте программу на кластере на полном объёме данных

Вопросы?