Лекция 10
Модель программирования MapReduce

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Высокопроизводительные вычислительные системы»
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)
Осенний семестр, 2015

Содержание лекции

- Модель программирования MapReduce
- Реализация модели MapReduce от Google
- Знакомство с Apache Hadoop

MapReduce

- **Модель** программирования для описания алгоритмов обработки больших массивов данных (Big Data)
- **Среда** выполнения для параллельной обработки больших объемов данных
- **Программная реализация** (Google, Apache Hadoop, Microsoft Dytona, ...)

Big Data

- Коммерческие приложения
  - Web
    - десятки миллиардов страниц, сотни терабайт текста
    - Google MapReduce: 100 TB данных в день (2004), 20 PB (2008)
- Социальные сети
  - Facebook – петабайты пользовательских данных (15 TB/день)
- Поведенческие данные пользователей (business intelligence)
- Научные приложения
  - Физика высоких энергий: Большой Адронный Коллайдер – 15 PB/год
  - Астрономия и астрофизика: Large Synoptic Survey Telescope (2015) – 1.28 PB/год
  - Биоинформатика: секвенирование ДНК, European Bioinformatics Institute – 5 PB (2009)

Современные тенденции – рост объемов данных

- **Мы можем/вынуждены хранить все больше данных**
  - Латентность и пропускная способность дисковых массивов (SSD/HDD, RAID) не успевают за ростом объема данных
- **Современные задачи намного превышают возможности одного узла (сервера)**
  - Требуются кластеры из сотен и тысяч узлов
  - Стратегия *горизонтального масштабирования* (увеличение числа узлов, scale out) выгоднее стратегии *вертикального масштабирования* (увеличение производительности одного узла, scale up)
- **Данные нельзя полностью разместить в оперативной памяти** (RAM), приходится обращаться к внешнему хранилищу (HDD/SSD, RAID)
  - Последовательные чтение и запись данных эффективнее случайного доступа

Современные тенденции – рост объемов данных

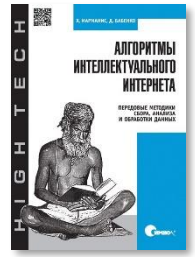
- **В большемасштабной системе отказы узлов – это нормальное явление**
  - 10 000 серверов с MTBF = 1 000 дней => ~ 10 отказов в день
  - Необходимы автоматическая обработка и восстановление после отказов (fault tolerance)
- **HPC-системы имеют выделенные системы хранения данных** (внешние хранилища, подключенные к узлам через высокоскоростной интерконнект)
  - Большие объемы данных эффективнее обрабатывать там же, где они хранятся
- **Разрабатывать приложения для подобных систем на низком уровне очень сложно**
  - Требуются высокоуровневые модели программирования, скрывающие детали системного уровня и учитывающие свойства большемасштабной системы (отказы узлов, латентность, …)

Web Search

- **Сбор документов Web (crawling)**
  - ❑ offline, загрузка большого объема данных, выборочное обновление, обнаружение дубликатов
- **Построение инвертированного индекса (indexing)**
  - ❑ offline, периодическое обновление, обработка большого объема данных, предсказуемая нагрузка
- **Ранжирование документов для ответа на запрос (retrieval)**
  - ❑ online, сотни миллисекунд, большое кол-во клиентов, пики нагрузки

❑ Кристофер Д. Маннинг, Прабхакар Рагхаван, Хайнрих Шютце. **Введение в информационный поиск**. – М.: Вильямс, 2011.

❑ Хараламбос Марманис, Дмитрий Бабенко. **Алгоритмы интеллектуального Интернета. Передовые методики сбора, анализа и обработки данных**. – М.: Символ-Плюс, 2011



Построение инвертированного индекса (Inverted index)

- **Имеется** коллекция документов (загружены роботом)

(docid1, content), (docid2, content), …, (docid*N*, content)

- **Необходимо** для каждого слова (term) построить список документов, в которых оно встречается

[(term, [<docid, tf>…])…]

- **Шаг 1.** Параллельная обработка документов
(у каждого вычислителя множество документов)

(docid, content) –> (term, [<docid1, tf1>,<docid2, tf2>...])

- **Шаг 2.** Параллельная агрегация промежуточных результатов для каждого терма
(собираем результаты со всех вычислителей)

(term, [<docid1, tf1>,<docid2, tf2>...]) –> (term, [<docid, tf>...])

Построение инвертированного индекса (Inverted index)

Шаг 1. (docid, content) –> (term, [<docid1,tf1>,<docid2,tf2>...])

- (кит, <docid1, tf>)
- (океан, <docid1, tf>)
- (вода, <docid1, tf>)
- (milk, <docid2, tf>)
- (drink, <docid2, tf>)

- (кит, <docid3, tf>, <docid5, tf>)
- (лев, <docid3, tf>)
- (лось, <docid3, tf>, <docid5, tf>)
- (lime, <docid4, tf>)
- (drink, <docid4, tf>)

- (milk, <docid6, tf>, <docid7, tf>, <docid8, tf>)
- (drink, <docid6, tf>, <docid7, tf>)
- (water, <docid7, tf>, <docid8, tf>)

Шаг 2. (term, [<docid1,tf1>,<docid2,tf2>...]) –> (term, [<docid,tf>...])

- (кит, <docid1, tf>, <docid3, tf>, <docid5, tf>)
- (океан, <docid1, tf>)
- (вода, <docid1, tf>)
- (milk, <docid2, tf>, <docid6, tf>, <docid7, tf>, <docid8, tf>)
- (drink, <docid2, tf>, <docid4, tf>, <docid6, tf>, <docid7, tf>)

- (лев, <docid3, tf>)
- (лось, <docid3, tf>, <docid5, tf>)
- (lime, <docid4, tf>)
- (water, <docid7, tf>, <docid8, tf>)

Подсчет частоты встречаемости слов (TF – Term Frequency)

- **Имеется** коллекция документов (загружены роботом)

(docid1, content), (docid2, content), …, (docid*N*, content)

- **Необходимо** для каждого слова (term) вычислить частоту его встречаемости в документах

[(term, tf)...]

- **Шаг 1.** Параллельная обработка документов (у каждого вычислителя множество документов)

(docid, content) –> [(term1,tf1), (term2, tf2), …]

- **Шаг 2.** Параллельная агрегация промежуточных результатов для каждого терма (собираем результаты со всех вычислителей)

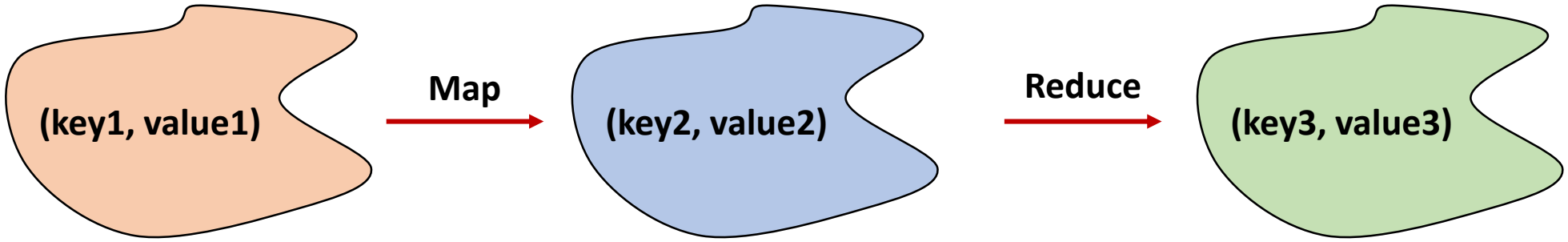
(term, [tf1, tf2, …]) –> (term, tf)

Модель программирования MapReduce

- Базовой структурой данных являются пары (ключ, значение)

- Программа описывается путем определения функций

  - **map**: (key1, value1) -> [(key2, value2)]
  - **reduce**: (key2, [value2, value2, …]) -> [(key3, value3)]



Пример: подсчет встречаемости слов

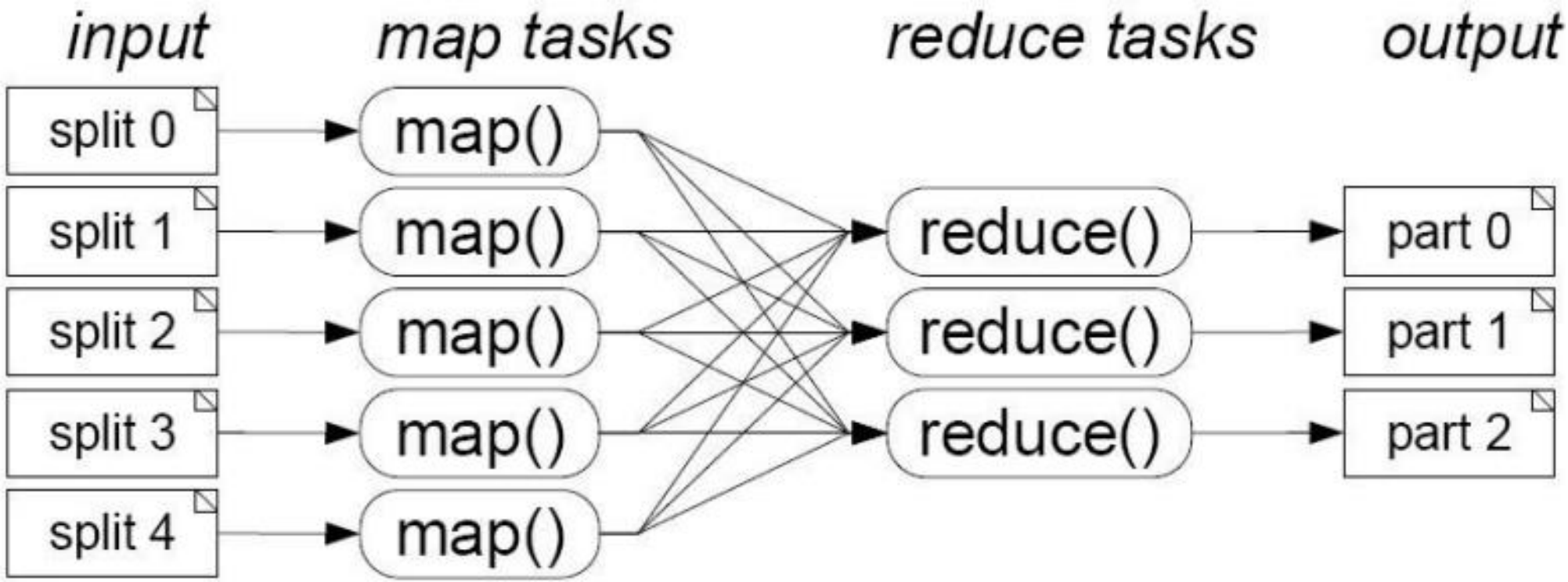
```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

Jimmy Lin and Chris Dyer. **Data-Intensive Text Processing with MapReduce**. – University of Maryland, 2010

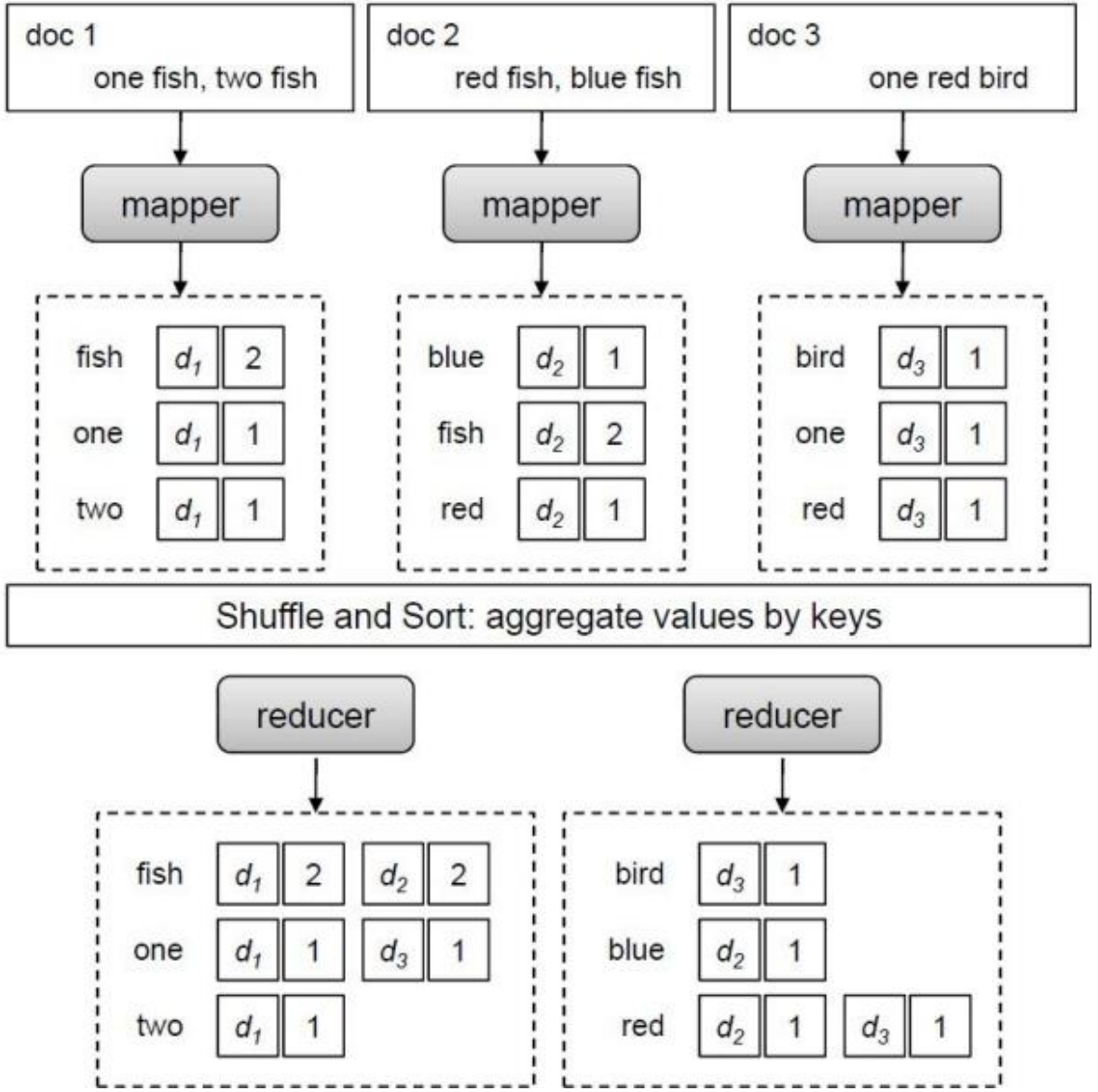
Другие примеры

- **Поиск в тексте**
  - map: (docid, content) → [(docid, line)]
  - reduce: нет
- **Группировка и сортировка по ключу**
  - map: (key, record) → (key, record)
  - reduce: (key, [record]) → (key, [record])
- **Обращение Web-графа**
  - map: (docid, content) → [(url, docid)]
  - reduce: (url, [docid]) → (url, [docid])
- **Анализ посещаемости сайта**
  - map: (logid, log) → [(url, visit_count)]
  - reduce: (url, [visit_count]) → (url, total_count)
- **Вычисление векторов ключевых слов по сайтам**
  - map: (docid, <url, content>) → (ostname, doc_term_vector)
  - reduce: (ostname, [doc_term_vector]) → (ostname, ost_term_vector)

Параллелизм по данным



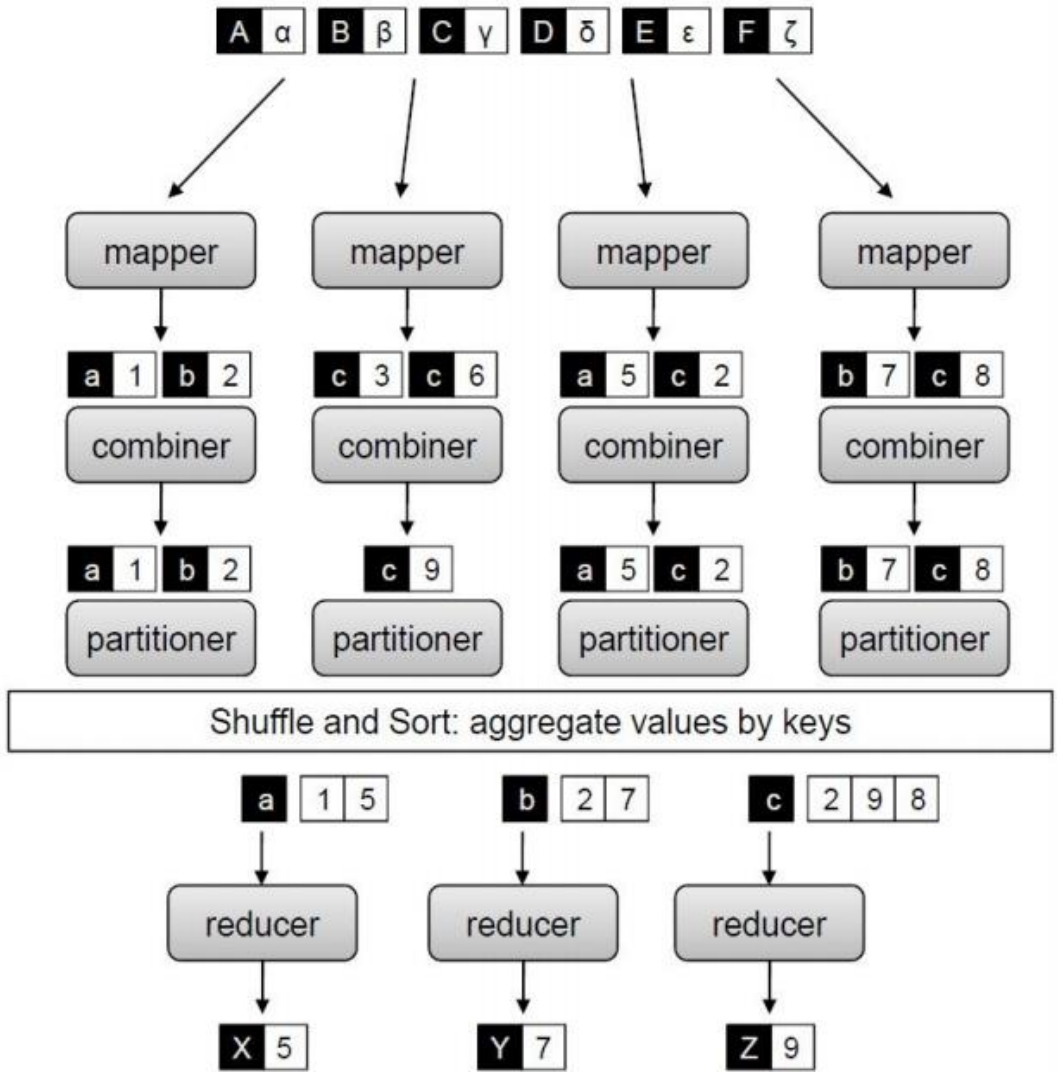
Реализация вычислений



Дополнительные функции

- **partition:** (k2, num_reducers) -> reducer_id
  - определяет распределение промежуточных данных между reduce-процессами
  - простейший случай: hash(k2) % num_reducers
- **combine:** (k2, [v2]) -> [(k2', v2')]
  - осуществляет локальную агрегацию промежуточных данных после map() в рамках одного map-процесса
  - для ассоциативных и коммутативных операций может использоваться reduce()
- **compare:** (k2, k2') -> {-1, 0, 1}
  - определяет отношение порядка между промежуточными ключами

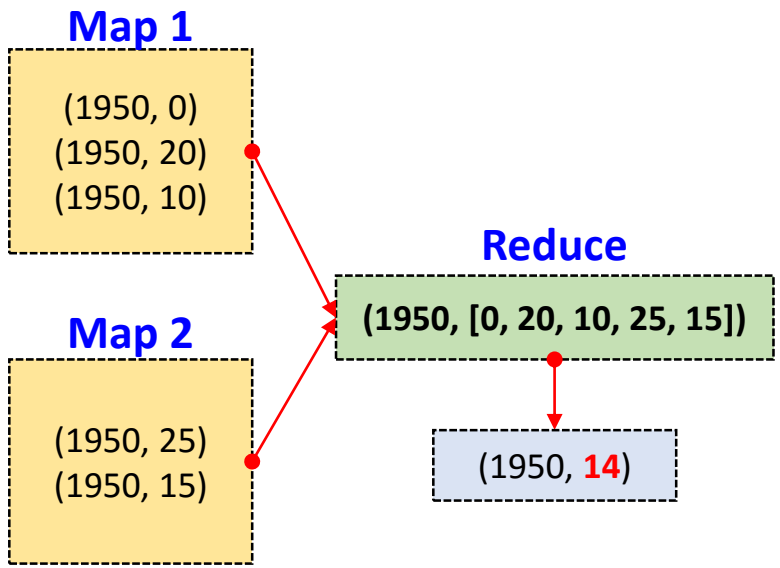
Детальная схема вычислений

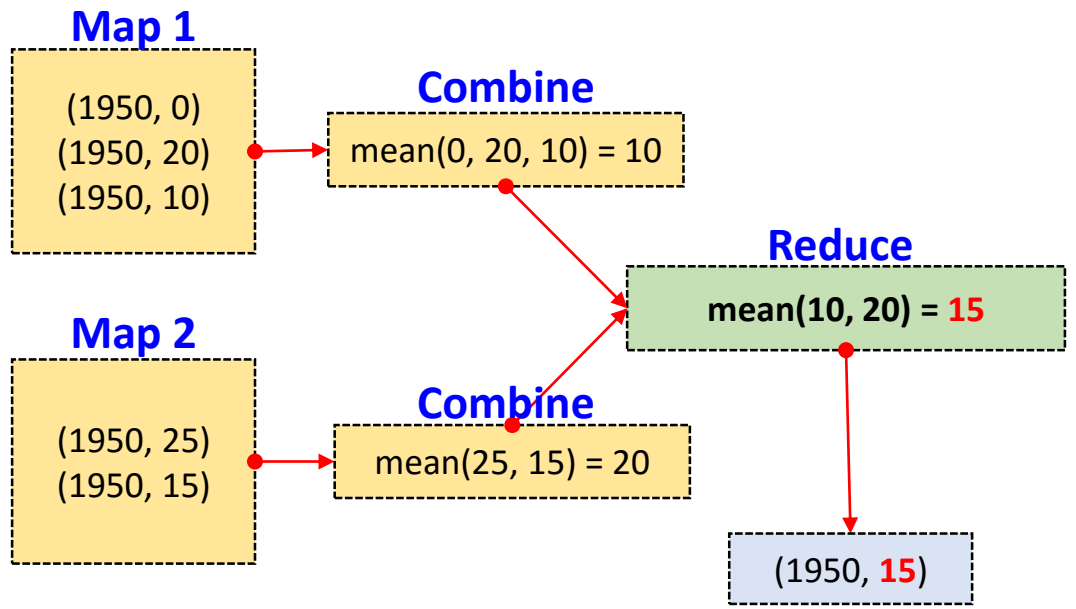


Jimmy Lin and Chris Dyer. **Data-Intensive Text Processing with MapReduce**. – University of Maryland, 2010

Функция Combine

- Функция Combine позволяет сократить объемы данных, передаваемых от фазы map к reduce
- Функция combine не всегда применима
- **Пример: поиск среднего значения в коллекции (год, температура)**





Запуск MapReduce-программы

- Конфигурация задания
  - Входные данные, способ получения (k1, v1)
  - Функции map, reduce, partition, combine, compare
  - Местоположение и формат выходных данных
  - Параметры запуска (количество map- и reduce-задач)
- Запуск задания: MapReduce.runJob(config)
- Остальное берет на себя реализация среды выполнения

За что отвечает реализация MapReduce

- Декомпозиция на параллельные подзадачи (map- и reduce-задачи)
- Запуск рабочих процессов
- Распределение задач по рабочим процессам и балансировка нагрузки
- Передача данных рабочим процессам (требуется минимизировать)
- Синхронизация и передача данных между рабочими процессами
- Обработка отказов рабочих процессов

Реализации MapReduce

- **Системы с распределенной памятью (вычислительные кластеры)**
  - **Google MapReduce** (C++, Python, Java)
  - **Apache Hadoop** (Java, Any)
  - Disco (Erlang / Python)
  - Skynet (Ruby)
  - Holumbus-MapReduce (Haskell)
  - FileMap: File-Based Map-Reduce
  - Yandex YT (Yandex MapReduce, C++/any) // http://www.slideshare.net/yandex/yt-26753367
- **Системы с общей памятью (SMP/NUMA-серверы)**
  - QtConcurrent (C++)
  - Phoenix (C, C++)
- **GPU**
  - Mars: A MapReduce Framework on Graphics Processors

Apache Hadoop

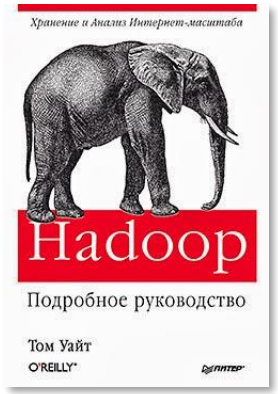
http://hadoop.apache.org

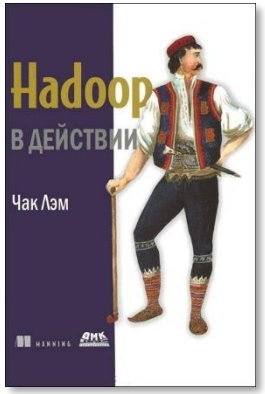
Apache Hadoop

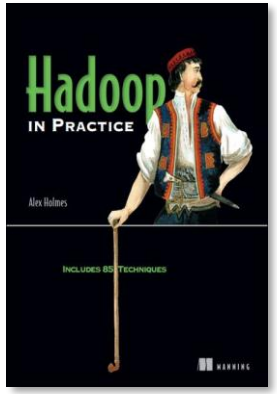
- **Apache Hadoop** – это открытая реализация MapReduce для отказоустойчивых, масштабируемых распределенных вычислений
- **Лицензия:** Apache License 2.0
- **Версии:** 2007 г. – версия 0.15.1; 2008 г. – 0.19.0; …; 2013 г. – 2.2.0; 2014 г. – **2.3.0**
- **Состав Apache Hadoop:**
  - Hadoop Common
  - Hadoop Distributed File System (HDFS) – распределенная файловая система
  - Hadoop YARN – подсистема управления заданиями и ресурсами кластера
  - Hadoop MapReduce – фреймворк для разработки MapReduce-программ

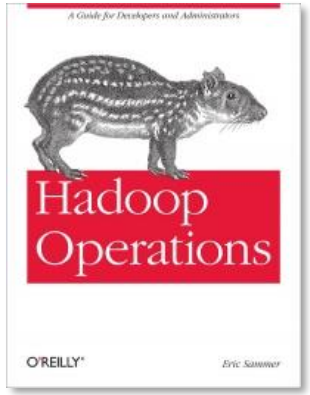
Документация

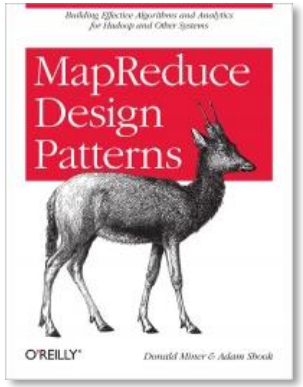
- **Apache Online:** http://hadoop.apache.org/docs/stable
- **Том Уайт. Hadoop. Подробное руководство. - СПб.: Питер, 2013.**
- Tom White. Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media, 2012.
- **Чак Лэм. Hadoop в действии. - М.: ДМК Пресс, 2012.**
- Chuck Lam. Hadoop in Action. Manning Publications, 2010.
- Alex Holmes. Hadoop in Practice. Manning Publications, 2012.







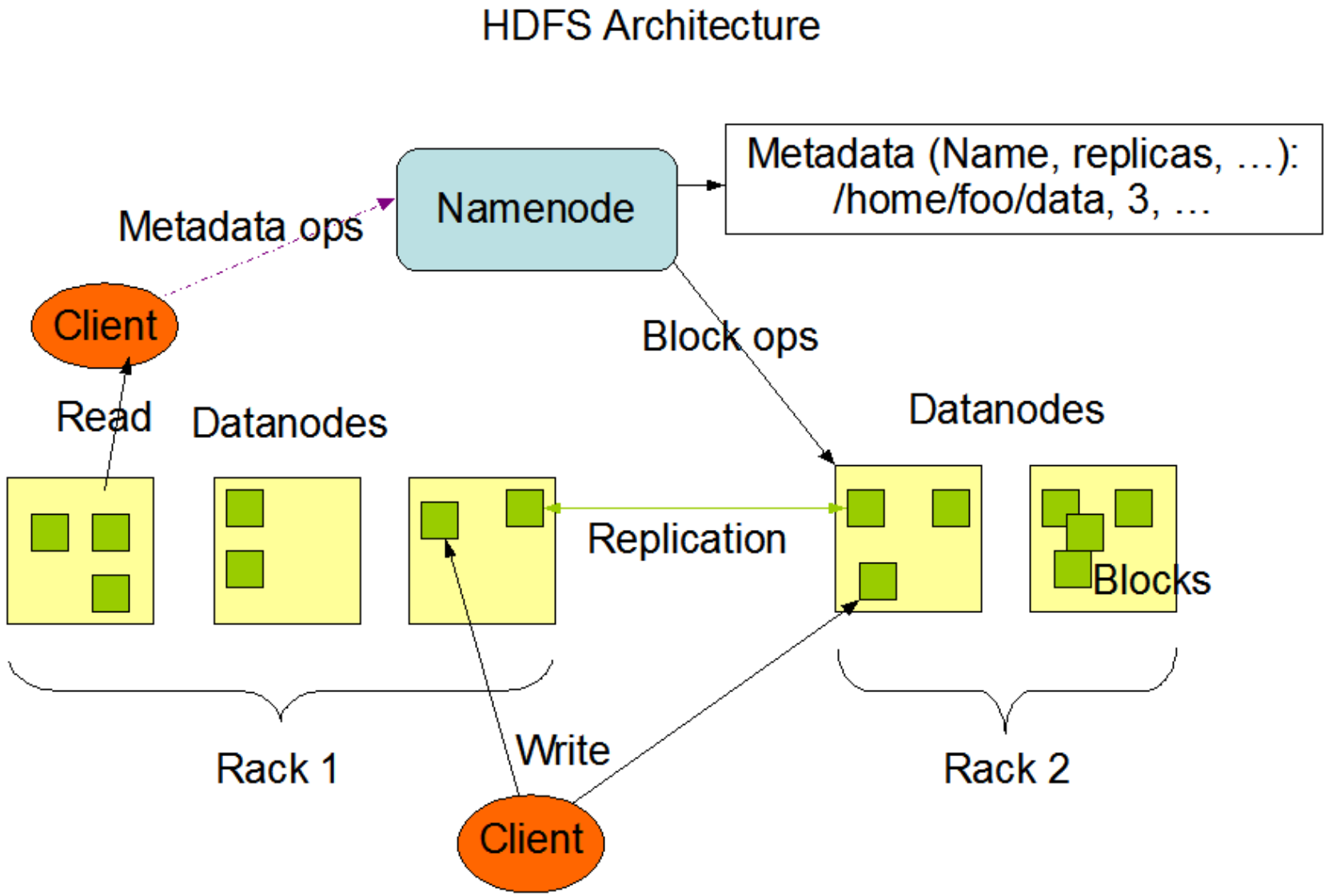




Архитектура HDFS (Hadoop Distributed File System)

- **Hadoop Distributed File System** (HDFS) – распределенная файловая система (отказоустойчивая, горизонтально масштабируемая, простая)
- Модель "write-once-read-many"
- Архитектура Master/Slave
- HDFS-кластер: 1 NameNode + *N* DataNode (на каждом узле)
- NameNode – сервер метаданных (file system namespace, контроль доступа к файлам, операции open, close, rename)
- DataNode – сервер управления локальным хранилищем (обрабатывает запросы на чтение/запись к локальному хранилищу)
- Файл разбивается на блоки фиксированного размера и распределяется по нескольким DataNode
- Размер файла может превышать размер жесткого диска одного узла!

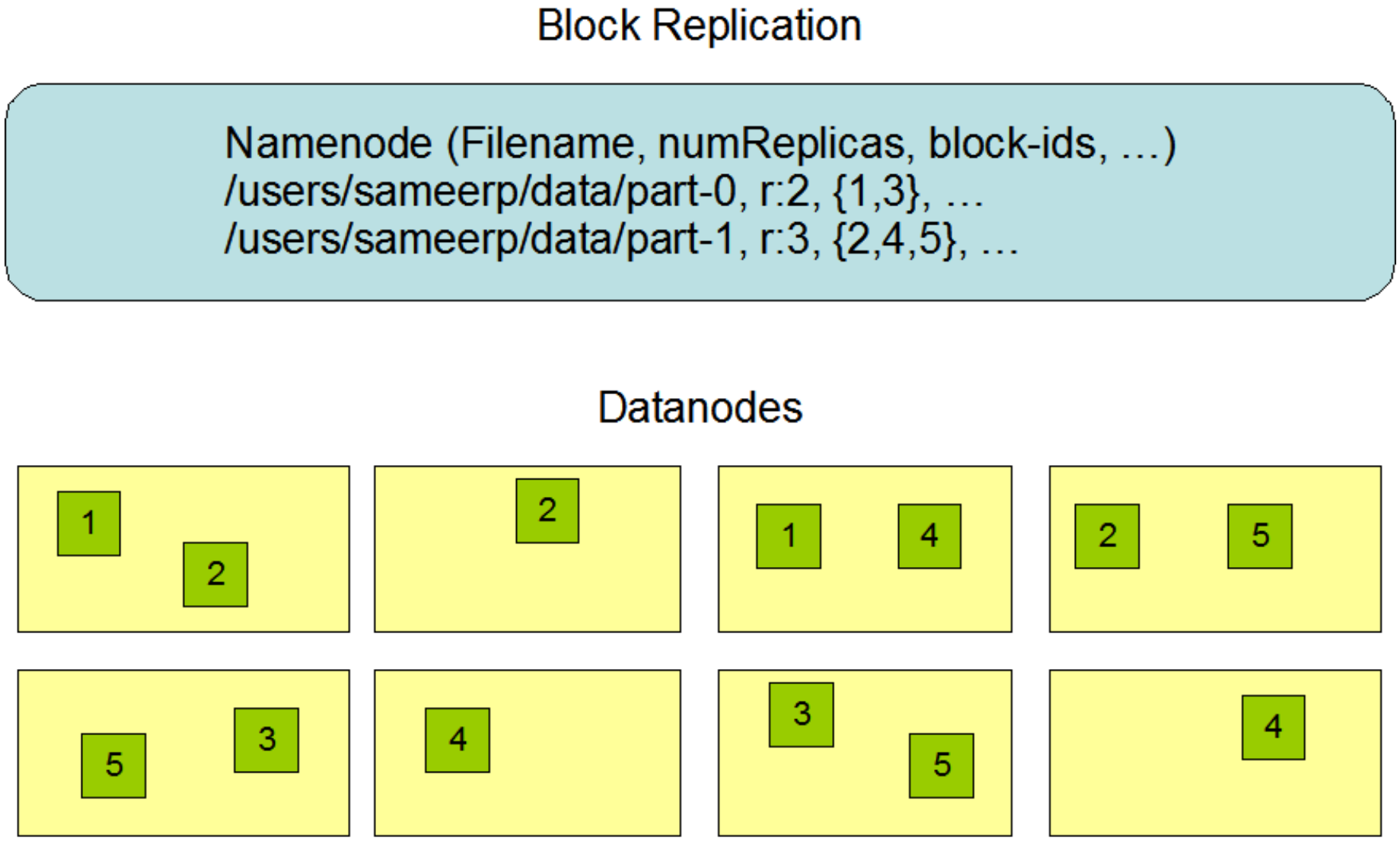
Архитектура HDFS (Hadoop Distributed File System)



Репликация данных (Data replication)

- Файл разбивается на блоки одинакового размера
  (за исключением последнего, по умолчанию 128 MiB)
- **Отказоустойчивость**
  - для каждого блока создается несколько реплик на разных узлах
    (настраиваемый параметр для каждого файла, по умолчанию 3)
  - NameNode периодически принимает от DataNode информацию о их состоянии
    (включая список блоков каждого узла)
  - Чтение осуществляется с ближайшей реплики
- **Целостность данных**: для каждого блока рассчитывается контрольная сумма,
  она проверяется при чтении блока (если не совпала можно прочитать с другой реплики)
- Если добавили новый узел или на диске узла осталось мало места,
  запускается процедура перераспределения блоков (rebalancing)

Архитектура HDFS (Hadoop Distributed File System)

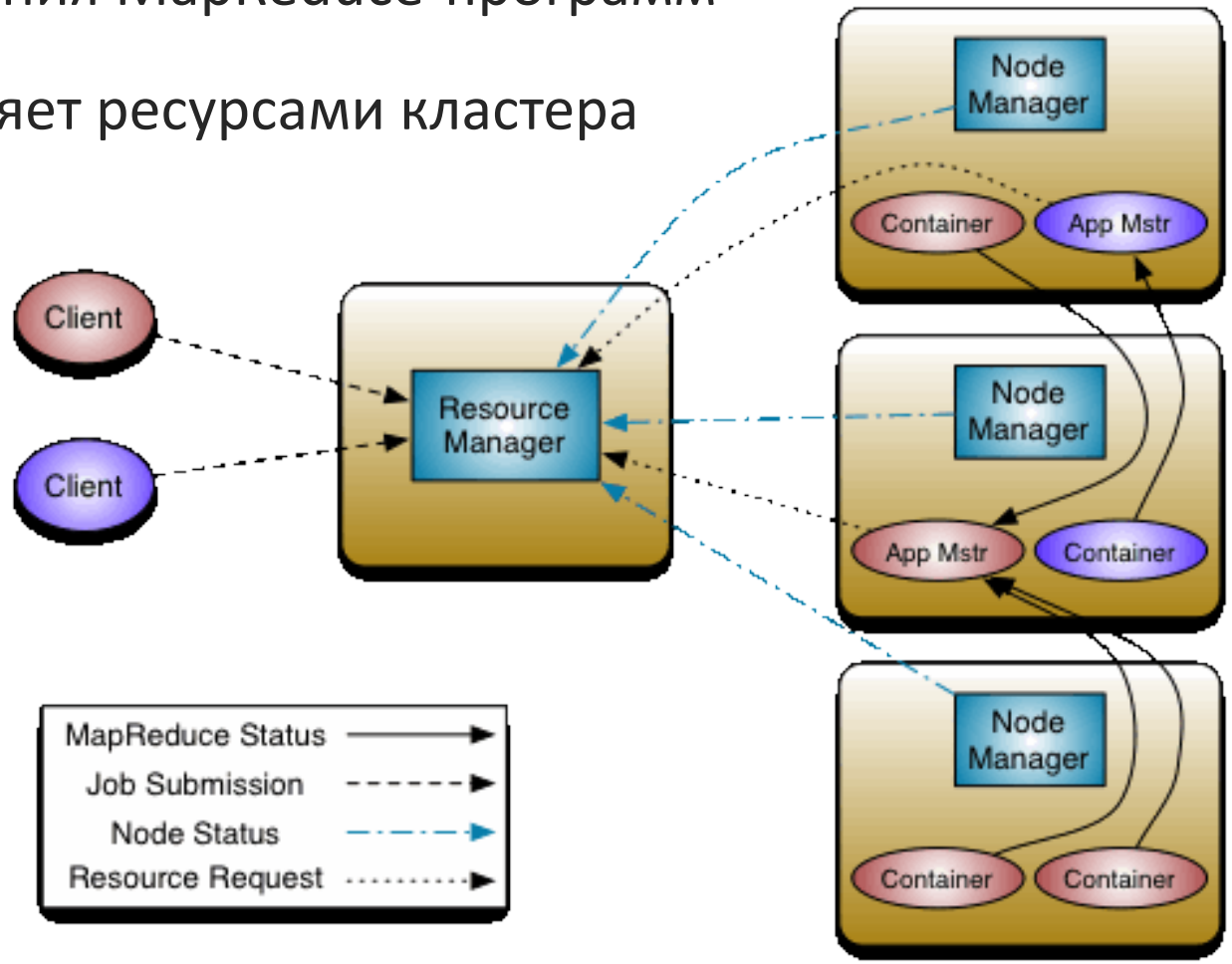


Работа с HDFS

- Командная строка
- Web-интерфейс (просмотр состояния)

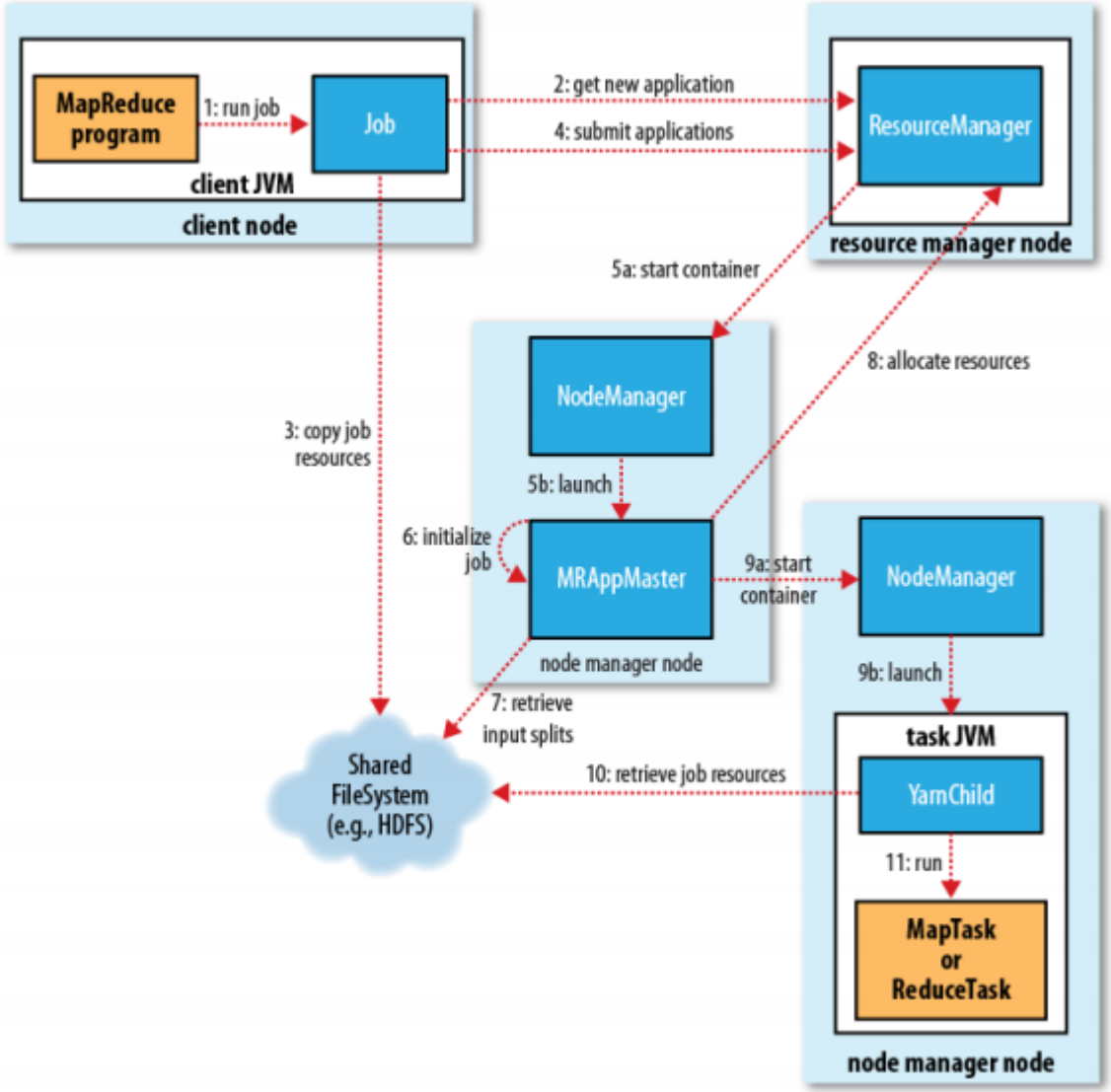
Apache YARN (Yet Another Resource Negotiator)

- **Apache YARN** – подсистема управления вычислительными ресурсами Hadoop-кластера и процессом выполнения MapReduce-программ
- Глобальный **ResourceManager** – управляет ресурсами кластера (Scheduler, ApplicationsManager)
- На каждом узле NodeManager



Hadoop MapReduce

- **ResourceManager** выделяет контейнер и запускает на нем п**роцесс MRAppMaster** для управления одной MapReduce-задачей
- **MRAppMaster** взаимодействует с **ResourceManager, NodeManager** и запускает на узлах задачи (map, reduce)



Tom White. Hadoop: **The Definitive Guide**, 3rd Edition, O'Reilly Media, 2012.

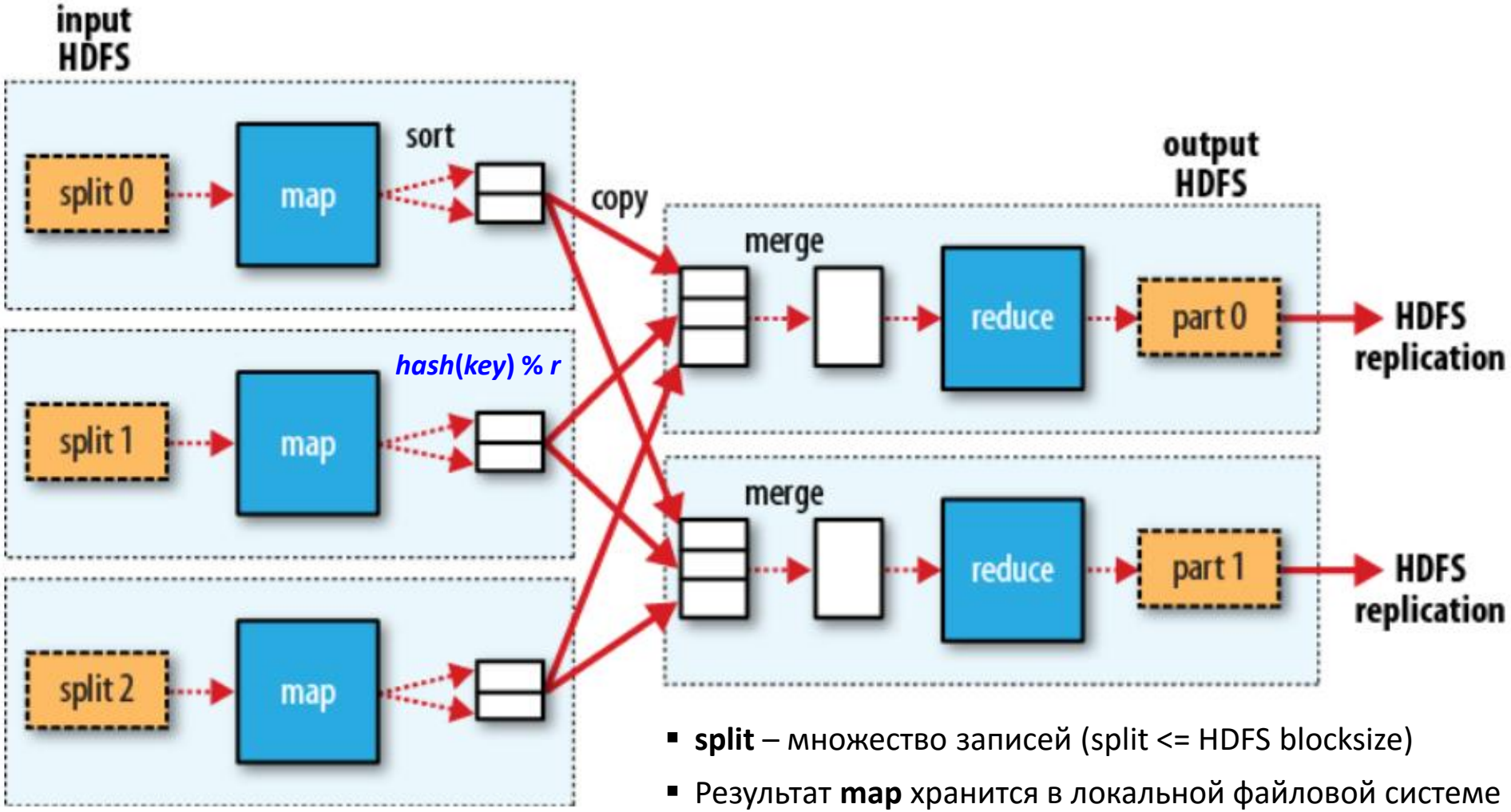
Разработка MapReduce-программ для Hadoop

- **Java**
  - Стандартный Java API
  - http://hadoop.apache.org/docs/current/api/index.html
  - Java-пакет **org.apache.hadoop.mapred**
- **Любые языки и скрипты**
  - Hadoop Streaming
- **C++ (и другие языки через SWIG)**
  - Hadoop Pipes

Общая структура MapReduce-программы

- Реализации Mapper и Reducer (Partitioner, Combiner...)
- Код формирования и запуска задания
- Ожидание результата или выход

Apache Hadoop Dataflow



Tom White. Hadoop: **The Definitive Guide**, 3rd Edition, O'Reilly Media, 2012.

- **split** – множество записей (split <= HDFS blocksize)
- Результат **map** хранится в локальной файловой системе (не в HDFS)
- Результат **map** разбивается (partitioned) – каждой reduce-задаче достается часть результата всех map-задач

Класс Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>

- **org.apache.hadoop.mapreduce.Mapper**
- Методы
  - void **setup**(Mapper.Context context)
    Called once at the beginning of the task
  - void **run**(Mapper.Context context)
  - void **map**(K1 key, V1 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

Реализация по умолчанию

```
protected void map(KEYIN key, VALUEIN value, Context context)
    throws IOException, InterruptedException {
    context.write((KEYOUT)key, (VALUEOUT)value);
}

protected void cleanup(Context context) throws IOException, InterruptedException {
    // NOTHING
}

public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    try {
        while (context.nextKeyValue()) {
            map(context.getCurrentKey(), context.getCurrentValue(), context);
        }
    } finally {
        cleanup(context);
    }
}
```

WordCount: Mapper

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {

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

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

Класс Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>

- **org.apache.hadoop.mapreduce.Reducer**
- Методы
  - void **setup**(Reducer.Context context)
  - void **run**(Reducer.Context context)
  - void **reduce**(K2 key, Iterable<V2> values, Reducer.Context context)
    This method is called once for each key
  - void **cleanup**(Reducer.Context context)

Реализация по умолчанию

```
protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)
    throws IOException, InterruptedException {
    for (VALUEIN value : values) {
        context.write((KEYOUT)key, (VALUEOUT)value);
    }
}

public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    try {
        while (context.nextKey()) {
            reduce(context.getCurrentKey(), context.getValues(), context);
            // If a back up store is used, reset it
            Iterator<VALUEIN> iter = context.getValues().iterator();
            if (iter instanceof ReduceContext.ValueIterator) {
                ((ReduceContext.ValueIterator<VALUEIN>)iter).resetBackupStore();
            }
        }
    } finally {
        cleanup(context);
    }
}
```

WordCount: Reducer

```
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

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

Конфигурация и запуск задания

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Входные и выходные данные

- (input) → (k1, v1) → map → (k2, v2) →
  combine → (k2, v2) → reduce → (k3, v3) → (output)

- **Базовые интерфейсы**
  - ❑ Входные данные: InputFormat
  - ❑ Выходные данные: OutputFormat
  - ❑ Ключи: WritableComparable
  - ❑ Значения: Writable

Типы данных (оптимизированы для сериализации)

- **Пакет org.apache.hadoop.io**
- Text
- BooleanWritable
- IntWritable
- LongWritable
- FloatWritable
- DoubleWritable
- BytesWritable
- ArrayWritable
- MapWritable

Класс InputFormat<K, V>

- Разбивает входные файлы на логические блоки InputSplit

- **TextInputFormat** (по умолчанию)
  - <LongWritable, Text> = <byte_offset, line>
- **KeyValueTextInputFormat**
  - <Text, Text>
  - Текстовый файл со строками вида: key [tab] value
- **SequenceFileInputFormat<K, V>**
  - Двоичный формат с поддержкой сжатия
- …

Класс OutputFormat<K, V>

- **TextOutputFormat<K, V>**
  - ❑ Текстовый файл со строками вида: key [tab] value
- **SequenceFileOutputFormat**

Запуск примера на кластере

- Компилируем и создаем JAR-файл
- Копируем JAR и исходные данные в домашнюю директорию на кластере по SCP
- Заходим на кластер по SSH
- Загружаем исходные данные в HDFS
- Запускаем MapReduce-задание
- Выгружаем результаты из HDFS

Компилируем WordCount.java

```
$ javac -classpath `hadoop classpath` WordCount.java
$ jar -cvf wordcount.jar .
$ ls
WordCount.class
WordCount$IntSumReducer.class
WordCount$TokenizerMapper.class
wordcount.jar
```

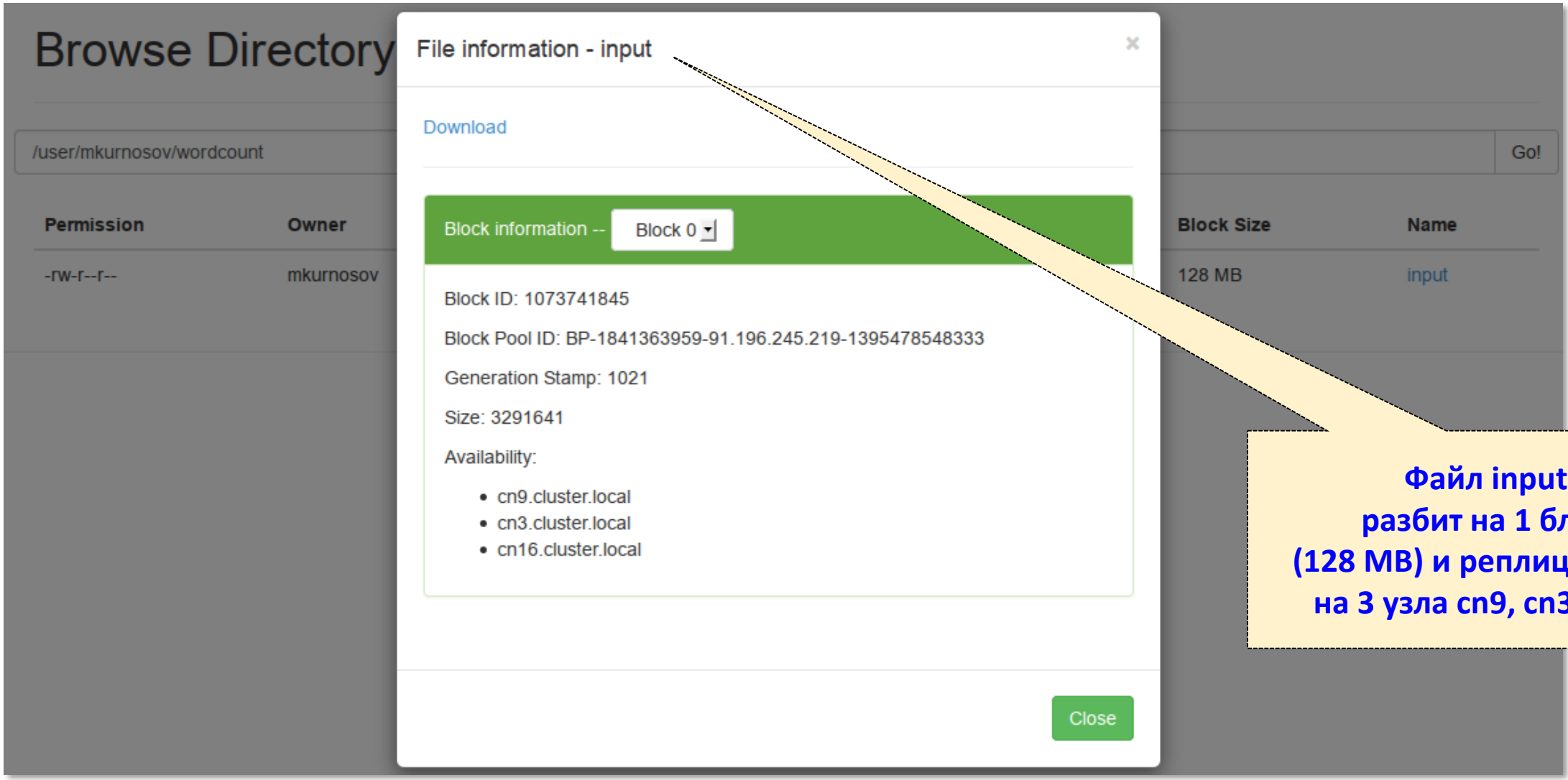
Загрузка данных в HDFS

- **hdfs dfs -put <local_dir> <hdfs_dir>**

```
# Создаем в HDFS каталог
$ hdfs dfs -mkdir ./wordcount

# Копируем файл в HDFS
$ hdfs dfs -put ~/data.txt ./wordcount/input
```

- Файл input будет разбит на блоки и распределен по узлам кластера
- Каждый блок будет реплицирован на несколько узлов
  (по умолчанию 3 экземпляра каждого блока)

Реплики блоков файла input



Файл input разбит на 1 блок (128 MB) и реплицирован на 3 узла cn9, cn3, cn16

Запуск задания

```
$ hadoop jar ./wordcount.jar pdccourse.lecture6.WordCount \
        -D mapred.reduce.tasks=1 \
        ./wordcount/input ./wordcount/output
```

```
Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /opt/hadoop-
2.3.0/lib/native/libhadoop.so.1.0.0 which might have disabled stack guard. The VM will try to fix the stack guard
now.
14/03/25 11:29:55 INFO mapreduce.Job: Running job: job_local670227185_0001
14/03/25 11:29:55 INFO mapred.LocalJobRunner: Waiting for map tasks
14/03/25 11:29:55 INFO mapred.LocalJobRunner: Starting task: attempt_local670227185_0001_m_000000_0
14/03/25 11:29:55 INFO mapred.MapTask: Processing split:
hdfs://frontend:50075/user/mkurnosov/wordcount/input:0+3291641
14/03/25 11:29:56 INFO mapred.MapTask: Starting flush of map output
14/03/25 11:29:56 INFO mapred.MapTask: Spilling map output
14/03/25 11:29:57 INFO mapred.LocalJobRunner: Finishing task: attempt_local670227185_0001_m_000000_0
14/03/25 11:29:57 INFO mapred.LocalJobRunner: map task executor complete.
14/03/25 11:29:57 INFO mapred.LocalJobRunner: Waiting for reduce tasks
14/03/25 11:29:57 INFO mapred.LocalJobRunner: Starting task: attempt_local670227185_0001_r_000000_0
14/03/25 11:29:57 INFO mapred.LocalJobRunner: Finishing task: attempt_local670227185_0001_r_000000_0
```

- В результате выполнения будет создан каталог ./wordcount/output

Количество задач

- **Maps**
  - Определяется количеством блоков во входных файлах, размером блока, параметрами mapred.min(max).split.size, реализацией InputFormat
- **Reduces**
  - По умолчанию 1 (на кластере переопределено)
  - Опция «-D mapred.reduce.tasks=N» или метод «job.setNumReduceTasks(int)»
  - Обычно подбирается опытным путем
  - Время выполнения reduce должно быть не менее минуты
  - 0, если фаза Reduce не нужна

Выгружаем данные из HDFS в локальную файловую систему

- **hdfs dfs -get <hdfs_src> <local_dst>**

```
$ hdfs dfs -ls ./wordcount/output
Found 2 items
-rw-r--r--   3 mkurnosov supergroup          0 2014-03-25 11:29 wordcount/output/_SUCCESS
-rw-r--r--   3 mkurnosov supergroup     467841 2014-03-25 11:29 wordcount/output/part-r-00000

$ hdfs dfs -get ./wordcount/output/part* result

$ hdfs dfs -cat ./wordcount/output/part*
"'Come  1
"'Dieu  1
"'Dio   1
"'From  1
"'Grant 1
"'I     4
"'No    1
```

Количество задач

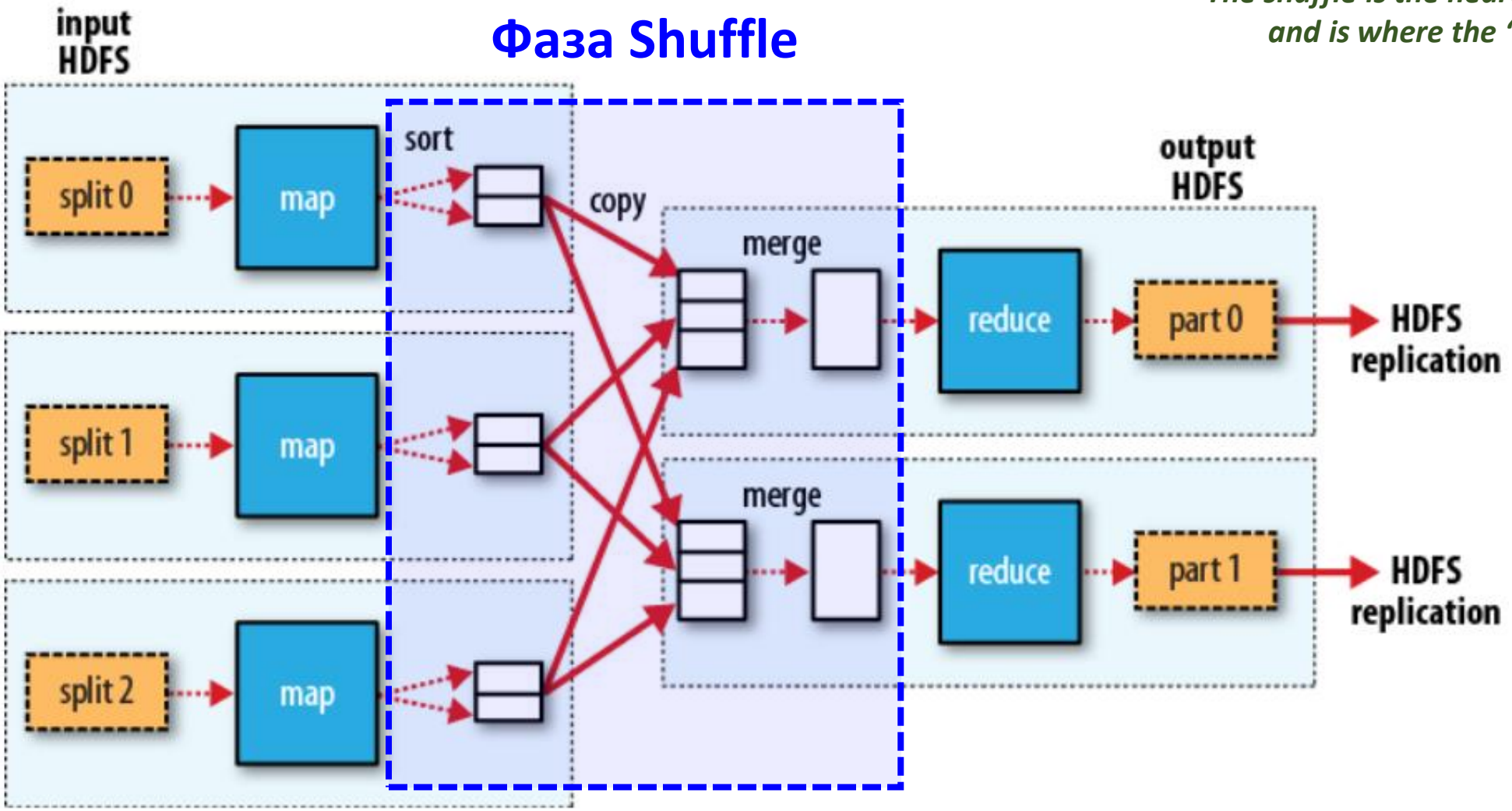
- **Maps**
  - Определяется количеством блоков во входных файлах, размером блока, параметрами mapred.min(max).split.size, реализацией InputFormat
  - Желательно время выполнения map >= 1 мин.
  - 10-100 maps per node
- **Reduces**
  - По умолчанию 1 (на кластере переопределено)
  - Опция «-D mapred.reduce.tasks=N» или метод «job.setNumReduceTasks(int)»
  - Обычно подбирается опытным путем
  - Время выполнения reduce желательно >= 1 мин.
  - 0, если фаза Reduce не нужна
  - Количество reduce: nodes * 0.95 или nodes * 1.75

Повторный запуск

- Перед каждым запуском надо удалять из HDFS output-директорию
  ```
  $ hdfs dfs -rm -r ./wordcount/output
  ```
- Или каждый раз указывать новую output-директорию
- **Если данные больше не нужны удаляйте их из HDFS!**

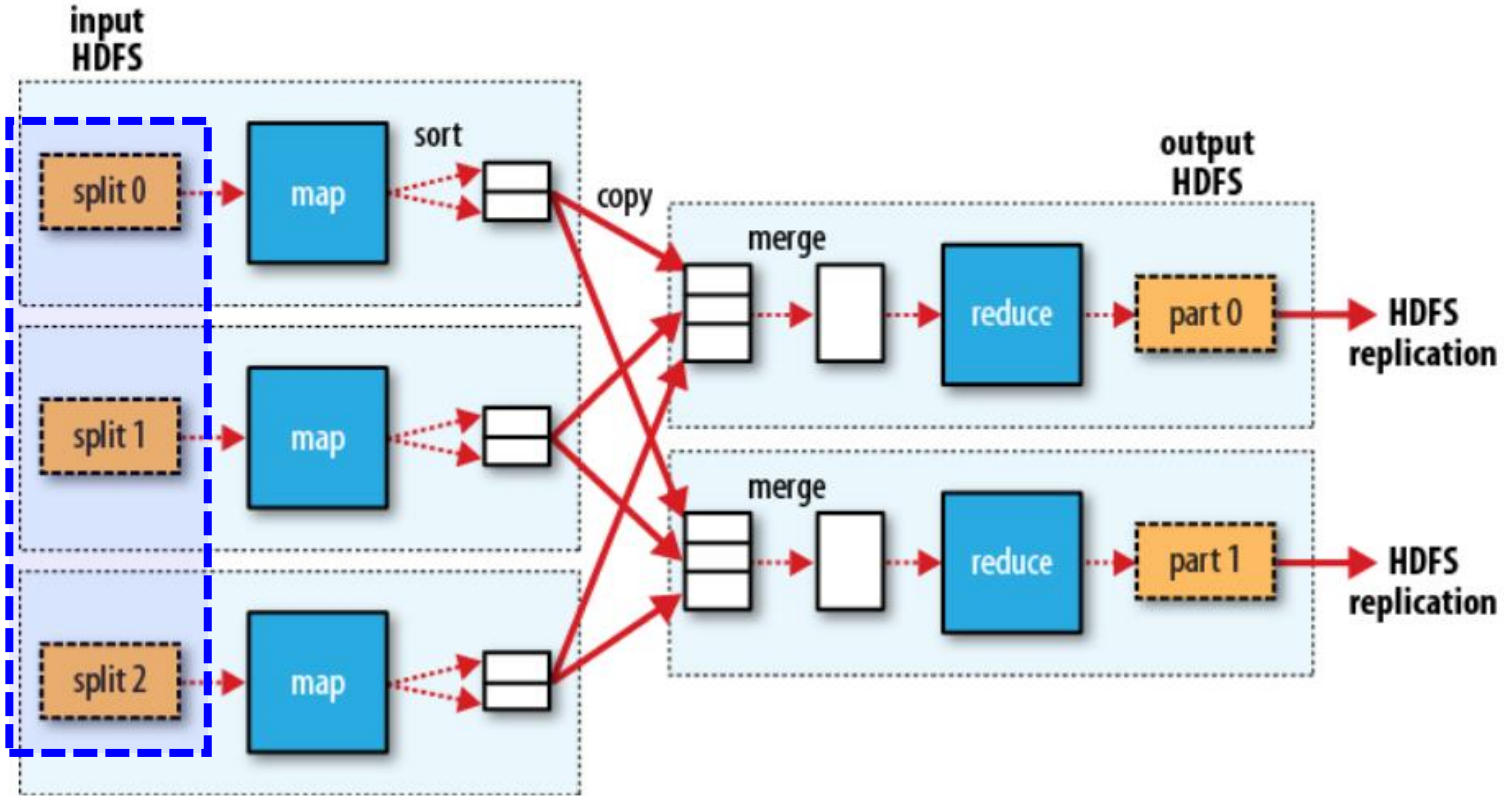
Apache Hadoop Dataflow

The shuffle is the heart of MapReduce and is where the "magic" happen



Tom White. Hadoop: **The Definitive Guide**,3rd Edition, O'Reilly Media, 2012.

Apache Hadoop: input



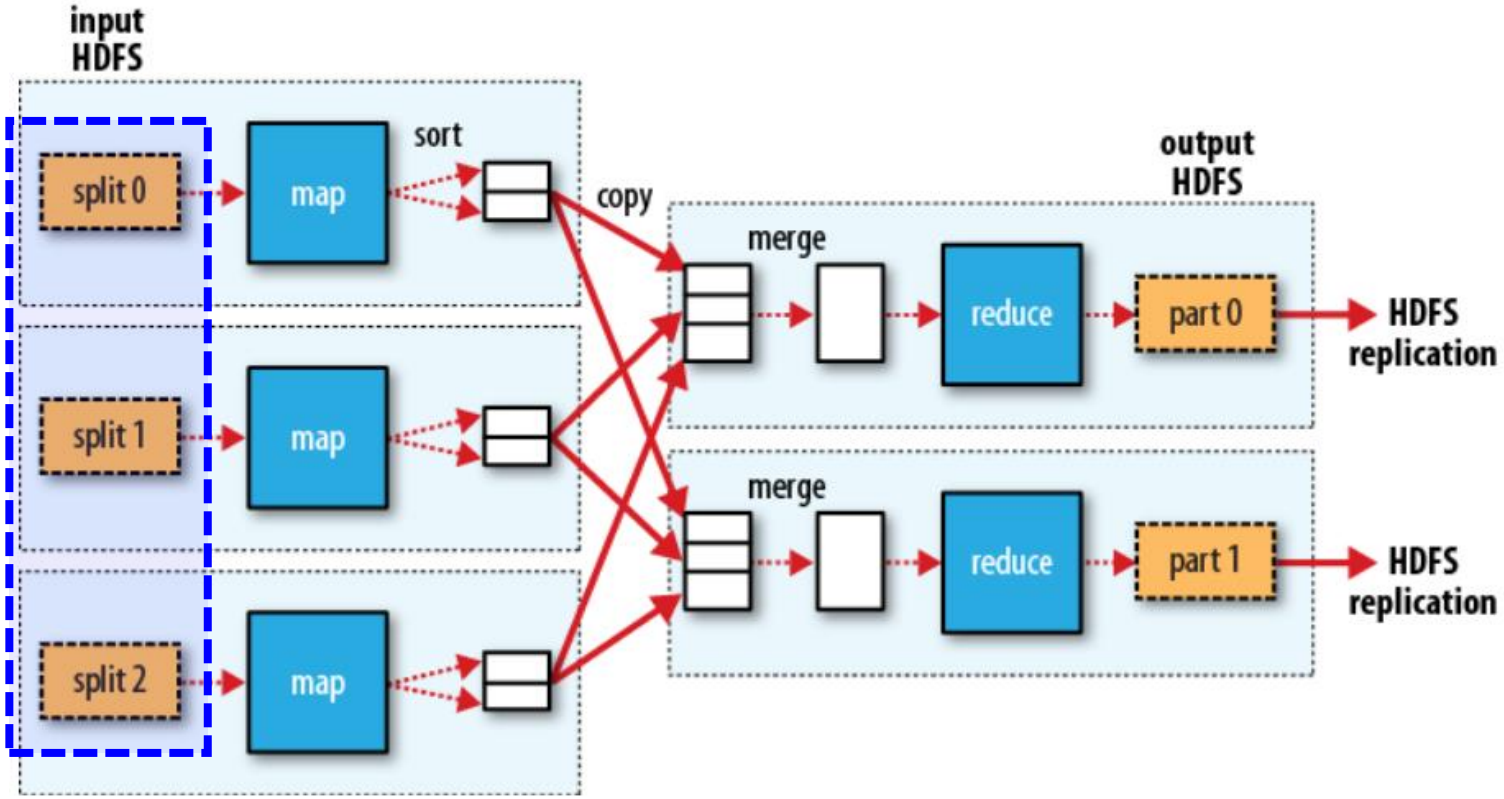
Tom White. Hadoop: **The Definitive Guide**, 3rd Edition, O'Reilly Media, 2012.

- Входные данные **разбиваются** на части split0, split1, …, split *M*
- Каждый split **обрабатывается** отдельной map-задачей
- Алгоритм вычисления split size реализован в InputFormat.computeSplitSize()
- Если файлы “маленькие” для каждого будет создана своя map-задача
- Эффективнее обрабатывать несколько больших файлов

```
// FileInputFormat.java [1]
long computeSplitSize(long blockSize, long minSize, long maxSize) {
    return Math.max(minSize, Math.min(maxSize, blockSize));
}
```

[1] hadoop-src/hadoop-mapreduce-project/hadoop-mapreduce-client/hadoop-mapreduce-client-core/src/main/java/org/apache/hadoop/mapreduce/lib/input

Apache Hadoop: input

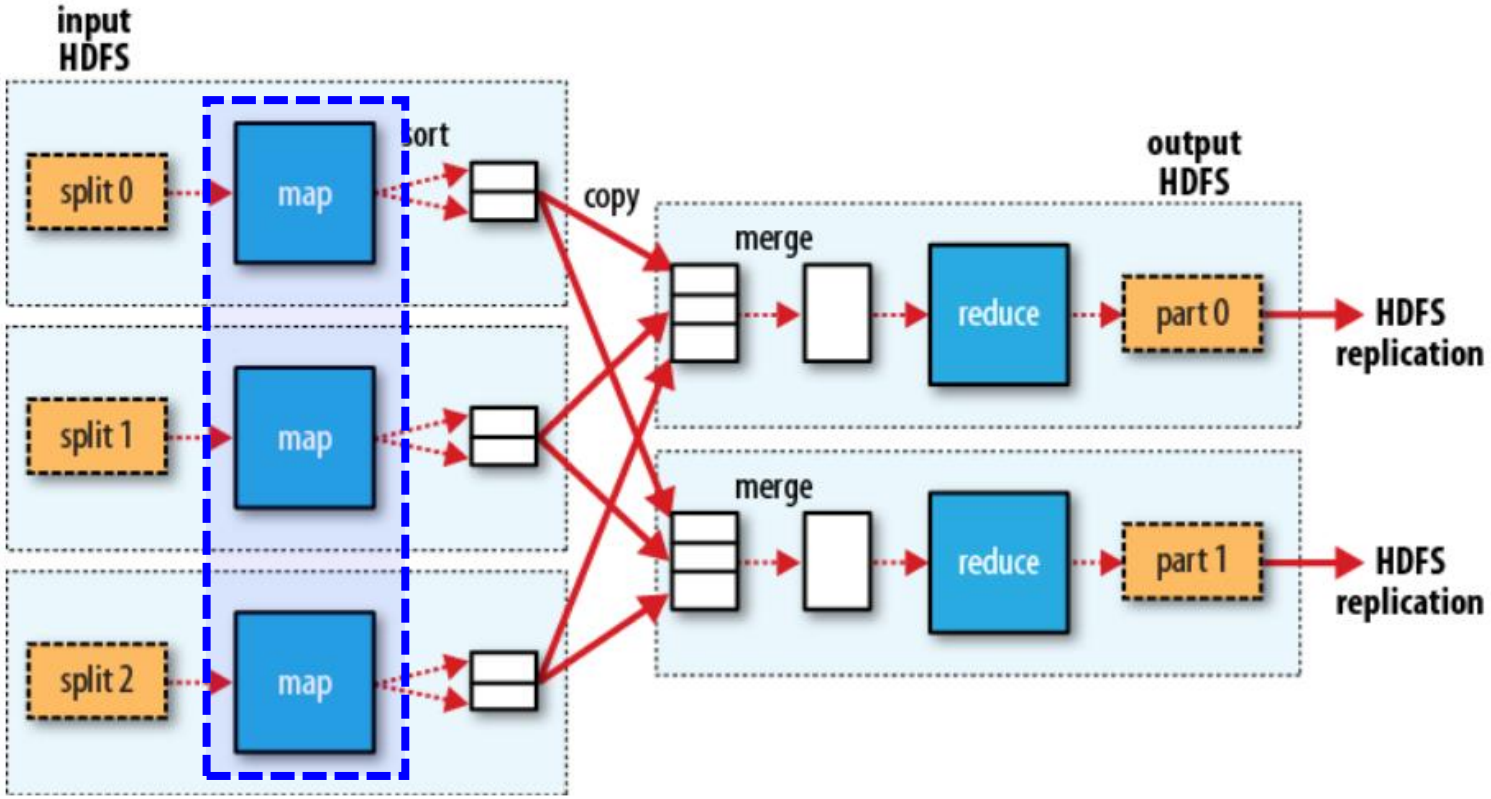


Пример

Требуется обработать 1 GiB данных

- **Данные в файле 1 GiB**
  Файл разбивается на 8 частей
  по 128 MiB => 8 map-задач
- **1024 файла по 1 MiB**
  1024 частей по 1 MiB =>
  1024 map-задач
  (накладные расходы на запуск задач будут значительными)
- **Эффективнее обрабатывать несколько больших файлов**
- Hadoop может объединить маленькие файлы в один split – класс CombineFileInputFormat

Apache Hadoop: map



Split (часть файла)

Лодка плыла по воде.
Солнце стояло высоко.
...

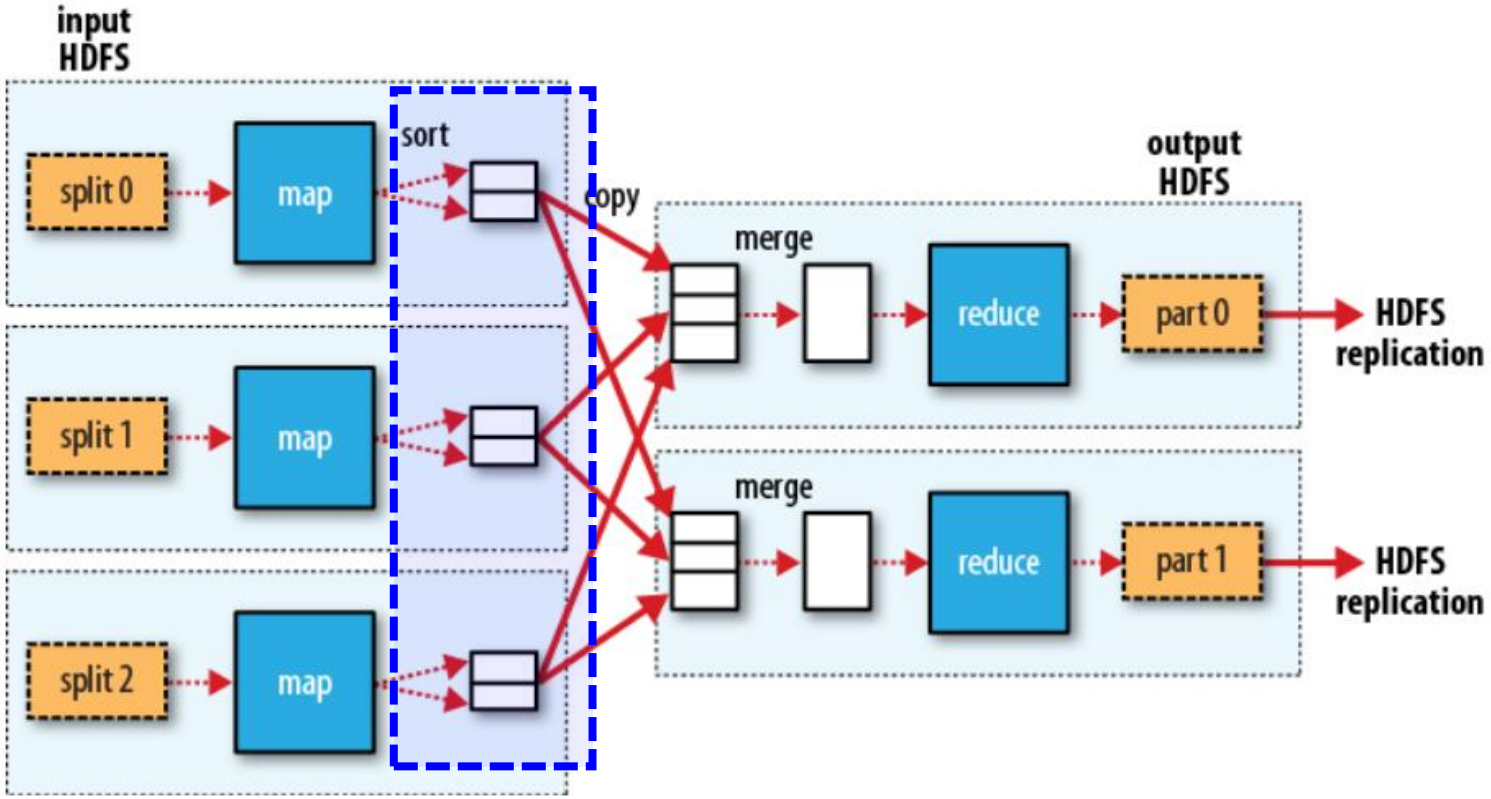
(*k*1, *v*1)

(0, Лодка плыла по воде.)
(21, Солнце стояло высоко.)

map(*k*1, *v*1) -> (*k*2, *v*2)

- **Split** – это совокупность записей (records)
- Метод **RecordReader.nextKeyValue()** реализует чтение split и возвращает (*k*1, *v*1), они передаются в **map**
- По умолчанию используется **LineRecordReader.nextKeyValue**() – читает файл по строкам:
  - *k*1 – смещение первого символа строки в файле (offset)
  - *v*1 – строка (line)

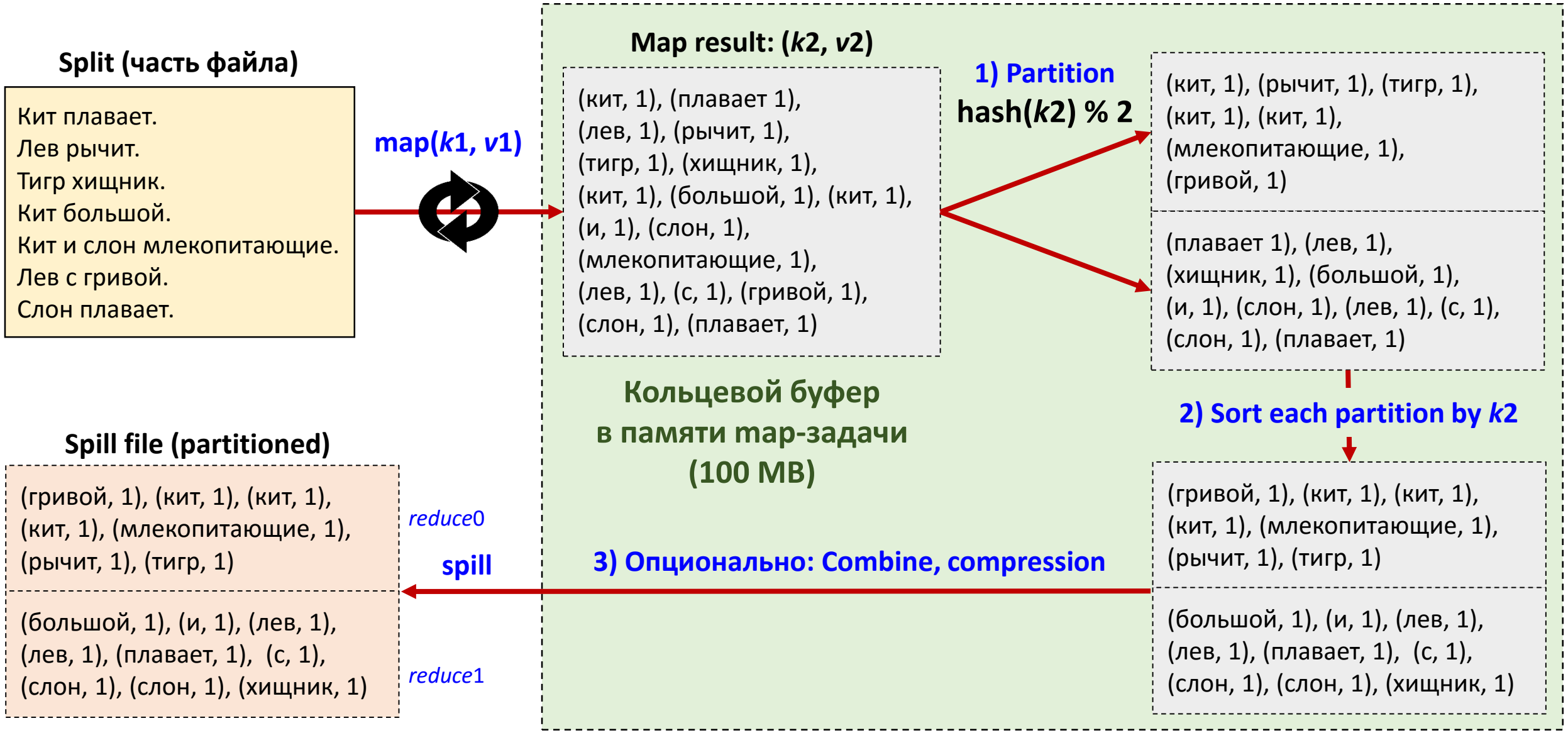
Apache Hadoop: map



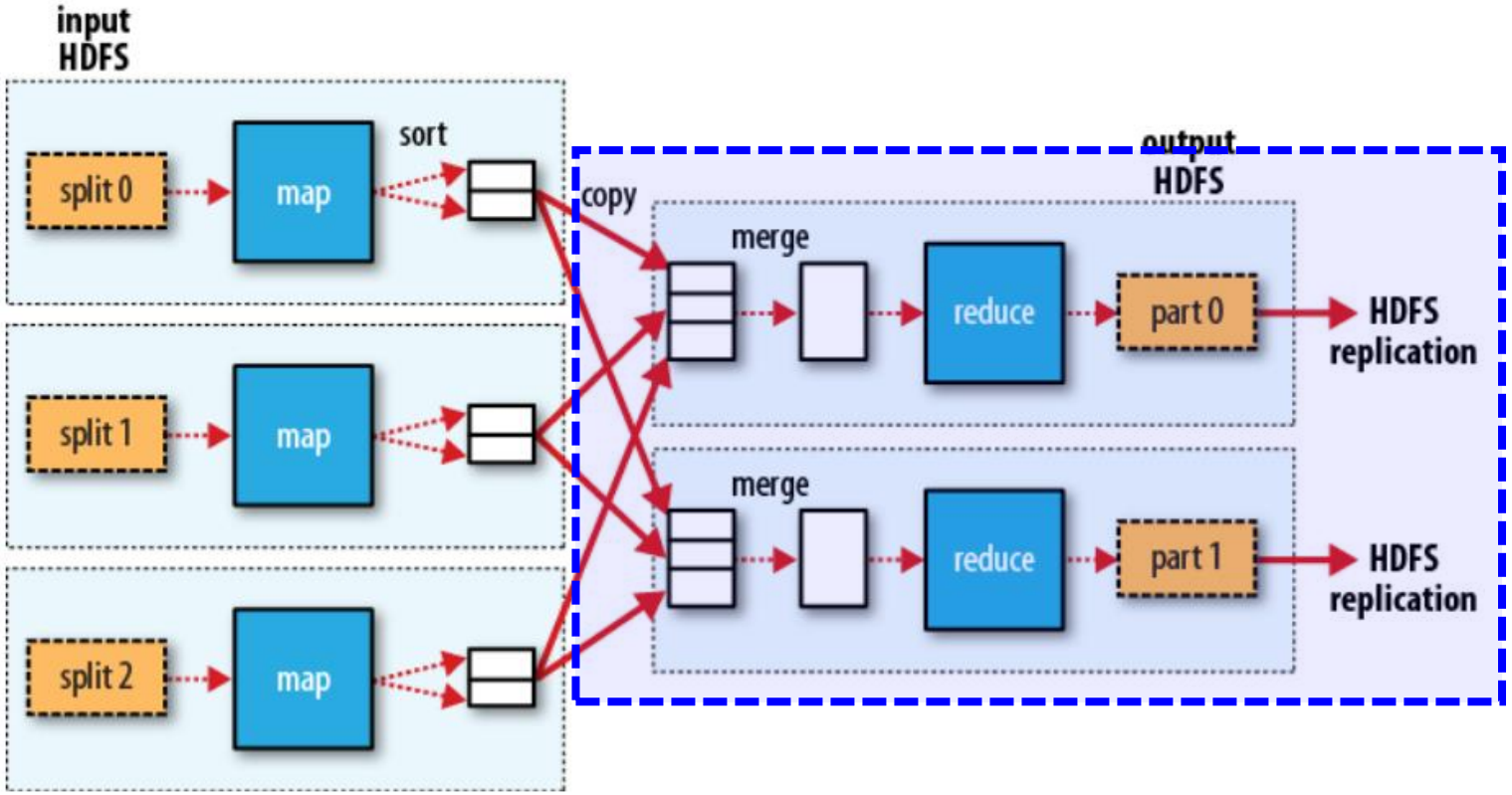
map(*k*1, *v*1) -> (*k*2, *v*2)

- Каждая map-задача записывает пары (*k*2, *v*2) в свой **циклический буфер** в памяти (100 MB, io.sort.mb)
- Если **буфер заполнен** на величину порогового значения (80%, io.sort.spill.percent) создается фоновый поток, который:
  - ***partition***: распределяет пары по подмножествам: *hash*(*k*2) % *nreduces*
  - ***sort***: сортирует в каждом подмножестве пары по ключам *k*2
  - ***combine***: если указан combiner он запускается для результата сортировки
  - результаты сбрасываются (spill) на диск в spill-файл
- Spill-файлы сливаются в один (с соблюдением распределения пар по reduce-задачам)

Apache Hadoop: map (WordCount)



Apache Hadoop: reduce



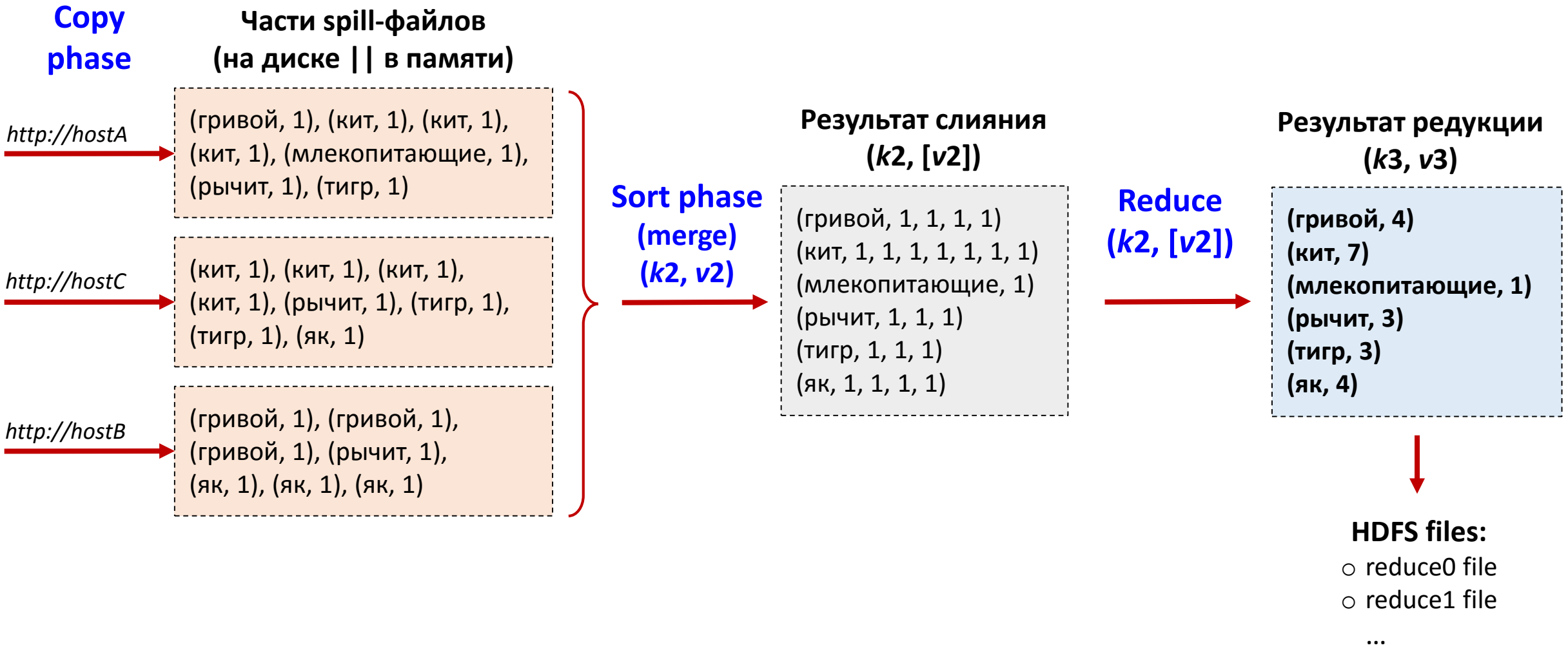
Copy phase

- Reduce-задача обращается к узлам map-задач и копирует по сети (HTTP) соответствующие части spill-файлов (на диск или в память)

Sort phase (merge)

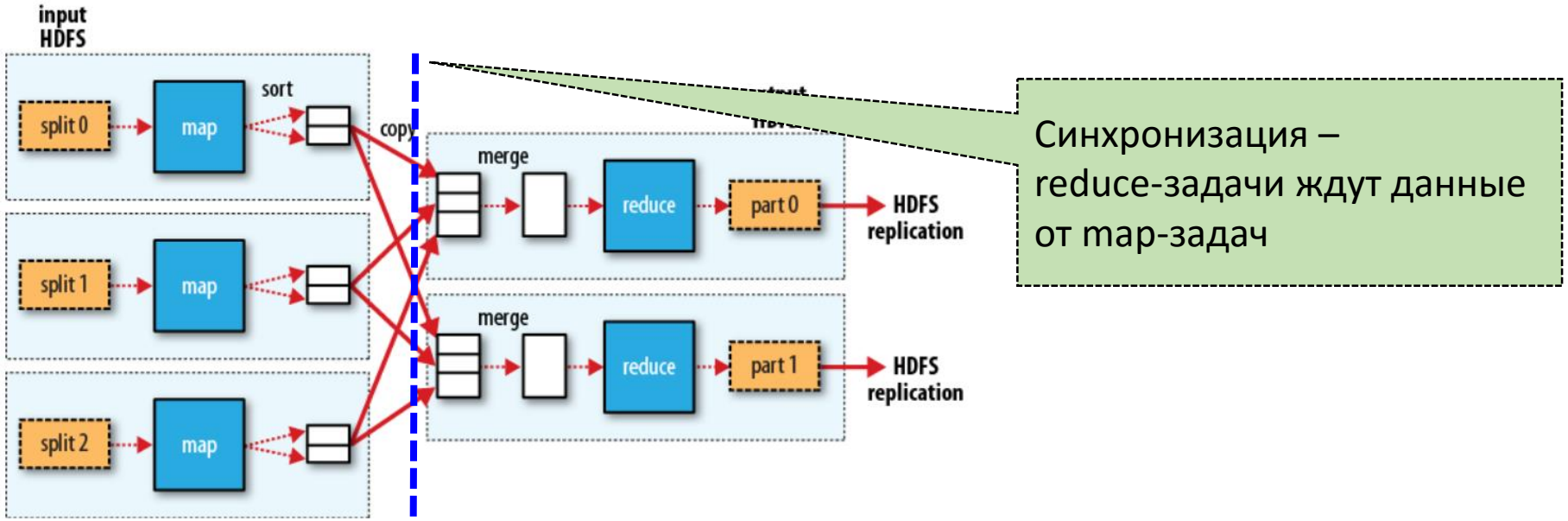
- Загруженные части spill-файлов сливаются за несколько раундов (merge factor)
- В конце фазы sort имеется merge factor файлов (10, mapreduce.task.io.sort.factor)
- Результаты финального раунда передаются в функцию reduce
- Результаты записываются в HDFS

Apache Hadoop: reduce (WordCount)



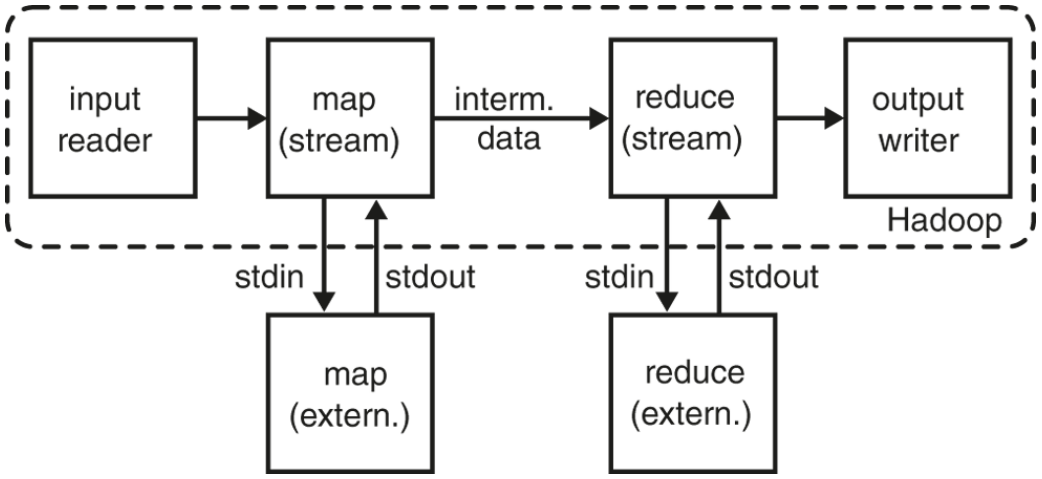
Ограничения MapReduce

- “Жесткая” модель параллельных вычислений
- Синхронизация между задачами только в фазе Shuffle (reduce-задачи ждут данные map-задач)
- Ограниченный контроль над тем, где, когда и какие данные будет обрабатывать конкретная задача



Hadoop Streaming

- Позволяет использовать в качестве реализаций Map и Reduce произвольные программы и скрипты
- Обмен данными между Hadoop и программой происходит через стандартные потоки ввода-вывода
  - ❑ stdin: входные данные в виде строк «key [SEPARATOR] value»
  - ❑ stdout: выходные данные в виде строк «key [SEPARATOR] value»



http://hadoop.apache.org/docs/stable/streaming.html

WordCount(2): Mapper (mapper.py)

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

for line in sys.stdin:
    words = line.lower().split()
    for word in words:
        print '%s\t%s' % (word, 1)
```

WordCount(2): Reducer (reducer.py)

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

current_word = None
current_count = 0

for line in sys.stdin:
    data = line.split('\t')
    word = data[0]
    count = int(data[1])

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

print '%s\t%s' % (current_word, current_count)
```

Запуск на кластере

```
$ hadoop jar /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.3.0.jar \
        -file ./mapper.py -mapper ./mapper.py -file ./reducer.py \
        -combiner ./reducer.py -reducer ./reducer.py \
        -input ./wordcount/input -output ./wordcount/output \
        -numReduceTasks 1
```

Hadoop Pipes

- C++ интерфейс для создания MapReduce-программ
- Взаимодействие программы и Hadoop осуществляется через сокет
- Ключи и значения передаются в программу в виде STL-строк
- Пример
  http://cs.smith.edu/dftwiki/index.php/Hadoop_Tutorial_2.2_--_Running_C%2B%2B_Programs_on_Hadoop

WordCount(3): Mapper

```
class WordCountMapper : public HadoopPipes::Mapper {
public:
    // Constructor: does nothing
    WordCountMapper(HadoopPipes::TaskContext& context) { }

    // Map function: receives a line, outputs (word,"1") to reducer
    void map(HadoopPipes::MapContext& context) {
        // get line of text
        string line = context.getInputValue();
        // split it into words
        vector<string> words = HadoopUtils::splitString(line, " ");
        // emit each word tuple (word, "1" )
        for (unsigned int i = 0; i < words.size(); i++) {
            context.emit(words[i], HadoopUtils::toString(1));
        }
    }
};
```

WordCount(3): Reducer

```
class WordCountReducer : public HadoopPipes::Reducer {
public:
    // Constructor: does nothing
    WordCountReducer(HadoopPipes::TaskContext& context) { }

    // Reduce function
    void reduce(HadoopPipes::ReduceContext& context) {
        int count = 0;
        // get all tuples with the same key, and count their numbers
        while (context.nextValue()) {
            count += HadoopUtils::toInt(context.getInputValue());
        }
        // emit (word, count)
        context.emit(context.getInputKey(), HadoopUtils::toString(count));
    }
};
```

WordCount(3): main

```
int main(intargc, char *argv[])
{
    return HadoopPipes::runTask(
                HadoopPipes::TemplateFactory<WordCountMapper,
                                             WordCountReducer>()
            );
}
```

Компиляция и запуск на кластере

- Компилировать необходимо на кластере (Makefile прилагается)
- После компиляции необходимо загрузить исполняемый файл в HDFS
  - `$ hdfs dfs -mkdir ./wordcount/bin`
  - `$ hdfs dfs -put ./wordcount ./wordcount/bin/wordcount`
- Запуск (в одну строку)

```
$ hadoop pipes -D hadoop.pipes.java.recordreader=true \
        -D hadoop.pipes.java.recordwriter=true \
        -input wordcount/input -output wordcount/output \
        -program wordcount/bin/wordcount -reduces 1
```