Map-Reduce API

Типы API

- *org.apache.hadoop.mapreduce*
  - Новое API, будем использовать в примерах
- *org.apache.hadoop.mapred*
  - Старое API, лучше не использовать

Класс ***Job***

Содержит описание MapReduce задачи:

- *input/output* пути
- Формат *input/output* данных
- Указания классов для *mapper, reducer, combiner* и *partitioner*
- Типы значений пар *key/value*
- Количество редьюсеров

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 reducer для ключа K

“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, sum);
```

WordCount: Configure Job

- Создание объекта Job:

```
Job job = Job.getInstance(getConf(), “WordCount");
```

- Определение jar для задачи:

```
job.setJarByClass(getClass());
```

WordCount: Configure Job

Определение input:

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

- в качестве пути может быть файл, директория, шаблон пути (*/path/to/dir/test_**)
- *TextInputFormat* читает входные данные как текстовый файл (key – LongWritable, value – Text)

WordCount: Configure Job

Определение output:

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

Если типы для *map* и *reduce* отличаются, то:

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(LongWritable.class);
```

WordCount: Configure Job

- Определение класса для **Mapper** и **Reducer**:

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

- Определение класса для **Combiner**:

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

WordCount: запуск задачи

Запускает задачу и ждет ее окончания:

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

– *true* в случае успеха, *false* в случае ошибки

```
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;
        }
}
```

```
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

- Наследник класса:

```
public class Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>
```

- Должен быть реализован метод map():

```
void map(KEYIN key, VALUEIN value, Context context){}
```

- Типы key / value (из org.apache.hadoop.io):
  - IntWritable
  - Text
  - ImmutableBytesWritable

```
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()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

WordCount: Reducer

- Наследник класса:

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
```

- Должен быть реализован метод reduce():

```
void reduce(KEYIN key, Iterable<VALUEIN> values,
            Context context){}
```

- Типы входных данных в Reducer должны совпадать с типами выходных данных Mapper

```
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

- Меньше данных отправляется на ***Reducer***
- Типы key/value в output у ***Reducer*** и ***Mapper*** равны
- 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

Простой способ:

- Закодировать в ***Text***
- Для раскодирования нужен специальный метод парсинга
- Просто, работает, но…

Комплексные типы данных в Hadoop

Сложный (правильный) способ:

- Определить реализацию своего типа ***Writable(Comprable)***
- Необходимо реализовать методы ***readFields, write, (compareTo)***
- Более производительное решение, но сложнее в реализации

Класс ***InputSplit***

- ***Split*** – это набор логически организованных записей
  - Строки в файле
  - Строки в выборке из БД
- Каждый экземпляр *Mapper* обрабатывает один split
  - Функция *map(k, v)* вызывается для каждой записи из split
- Сплиты реализуются расширением класса ***InputSplit****:*
  - FileSplit
  - TableSplit

Класс ***InputFormat***

- Создает *input splits*
- Определяет, как читать каждый *split*

```
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;
}
```

- Готовые классы-реализации ***InputFormat***:
  - **TextInputFormat**
    - LongWritable / Text
  - **NLineInputFormat**
    - *NLineInputFormat.setNumLinesPerSplit(job, 100);*
  - **DBInputFormat**
  - **TableInputFormat** (HBASE)
    - ImmutableBytesWritable / Result
  - **SequenceFileInputFormat**
- Выбор нужного формата:

```
job.setInputFormatClass(*InputFormat.class);
```

Класс ***OutputFormat***

- Определяет формат выходных данных
- Реализация интерфейса класса ***OutputFormat***
  - Проверяет output для задачи
  - Создает реализацию ***RecordWriter***
  - Создает реализацию ***OutputCommitter***

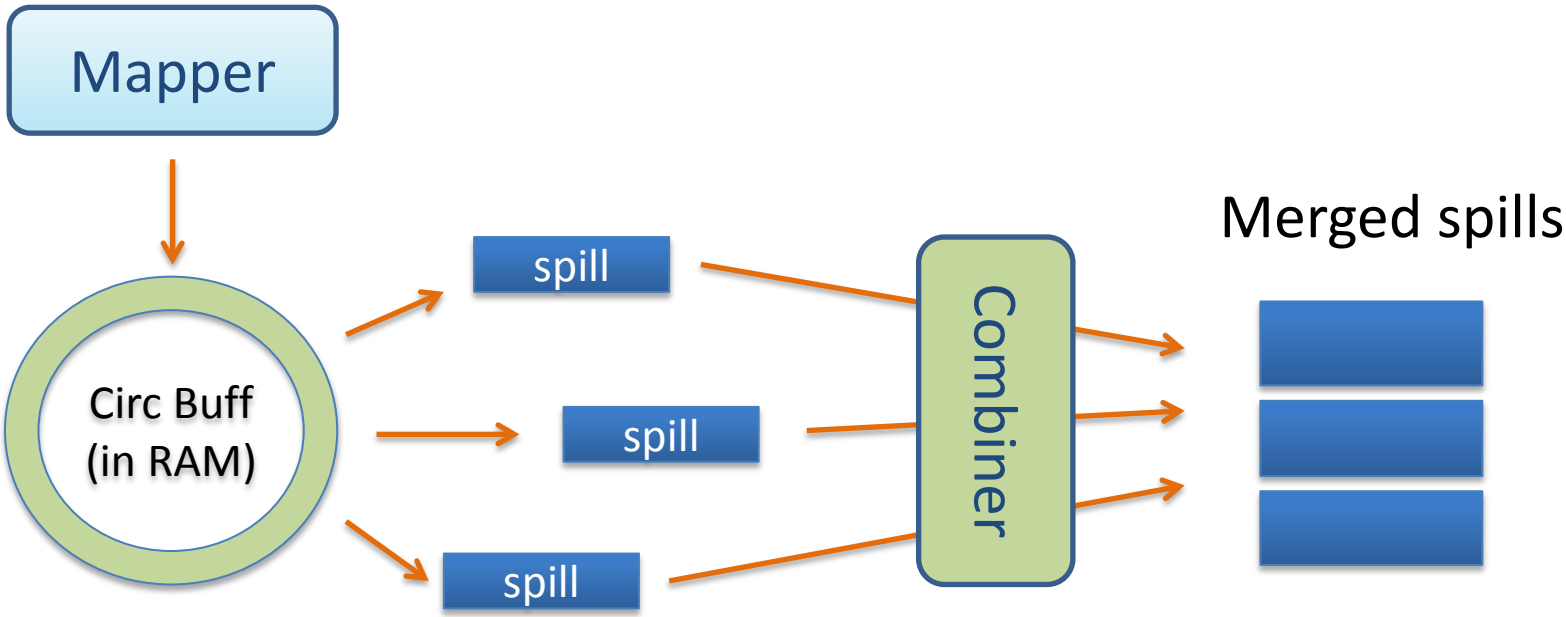
- Готовые классы-реализации ***OutputFormat***:
  - *TextOutputFormat*
  - *DBOutputFormat*
  - *TableOutputFormat* (HBASE)
  - *SequenceFileOutputFormat*
  - *NullOutputFormat*
- Выбор нужного формата:

```
job.setOutputFormatClass(*OutputFormat.class);
job.setOutputKeyClass(*Key.class);
job.setOutputValueClass(*Value.class);
```

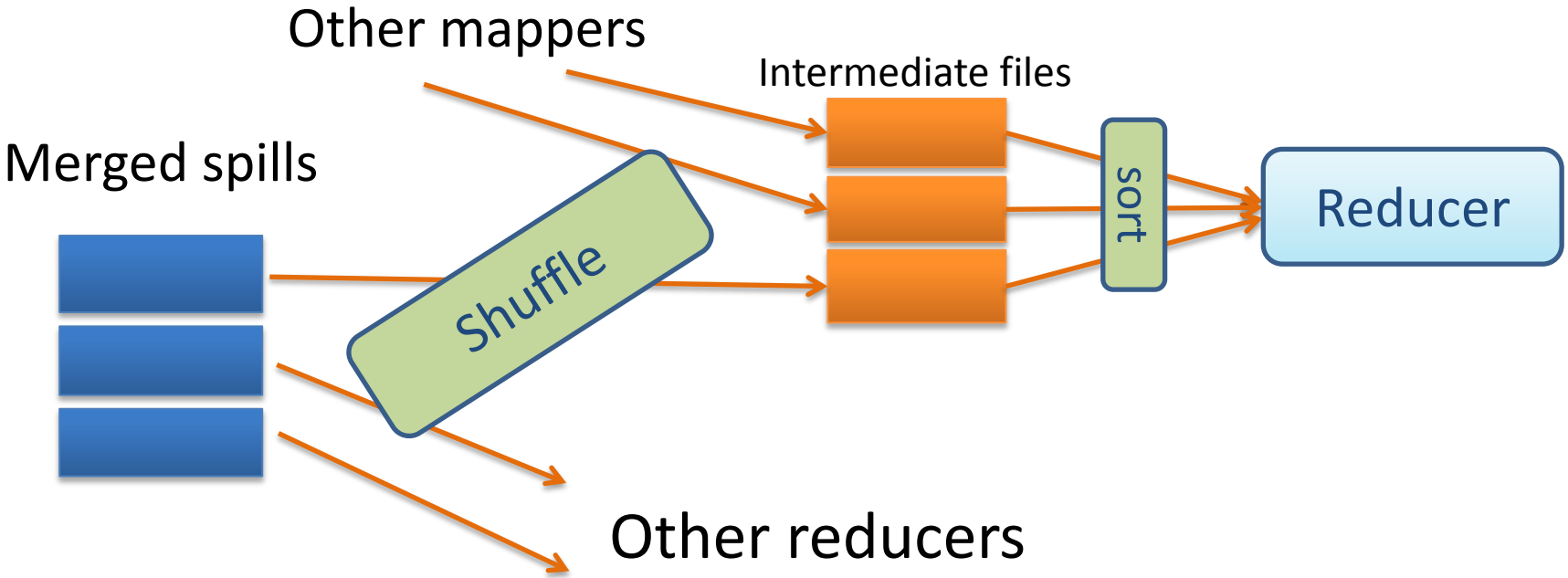
Shuffle и Sort в Hadoop

- На стороне ***Map***
  - Выходные данные сохраняются в памяти в циклическом буфере
  - Когда размер буфера достигает предела, данные “скидываются” (*spilled*) на диск
  - Все “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



Shuffle и Sort в Hadoop



$ hadoop jar <jar> [mainClass] args...

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*
-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*

$ hadoop jar file.jar org.my.main.class –files dict.txt –D send.stat=true

Отладка задач в Hadoop

- Логирование
  - ***System.out.println***
  - Доступ к логам через веб-интерфейс
  - Лучше использовать отдельный класс-логер (log4j)
  - Аккуратней с количество данных в логах
- Использование счетчиков:

```
context.getCounter("GROUP", "NAME").increment(1);
```

Hadoop Streaming

Hadoop Streaming

- Используется стандартный механизм ввода/вывода в Unix для взаимодействия программы и Hadoop
- Разработка MR задачи почти на любом языке программирования
- Обычно используется:
  - Для обработки текста
  - При отсутствии опыта программирования на Java
  - Для 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'
```

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 и отладка

Запуск задачи через Streaming Framework:

```
hadoop jar $HADOOP_HOME/hadoop/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);
                }
}
```