Модель программирования MapReduce (часть 2)

Михаил Георгиевич Курносов

Email: mkurnosov@gmail.com

WWW: http://www.mkurnosov.net

Курс «Параллельные и распределенные вычисления»

Школа анализа данных Яндекс (Новосибирск)

Весенний семестр, 2015

Модель программирования MapReduce (часть 2)

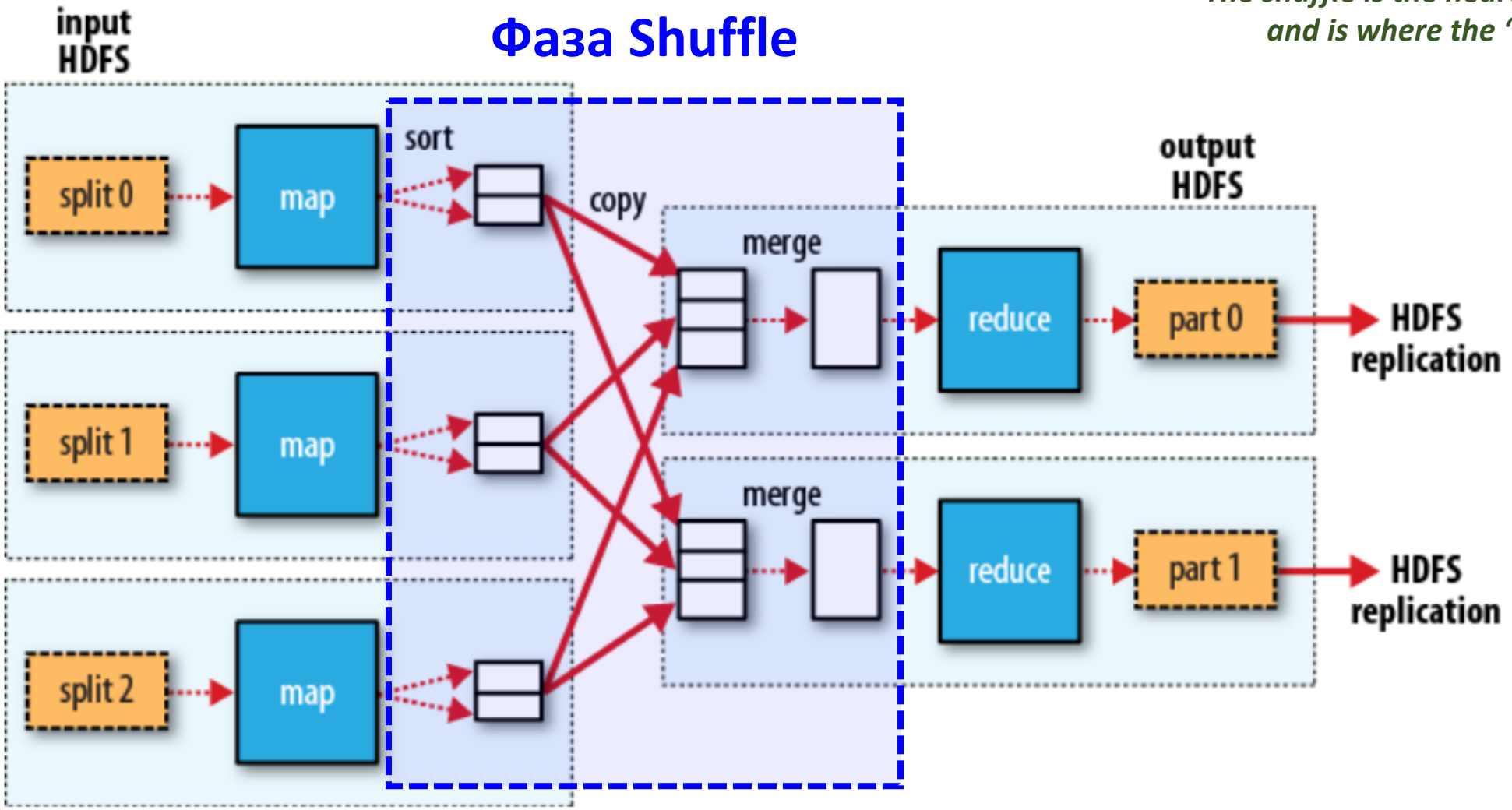
Курносов Михаил Георгиевич

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

Курс «Распределенная обработка информации»
Сибирский государственный университет телекоммуникаций и информатики
Весенний семестр, 2020

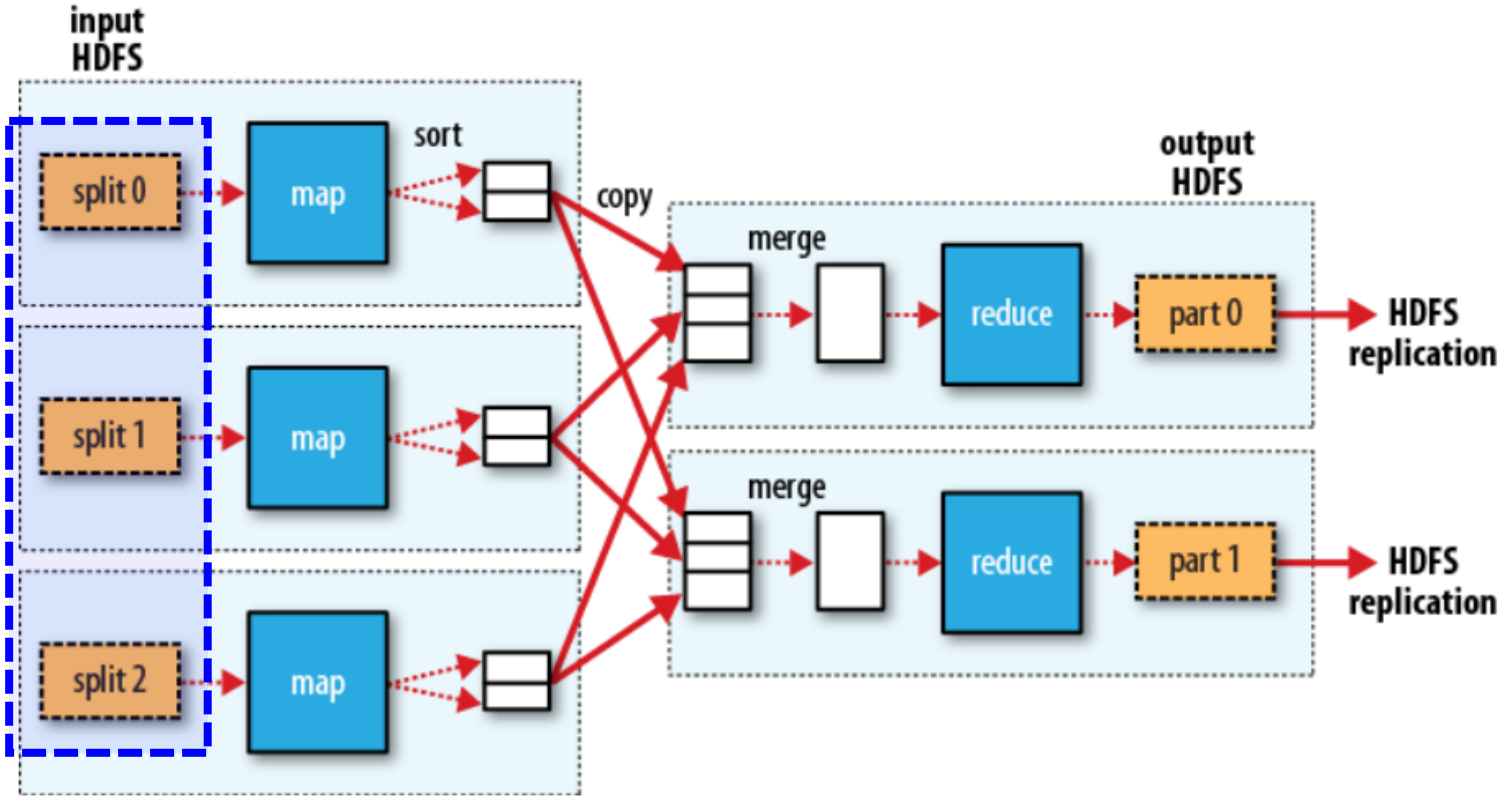
Apache Hadoop Dataflow

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



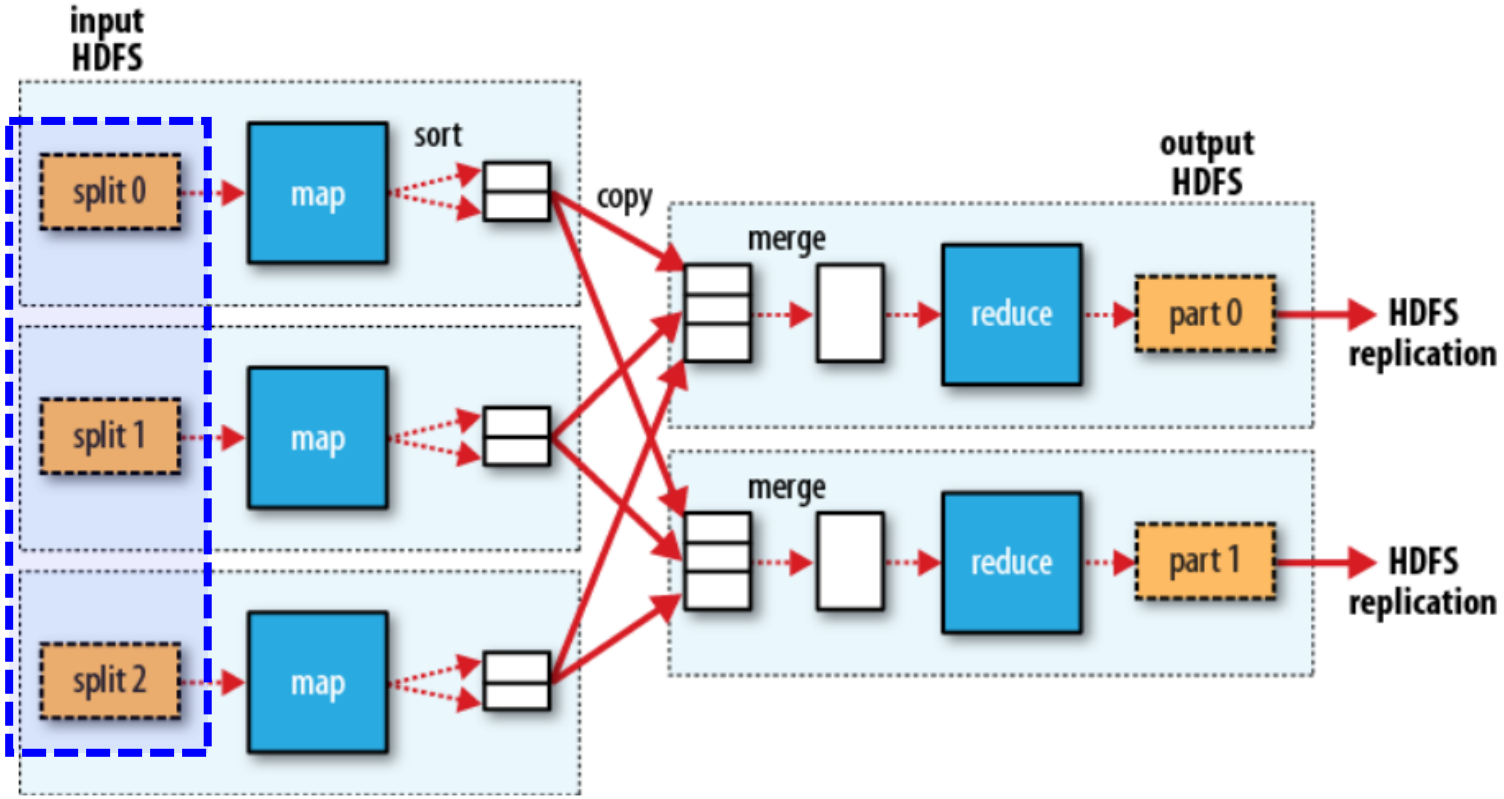
Tom White. Hadoop: **The Definitive Guide**, 3rd Edition, O'Reilly Media, 2012.

- 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

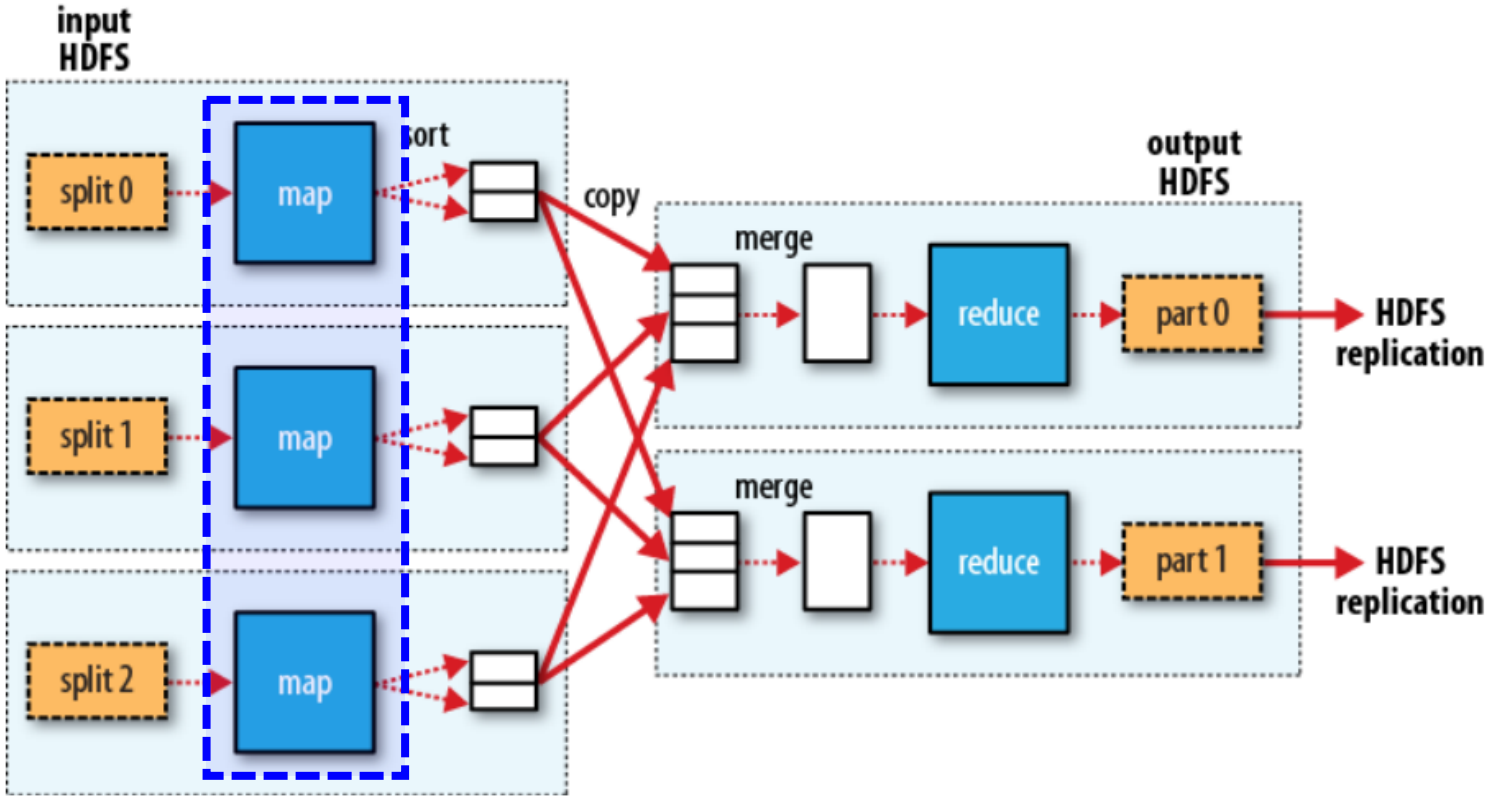


Пример

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.
...

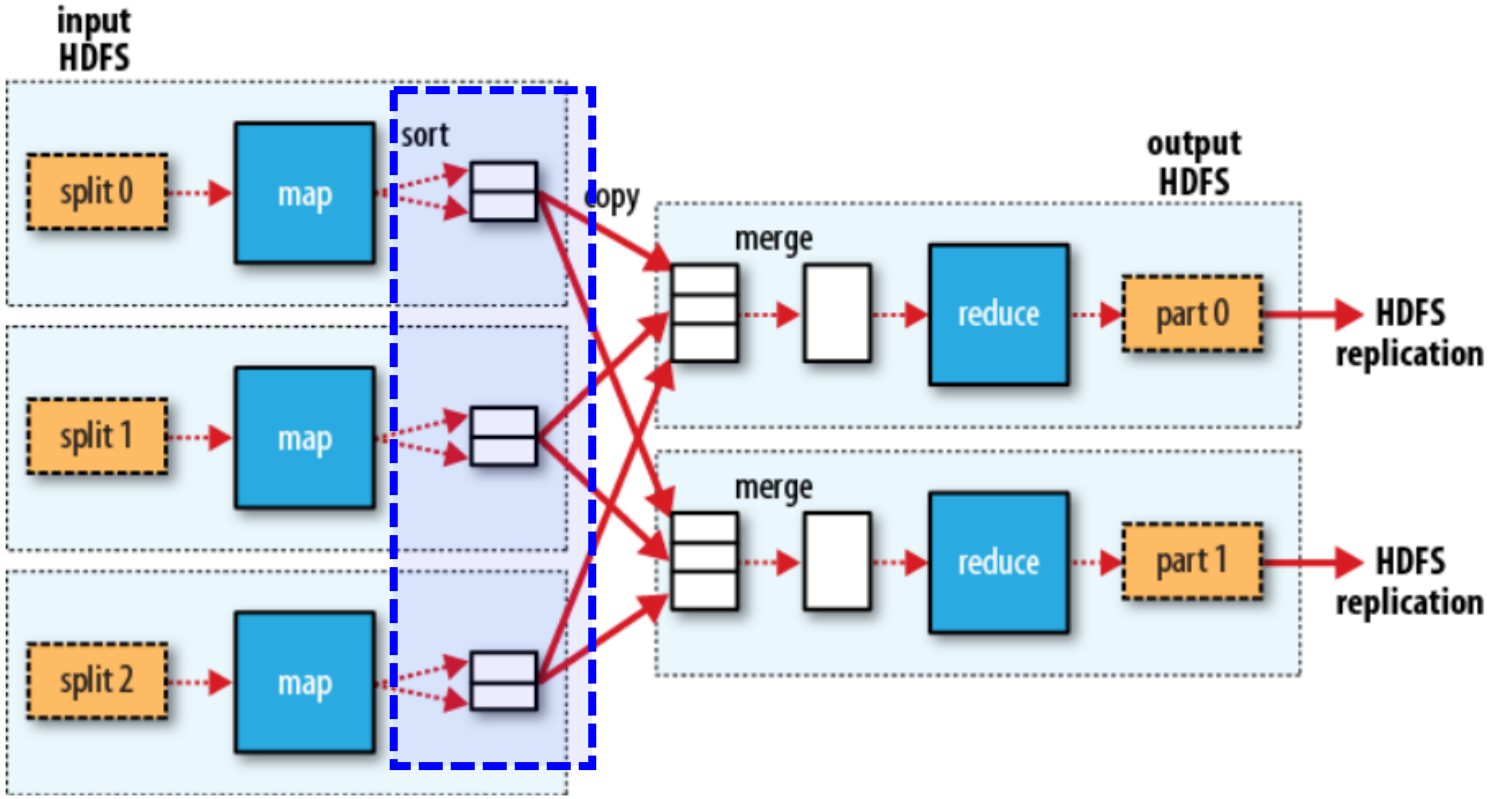
(*k*1, *v*1)

(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)

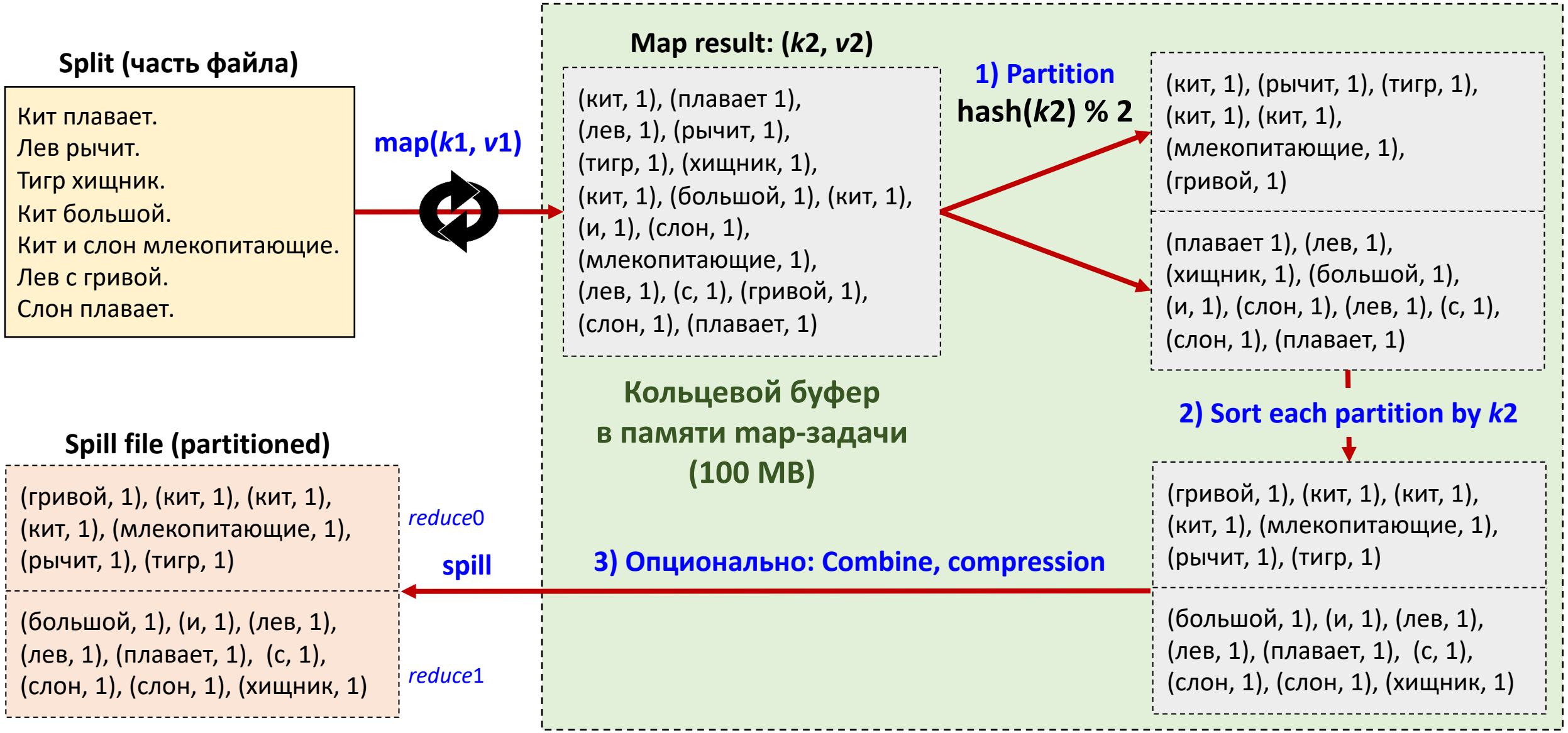
Apache Hadoop: map



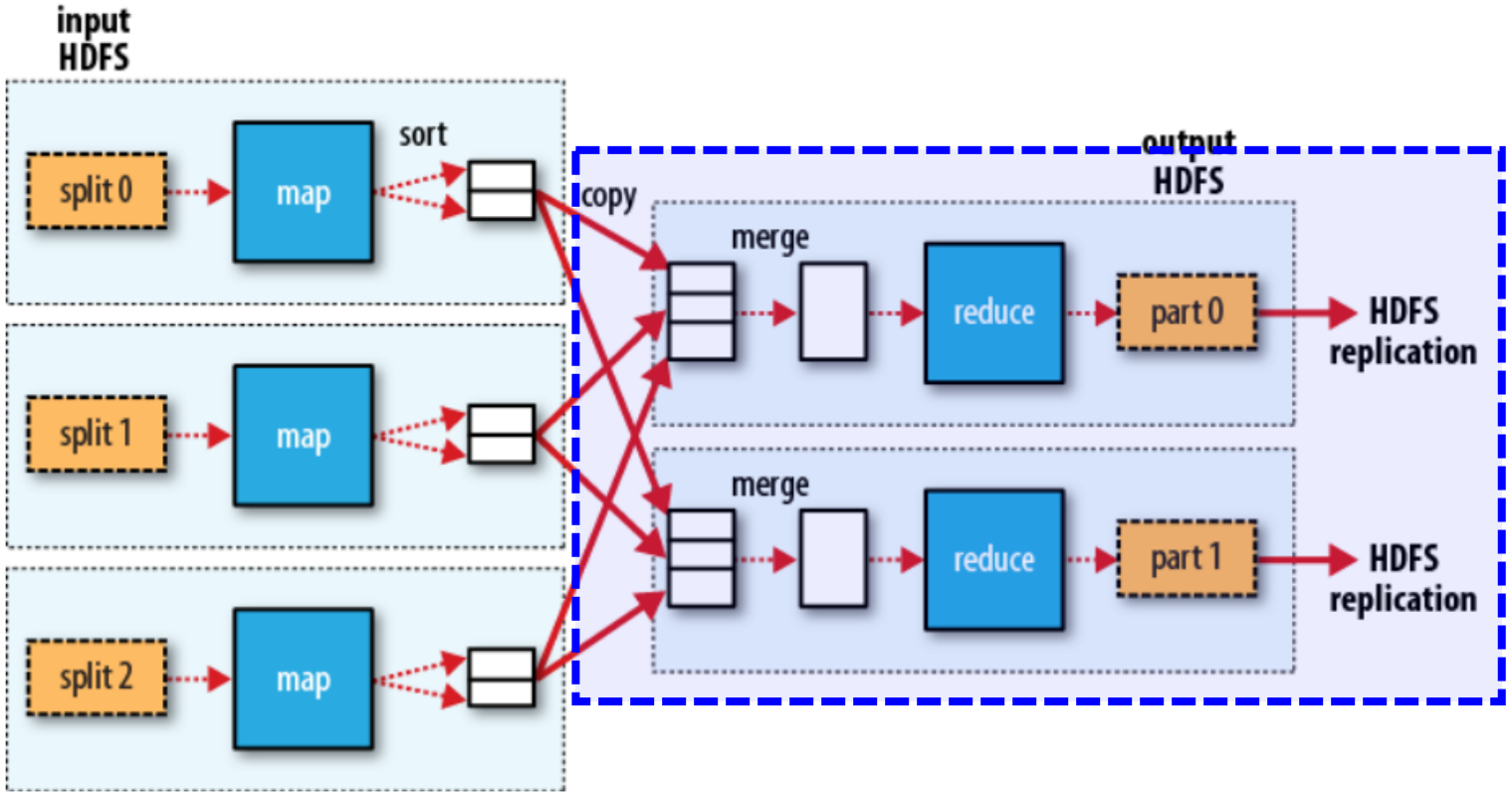
map(*k*1, *v*1) -> (*k*2, *v*2)

- 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



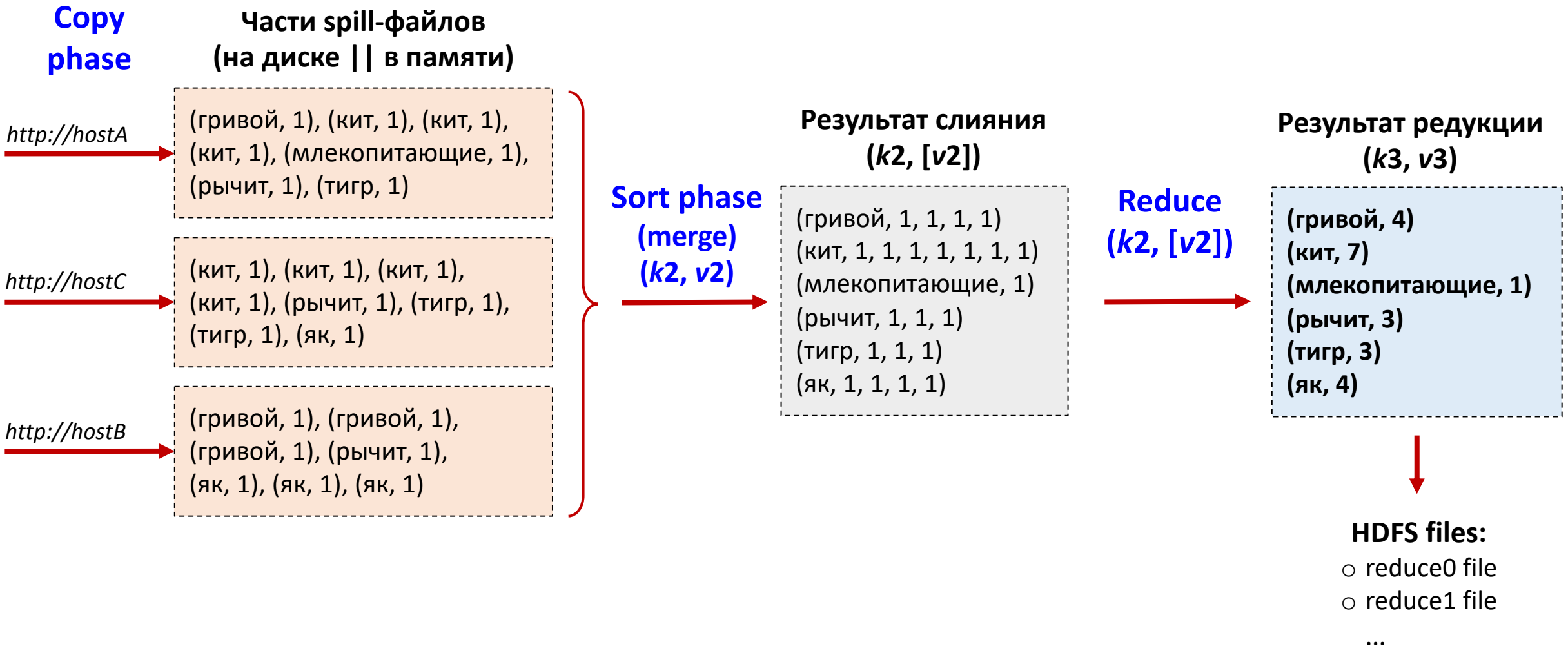
Copy phase

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

Sort phase (merge)

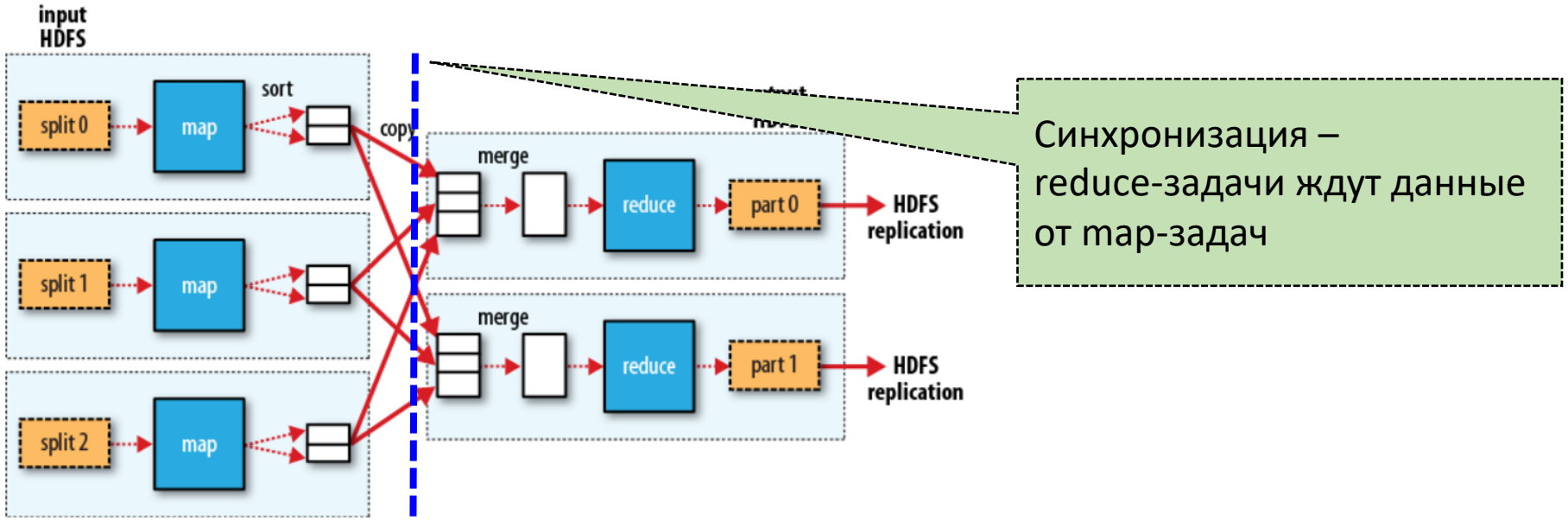
- Загруженные части spill-файлов сливаются за несколько раундов (merge factor)
- В конце фазы sort имеется merge factor файлов (10, mapreduce.task.io.sort.factor)
- Результаты финального раунда передаются в функцию reduce
- Результаты записываются в HDFS

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



Возможности MapReduce

- Использование сложных ключей и значений для управления процессом вычислений
- Выполнение заданного кода при инициализации и завершении map- и reduce-задач
- Сохранение состояния внутри map- и reduce-задач при обработке группы записей
- Определение порядка сортировки промежуточных ключей
- Определение разбиения пространства промежуточных ключей между reduce-задачами
- Композиция нескольких MapReduce-заданий

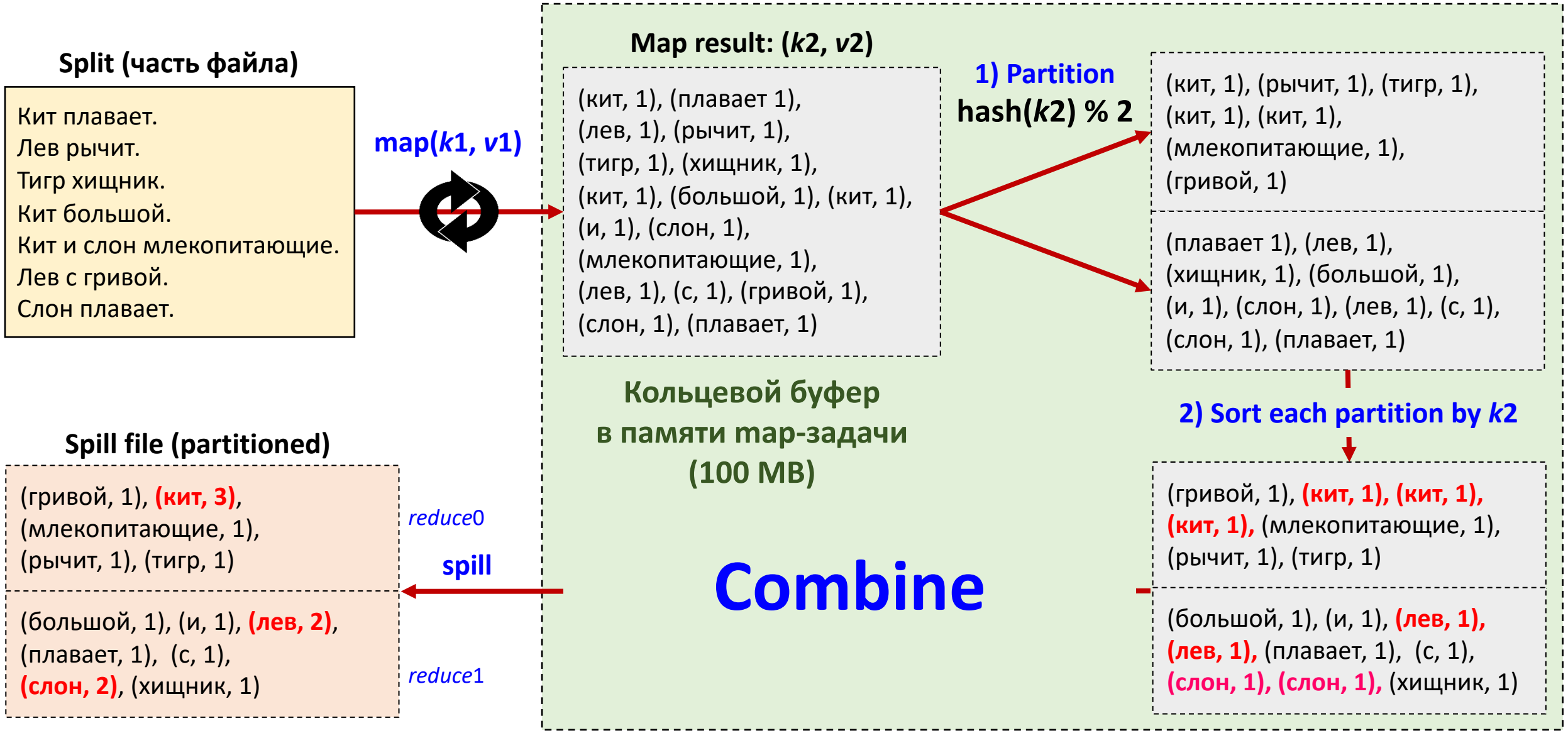
Показатели эффективности MapReduce-программ

- **Коэффициент ускорения (speedup, масштабируемость, scalability)**
  - ❑ В идеале – линейное ускорение (linear speedup)
    - ○ При двукратном увеличении количества машин происходит двукратное уменьшение времени работы
- **Эффективное использование ресурсов кластера**
  - ❑ Процессоров (время работы map/reduce-задач)
  - ❑ Памяти (буферы сортировки и слияния)
  - ❑ Сетевой и дисковой подсистемы (сжатие данных, data-aware распределение map-задач по узлам)

Локальная агрегация промежуточных данных

- После выполнения map-задачи применяем агрегацию данных => сокращается размер spill-файла (множество ключей (k2, v2))
- Ключевой прием при реализации эффективных MapReduce-программ
- Позволяет уменьшить накладные расходы на передачу данных между задачами
  - ❑ Сохранение данных на диск
  - ❑ Передача по сети
- Позволяет сбалансировать reduce-задачи

Локальная агрегация промежуточных данных



Combiner

- Для ассоциативных и коммутативных реализаций reduce
  - combiner == reducer
  - Ассоциативные и коммутативные операции: +, *, min/max, OR, AND
  - Не ассоциативные операции: mean(a, b)
- Не должен менять типы ключей и значений между map и reduce
  - На входе – результат map – пары (k2, v2)
  - На выходе – входные типы для reduce – пары (k2, v2)
- Hadoop: динамически использует combine как дополнительную оптимизацию
  - Может вообще не запускаться или запускаться несколько раз, в том числе на стороне reduce

Прием In-mapper Combining

```
1: class Mapper
2:     method Initialize
3:         H ← new AssociativeArray
4:     method Map(docid a, doc d)
5:         for all term t ∈ doc d do
6:             H{t} ← H{t} + 1
7:     method Close
8:         for all term t ∈ H do
9:             Emit(term t, count H{t})
```

- **Плюсы**
  - ❑ сокращается время выполнения программы
  - ❑ Не нужен отдельный combiner
- **Минусы**
  - ❑ дополнительный расход памяти порядка $O(|SplitSize|)$
  - ❑ можно сбрасывать массив при достижении порогового значения числа ключей в нем

- **Apache Hadoop: Mapper**
  - `void setup(Mapper.Context context)`
  - `void map(K1 key, V1 value, Mapper.Context context)`
  - `void cleanup(Mapper.Context context)`

[1] Jimmy Lin and Chris Dyer. **Data-Intensive Text Processing with MapReduce**, Morgan & Claypool Publishers, 2010.

Пример: вычисление среднего значения

- **Имеется** лог посещения сайта пользователями:
  - key – это userid
  - value – среднее время пребывания пользователя на сайте (время сессии)
- **Log [(userid, time)]:** (34102, 15), (34242, 29), …, (10023, 102)
- **Необходимо** для каждого пользователя вычислить среднее время сессии
- **Вычисление среднего значения для каждого ключа**
  - На входе: (string key, int val)
  - На выходе: (string key, int mean_val)
- **Нельзя использовать reduce в качестве combine**
  - mean(1, 2, 3, 4, 5) != mean(mean(1, 2), mean(3, 4, 5))

Пример: вычисление среднего значения (v1)

```
class Mapper
    method Map(string t, integer r)
        Emit(string t, integer r)

class Reducer
    method Reduce(string t, integers [r_1, r_2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r_1, r_2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

- Корректное, но не эффективное решение
- Reduce-задачи будут копировать блоки с большим числом одинаковых ключей: (1023, 34), (1023, 55), (1023, 15), …
- Нужен combiner

Пример: вычисление среднего значения (v2)

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Пример: вычисление среднего значения (v2)

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

- **Некорректное решение!**
- Combine – это оптимизация, вызывается по решению runtime-системы
- Combine должен принимать на вход пары от map и возвращает пары ***такого же типа***!
- Корректная последовательность: (k1, v1) -> map(k1, v1) -> **(k2, v2)** -> reduce(k2, [v2]) -> (k2, v2)
- **Как модифицировать алгоритм?**

Пример: вычисление среднего значения (v3)

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))
```

```
1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...]
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))
```

```
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

- Корректное решение
- map(k1, v1) -> (k2, <v2, 1>)

Пример: вычисление среднего значения (v4)

```
1:  class Mapper
2:      method Initialize
3:          S ← new AssociativeArray
4:          C ← new AssociativeArray
5:      method Map(string t, integer r)
6:          S{t} ← S{t} + r
7:          C{t} ← C{t} + 1
8:      method Close
9:          for all term t ∈ S do
10:             Emit(term t, pair (S{t}, C{t}))
```

- Корректное решение
- Использование паттерна in-mapper combining

Пример: Word co-occurrence matrix

- Имеется корпус текстов, состоящий из *N* слов
- Требуется построить матрицу *M* размера *N*x*N* (co-occurrence matrix)
- Элемент m_{ij} – это число совместных появлений (употреблений) слова *i* со словом *j* в пределах определенного контекста: в одном предложении, абзаце или фиксированном окне из *k* слов

В кабинете перед столом стоял председатель домкома ***Швондер*** *в кожаной тужурке. Доктор Борменталь сидел в кресле. При этом на румяных от мороза щеках доктора (он только что вернулся) было столь же растерянное выражение, как и у Филиппа Филипповича, сидящего рядом.*

- Как же писать? — Нетерпеливо спросил он.

- Что же, — заговорил Швондер, — дело не сложное. Пишите удостоверение, гражданин профессор. Что так, мол, и так предьявитель сего действительно Шариков Полиграф Полиграфович, гм… Зародившийся в вашей, мол, квартире.

	В	Дело	Домком	…
…				
Швондер	2	1	1	…
…				
…				

Word co-occurrence matrix: стратегия Pairs (сложный ключ)

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

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

- ***map***(*docid*, *content*) -> (<*w*, *u*>, 1)
- ***Neighbors***(*w*) –
  локальная окрестность слова *w*
  (множество соседних слов) –
  предложение, абзац, окно из *k*
  слов

Word co-occurrence matrix: стратегия Stripes

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term w ∈ doc d do
4:             H ← new AssociativeArray
5:             for all term u ∈ Neighbors(w) do
6:                 H{u} ← H{u} + 1
7:             Emit(Term w, Stripe H)
```

```
1: class Reducer
2:     method Reduce(term w, stripes [H_1, H_2, H_3, ...])
3:         H_f ← new AssociativeArray
4:         for all stripe H ∈ stripes [H_1, H_2, H_3, ...] do
5:             Sum(H_f, H)
6:         Emit(term w, stripe H_f)
```

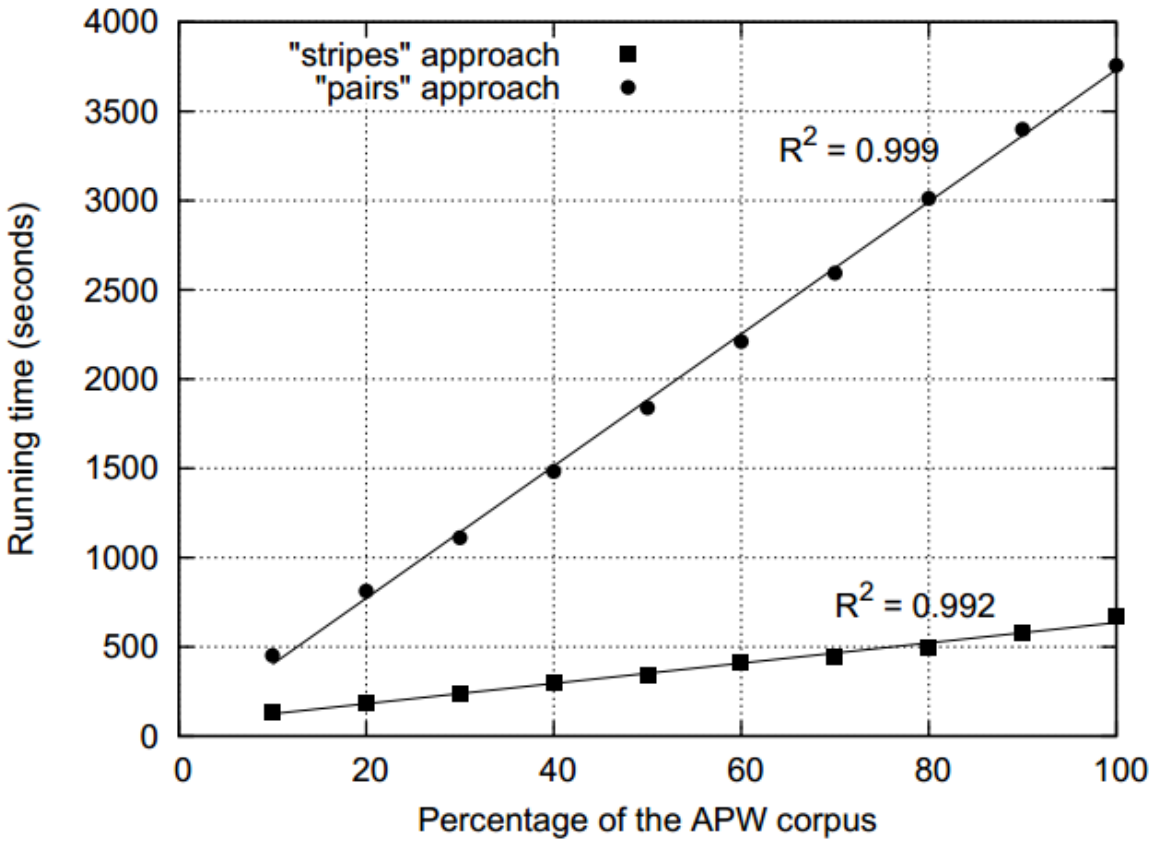
- ***map***(*docid*, *content*) ->
  (*w*, [<u_1, c_1>, <u_2, c_2>, …, <u_z, c_z>])

Word co-occurrence matrix: какая стратегия лучше?

- **Стратегия Pairs**
  - ❑ Простая реализация
  - ❑ Генерирует больше данных
  - ❑ Ключ – пара из 2 слов: дольше выполнять операцию сравнения (Sort)

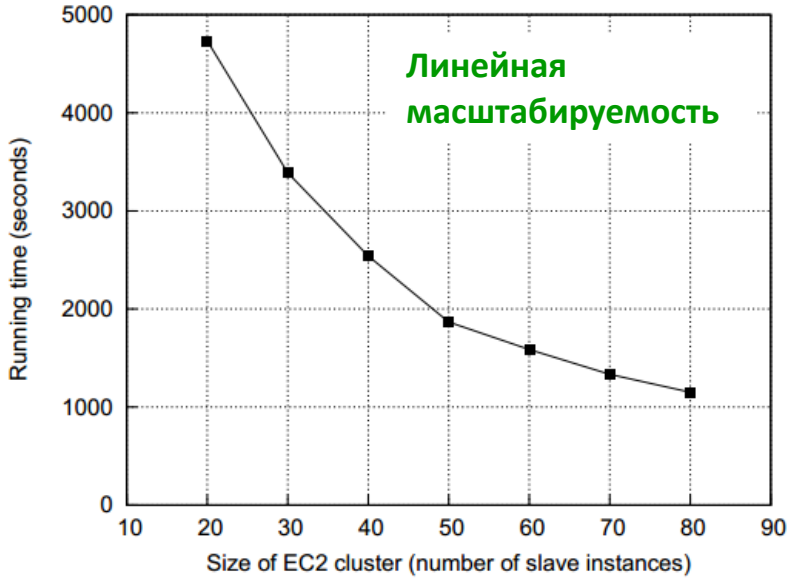
- **Стратегия Stripes**
  - ❑ Генерирует меньше промежуточных ключей
  - ❑ Относительно компактный ключ
  - ❑ Больше возможностей для применения локальной агрегации
  - ❑ Выше требования к памяти

Word co-occurrence matrix: какая стратегия лучше?



Jimmy Lin and Chris Dyer. **Data-Intensive Text Processing with MapReduce**, Morgan & Claypool Publishers, 2010.

- Реализация стратегий на Apache Hadoop
- Корпус из 2.27 миллионов документов (Associated Press Worldstream – APW)
- Hadoop-кластер из 19 узлов (2 ядра, 2 диска)



Стратегия Stripes, кластер на базе Amazon's EC2

Word co-occurrence matrix: вычисление относительных частот

- Вычисление относительной частоты слова w_j в контексте слова w_i

$$f(w_j|w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

- $N(w_i, w_j)$ – это количество совместных употреблений слов w_i, w_j в корпусе текстов
- **Например**: сколько раз встречалось слово «*atomic*» в контексте слова «*OpenMP*»?

$$f(w_j|w_i) = \frac{N(atomic, openmp)}{\sum_{w'} N(openmp, w')}$$

Сколько раз встречалось слово atomic вместе с openmp

Сколько раз встречалось слово openmp с другими словами

Вычисление относительных частот: стратегия Stripes

- Вычисление относительной частоты слова w_j в контексте слова w_i

$$f(w_j|w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

- ***map***(*docid*, *content*) -> (w_i, H = [<u_1, c_1>, <u_2, c_2>, …, <u_z, c_z>])
- ***reduce***(w_i, H_1, H_2, …, H_d)
  - Вычисляем $N(w_i, w_j)$ – используя массивы H_1, H_2, …, H_d
  - Вычисляем Sum $N(w_i, w')$ – используя массивы H_1, H_2, …, H_d
  - Выдаем результат

Вычисление относительных частот: стратегия Pairs

- Вычисление относительной частоты слова w_j в контексте слова w_i

$$f(w_j|w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

- ***map***(*docid*, *content*) -> (<w_i, w_j>, 1)
- ***reduce***((w_i, w_j), [c_1, c_2, …, c_d])
  - Вычисляем $N(w_i, w_j)$ – используя [c_1, c_2, …, c_d]
  - Как вычислить Sum $N(w_i, w')$?

Jimmy Lin and Chris Dyer. **Data-Intensive Text Processing with MapReduce**, Morgan & Claypool Publishers, 2010, **P. 58**

Вычисление относительных частот: стратегия Pairs

key	values	
$(\text{dog}, *)$	$[6327, 8514, \ldots]$	compute marginal: $\sum_{w'} N(\text{dog}, w') = 42908$
$(\text{dog}, \text{aardvark})$	[2,1]	$f(\text{aardvark}\|\text{dog}) = 3/42908$
$(\text{dog}, \text{aardwolf})$	[1]	$f(\text{aardwolf}\|\text{dog}) = 1/42908$
...		
$(\text{dog}, \text{zebra})$	[2,1,1,1]	$f(\text{zebra}\|\text{dog}) = 5/42908$
$(\text{doge}, *)$	$[682, \ldots]$	compute marginal: $\sum_{w'} N(\text{doge}, w') = 1267$
...		

- **Прием Order inversion**
  - ❑ Распределение промежуточных ключей с одинаковым первым словом на один Reducer
  - ❑ Выдача Map вспомогательных записей с ключами (w_i, *)
  - ❑ Сортировка промежуточных ключей по первому слову так, чтобы вспомогательные значения были первыми
  - ❑ Хранение внутреннего состояния в Reduce

Apache Hadoop: сложный тип ключа (string, string)

```
public static class MyKey implements WritableComparable<MyKey> {

    private String first;
    private String second;

    public void set(String first, String second) {
        this.first = first;
        this.second = second;
    }

    public String getFirst() { return first; }
    public String getSecond() { return second; }

    public int compareTo(MyKey o) {
        if (!first.equals(o.first)) {
            return first.compareTo(o.first);
        } else {
            return second.compareTo(o.second);
        }
    }
```

Apache Hadoop: сложный тип ключа (string, string)

```
    @Override
    public void readFields(DataInput in) throws IOException {
        first = Text.readString(in);
        second = Text.readString(in);
    }

    @Override
    public void write(DataOutput out) throws IOException {
        Text.writeString(out, first);
        Text.writeString(out, second);
    }

    @Override
    public String toString() {
        return first + "+" + second;
    }

    @Override
    public int hashCode() { return toString().hashCode(); }
}
```

Apache Hadoop: Partitioner – распределение ключей по reducers

- **Класс org.apache.hadoop.mapreduce.Partitioner<K, V>**
  - ❑ abstract int getPartition(KEY key, VALUE value, int numPartitions);

- По умолчанию используется реализация **HashPartitioner**
  - ❑ return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  - ❑ можно использовать путем переопределения метода key.hashCode()

- **Установка**
  - ❑ job.setPartitionerClass(*MyPartitioner.class*)

- **Hadoop Streaming**
- – см. KeyFieldBasedPartitioner

Apache Hadoop: Sort Comparator

- **Сортировка ключей**: recursive QuickSort + HeapSort
  src/hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/util/QuickSort.java
- **Вызывается при сортировке ключей**
  - ❑ переопределения порядка сортировки ключей
  - ❑ оптимизации: RawComparator, не требующий десериализации ключей в объекты
- **Устанавливается** с помощью Job.setSortComparatorClass(*MyComparator.class*)
- **Готовые реализации**: LongWritable.DecreasingComparator
- **Hadoop Streaming**
  - ❑ см. KeyFieldBasedComparator

Apache Hadoop: Secondary Sort

- Иногда требуется упорядочить промежуточные значения (values) для данного ключа на входе Reduce
- Возможные решения
  - Сортировка в памяти внутри Reduce
  - Прием Value-to-key conversion: (key, value) -> (<key, subkey>, value)
- Apache Hadoop
  - Использование GroupingComparator для вызова reduce один раз для каждого значения key
  - см. стандартный пример SecondarySort.java

Apache Hadoop Java API: объект context

- Передается в функции map и reduce
- Предоставляет доступ к конфигурации задания
  - Context.getConfiguration()
- Принимает пары (key,value) на выходе
  - Context.write(key, value)
- Позволяет обновлять статус задания и счетчики задания (counters)

Apache Hadoop: состояние задания и счетчики

- **Уведомление** мастера о том, что задача «жива»
  - ❑ mapreduce.task.timeout = 10 минут по умолчанию
- **Обновление статуса задачи**
  - ❑ context.setStatus(String status)
- **Глобальные счетчики**

```
static enum MyCounters { INPUT_WORDS, ... }
// ...
context.getCounter(MyCounters.INPUT_WORDS).increment(1);
```

Apache Hadoop Streaming: состояние задания и счетчики

- **Статус задания**
  - ❑ std::cerr << *reporter:status:message*
- **Счетчики**
  - ❑ std::cerr << *reporter:counter:group,counter,amount*

Apache Hadoop: Distributed Cache

- **Загрузка дополнительных файлов (read only) на узлы кластера вместе с заданием**
  - Вспомогательные данные, текстовые файлы, архивы, библиотеки
- Файлы должны быть предварительно размещены в HDFS
- Или переданы пути к локальным файлам с помощью
  - ❑ Опций командной строки "-files, -archives, -libjars"
  - ❑ Java API: conf.set("tmpfiles(tmparchives,tmpjars)", "file://path...")
- Файлы копируются один раз перед началом выполнения задания
- Не допускается изменение данных файлов во время выполнения задания

https://hadoop.apache.org/docs/r2.3.0/api/org/apache/hadoop/filecache/DistributedCache.html

Apache Hadoop: разнородные данные на входе и выходе

- **Входные данные**
  - По умолчанию все файлы имеют один формат, обрабатываются одним Map-классом
- **Несколько типов входных файлов**
  - org.apache.hadoop.mapreduce.lib.input.MultipleInputs
  - Для каждого набора можно указать свой формат и Map-класс
- **Выходные данные**
  - По умолчанию один файл на reducer: part-r-00000, part-r-00001, ...
- **Несколько типов файлов на выходе**
  - org.apache.hadoop.mapreduce.lib.output.MultipleOutputs
  - Каждый файл может иметь свой формат и типы ключей-значений

Apache Hadoop: цепочки задач

- **Цепочки заданий**
  - ❑ Выходные данные задания являются входными для следующего задания
- **Объединение map-функций в одну логическую map-задачу**
  - ❑ org.apache.hadoop.mapreduce.lib.chain.ChainMapper/ChainReducer
  - ❑ [MAP+ / REDUCE MAP*]
- **Более сложные зависимости между заданиями (DAG)**
  - ❑ org.apache.hadoop.mapreduce.lib.jobcontrol

Задачи: 100 термометров

- **Имеется** лог с показанием 100 термометров (сеть датчиков)
- Формат записи: (id, timestamp, temperature)
  - (34, 10:34:15, 28.0)
  - (12, 03:59:22, 2.0)
  - (1, 12:04:39, 32.0)
  - …
  - (5, 14:50:01, 33.0)
- **Необходимо** найти записи с аномальной температурой (temperature > t_critical)
- **Выдать** (date, temperature)

Задачи: 100 термометров

- **Имеется** лог с показанием 100 термометров (сеть): (id, timestamp, temperature). Необходимо найти записи с аномальной температурой (temperature > t_critical)
- **Возможное решение**
  - **map**(offset, line) -> (id, <date, t>) - выдать *if t > t_critical*
  - **reduce**(id, [<date, t>, …]) – например, выдать только за последние 10 дней

Задачи: определить “гендерную принадлежность” сайта

- **Имеются** обработанные логи посещения веб-сайтов (timestamp, user_sex, url):
  - 12:34 М www.kernel.org
  - 14:01 Ж www.passion.ru
  - 14:01 М www.sportbox.ru
  - …
- **Требуется** определить пол сайта:
  - выдать www.kernel.org **95%** М, **5%** Ж

Задачи: определить “гендерную принадлежность” сайта

- **Имеются** обработанные логи посещения веб-сайтов (timestamp, user_sex, url):
  - 12:34 М www.kernel.org
  - 14:01 Ж www.passion.ru
  - 14:01 М www.sportbox.ru
  - …
- **Требуется** определить пол сайта: выдать www.kernel.org **95%** М, **5%** Ж
- **Возможное решение (в лоб)**
  - **map**(offset, line) -> (url, <men, women>) -- выдать (www.kernel.org, <1, 0>)
  - **reduce**(url, [<men, women>, …]) –
    выдать (url, <men/n_url_visits * 100, women/n_url_visits * 100>)

Пример: Биграммы (Bigram)

- **Биграмма** (bigram, digram, 2-gram) – два соседних символа (слова, слога) в заданном корпусе текстов (коллекции документов)
- Частотное распределение биграмм используется в статистическом анализе текстов (лингвистика, машинный перевод, криптография, распознавание речи, выявление фразеологизмов, SEO-оптимизация сайтов)

Корпус текстов

“Сквозь волнистые туманы
Пробирается луна,
На печальные поляны
Льет печально свет она.”

Биграммы из слов

- (сквозь, волнистые)
- (волнистые, туманы)
- …
- (свет, она)

Пример: подсчет всех биграмм (BigramCount)

- **Входные данные:** текстовые файлы (FileInputFormat)
  - (offset, line) -> **map**(line_offset, line)
- **map**(line_offset, line) -> (<w1, w2>, 1)
  - Перебираем слова в строке: Emit(“prev_word+curr_word”, 1)
- **reduce**(<w1, w2>, [c1, c2, …, cn])
- **Выходные данные:** текстовые файлы (<w1, w2>, count)

about+attack	**1**
about+eurasia	**1**
about+five	**1**
about+forty	**1**
about+four	**1**
about+him	**3**
about+his	**1**
about+iron	**1**

Пример: подсчет биграмм (BigramCount, BigramTop)

- **Реализация Bigram**
  - ❑ Доступ к конфигурации задания в Mapper/Reduce.configure()
  - ❑ Реализация сложных типов ключей
  - ❑ Реализация Partitioner
  - ❑ Реализация Comparator
  - ❑ Статус и счетчики
  - ❑ Передача дополнительных файлов (DistributedCache)
  - ❑ Паттерн In-mapper combining
  - ❑ Стратегия Pairs
  - ❑ Secondary Sort

Запуск примеров

- **BigramCount - подсчет всех биграмм**

```
$ hadoop jar ./bigramcount.jar pdccourse.lecture7.BigramCount \
             -D mapreduce.job.reduces=4 /pub/etwiki.xml bigram/bg ./skip.en
```

- **BigramTop - выдача TOP-биграмм для каждого слова**

```
$ hadoop jar ./bigramtop.jar pdccourse.lecture7.BigramTop \
             -D mapreduce.job.reduces=4 bigram/bg bigram/bg-top 5  100
```

about+attack	**1**
about+eurasia	**1**
about+five	**1**
about+forty	**1**
about+four	**1**
about+him	**3**
…	

away+from=8
because+they=5
behind+him=8
big+brother=36
can+you=6
…

Вывести биграммы
с числом повторений >= 5

Вывести <= 100
биграмм

Домашнее чтение

- Jimmy Lin, Chris Dyer. **Data-Intensive Text Processing with MapReduce**
  - ❑ http://lintool.github.com/MapReduceAlgorithms/
  - ❑ Глава 3