ТЕХНОСФЕРА

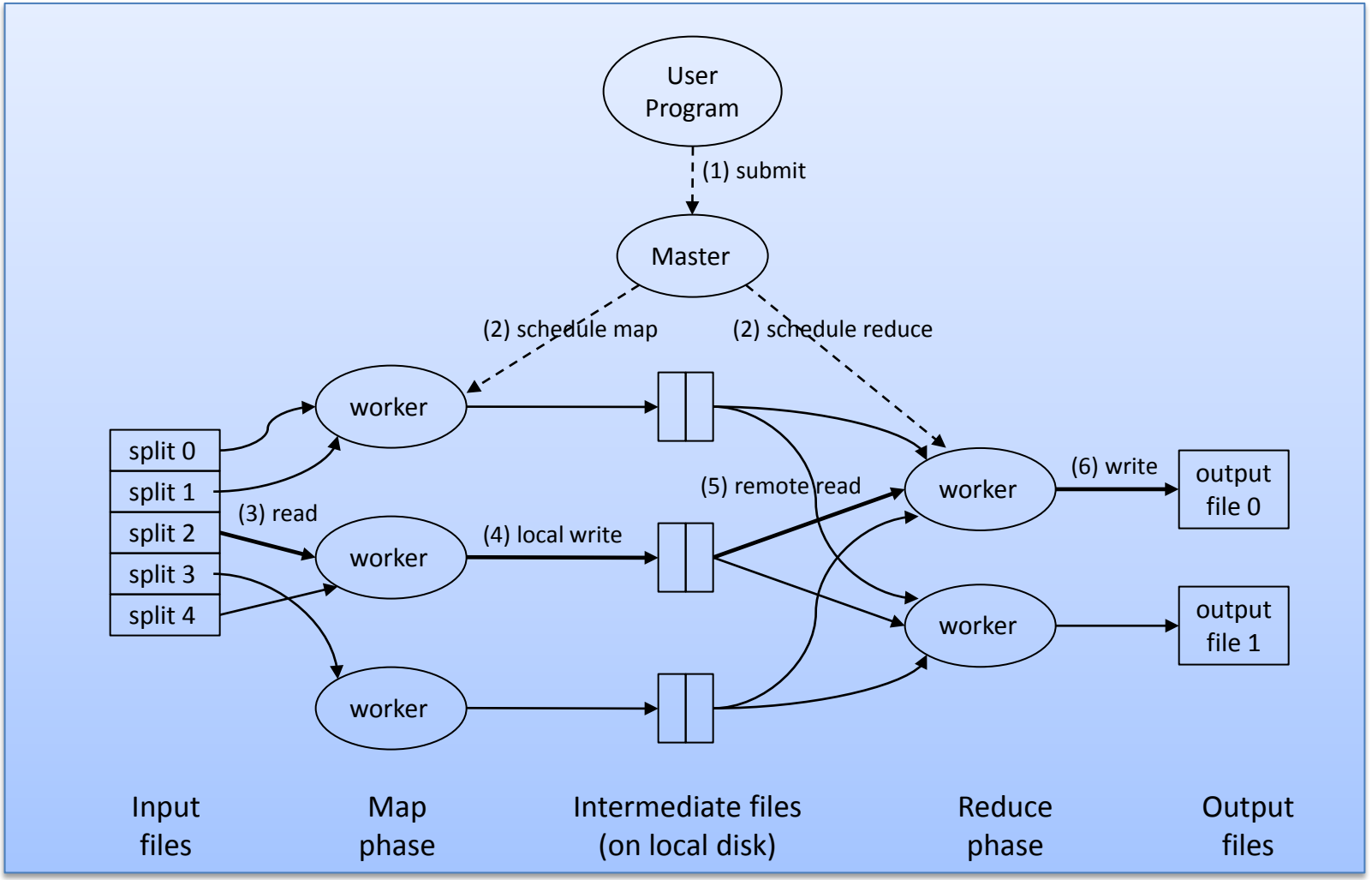
Методы распределенной обработки больших объемов данных в Hadoop

Лекция 4: MapReduce в Hadoop, введение

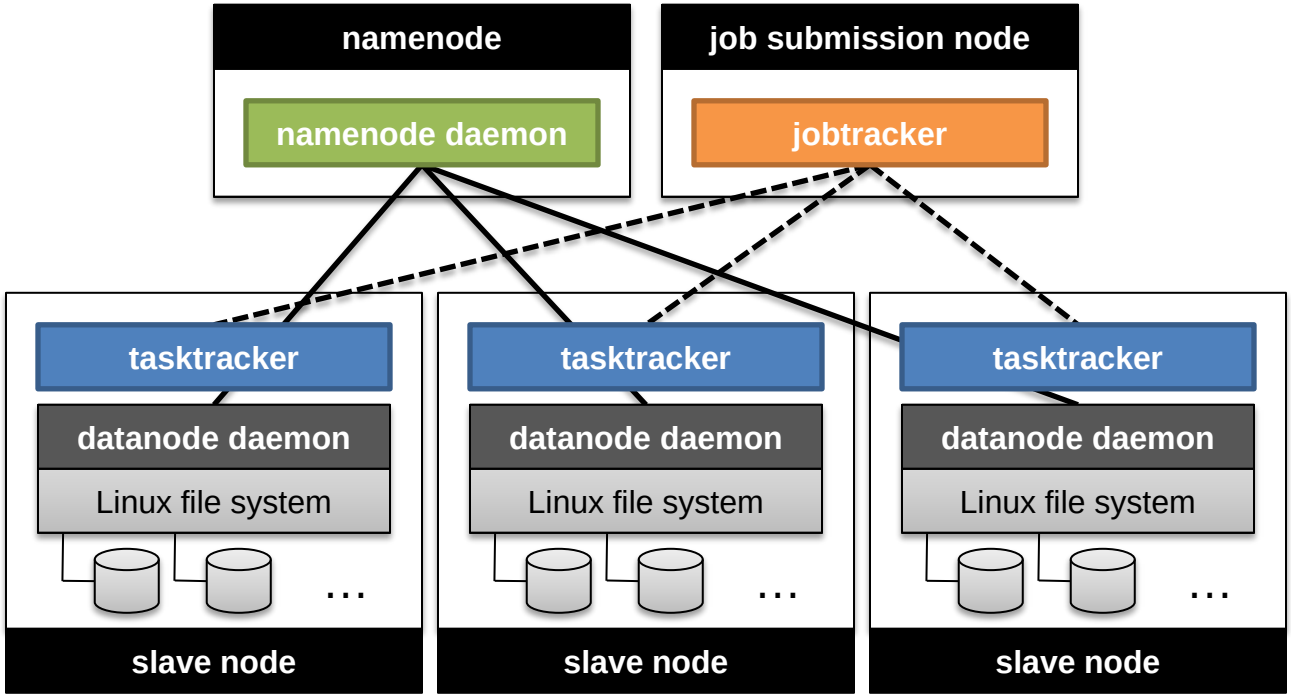
Москва 2014

MapReduce

MapReduce workflow



Hadoop MapReduce & HDFS

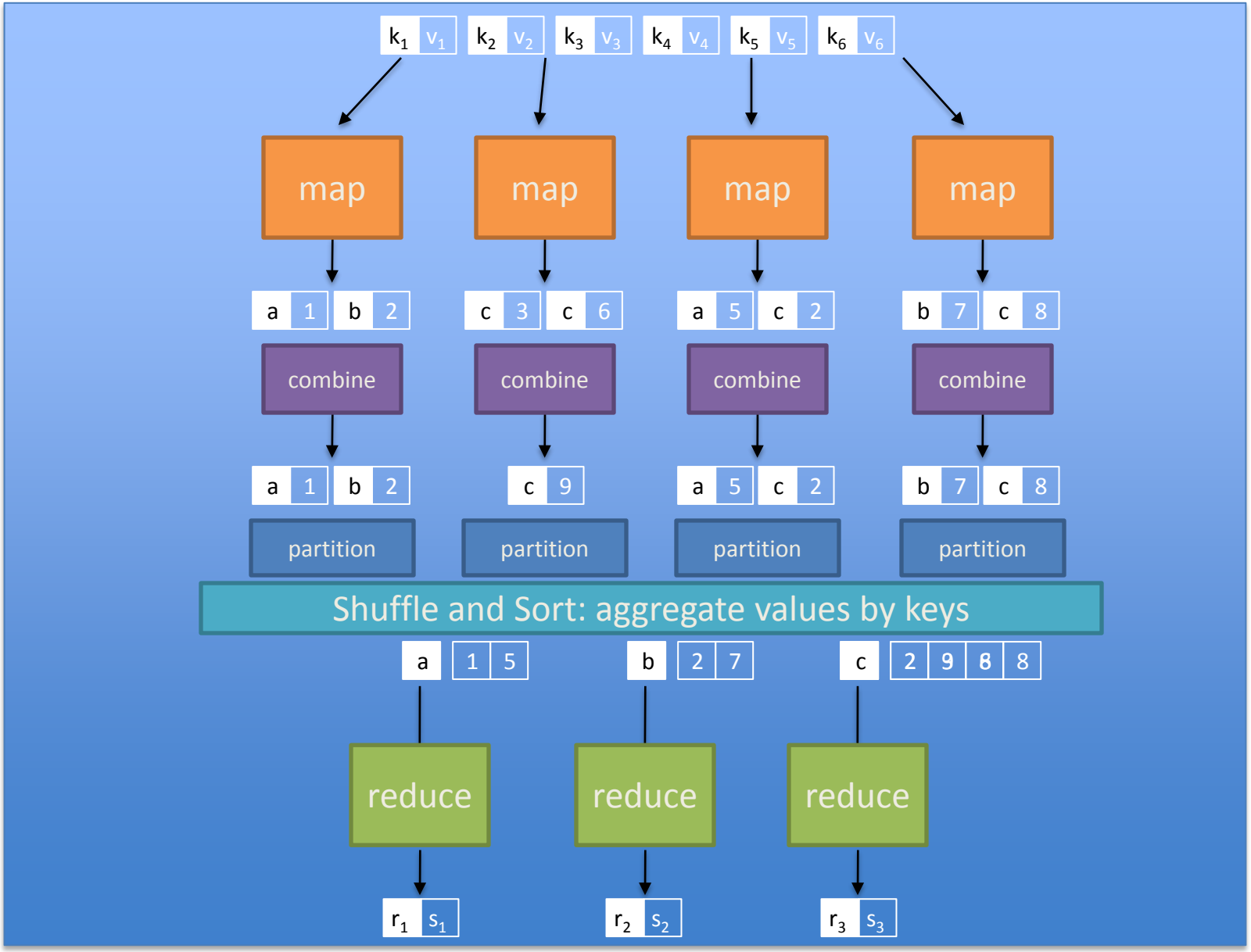


MapReduce “Runtime”

- Управление запуском задач
  - Назначает воркерам таски *map* или *reduce*
- Управление “*data distribution*”
  - Перемещает код к данным
  - Запускает таски (по возможности) локально с данными
- Управление синхронизацией
  - Собирает, сортирует и объединяет промежуточные данные
- Управление обработкой ошибкой и отказов
  - Определяет отказ воркера и перезапускает таск
- Все работает поверх распределенной FS

Hadoop MapReduce (ver.1)

- MapReduce работает поверх демонов
  - ***JobTracker*** и ***TaskTracker***
- ***JobTracker***
  - Управляет запуском тасков и определяет, на каком ***TaskTracker*** таск будет запущен
  - Управляет процессом работы MapReduce задач (*jobs*)
  - Мониторит прогресс выполнения задач
  - Перезапускает зафейленные или медленные таски
- MapReduce имеет систему ресурсов основанную на слотах (*slots*)
  - На каждом ***TaskTracker*** определяется, сколько будет запущено слотов
  - Таск запускается в одном слоте
  - M мапперов + R редьюсеров = N слотов
  - Для каждого слота определяется кол-во потребляемой ОЗУ
  - Такая модель не слишком удобная с точки зрения утилизации ресурсов


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce

- Программист определяет две основные функции:
  map (k1, v1) → list(k2, v2)
  reduce (k2, list(v2*) → list(k3, v3)
  – Все значения с одинаковым ключом отправляются на один и тот же reducer
- … и опционально:
  partition (k2, number of partitions) → partition for k2
  – Часто просто хеш от key, напр., *hash(k2) mod n*
  – Разделяет множество ключей для параллельных операций reduce
  combine (k2, v2) → list(k2, v2')
  – Мини-reducers которые выполняются после завершения фазы map
  – Используется в качестве оптимизации для снижения сетевого трафика на reduce

Hadoop API

- Типы ***API***
  - *org.apache.hadoop.mapreduce*
    - Новое API, будем использовать в примерах
  - *org.apache.hadoop.mapred*
    - Старое API, лучше не использовать
- Класс ***Job***
  - Представляет сериализованную Hadoop-задачу для запуска на кластере
  - Необходим для указания путей для *input/output*
  - Необходим для указания формата данных для *input/output*
  - Необходим для указания типов классов для *mapper, reducer, combiner* и *partitioner*
  - Необходим для указания типов значений пар *key/value*
  - Необходим для указания количества редьюсеров (но не мапперов!)

Hadoop API

- Класс ***Mapper***
  - *void setup(Mapper.Context context)*
    - Вызывается один раз при запуске таска
  - *void map(K key, V value, Mapper.Context context)*
    - Вызывается для каждой пары key/value из *input split*
  - *void cleanup(Mapper.Context context)*
    - Вызывается один раз при завершении таска
- Класс ***Reducer/Combiner***
  - *void setup(Reducer.Context context)*
    - Вызывается один раз при запуске таска
  - *void reduce(K key, Iterable<V> values, Reducer.Context context)*
    - Вызывается для каждого *key*
  - *void cleanup(Reducer.Context context)*
    - Вызывается один раз при завершении таска
- Класс ***Partitioner***
  - int getPartition(K key, V value, int numPartitions)
    - Возвращает номер партиции (reducer) для конкретного ключа

“Hello World”: Word Count

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```

WordCount, Configure Job

- *Job job = Job.getInstance(getConf(), “WordCount");*
  - Инкапсулирует все, что связано с задачей
  - Контролирует процесс выполнения задачи
- Код задачи упаковывается в *jar*-файл
  - Фреймворк *MapReduce* отвечает за дистрибуцию *jar*-файла по кластеру
  - Самый простой способ определить *jar*-файл для класса это вызвать
    job.setJarByClass(getClass());

WordCount, Configure Job

- Определить input
  - Может быть либо путь к файлу, директории или шаблон пути
    - Напр.: */path/to/dir/test_**
  - Директория преобразуется ко списку файлов внутри
  - Input определяется через класс имплементации ***InputFormat.*** В данном случае ***TextInputFormat***
    - Отвечает за сплит входных данных и чтение записей
    - Определяет тип входных данных для пар key/value (***LongWritable*** и ***Text***)
    - Разбивает файл на записи, в данном случае по-строчно.
    - Каждый вызов маппера получает на вход одну строку

```
TextInputFormat.addInputPath(job, new Path(args[0]));
job.setInputFormatClass(TextInputFormat.class);
```

WordCount, Configure Job

- Определить output
  - Определить типы для *output key* и *values* для функций *map* и *reduce*
    - В случае, если типы для *map* и *reduce* отличаются, то
      - ***setMapOutputKeyClass()***
      - ***setMapOutputValueClass()***

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

WordCount, Configure Job

- Определить классы для ***Mapper*** и ***Reducer***
  - Как минимум, надо будет реализовать эти классы

```
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
```

- Опциально, надо определить класс для Combiner
  - Часто он будет совпадать с классом ***Reducer*** (но не всегда!)

```
job.setCombinerClass(WordCountReducer.class);
```

- Запуск задачи
  - Запускает задачу и ждет окончания ее работы
    - true в случае успеха, false в случае ошибки

```
job.waitForCompletion(true)
```

WordCount, Configure Job

```
public class WordCountJob extends Configured implements Tool{
        @Override
        public int run(String[] args) throws Exception {
                Job job = Job.getInstance(getConf(), "WordCount");
                job.setJarByClass(getClass());

                TextInputFormat.addInputPath(job, new Path(args[0]));
                job.setInputFormatClass(TextInputFormat.class);

                job.setMapperClass(WordCountMapper.class);
                job.setReducerClass(WordCountReducer.class);
                job.setCombinerClass(WordCountReducer.class);

                TextOutputFormat.setOutputPath(job, new Path(args[1]));
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

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

WordCount, Configure Job

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

WordCount, Mapper

- Класс ***Mapper*** имеет 4 *generic* параметра
  - Параметры
    - input key
    - input value
    - output key
    - output value
  - Используются типы из hadoop's IO framework
    - org.apache.hadoop.io
- В задаче должен быть реализован метод ***map()***
- В ***map()*** передается
  - пара key/value
  - Объект класса Context
    - Используется для записи в output новой пары key/value
    - Работы со счетчиками
    - Отображения статуса таска

WordCount, Mapper

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

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

        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
                StringTokenizer tokenizer = new StringTokenizer(value.toString());
                while (tokenizer.hasMoreTokens()) {
                        text.set(tokenizer.nextToken());
                        context.write(text, one);
                }
        }
}
```

WordCount, Reducer

- Класс ***Reducer*** имеет 4 *generic* параметра
  - Параметры
    - input key
    - input value
    - output key
    - output value
  - Типы параметров из *map output* должны соответствовать *reduce input*
- MapReduce группирует пары *key/value* от мапперов по ключу
  - Для каждого ключа будет набор значений
  - *Reducer input* отсортирован по ключу
  - Сортировка значений для ключа не гаратируется
- Объект типа Context используется с той же целью, что и в ***Mapper***

WordCount, Reducer

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

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

Reducer в качестве Combiner

- Объединяет данные на стороне ***Mapper*** для уменьшения количества передаваемых данных на ***Reducer***
  - Годится в том случае, когда тип пары key/value в output у ***Reducer*** такие же, как и у ***Mapper***
- Фреймворк MapReduce не гарантирует вызов ***Combiner***
  - Нужно только с точки зрения оптимизации
  - Логика приложения не должна зависеть от вызова вызов ***Combiner***

Типы данных в Hadoop

- ***Writable***
  - Определяет протокол де- сериализации. Каждый тип в Hadoop должен быть ***Writable***
- ***WritableComprable***
  - Определяет порядок сортировки. Все ключи должны быть ***WritableComprable*** (но не значения!)
- ***Text, IntWritable, LongWritable*** и т.д.
  - Конкретные реализации для конкретных типов
- ***SequenceFiles***
  - Бинарно-закодированная последовательность пар *key/value*

Комплексные типы данных в Hadoop

- Как можно реализовать комплексный тип?
- Простой способ:
  - Закодировать в ***Text***
    - Напр., (x,10) = “x:10”
  - Для раскодирования использовать специальный метод парсинга
  - Просто, работает, но…
- Сложный (правильный) способ:
  - Определить реализацию своего типа ***Writable(Comprable)***
  - Необходимо реализовать методы ***readFields, write, (compareTo)***
  - Более производительное решение, но сложнее в реализации

InputSplit

- ***Split*** – это набор логически организованных записей
  - Строки в файле
  - Строки в выборке из БД
- Каждый экземпляр *Mapper* будет обрабатывать один split
  - Функция *map(k, v)* обрабатывать только одну пару key/value
  - Функция *map(k, v)* вызывается для каждой записи из split
- Различные сплиты реализуются расширением класса ***InputSplit***
  - Hadoop предлагает много различных реализаций ***InputSplit***
    - FileSplit, TableSplit и т.д.

InputFormat

- Определяет формат входных данных
- Создает *input splits*
- Определяет, как читать каждый *split*
  - Разбивает каждый *split* на записи
  - Предоставляет реализацию класса ***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;
}
```

InputFormat

- Hadoop предоставляет много готовых классов-реализаций ***InputFormat***
  - TextInputFormat
    - LongWritable/Text
  - NLineInputFormat
    - *NLineInputFormat.setNumLinesPerSplit(job, 100);*
  - DBInputFormat
  - TableInputFormat (HBASE)
    - ImmutableBytesWritable/Result
  - StreamInputFormat
  - SequenceFileInputFormat
- Выбор нужного формата производится через
  - *job.setInputFormatClass(*InputFormat.class);*

OutputFormat

- Определяет формат выходных данных
- Реализация интерфейса класса ***OutputFormat<K,V>***
  - Проверяет output для задачи
    - Напр. путь в HDFS
  - Создает реализацию ***RecordWriter***
    - Непосредственно пишет данные
  - Создает реализацию ***OutputCommitter***
    - Отменяет *output* в случае ошибки таска или задачи

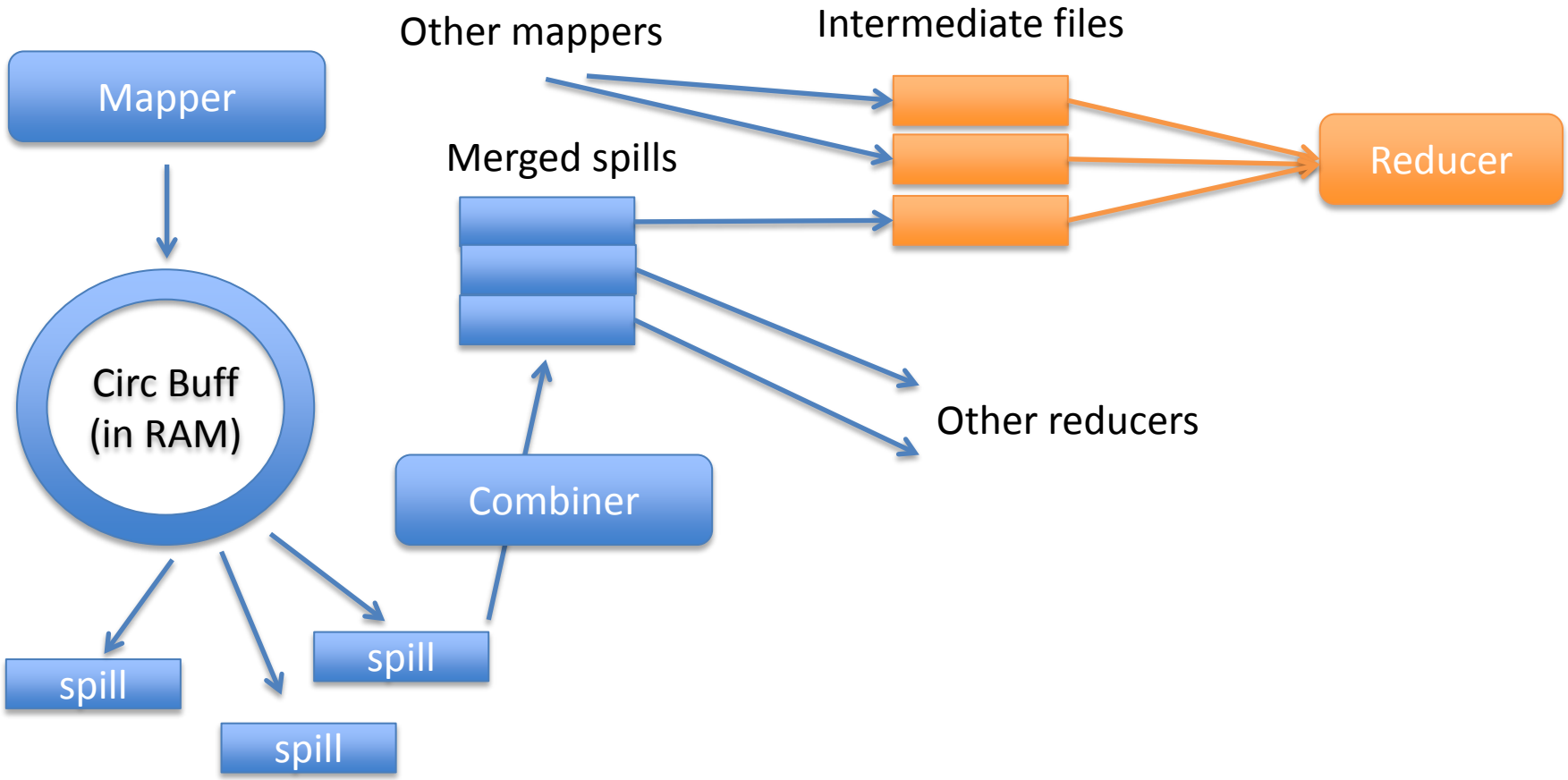
OutputFormat

- Hadoop предоставляет много готовых классов-реализаций ***OutputFormat***
  - TextOutputFormat
  - DBOutputFormat
  - TableOutputFormat (HBASE)
  - MapFileOutputFormat
  - SequenceFileOutputFormat
  - NullOutputFormat
- Выбор нужно формата производится через
  - *job.setOutputFormatClass(*OutputFormat.class);*
  - *job.setOutputKeyClass(*Key.class);*
  - *job.setOutputValueClass(*Value.class);*

Shuffle и Sort в Hadoop

- На стороне ***Map***
  - Выходные данные буферизуются в памяти в циклическом буфере
  - Когда размер буфера достигает предела, данные “скидываются” (*spilled*) на диск
  - Затем все такие “сброшенные” части объединяются (*merge*) в один файл, разбитый на части
    - Внутри каждой части данные отсортированы
    - ***Combiner*** запускается во время процедуры объединения
- На стороне ***Reduce***
  - Выходные данные от мапперов копируются на машину, где будет запущен редьюсер
  - Процесс сортировки (*sort*) представляет собой многопроходный процесс объединения (*merge*) данных от мапперов
    - Это происходит в памяти и затем пишется на диск
  - Итоговый результат объединения отправляется непосредственно на редьюсер

Shuffle и Sort в Hadoop



Запуск задач в Hadoop

$ yarn command [genericOptions] [commandOptions]

Generic Option	Описание
-conf <conf_file.xml>	Добавляет свойства конфигурации из указанного файла в объект *Configuration*
-Dproperty=value	Устанавливает значение свойства конфигурации в объекте *Configuration*
-fs URI	Переопределяет URI файловой системы по-умолчанию (*-Dfs.default.name=URI*)
-files <file,file,file>	Предоставляет возможность использовать указанные файлы в задаче MapReduce через *DistributedCache*
-libjars <f.jar, f2.jar>	Добавляет указанные jars к переменной CLASSPATH у тасков задачи и копирует их через *DistributedCache*

Отладка задач в Hadoop

- Логгирование
  - ***System.out.println***
  - Доступ к логам через веб-интерфейс
  - Лучше использовать отдельный класс-логер (log4j)
    - Настройка типов логгирования (TRACE, DEBUG, ERROR, INFO)
  - Аккуратней с количество данных в логах
- Использование счетчиков
  - ***context.getCounter("GROUP", "NAME").increment(1);***
  - *Аккуратней с количеством различных счетчиков*
- Программирование такое программирование
  - Hadoop – это все лишь "прослойка" для масштабирования
  - Весь функционал лучше реализовывать отдельно от мапперов и редьюсеров
  - Т.о. можно отдельно тестировать как обычный код
  - map() и reduce() всего лишь обертки для получения/передачи данных

Hadoop Streaming

Hadoop Streaming

- Используется стандартный механизм ввода/вывода в Unix для взаимодействия программы и Hadoop
- Позволяет разрабатывать MR задачи почти на любом языке программирования, который умеет работать со стандартным вводом/выводом
- Обычно используется:
  - Для обработки текста
  - При отсутствии опыта программирования на Java
  - Быстрого написания прототипа

Streaming в MapReduce

- На вход функции *map()* данные подаются через стандартный ввод
- В *map()* обрабатываются они построчно
- Функция *map()* пишет пары *key/value*, разделяемые через символ табуляции, в стандартный вывод
- На вход функции *reduce()* данные подаются через стандартный ввод, отсортированный по ключам
- Функция *reduce()* пишет пары *key/value* в стандартный вывод

WordCount на Python

Map: countMap.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token: print token + '\t1'
```

WordCount на Python

Reduce: countReduce.py

```
#!/usr/bin/python
import sys

(lastKey, sum)=(None, 0)

for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    if lastKey and lastKey != key:
        print lastKey + '\t' + str(sum)
        (lastKey, sum) = (key, int(value))
    else:
        (lastKey, sum) = (key, sum + int(value))
if lastKey:
    print lastKey + '\t' + str(sum)
```

Запуск и отладка

Тест в консоли перед запуском

```
$ cat test.txt | countMap.py | sort | countReduce.py
```

Запуск задачи через Streaming Framework

```
yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-D mapred.job.name=“WordCount Job via Streaming" \
-files countMap.py, countReduce.py \
-input text.txt \
-output /tmp/wordCount/ \
-mapper countMap.py \
-combiner countReduce.py \
-reducer countReduce.py
```

Python vs. Java

```
#!/usr/bin/python
import sys

for line in sys.stdin:
   for token in line.strip().split(" "):
      if token: print token + '\t1'

#!/usr/bin/python
import sys

(lastKey, sum)=(None, 0)

for line in sys.stdin:
   (key, value) = line.strip().split("\t")
   if lastKey and lastKey != key:
      print lastKey + '\t' + str(sum)
      (lastKey, sum) = (key, int(value))
   else:
      (lastKey, sum) = (key, sum + int(value))
if lastKey:
   print lastKey + '\t' + str(sum)
```

```
public class WordCountJob extends Configured implements Tool{
        static public class WordCountMapper
                extends Mapper<LongWritable, Text, Text, IntWritable> {

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

                @Override
                protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                        StringTokenizer tokenizer = new StringTokenizer(value.toString());
                        while (tokenizer.hasMoreTokens()) {
                                text.set(tokenizer.nextToken());
                                context.write(text, one);
                        }
                }
        }
        static public class WordCountReducer
                extends Reducer<Text, IntWritable, Text, IntWritable> {

                @Override
                protected void reduce(Text key, Iterable<IntWritable> values,
                                                        Context context)
                        throws IOException, InterruptedException {
                        int sum = 0;
                        for (IntWritable value : values) {
                                sum += value.get();
                        }
                        context.write(key, new IntWritable(sum));
                }
        }
        @Override
        public int run(String[] args) throws Exception {
                Job job = Job.getInstance(getConf(), "WordCount");
                job.setJarByClass(getClass());

                TextInputFormat.addInputPath(job, new Path(args[0]));
                job.setInputFormatClass(TextInputFormat.class);

                job.setMapperClass(WordCountMapper.class);
                job.setReducerClass(WordCountReducer.class);
                job.setCombinerClass(WordCountReducer.class);

                TextOutputFormat.setOutputPath(job, new Path(args[1]));
                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 WordCountJob(), args);
                System.exit(exitCode);
        }
}
```

Отладка в Streaming

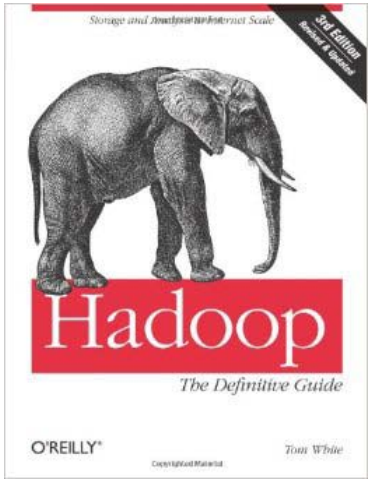
- Можно обновлять счетчики и статус таска
  - Для этого надо отправить строку в *standard error* в формате *“streaming reporter”*

    reporter:counter:<counter_group>,<counter>,<increment_by>

```
#!/usr/bin/python
import sys

for line in sys.stdin:
   for token in line.strip().split(" "):
      if token:
         sys.stderr.write("reporter:counter:Tokens,Total,1\n")
         print token[0] + '\t1'
```

Книги



Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition

Chapter 2: MapReduce
Chapter 5: Developing a MapReduce Application
Chapter 7: MapReduce Types and Formats

Data-Intensive Text Processing with MapReduce
Jimmy Lin and Chris Dyer (Authors) (April, 2010)

Chapter2: MapReduce Basics

Вопросы?