

MapReduce

Описание подхода и паттернов

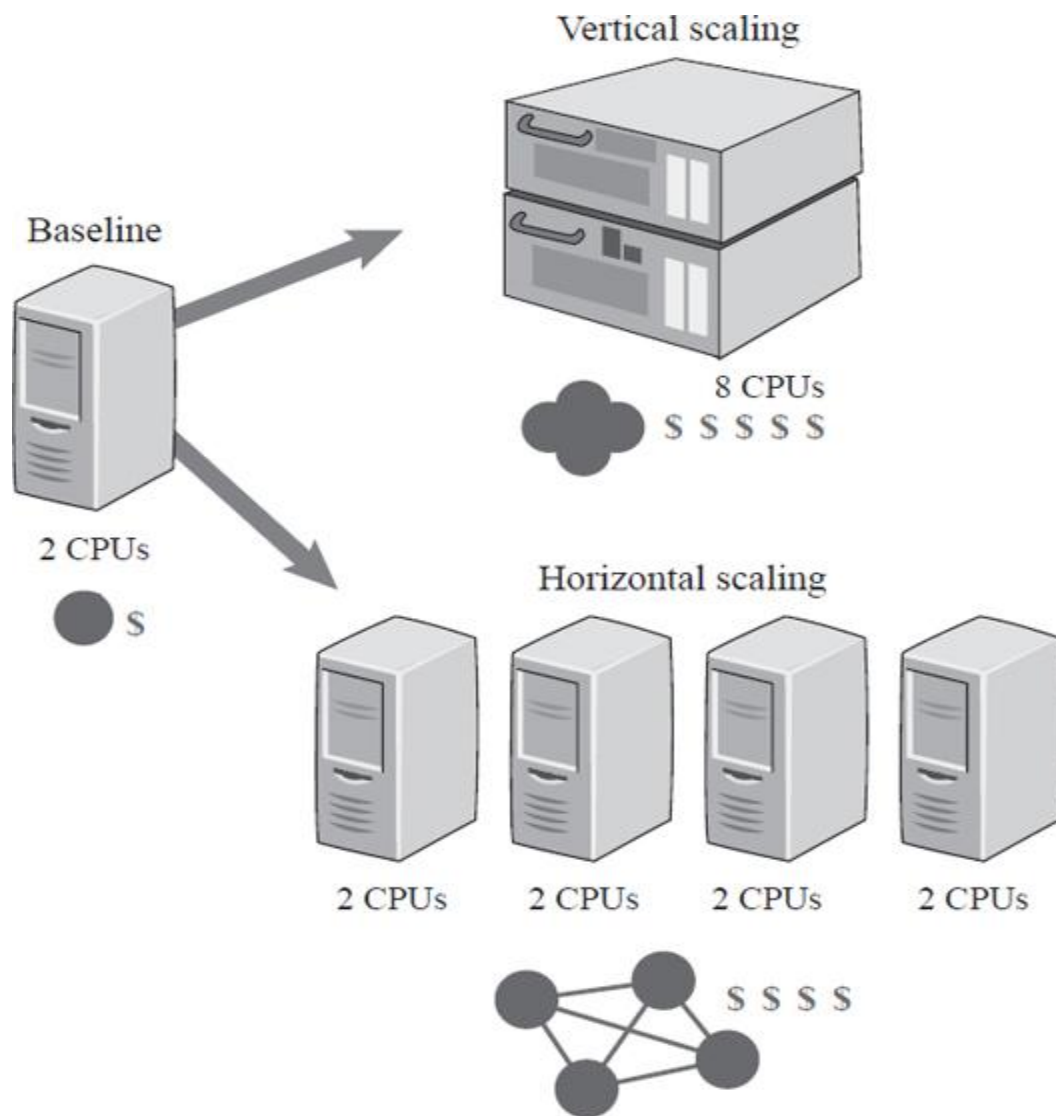
mrjob

Apache Hadoop

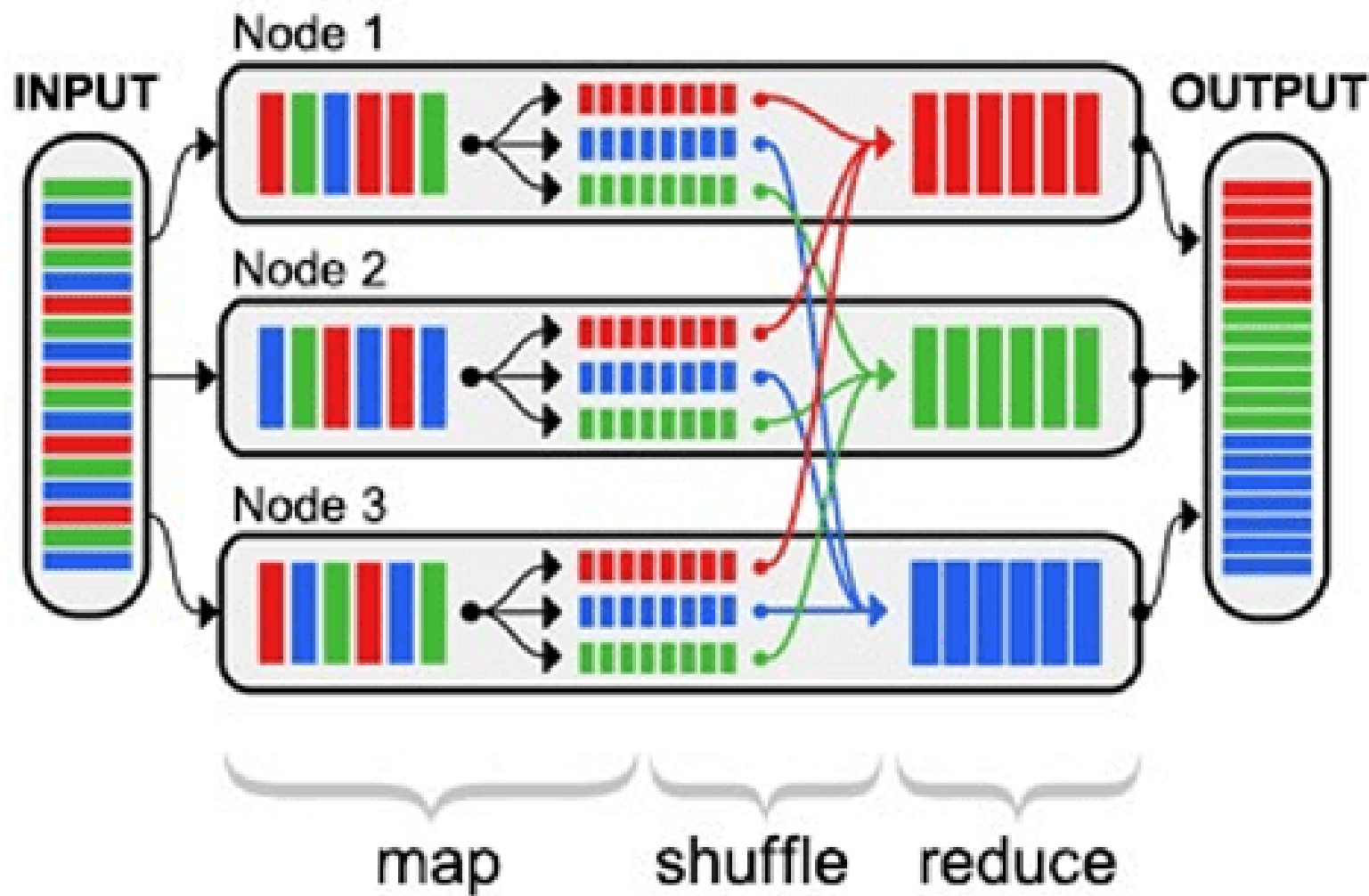
Spark

Hive

Еще раз о масштабировании



MapReduce



MapReduce

1. Стадия Map. На этой стадии данные преобразуются при помощи функции `map()`, которую определяет пользователь. Работа этой стадии заключается в преобразовке и фильтрации данных. Работа очень похожа на операцию `map` в функциональных языках программирования – пользовательская функция применяется к каждой входной записи.

Функция `map()` примененная к одной входной записи и выдаёт множество пар ключ-значение.

Множество – т.е. может выдать только одну запись, может не выдать ничего, а может выдать несколько пар ключ-значение. Что будет находится в ключе и в значении – решать пользователю, но ключ – очень важная вещь, так как данные с одним ключом в будущем попадут в один экземпляр функции `reduce`.

2. Стадия Shuffle. Проходит незаметно для пользователя. В этой стадии вывод функции `map` «разбирается по корзинам» – каждая корзина соответствует одному ключу вывода стадии `map`. В дальнейшем эти корзины послужат входом для `reduce`.

3. Стадия Reduce. Каждая «корзина» со значениями, сформированная на стадии `shuffle`, попадает на вход функции `reduce()`.

Функция `reduce` задаётся пользователем и вычисляет финальный результат для отдельной «корзины». Множество всех значений, возвращённых функцией `reduce()`, является финальным результатом MapReduce-задачи.

<https://habr.com/ru/company/dca/blog/267361/>

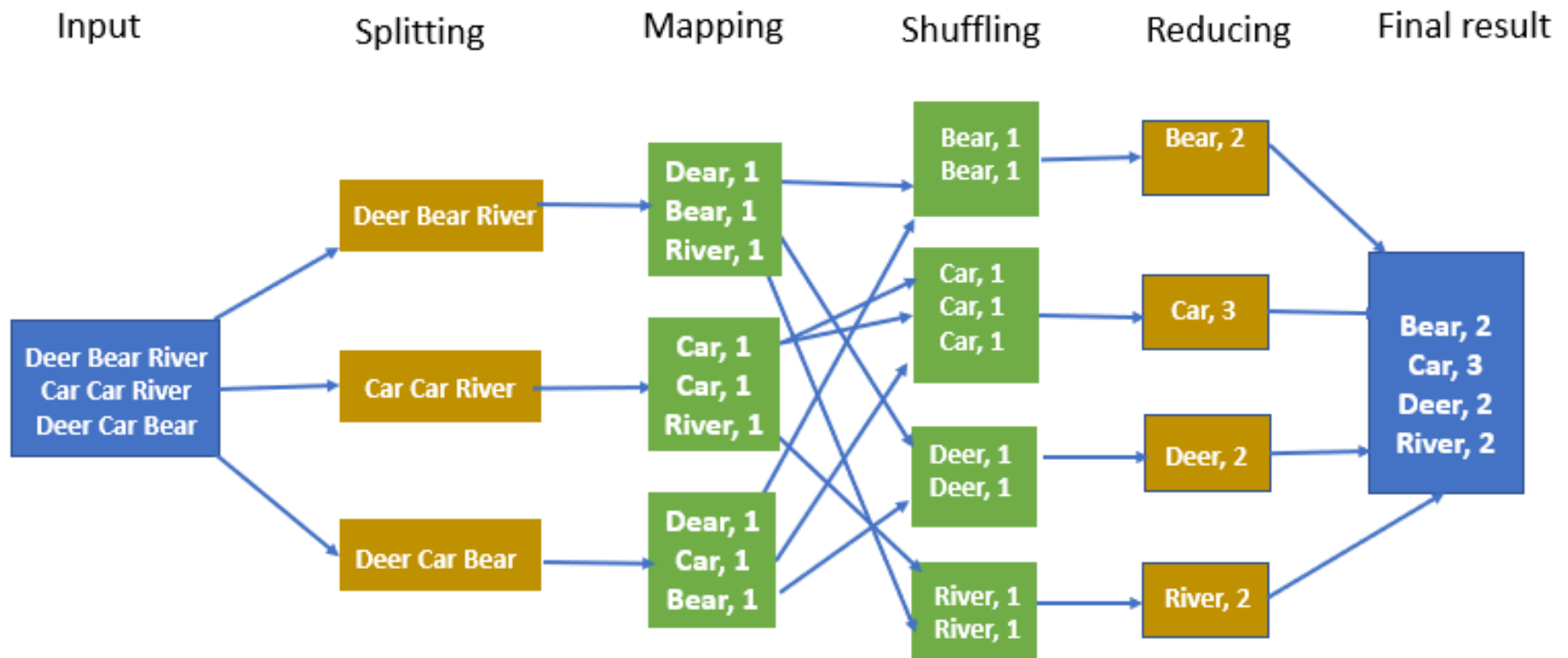
Задача 1 (классика)

```
def map(doc):  
    for word in doc:  
        yield word, 1
```

```
def reduce(word, values):  
    yield word, sum(values)
```

Функция **map** превращает входной документ в набор пар (слово, 1), **shuffle** прозрачно для нас превращает это в пары (слово, [1,1,1,1,1,1]), **reduce** суммирует эти единички, возвращая финальный ответ для слова.

Иллюстрация решения задачи 1



Задача 2

Задача: имеется csv-лог рекламной системы вида:

```
<user_id>,<country>,<city>,<campaign_id>,<creative_id>,<payment></p>

11111,RU,Moscow,2,4,0.3
22222,RU,Voronezh,2,3,0.2
13413,UA,Kiev,4,11,0.7
...
```

Необходимо рассчитать среднюю стоимость показа рекламы по городам России.

Задача 2 (решение)

```
def map(record):  
    user_id, country, city, campaign_id, creative_id, payment = record.split(",")  
    payment=float(payment)  
    if country == "RU":  
        yield city, payment
```

```
def reduce(city, payments):  
    yield city, sum(payments)/len(payments)
```

Функция **map** проверяет, нужна ли нам данная запись – и если нужна, оставляет только нужную информацию (город и размер платежа). Функция **reduce** вычисляет финальный ответ по городу, имея список всех платежей в этом городе.

Map only job

<https://habr.com/ru/post/270453/>

Как мы помним, MapReduce состоит из стадий Map, Shuffle и Reduce. Как правило, в практических задачах **самой тяжёлой оказывается стадия Shuffle**, так как на этой стадии происходит сортировка данных. На самом деле существует ряд задач, в которых можно обойтись только стадией Map. Вот примеры таких задач:

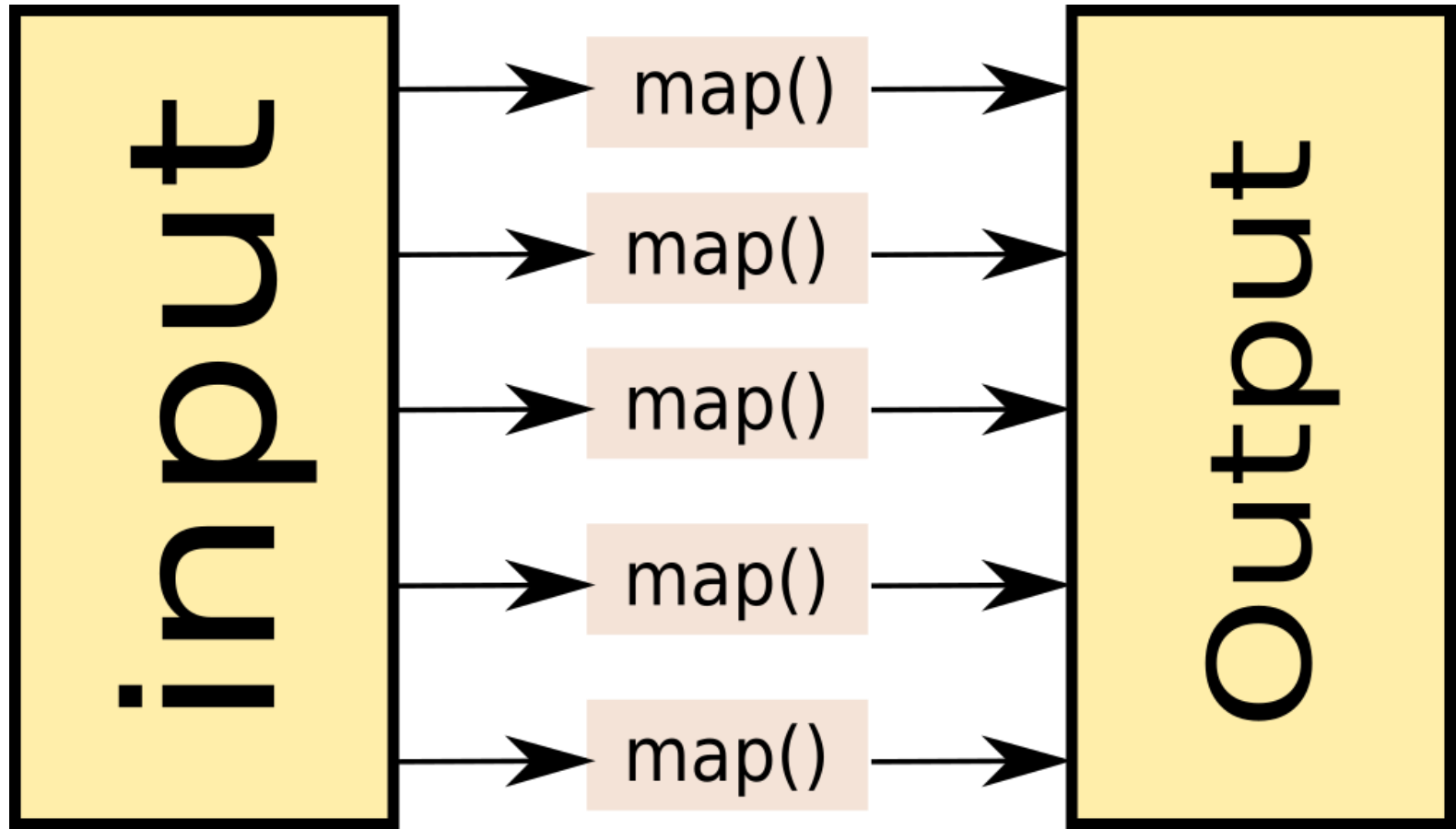
Фильтрация данных (например, «Найти все записи с IP-адреса 123.123.123.123» в логах web-сервера);

Преобразование данных («Удалить колонку в csv-логах»);

Загрузка и выгрузка данных из внешнего источника («Вставить все записи из лога в базу данных»).

Такие задачи решаются при помощи Map-Only. При создании Map-Only задачи в Hadoop нужно указать нулевое количество reducer'ов:

MapReduce



MR job chains

<https://habr.com/ru/post/270453/>

Бывают ситуации, когда для решения задачи одним MapReduce не обойтись. Например, рассмотрим немного видоизмененную задачу WordCount: имеется набор текстовых документов, необходимо посчитать, сколько слов встретилось от 1 до 1000 раз в наборе, сколько слов от 1001 до 2000, сколько от 2001 до 3000 и так далее.

Для решения нам потребуется 2 MapReduce job'a:

Видоизменённый wordcount, который для каждого слова рассчитает, в какой из интервалов оно попало;

MapReduce, подсчитывающий, сколько раз в выходе первого MapReduce встретился каждый из интервалов.

MR job chains

<https://habr.com/ru/post/270453/>

```
#map1
def map(doc):
    for word in doc:
        yield word, 1
```

```
#reduce1
def reduce(word, values):
    yield int(sum(values)/1000), 1
```

```
#map2
def map(doc):
    interval, cnt = doc.split()
    yield interval, cnt
```

```
#reduce2
def reduce(interval, values):
    yield interval*1000, sum(values)
```

MR Patterns: 1) count unique items

<https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/>

Phase I:

```
1  class Mapper
2      method Map(record [value f, categories [g1, g2,...]])
3          for all category g in [g1, g2,...]
4              Emit(record [g, f], count 1)
5
6  class Reducer
7      method Reduce(record [g, f], counts [n1, n2, ...])
8          Emit(record [g, f], null )
```

Phase II:

```
1  class Mapper
2      method Map(record [f, g], null)
3          Emit(value g, count 1)
4
5  class Reducer
6      method Reduce(value g, counts [n1, n2,...])
7          Emit(value g, sum( [n1, n2,...] ) )
```

MR Patterns: 2) cross-correlation

Problem Statement: There is a set of tuples of items. For each possible pair of items calculate a number of tuples where these items co-occur. If the total number of items is N then $N*N$ values should be reported.

This problem appears in text analysis (say, items are words and tuples are sentences), market analysis (customers who buy *this* tend to also buy *that*). If $N*N$ is quite small and such a matrix can fit in the memory of a single machine, then implementation is straightforward.

```
1  class Mapper
2      method Map(null, items [i1, i2,...] )
3          for all item i in [i1, i2,...]
4              for all item j in [i1, i2,...]
5                  Emit(pair [i j], count 1)
6
7  class Reducer
8      method Reduce(pair [i j], counts [c1, c2,...])
9          s = sum([c1, c2,...])
10         Emit(pair[i j], count s)
```

Relational MR Patterns

Selection

```
1 | class Mapper
2 |     method Map(rowkey key, tuple t)
3 |         if t satisfies the predicate
4 |             Emit(tuple t, null)
```

Projection

Projection is just a little bit more complex than selection, but we should use a Reducer in this case to eliminate possible duplicates.

```
1 | class Mapper
2 |     method Map(rowkey key, tuple t)
3 |         tuple g = project(t) // extract required fields to tuple g
4 |         Emit(tuple g, null)
5 |
6 | class Reducer
7 |     method Reduce(tuple t, array n) // n is an array of nulls
8 |         Emit(tuple t, null)
```

Relational MR Patterns

Union

Mappers are fed by all records of two sets to be united. Reducer is used to eliminate duplicates.

```
1  class Mapper
2      method Map(rowkey key, tuple t)
3          Emit(tuple t, null)
4
5  class Reducer
6      method Reduce(tuple t, array n)    // n is an array of one or two nulls
7          Emit(tuple t, null)
```

Intersection

Mappers are fed by all records of two sets to be intersected. Reducer emits only records that occurred twice. It is possible only if both sets contain this record because record includes primary key and can occur in one set only once.

```
1  class Mapper
2      method Map(rowkey key, tuple t)
3          Emit(tuple t, null)
4
5  class Reducer
6      method Reduce(tuple t, array n)    // n is an array of one or two nulls
7          if n.size() = 2
8              Emit(tuple t, null)
```


PageRank algorithm

<https://medium.com/swlh/pagerank-on-mapreduce-55bcb76d1c99>

PageRank (or PR in short) is a recursive algorithm developed by Google founder **Larry Page** to assign a real number to each page in the Web so they can be ranked based on these scores, where the higher the score of a page, the more “important” it is.

$$PR_a = (1 - d)/N + d \sum_{i=1}^n \frac{PR_i}{C_i},$$

1. PR_a – PageRank рассматриваемой страницы
2. d – коэффициент затухания
3. N – общее количество документов
4. PR_i – PageRank i -й страницы, ссылающейся на страницу a
5. C_i – общее число ссылок на i -й странице.

Hadoop

Изначально Hadoop был, в первую очередь, инструментом для хранения данных и запуска MapReduce-задач, сейчас же Hadoop представляет собой большой стек технологий, так или иначе связанных с обработкой больших данных (не только при помощи MapReduce).

Основными (core) компонентами Hadoop являются:

- * [Hadoop Distributed File System \(HDFS\)](#) – распределённая файловая система, позволяющая хранить информацию практически неограниченного объёма.
- * [Hadoop YARN](#) – фреймворк для управления ресурсами кластера и менеджмента задач, в том числе включает фреймворк MapReduce.
- * Hadoop common

Hadoop Streaming

Самый простой способ запустить MapReduce-программу на Hadoop – воспользоваться streaming-интерфейсом Hadoop. Streaming-интерфейс предполагает, что map и reduce реализованы в виде программ, которые принимают данные с **stdin** и выдают результат на **stdout**.

Программа, которая исполняет функцию map называется **mapper**. Программа, которая выполняет reduce, называется, соответственно, **reducer**.

Streaming интерфейс предполагает по умолчанию, что одна входящая строка в mapper или reducer соответствует одной входящей записи для map.

Вывод mapper'а попадает на вход reducer'у в виде пар (ключ, значение), при этом все пары соответствующие одному ключу:

Гарантированно будут обработаны одним запуском reducer'а; Будут поданы на вход подряд (то есть если один reducer обрабатывает несколько разных ключей – вход будет сгруппирован по ключу).

Hadoop Streaming

```
#mapper.py
import sys

def do_map(doc):
    for word in doc.split():
        yield word.lower(), 1

for line in sys.stdin:
    for key, value in do_map(line):
        print(key + "\t" + str(value))

#reducer.py
import sys

def do_reduce(word, values):
    return word, sum(values)

prev_key = None
values = []

for line in sys.stdin:
    key, value = line.split("\t")
    if key != prev_key and prev_key is not None:
        result_key, result_value = do_reduce(prev_key, values)
        print(result_key + "\t" + str(result_value))
        values = []
    prev_key = key
    values.append(int(value))

if prev_key is not None:
    result_key, result_value = do_reduce(prev_key, values)
    print(result_key + "\t" + str(result_value))
```

Hadoop Streaming

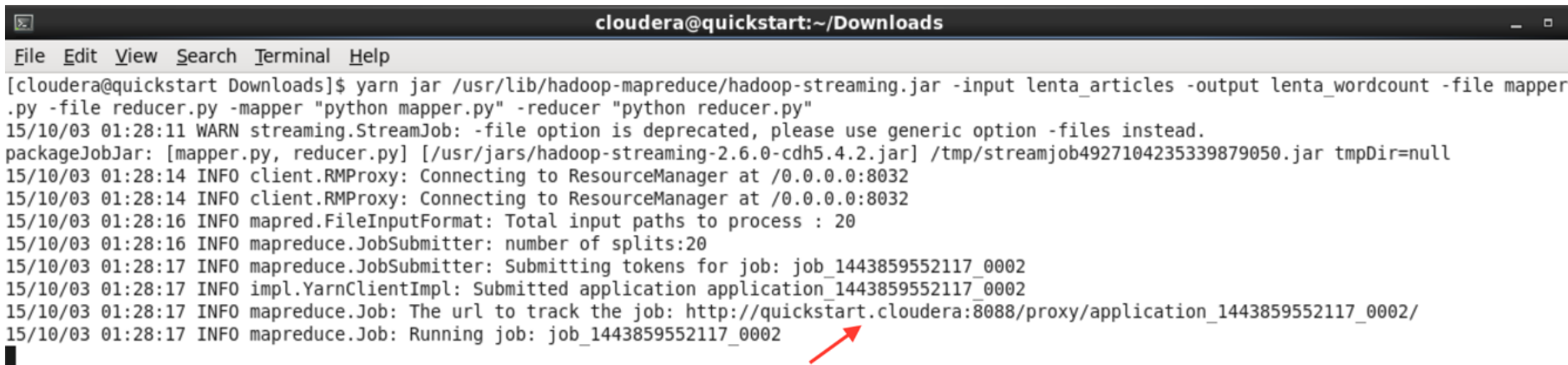
Данные, которые будет обрабатывать Hadoop должны храниться на HDFS. Загрузим наши статьи и положим на HDFS. Для этого нужно воспользоваться командой `hadoop fs`:

```
wget  
https://www.dropbox.com/s/opp5psid1x3jt41/lenta_articles.tar.gz  
tar xzvf lenta_articles.tar.gz  
hadoop fs -put lenta_articles
```

Утилита `hadoop fs` поддерживает большое количество методов для манипуляций с файловой системой.

Hadoop Streaming

```
yarn jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar\  
-input lenta_articles\  
-output lenta_wordcount\  
-file mapper.py\  
-file reducer.py\  
-mapper "python mapper.py"\  
-reducer "python reducer.py"
```



A terminal window titled "cloudera@quickstart:~/Downloads" showing the execution of a Hadoop Streaming job. The command entered is: `yarn jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -input lenta_articles -output lenta_wordcount -file mapper.py -file reducer.py -mapper "python mapper.py" -reducer "python reducer.py"`. The output shows a warning about the deprecated `-file` option, followed by logs for job submission and tracking. A red arrow points to the URL `http://quickstart.cloudera:8088/proxy/application_1443859552117_0002/` in the logs.

```
cloudera@quickstart:~/Downloads  
File Edit View Search Terminal Help  
[cloudera@quickstart Downloads]$ yarn jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -input lenta_articles -output lenta_wordcount -file mapper  
.py -file reducer.py -mapper "python mapper.py" -reducer "python reducer.py"  
15/10/03 01:28:11 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.  
packageJobJar: [mapper.py, reducer.py] [/usr/jars/hadoop-streaming-2.6.0-cdh5.4.2.jar] /tmp/streamjob4927104235339879050.jar tmpDir=null  
15/10/03 01:28:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032  
15/10/03 01:28:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032  
15/10/03 01:28:16 INFO mapred.FileInputFormat: Total input paths to process : 20  
15/10/03 01:28:16 INFO mapreduce.JobSubmitter: number of splits:20  
15/10/03 01:28:17 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1443859552117_0002  
15/10/03 01:28:17 INFO impl.YarnClientImpl: Submitted application application_1443859552117_0002  
15/10/03 01:28:17 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1443859552117_0002/  
15/10/03 01:28:17 INFO mapreduce.Job: Running job: job_1443859552117_0002
```

URL для
трекинга

Hadoop Streaming


MapReduce Job job_144... x

quickstart.cloudera:8088/proxy/application_1443859552117_0002/mapreduce/job/job_1443859552

Search

☆ 📄 ⬇️ 🏠 🗨️ ☰

☐ Cloudera ☐ Hue ☒ Hadoop ☒ HBase ☒ Impala ☒ Spark ☐ Solr ☐ Oozie ☐ Cloudera Manager ☐ Getting Started

 MapReduce Job job_1443859552117_0002

Cluster

Application

Job

- Overview
- Counters
- Configuration
- Map tasks
- Reduce tasks
- AM Logs

Tools

Job Overview

Job Name: streamjob4927104235339879050.jar

State: RUNNING

Uberized: false

Started: Sat Oct 03 01:28:28 PDT 2015

Elapsed: 1mins, 18sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Oct 03 01:28:20 PDT 2015	quickstart.cloudera:8042	logs

Task Type	Progress	Total	Pending	Running	Complete
Map	<div><div></div></div>	20	9	5	6
Reduce	<div><div></div></div>	1	0	1	0

Attempt Type	New	Running	Failed	Killed	Successful
Maps	9	5	0	0	6
Reduces	0	1	0	0	0

MapReduce Job job_14...

Software Update

cloudera@quickstart:...



mrjob

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner(self, word, counts):
        yield (word, sum(counts))

    def reducer(self, word, counts):
        yield (word, sum(counts))

if __name__ == '__main__':
    MRWordFreqCount.run()
```


mrjob

locally

```
python mrjob/examples/mr_word_freq_count.py README.rst > counts
```

on EMR (Elastic MapReduce © Amazon)

```
python mrjob/examples/mr_word_freq_count.py README.rst -r emr > counts
```

on Dataproc

```
python mrjob/examples/mr_word_freq_count.py README.rst -r dataproc > counts
```

on your Hadoop cluster

```
python mrjob/examples/mr_word_freq_count.py README.rst -r hadoop > counts
```

```

class MRMostUsedWord(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                    combiner=self.combiner_count_words,
                    reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)
        ]

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)

```



Hive

Hive is an open-source distributed data warehousing database which operates on HDFS. Hive was built for querying and analyzing big data. The data is stored in the form of tables (just like RDBMS). Data operations can be performed using a SQL interface called [HiveQL](#). Hive brings in SQL capability on top of Hadoop, making it a horizontally scalable database and a great choice for DWH environments.

The word count program counts the number of times each word occurs in the input. The word count can be written in HiveQL as:

```
1 DROP TABLE IF EXISTS docs;
2 CREATE TABLE docs (line STRING);
3 LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
4 CREATE TABLE word_counts AS
5 SELECT word, count(1) AS count FROM
6 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
7 GROUP BY word
8 ORDER BY word;
```

Spark

Spark is a distributed big data framework which helps extract and process large volumes of data in RDD format for analytical purposes. In short, it is not a database, but rather a framework which can access external distributed data sets using RDD (Resilient Distributed Data) methodology from data stores like Hive, Hadoop, and HBase. Spark operates quickly because it performs complex analytics in-memory.

