

УДК 004.65

ББК 32.073

Л-33

Лебедев А.С. Методы Big Data [Электронный ресурс]: Учебно-методическое пособие / Лебедев А.С., Магомедов Ш.Г. – М.: МИРЭА – Российский технологический университет, 2021. – 1 электрон. опт. диск (CD-ROM).

Целью настоящего учебно-методического пособия является изучение современных фреймворков для обработки больших данных, таких как Hadoop, Hive, Pig, Spark, а также изучение принципов разработки параллельных программ с применением паттерна MapReduce для организации отказоустойчивых вычислений. Приводится детальное описание примеров работы с файловой системой HDFS, инструментарием Pig, Hive, Spark, а также примеры программ в модели программирования MapReduce.

Учебное пособие предназначено для бакалавров, обучающихся по направлению 09.03.02 «Информационные системы и технологии», и специалистов, обучающихся по специальности 10.05.05 «Безопасность информационных технологий» в правоохранительной сфере».

Учебно-методическое пособие издаётся в авторской редакции.

Авторский коллектив: Лебедев Артем Сергеевич, Магомедов Шамиль Гасангусейнович

Рецензенты:

Аляева Юлия Владимировна, к.т.н., доцент кафедры вычислительных машин, систем и сетей, ФГБОУ ВО «Национальный исследовательский университет МЭИ»
Назаренко Максим Анатольевич, зав. кафедрой управления качеством и сертификации ФГБОУ ВО РТУ МИРЭА, к.ф-м.н., доцент.

Системные требования:

Наличие операционной системы Windows, поддерживаемой производителем.

Наличие свободного места в оперативной памяти не менее 128 Мб.

Наличие свободного места в памяти постоянного хранения (на жестком диске) не менее 30 Мб.

Наличие интерфейса ввода информации.

Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА — Российский технологический университет.

Объем: 5.01 мб

Тираж: 10

Оглавление

1.	Знакомство с Hadoop, MapReduce и файловой системой HDFS	5
	Технологии больших данных в Hadoop	5
	Установка и настройка Hadoop на Linux	6
	Задача WordCount.....	8
	Пример реализации на Java.....	10
	Пример реализации на Python2 (Hadoop Streaming).....	12
	Интерфейс WebHDFS	16
	Задание. Разработка клиента HDFS	18
2.	Разработка приложений с помощью паттерна MapReduce.....	20
	Модель программирования MapReduce в Hadoop	20
	Пример вычисления метрики TF-IDF на Java	21
	Этап 1. Вычисление числителей TF	22
	Этап 2. Вычисление знаменателей TF	24
	Этап 3. Вычисление TF-IDF.....	25
	Технология Hadoop Streaming.....	30
	Пример реализации байесовской фильтрации спама на Python.....	30
	Описание модели классификации	30
	Описание набора данных	32
	Подготовка данных	33
	Этап 1. Вычисление TF	34
	Этап 2. Вычисление $P(W_i S)$ и $P(W_i H)$ (только числители)	36
	Этап 3. Вычисление $P(W_i S)$ и $P(W_i H)$ (только знаменатели).....	38
	Этап 4. Вычисление вероятностей $P(S W_i)$ и $P(H W_i)$	39
	Этап 5. Классификация сообщений тестовой выборки.....	40
	Задание. Алгоритм кросс-корреляции	43
3.	Обработка реляционных данных в экосистеме Hadoop	45
	Технология Hive	45

Установка и настройка Hive на Hadoop	46
Примеры работы с Hive	48
Пример разбиения таблицы на разделы (partitions).....	48
Пример разбиения таблицы на гнезда (buckets)	53
Технология Pig.....	58
Установка и настройка Pig на Hadoop	59
Pig Latin: отношения, сумки, кортежи, поля	61
Пример обработки журнала веб-сервера с помощью Pig	64
Задание. Обработка реляционных данных с применением Hive, Pig, MapReduce.....	68
4. Анализ данных с помощью Spark	69
Технология Spark.....	69
Установка и настройка Spark	69
Автономный режим	69
Режим Spark on YARN.....	71
Настройка версии Python для pyspark	72
Примеры работы с реляционными данными.....	73
Максимальная годовая температура	73
Анализ социальной активности на примере дампа twitter.....	76
Задание. Анализ социальной активности на примере дампа twitter	80
Литература	81
Сведения об авторах	81
Приложение 1. Скрипты для управления сервисами Hadoop	82
Приложение 2. Исходный текст примера WordCount.java	84
Приложение 3. Исходный текст примера TFIDF	85
Приложение 4. Тестовые примеры для Pig	90
Приложение 5. Тестовый пример для Spark.....	91

1. Знакомство с Hadoop, MapReduce и файловой системой HDFS

Технологии больших данных в Hadoop

Мы живем в эпоху, когда объемы данных, с которыми нужно работать ежедневно, превзошли возможности одной сколь угодно мощной вычислительной машины. Большие данные сопряжены не только с двумя фундаментальными проблемами их хранения и обработки, но, что более важно, сложностями в их интерпретации и понимании, как превращать их в конкурентное преимущество. Hadoop заполняет пробел на рынке, предоставляя эффективные средства хранения и управления вычислительными ресурсами для обработки значительных объемов данных. Это распределенная система, которая предлагает способ распараллеливания и выполнения программ на кластере машин. Hadoop был принят такими технологическими гигантами, как Yahoo, Facebook и Twitter для удовлетворения своих потребностей в больших данных, и он продолжает проникать во все отрасли промышленности.

Hadoop — это платформа, которая обеспечивает как распределенное хранилище, так и вычислительные возможности. Первоначально Hadoop был задуман для устранения проблемы масштабируемости, которая существовала в Nutch — краулере с открытым исходным кодом и поисковой системе. В то время Google опубликовал статьи, в которых описывалась его новая распределенная файловая система Google (GFS), а также MapReduce — вычислительная среда для параллельной обработки. Успешная реализация концепций этих документов в Nutch привела к его разделению на два отдельных проекта, второй из которых стал Hadoop — проектом первого уровня Apache.

Собственно, Hadoop — это распределенная архитектура «главный-подчиненный», которая состоит из распределенной файловой системы Hadoop (HDFS) для хранения и MapReduce для обеспечения вычислительных возможностей. Характерными чертами Hadoop являются разделение данных и параллельные вычисления над большими наборами данных. Его хранилище и вычислительные возможности масштабируются с добавлением узлов в кластер, и могут обрабатывать петабайты данных в кластерах с тысячами узлов.

HDFS — это компонент хранения Hadoop, распределенная файловая система, вдохновленная статьей о файловой системе Google (GFS). HDFS оптимизирована для обеспечения высокой пропускной способности и лучше всего работает при чтении и записи больших файлов (гигабайт и больше). Для поддержки этой пропускной способности HDFS использует необычно большие

(для файловой системы) размеры блоков и оптимизацию локализации данных для уменьшения сетевого ввода/вывода.

Масштабируемость и доступность также являются ключевыми характеристиками HDFS, что отчасти достигается за счет репликации данных и отказоустойчивости. HDFS реплицирует файлы заданное количество раз, устойчив к программным и аппаратным сбоям, а также автоматически реплицирует блоки данных на вышедших из строя узлах.

Установка и настройка Hadoop на Linux

— команда от имени суперпользователя (root)

\$ — команда от имени обычного пользователя

Как стать root?

\$ su

(нужно ввести пароль root, если он был задан)

или

\$ sudo su

(нужно ввести свой пароль, если Вы являетесь sudoer)

Если Вы нигде не задавали пароль root, значит, скорее всего, Вы — sudoer, и имеете административные права. Это стандартная ситуация для дистрибутивов (например, Ubuntu).

В таком случае можно выполнять административные команды, приписав в начале sudo. Но, чтобы не приписывать каждый раз, легче переключиться на root. Закончив выполнение команд от имени root, можно вернуться в свой терминал набрав exit или нажав ctrl+d.

1. Установите необходимое ПО. Ознакомьтесь с его возможностями.

yum install mc wget curl ssh rsync screen make

Команда приведена для дистрибутива CentOS 7.

Далее все команды выполняются от имени обычного пользователя.

Перейдите в домашний каталог, если Вы не там:

\$ cd ~

2. Скачайте Java SE Development Kit 8 с сайта oracle.com.

Нужный вариант — для Linux x64 в архиве .tar.gz. Переместите архив в домашний каталог. Распаковать архив можно, например, так:

\$ tar -xvf jdk-8u221-linux-x64.tar.gz

3. Скачайте Hadoop.

Воспользуйтесь утилитой wget для скачивания архива:

\$ wget http://mirror.linux-ia64.org/apache/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz

Распаковать архив можно так:

\$ tar -xvf hadoop-2.9.2.tar.gz

4. Настройте переменные окружения.

Необходимо отредактировать специальный файл в домашнем каталоге, который исполняется оболочкой при Вашем входе. Для Centos/Fedora это файл `~/.bash_profile`, для Ubuntu это файл `~/.bashrc`. Убедитесь, что этот файл имеет следующий текст в конце:

```
# User specific environment and startup programs

JAVA_HOME=$HOME/jdk1.8.0_221
HADOOP_HOME=$HOME/hadoop-2.9.2
PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin

export JAVA_HOME
export HADOOP_HOME
export PATH
```

Можно отредактировать его с помощью нажатия F4 в менеджере MidnightCommander (запускается `mc`), а можно вызвать редактор из командной строки:

```
$ mcedit ~/.bash_profile
```

Изменения вступят в силу только тогда, когда вы повторно запустите сеанс пользователя. Этот этап можно отложить на самый конец процесса установки и конфигурации.

5. Настройте беспарольный доступ по SSH.

```
$ mkdir ~/.ssh
$ chmod 700 ~/.ssh
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

Команда

```
$ ssh localhost
```

должна давать Вам доступ по сети (локальная петля) на Вашу машину без пароля. После подключения нажмите `ctrl+d` или наберите `exit`.

6. Настройте Hadoop.

Для удобства можно перейти в каталог `~/hadoop-2.9.2` и отредактировать конфигурационные файлы так:

```
$ mcedit имя_файла
etc/hadoop/hadoop-env.sh
```

Указать в этом файле `JAVA_HOME=$HOME/jdk1.8.0_221`

`etc/hadoop/core-site.xml`

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

etc/hadoop/hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

etc/hadoop/mapred-site.xml

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

etc/hadoop/yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Задание. Ознакомьтесь с документацией Hadoop ~/hadoop-2.9.2/share/doc/hadoop/index.html и объясните назначение каждого конфигурационного параметра.

7. Установите вспомогательные скрипты из архива scripts.tar.gz.

Файлы должны быть размещены в каталоге ~/bin. Текст скриптов приведен в приложении 1.

- hdpStart.sh — старт HDFS и yarn;
- hdpStop.sh — остановка yarn и HDFS;
- hdpFormat.sh — форматирование HDFS, необходимо перед стартом первый раз;
- hdpCompileJava.sh — скрипт компиляции .java в .jar, поддерживает один параметр — имя входного .java-файла.

Задача WordCount

Есть коллекция документов. Каждый документ — это набор термов (слов). Необходимо подсчитать количество вхождений каждого терма во всех документах. Псевдокод “наивного” решения задачи представлен на рисунке (Рисунок 1).

Распакуйте и запустите пример WordCount. Если архив лежит в домашнем каталоге, и Вы находитесь в нем же:

```
$ mkdir mysources
$ tar -xvf WordCount.tar.gz
$ mv WordCount mysources
```

```

class Mapper
    method Map (docid id, doc d)
        for all term t in doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce (term t, counts [c1, c2,...])
        sum = 0
        for all count c in [c1, c2,...] do
            sum = sum + c
        Emit(term t, count sum)

```

Рисунок 1 — Псевдокод наивного решения задачи WordCount

Выполните подготовительную работу.

Отформатируйте hdfs:

```
$ hdpFormat.sh
```

Запустите сервисы hadoop:

```
$ hdpStart.sh
```

Перейдите в каталог примера:

```
$ cd ~/mysources/WordCount
```

Содержимое архива проиллюстрировано на рисунке (Рисунок 2).

```
[aslebedev@tementy WordCount]$ ls
copyInput.sh  countReduce.py  file02    runJar.sh  WordCount.jar
countMap.py   file01       Makefile  runPy.sh   WordCount.java
[aslebedev@tementy WordCount]$
```

Рисунок 2 — Файлы WordCount

copyInput.sh — скрипт, копирующий 2 тестовых документа в HDFS (Рисунок 3).

```

WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
copyInput.sh      [----]  0 L:[ 1+ 7  8/  8] *(148 / 148b) <EOF>      [*][X]
#!/bin/bash

hdfs dfs -mkdir wordcount
hdfs dfs -mkdir wordcount/input

hdfs dfs -put file01 wordcount/input/
hdfs dfs -put file02 wordcount/input/

```

Рисунок 3 — Пример копирования исходных данных в HDFS

Запустите копирование командой ./copyInput.sh, и убедитесь, что содержимое в HDFS целостно (Рисунок 4).

```
WordCount : bash — Konsole
File Edit View Bookmarks Settings Help
drwxr-xr-x - aslebedev supergroup 0 2019-10-09 14:36 wordcount/input
[aslebedev@tementy WordCount]$ hdfs dfs -ls wordcount/input
Found 2 items
-rw-r--r-- 1 aslebedev supergroup 25 2019-10-09 14:36 wordcount/input/
file01
-rw-r--r-- 1 aslebedev supergroup 27 2019-10-09 14:36 wordcount/input/
file02
[aslebedev@tementy WordCount]$ hdfs dfs -cat wordcount/input/file01
Hello World Goodbye World[aslebedev@tementy WordCount]$
[aslebedev@tementy WordCount]$
[aslebedev@tementy WordCount]$ hdfs dfs -cat wordcount/input/file02
Hello Hadoop Goodbye Hadoop[aslebedev@tementy WordCount]$
[aslebedev@tementy WordCount]$ cat file01
Hello World Goodbye World[aslebedev@tementy WordCount]$
[aslebedev@tementy WordCount]$ cat file02
Hello Hadoop Goodbye Hadoop[aslebedev@tementy WordCount]$
[aslebedev@tementy WordCount]$
```

Рисунок 4 — Пример вывода содержимого HDFS на терминал

Пример реализации на Java

Задание. Изучите исходный текст WordCount.java (приложение 2). Пользуясь документацией Hadoop, объясните архитектурные решения и использование специфичных классов.

Собрать Java-код можно командой make (Рисунок 5):

```
[aslebedev@tementy WordCount]$ make
hdpCompileJava.sh WordCount.java && rm -f WordCount*.class
[aslebedev@tementy WordCount]$
```

Рисунок 5 — Сборка WordCount.java

Задание. Изучите скрипт запуска задачи runJar.sh (Рисунок 6). Объясните назначение каждой команды и ее параметров.

```
WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
runJar.sh      [----] 0 L:[ 1+ 7 8/ 8] *(160 / 160b) <EOF> [*][X]
#!/bin/bash

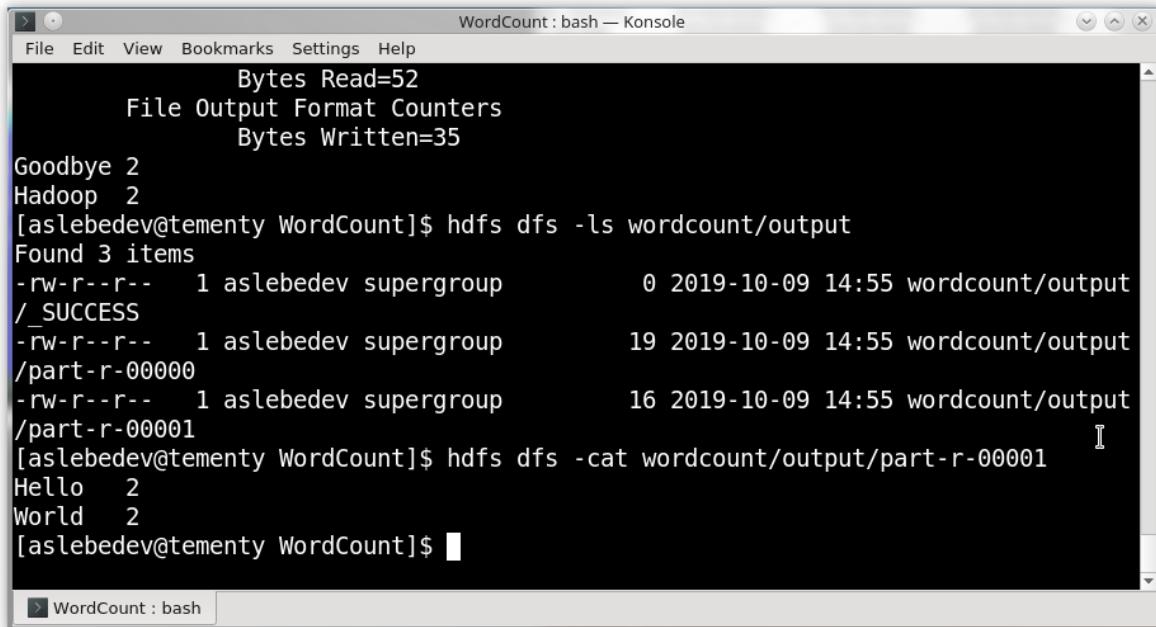
hdfs dfs -rm -r wordcount/output
hadoop jar WordCount.jar WordCount wordcount/input wordcount/output
hdfs dfs -cat wordcount/output/part-r-00000
```

Рисунок 6 — Скрипт запуска реализации WordCount на Java

Запустите расчет:

```
$ ./runJar.sh
```

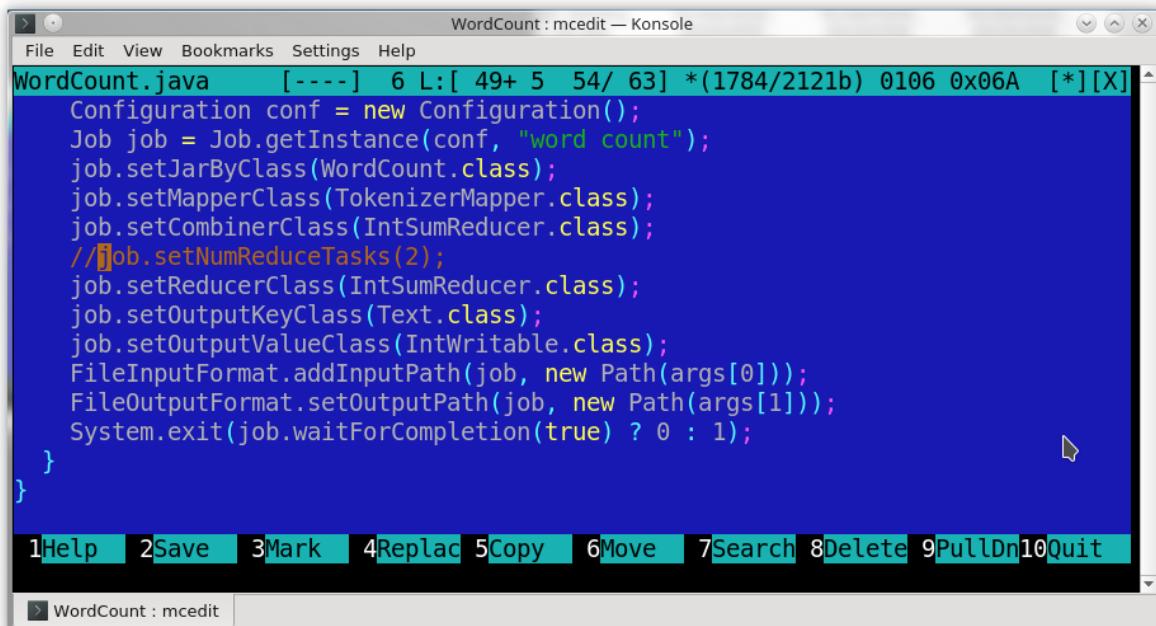
Задание. Объясните, почему результат разбит на 2 файла (Рисунок 7).



```
WordCount : bash — Konsole
File Edit View Bookmarks Settings Help
Bytes Read=52
File Output Format Counters
Bytes Written=35
Goodbye 2
Hadoop 2
[aslebedev@tementy WordCount]$ hdfs dfs -ls wordcount/output
Found 3 items
-rw-r--r-- 1 aslebedev supergroup 0 2019-10-09 14:55 wordcount/output/_SUCCESS
-rw-r--r-- 1 aslebedev supergroup 19 2019-10-09 14:55 wordcount/output/part-r-00000
-rw-r--r-- 1 aslebedev supergroup 16 2019-10-09 14:55 wordcount/output/part-r-00001
[aslebedev@tementy WordCount]$ hdfs dfs -cat wordcount/output/part-r-00001
Hello 2
World 2
[aslebedev@tementy WordCount]$
```

Рисунок 7 — Запуск примера WordCount.java с двумя экземплярами reducer

Уменьшите количество экземпляров reducer до умалчиваемого (Рисунок 8).



```
WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
WordCount.java [----] 6 L:[ 49+ 5 54/ 63] *(1784/2121b) 0106 0x06A [*][X]
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
//job.setNumReduceTasks(2);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}

1Help | 2Save | 3Mark | 4Replac 5Copy | 6Move | 7Search | 8Delete | 9PullDn| 10Quit
```

Рисунок 8 — Редактирование WordCount.java

Пересоберите и перезапустите пример. Объясните вывод (Рисунок 9).

```
WordCount : bash — Konsole
File Edit View Bookmarks Settings Help
Total committed heap usage (bytes)=531103744
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=52
File Output Format Counters
Bytes Written=35
Goodbye 2
Hadoop 2
Hello 2
World 2
[aslebedev@tementy WordCount]$
```

Рисунок 9 — Запуск примера WordCount.java с одним экземпляром reducer

Пример реализации на Python2 (Hadoop Streaming)

Задание. Изучите исходные тексты countMap.py (Рисунок 10) и countReduce.py (Рисунок 11). Пользуясь документацией по Hadoop Streaming, объясните механизм передачи данных от фазы Map к фазе Reduce.

```
WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
countMap.py [----] 37 L:[ 1+ 6 7/ 8 ] *(133 / 134b) 0010 0x00A [*][X]
#!/usr/bin/python

import sys

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

Рисунок 10 — Реализация Mapper примера WordCount на Python2

```
WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
countReduce.py [----] 35 L:[ 1+14 15/ 16 ] *(348 / 349b) 0010 0x00A [*][X]
#!/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)
1Help 2Save 3Mark 4Replace 5Copy 6Move 7Search 8Delete 9PullDown 10Quit
```

Рисунок 11 — Реализация Reducer примера WordCount на Python2

Задание. Изучите скрипт запуска задачи runPy.sh (Рисунок 12). Объясните назначение каждой команды и ее параметров.

```
WordCount : mcedit — Konsole
File Edit View Bookmarks Settings Help
runPy.sh [----] 11 L:[ 1+ 0 1/ 15 ] *(11 / 407b) 0010 0x00A [*][X]
#!/bin/bash

hdfs dfs -rm -r wordcount/output

yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="WordCount Job via Streaming" \
-files `pwd`/countMap.py,`pwd`/countReduce.py \
-input wordcount/input/ \
-output wordcount/output/ \
-mapper `pwd`/countMap.py \
-combiner `pwd`/countReduce.py \
-reducer `pwd`/countReduce.py

hdfs dfs -cat wordcount/output/part-00000
1Help 2Save 3Mark 4Replace 5Copy 6Move 7Search 8Delete 9PullDown 10Quit
```

Рисунок 12 — Скрипт запуска реализации WordCount на Python

Запустите расчет:

```
$ ./runPy.sh
```

Задание. Объясните, почему результат расчета сформирован в 1 файле (Рисунок 13).

```
WordCount : bash — Konsole
File Edit View Bookmarks Settings Help
Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=52
File Output Format Counters
    Bytes Written=35
19/10/09 14:51:30 INFO streaming.StreamJob: Output directory: wordcount/output/
Goodbye 2
Hadoop 2
Hello 2
World 2
[aslebedev@tementy WordCount]$
```

Рисунок 13 — Запуск примера WordCount на Python

Перейдите по ссылке <http://localhost:50070> для отображения информации о NameNode и перейдите к обзору HDFS (Рисунок 14). Предполагается, что узел имен доступен локально (localhost) и порт для взаимодействия стандартный (50070).

The screenshot shows a web browser window titled "Namenode information". The URL in the address bar is localhost:50070/dfshealth.html#tab-overview. The page has a green header bar with tabs: "Hadoop" (selected), "Overview", "Datanodes", "Datanode Volume Failures", "Snapshot", "Startup Progress", and "Utilities". A dropdown menu under "Utilities" shows "Browse the file system" and "Logs". The main content area is titled "Overview 'localhost:9000' (active)". It displays the following information:

Started:	Wed Oct 09 14:25:58 +0300 2019
Version:	2.9.1, re30710aea4e6e55e69372929106cf119af06fd0e
Compiled:	Mon Apr 16 12:33:00 +0300 2018 by root from branch-2.9.1
Cluster ID:	CID-1671e6ac-f3bd-406e-a97d-2e6fab3313c6
Block Pool ID:	BP-602880187-192.168.1.247-1554813066359

Below this is a section titled "Summary" with the note "Security is off." and the URL "localhost:50070/explorer.html".

Рисунок 14 — Информация о NameNode

Пройдите по ссылкам до каталога wordcount/output (Рисунок 15).

Name	Size	Last Modified	Replication	Block Size
_SUCCESS	0 B	Mar 05 10:44	1	128 MB
part-r-00000	19 B	Mar 05 10:44	1	128 MB
part-r-00001	16 B	Mar 05 10:44	1	128 MB

Рисунок 15 — Результат работы MapReduce для задачи WordCount в HDFS

Реализована функциональность утилит head и tail для просмотра содержимого файлов в веб-интерфейсе (Рисунок 16).

Block information	Block 0
Block ID:	1073744068
Block Pool ID:	BP-1187330923-192.168.43.119-1583583500802
Generation Stamp:	3248
Size:	16
Availability:	• tementy

File contents:

```
Goodbye 2
Hadoop 2
```

Рисунок 16 — Просмотр содержимого файлов в HDFS

Вы можете установить идентификатор пользователя, от имени которого работают веб-интерфейсы Hadoop, задав свойство `hadoop.http.staticuser.user` в `$HADOOP_HOME/etc/hadoop/core-site.xml`. По умолчанию это `dr.who`, который не является суперпользователем, поэтому системные файлы недоступны через веб-интерфейс. Можно указать имя unix-пользователя, от имени которого функционирует псевдораспределенная установка Hadoop, тем самым получив те же права доступа, что и при работе с утилитами командной строки.

Интерфейс WebHDFS

Интерфейс командной строки hdfs dfs подходит для интерактивного взаимодействия с файловой системой HDFS: создание каталогов, копирование файлов, вывод результатов вычислений на терминал. Однако, вызов утилиты командной строки не является эффективным решением для организации взаимодействия с HDFS из программного кода приложений в силу его опосредованности. Помимо интерфейса командной строки существует REST API WebHDFS, с помощью которого возможно выполнение операций с удаленной файловой системой посредством отправки запросов по протоколу HTTP.

Подробнее ознакомиться с документацией функций можно в разделе «WebHDFS REST API» (`$HADOOP_HOME/share/doc/hadoop/hadoop-project-dist/hadoop-hdfs/WebHDFS.html`).

Для обеспечения работоспособности WebHDFS, необходимо в файле `$HADOOP_HOME/etc/hadoop/hdfs-site.xml` прописать в блоке configuration следующий фрагмент кода:

```
<property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
</property>
```

Если в блоке configuration уже есть блок property, указанный выше фрагмент вписывается до или после него.

При изучении API WebHDFS можно воспользоваться утилитой curl. Приведем пример загрузки файла в HDFS с помощью HTTP-запросов.

Этап 1. Взаимодействие с узлом имен.

Если предполагается создание файла **file01** в каталоге **/tmp**, необходимо сообщить об этом компоненту NameNode следующим запросом PUT:

```
$ curl -i -X PUT "http://localhost:50070/webhdfs/v1/tmp/file01?\nuser.name=aslebedev&op=CREATE&overwrite=true"
```

Предполагается, что узел имен доступен локально (localhost) и порт для взаимодействия стандартный (50070). Код операции CREATE передается в параметре op, флаг перезаписи существующего файла передается в параметре overwrite. При этом также необходимо указать в параметрах учетную запись, от имени которой происходит операция. При работе с утилитами командной строки идентификатор пользователя, который Hadoop использует для разрешений в HDFS, определяется путем выполнения команды whoami в клиентской системе. Точно так же имена групп получаются из выходных данных команды groups. В параметре user.name можно указать имя unix-пользователя, от имени которого функционирует псевдораспределенная установка Hadoop, тем самым получив те же права доступа, что и при работе с утилитами командной строки.

Узел имен дает следующий ответ в виде перенаправления (redirect):

```
HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 04 Mar 2021 13:22:41 GMT
Date: Thu, 04 Mar 2021 13:22:41 GMT
Pragma: no-cache
Expires: Thu, 04 Mar 2021 13:22:41 GMT
Date: Thu, 04 Mar 2021 13:22:41 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie:
hadoop.auth="u=aslebedev&p=aslebedev&t=simple&e=1614900161437&s=d1JPYrbC9ZJ1VC32
0A+xR6BEF0U="; Path=/; HttpOnly
Location:
http://tementy:50075/webhdfs/v1/tmp/file01?op=CREATE&user.name=aslebedev&namenod
erpcaddress=localhost:9000&createflag=&createparent=true&overwrite=true
Content-Length: 0
```

При этом в поле Location передается URL узла данных, на который необходимо в дальнейшем отправлять содержимое файла.

Этап 2. Взаимодействие с узлом данных.

Осуществить загрузку содержимого файла `~/mysources/WordCount/file01` можно следующим запросом:

```
$ curl -i -X PUT -T ~/mysources/WordCount/file01 \
"http://tementy:50075/webhdfs/v1/tmp/file01?op=CREATE&user.name=aslebedev&\
namenoderpcaddress=localhost:9000&createflag=&createparent=true&overwrite=true"
HTTP/1.1 100 Continue
```

```
HTTP/1.1 201 Created
Location: hdfs://localhost:9000/tmp/file01
Content-Length: 0
Access-Control-Allow-Origin: *
Connection: close
```

Проверить результат можно командой:

```
$ hdfs dfs -cat /tmp/file01
Hello World Goodbye World
```

Создание нового каталога в HDFS. Создать каталог `dir` в домашнем каталоге `/user/aslebedev` можно следующим запросом:

```
$ curl -i -X PUT "http://localhost:50070/webhdfs/v1/user/aslebedev/dir?\
user.name=aslebedev&op=MKDIRS"
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 04 Mar 2021 13:53:14 GMT
Date: Thu, 04 Mar 2021 13:53:14 GMT
Pragma: no-cache
Expires: Thu, 04 Mar 2021 13:53:14 GMT
Date: Thu, 04 Mar 2021 13:53:14 GMT
Pragma: no-cache
Content-Type: application/json
X-FRAME-OPTIONS: SAMEORIGIN
```

```
Set-Cookie:  
hadoop.auth="u=aslebedev&p=aslebedev&t=simple&e=1614901994355&s=CKDDErY1GDDIyBBL  
FVWenOesZcE="; Path  
h=/; HttpOnly  
Transfer-Encoding: chunked
```

```
{"boolean":true}
```

При этом также необходимо указывать имя пользователя для идентификации.

Вывод содержимого каталога. Вывести содержимое, например, каталога **/user/aslebedev/wordcount** можно следующим запросом:

```
$ curl -i "http://localhost:50070/webhdfs/v1/user/aslebedev/wordcount?\  
user.name=aslebedev&op=LISTSTATUS"  
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Expires: Thu, 04 Mar 2021 13:51:25 GMT  
Date: Thu, 04 Mar 2021 13:51:25 GMT  
Pragma: no-cache  
Expires: Thu, 04 Mar 2021 13:51:25 GMT  
Date: Thu, 04 Mar 2021 13:51:25 GMT  
Pragma: no-cache  
Content-Type: application/json  
X-FRAME-OPTIONS: SAMEORIGIN  
Set-Cookie:  
hadoop.auth="u=aslebedev&p=aslebedev&t=simple&e=1614901885245&s=HIWeGbJTMb9Oif9G  
CcW6SgnmdzU="; Path  
h=/; HttpOnly  
Transfer-Encoding: chunked
```

```
{"FileStatuses": [{"FileStatus": [  
 {"accessTime": 0, "blockSize": 0, "childrenNum": 2, "fileId": 16395, "group": "supergroup",  
 "length": 0, "modificationTime":  
 "1583583568078, "owner": "aslebedev", "pathSuffix": "input", "permission": "755", "replication": 0, "storagePolicy": 0,  
 "type": "DIRECTORY"},  
 {"accessTime": 0, "blockSize": 0, "childrenNum": 3, "fileId": 20463, "group": "supergroup",  
 "length": 0, "modificationTime":  
 "1613979238838, "owner": "aslebedev", "pathSuffix": "output", "permission": "755", "replication": 0, "storagePolicy": 0  
 , "type": "DIRECTORY"}]}]
```

При этом также необходимо указывать имя пользователя для идентификации.

Задание. Разработка клиента HDFS

Разработать клиент HDFS, поддерживающий операции:

- **mkdir <имя каталога в HDFS>** (создание каталога в HDFS);
- **put <имя локального файла>** (загрузка файла в HDFS);
- **get <имя файла в HDFS>** (скачивание файла из HDFS);

- append <имя локального файла> <имя файла в HDFS> (конкатенация файла в HDFS с локальным файлом);
- delete <имя файла в HDFS> (удаление файла в HDFS);
- ls (отображение содержимого текущего каталога в HDFS с разделением файлов и каталогов);
- cd <имя каталога в HDFS> (переход в другой каталог в HDFS, ".." — на уровень выше);
- llst (отображение содержимого текущего локального каталога с разделением файлов и каталогов);
- lcd <имя локального каталога> (переход в другой локальный каталог, ".." — на уровень выше).

Имена файлов и каталогов не содержат путей и символа "/".

Параметры командной строки клиента: сервер, порт, имя пользователя.

Пример запуска клиента: ./myhdfscli.rb localhost 50070 aslebedev

Использовать WebHDFS REST API и любой язык программирования.

2. Разработка приложений с помощью паттерна MapReduce

Модель программирования MapReduce в Hadoop

MapReduce — модель программирования, ориентированная на обработку данных. Эта модель проста, но не настолько, чтобы в ее контексте нельзя было реализовать полезные программы. Hadoop позволяет запускать программы MapReduce, написанные на различных языках. Все программы MapReduce параллельны по своей природе, следовательно, крупномасштабный анализ данных становится доступным для всех, у кого в распоряжении имеется достаточно вычислительных машин. Достоинства MapReduce в полной мере проявляются в работе с большими наборами данных.

Работа MapReduce основана на разбиении обработки данных на две фазы: фазу отображения (map) и фазу свертки (reduce). Каждая фаза использует в качестве входных и выходных данных пары «ключ-значение», типы которых могут быть выбраны программистом. Программист также задает две функции: отображения и свертки:

$$\begin{aligned} \text{map: } & (k_1, v_1) \rightarrow [(k_2, v_2)]; \\ \text{reduce: } & (k_2, [v_2]) \rightarrow [(k_3, v_3)], \end{aligned}$$

где k_1, k_2, k_3 и v_1, v_2, v_3 — обозначения типов данных ключей и значений соответственно. MapReduce гарантирует, что вход каждого reducer отсортирован по ключу. Процесс, посредством которого система выполняет сортировку и передает выходные данные map в reduce в качестве входных данных, известен как тасовка и сортировка (Shuffle and Sort). На этом этапе те пары «ключ-значение», у которых ключ совпадает, объединяются: $[(k_2, v_2)] \rightarrow (k_2, [v_2])$, затем разделяются между экземплярами reduce и сортируются по ключу. Каждый экземпляр reduce получает все значения, связанные с одним и тем же ключом.

Многие задания MapReduce ограничиваются по пропускной способности, доступной в кластере, поэтому передачу данных между задачами отображения и свертки желательно свести к минимуму. Hadoop позволяет пользователю задать комбинирующую функцию, которая будет выполняться для выходных данных отображения: $\text{combine: } (k_2, [v_2]) \rightarrow [(k_2, v_2)]$.

Выходные данные комбинирующей функции образуют ввод функции свертки. Так как комбинирующая функция является оптимизацией, Hadoop не предоставляет гарантий относительно того, сколько раз она будет вызвана для конкретной записи выходных данных отображения, и будет ли вызвана вообще. Другими словами, при вызове комбинирующей функции нуль, один или несколько раз функция свертки должна выдавать одинаковый результат.

Пример вычисления метрики TF-IDF на Java

Расчет TF-IDF используется при работе с текстом.

TF (term frequency — частота слова) — отношение числа вхождения некоторого слова к общему количеству слов документа. Таким образом, оценивается важность слова t в пределах отдельного документа d :

$$TF(t, d) = \frac{n_t}{\sum_k n_k}, \quad (2.1)$$

где n_t есть число вхождений слова t в документ d , а в знаменателе — общее число слов в данном документе d .

IDF (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции. Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF:

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D | t \in d\}|}, \quad (2.2)$$

где $|D|$ — число документов в коллекции; $|\{d \in D | t \in d\}|$ — число документов из коллекции D , в которых встречается t (когда $n_t > 0$). Выбор основания логарифма в формуле не имеет значения, поскольку изменение основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D). \quad (2.3)$$

Рассмотрим вычисление метрики TF-IDF на Java. В качестве входных данных выступает та же коллекция документов, что и для задачи WordCount. Каталог с коллекцией документов передается первым параметром командной строки, выходной каталог для всех заданий MapReduce — вторым (Рисунок 17). В нем будут созданы подкаталоги для записи результата работы каждого задания MapReduce. Имена подкаталогов и заданий MapReduce являются одинаковыми в целях упрощения интерпретации результатов.

```
#!/bin/bash
hdfs dfs -rm -r wordcount/output
hadoop jar out/artifacts/tfidf_jar/tfidf.jar wordcount/input wordcount/output
```

Рисунок 17 — Исходный текст скрипта run.sh для запуска примера TFIDF

Этап 1. Вычисление числителей TF

Фаза Map. Формат ввода по умолчанию — TextInputFormat, который использует байтовое смещение строки в файле, представляемое как LongWritable, и применяемое в качестве ключа. В качестве значения выступает сама строка, представляемая как Text. В этой фазе реализуется отображение:

$$(\text{offset}, \text{line}) \rightarrow [(\text{word}, \text{document}), 1].$$

Смещение offset имеет тип Object и не используется в программном коде. Стока line имеет тип Text. Она разбивается на слова с помощью StringTokenizer. При этом для каждого слова в контекст записывается пара ключ-значение, в которой в качестве ключа выступает кортеж из слова и имени документа, в котором оно встретилось, а в качестве значения — единица, завернутая в тип IntWritable. Она фиксирует факт того, что слово word встретилось в документе document. В качестве имени документа выступает имя файла в HDFS, которое можно вычленить, обратившись к методу getPath у экземпляра класса FileSplit, представляющего входной фрагмент данных. Кортеж из word и document представляется типом WordDocumentTuple. Это класс-обертка, реализующий интерфейс WritableComparable (серIALIZАЦИЯ/десериализация содержимого; компаратор для определения отношения порядка, необходимого при сортировке в фазе Shuffle&Sort). Подробная реализация WordDocumentTuple приведена в приложении 3. Исходный текст фазы Map приведен на рисунке (Рисунок 18).

```
public static class TokenizerMapper extends Mapper<Object, Text, WordDocumentTuple,
IntWritable> {
    private WordDocumentTuple wdt = new WordDocumentTuple();
    private final static IntWritable one = new IntWritable(1);

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
        String docname = ((FileSplit) context.getInputSplit()).getPath().getName();

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            wdt.set(itr.nextToken(), docname);
            context.write(wdt, one);
        }
    }
}
```

Рисунок 18 — Фаза Map этапа 1

Фаза Reduce. В этой фазе выполняется вычисление числителей TF (количество вхождений слова в документ): для тех пар ключ-значение, у которых ключ совпадает, выполняется суммирование значений:

$$((\text{word}, \text{document}), [\text{integer value}]) \rightarrow ((\text{word}, \text{document}), \text{sum}).$$

Тип WritableComparable входного ключа фазы Reduce является совместимым с типом WordDocumentTuple выходного ключа фазы Map,

поскольку класс WordDocumentTuple реализует интерфейс WritableComparable. Исходный текст фазы Reduce приведен на рисунке (Рисунок 19).

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

    public void reduce(WritableComparable 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);
    }
}
```

Рисунок 19 — Фаза Reduce этапа 1

Фаза Combine. На этой фазе полезно сократить объем данных, поступающих на фазу Reduce, посчитав частичные суммы для числителей TF (количество вхождений слова в сообщение). Итоговые суммы будут вычислены в фазе Reduce. Редукцию частичных результатов, полученных от экземпляров map, можно выполнить универсальным алгоритмом IntSumReducer (Рисунок 19), складывающим целочисленные значения для тех пар ключ-значение, у которых ключ совпадает: ((word, document), [1]) → ((word, document), sum).

Запуск вычислений. На рисунке (Рисунок 20) приведен начальный фрагмент функции main, отвечающей за создание объекта конфигурации, и запуск задания MapReduce первого этапа. Класс IntSumReducer используется и в фазе Reduce, и в фазе Combine.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    // phase 1
    String job1Name = "tfidf-phase1-tf-numerators";
    String inputPath = args[0];
    String outputPath = args[1] + "/" + job1Name;

    Job job = Job.getInstance(conf, job1Name);
    job.setJarByClass(TFIDF.class);

    job.setMapperClass(TokenizerMapper.class);
    job.setMapOutputKeyClass(WordDocumentTuple.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(WordDocumentTuple.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(inputPath));
    FileOutputFormat.setOutputPath(job, new Path(outputPath));
    if (!job.waitForCompletion(true))
        System.exit(1);
```

Рисунок 20 — Запуск задания MapReduce этапа 1

Результат работы задания tfidf-phase1-tf-numerators представлен на рисунке (Рисунок 21).

Goodbye	file01	1
Hello	file01	1
World	file01	2
Goodbye	file02	1
Hadoop	file02	2
Hello	file02	1

Рисунок 21 — Результат работы этапа 1 для набора данных WordCount

Этап 2. Вычисление знаменателей TF

Фаза Map. В этой фазе обрабатывается результат, полученный на этапе 1. Реализуется следующее отображение: $(\text{offset}, \text{line}) \rightarrow [\text{document}, \text{wordcount}]$.

Смещение offset имеет тип Object и не используется в программном коде. Стока line имеет тип Text. Она разбивается на термы (word, document, wordcount) с помощью split. При этом для каждой строки в контекст записывается пара ключ-значение, в которой в качестве ключа выступает имя документа document, а в качестве значения — счетчик wordcount вхождений слова word в документ document, завернутый в тип IntWritable. Фактически происходит отбрасывание информации о слове из каждой строки выхода этапа 1, остается только имя документа и счетчик вхождений (числитель TF). Исходный текст фазы Map приведен на рисунке (Рисунок 22).

```
public static class P2Mapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text document = new Text();
    private IntWritable wordcount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
        String[] line = value.toString().split("\t");
        document.set(line[1]);
        wordcount.set(Integer.parseInt(line[2]));
        context.write(document, wordcount);
    }
}
```

Рисунок 22 — Фаза Map этапа 2

Фазы Reduce и Combine. Редукцию частичных результатов, полученных от экземпляров map, а также итоговую редукцию, можно выполнить универсальным алгоритмом IntSumReducer (Рисунок 19), складывающим целочисленные значения для тех пар ключ-значение, у которых ключ совпадает, тем самым осуществив расчет количества слов в документах:

$$(\text{document}, [\text{wordcount}]) \rightarrow (\text{document}, \text{sum}).$$

Запуск вычислений. На рисунке (Рисунок 23) приведен фрагмент функции main, отвечающий за запуск задания MapReduce второго этапа. Класс IntSumReducer используется и в фазе Reduce, и в фазе Combine.

```
// phase 2

String job2Name = "tfidf-phase2-tf-denominators";
inputPath = args[1] + "/" + job1Name;
outputPath = args[1] + "/" + job2Name;

job = Job.getInstance(conf, job2Name);
job.setJarByClass(TFIDF.class);

job.setMapperClass(P2Mapper.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileInputFormat.setInputPaths(job, new Path(inputPath));
FileOutputFormat.setOutputPath(job, new Path(outputPath));

if (!job.waitForCompletion(true))
    System.exit(2);
```

Рисунок 23 — Запуск задания MapReduce этапа 2

Результат работы задания tfidf-phase2-tf-denominators представлен на рисунке (Рисунок 24).

file01	4
file02	4

Рисунок 24 — Результат работы этапа 2 для набора данных WordCount

Этап 3. Вычисление TF-IDF

Фаза Map. В этой фазе обрабатывается результат, полученный на этапе 1. Реализуется следующее отображение:

$$(\text{offset}, \text{line}) \rightarrow [\text{word}, (\text{document}, \text{wordcount})].$$

Смещение offset имеет тип Object и не используется в программном коде. Стока line имеет тип Text. Она разбивается на термы (word, document, wordcount) с помощью split. При этом для каждой строки в контекст записывается пара ключ-значение, в которой в качестве ключа выступает слово word, а в качестве значения — кортеж из имени документа document и счетчика wordcount вхождений слова word в документ document. Кортеж из document и wordcount представляется типом DocumentCounterTuple. Это класс-обертка, реализующий интерфейс WritableComparable (сериализация/десериализация содержимого; компаратор для определения отношения порядка, необходимого

при сортировке в фазе Shuffle&Sort). Подробная реализация DocumentCounterTuple приведена в приложении 3. Исходный текст фазы Map приведен на рисунке (Рисунок 25).

```
public static class P3Mapper extends Mapper<Object, Text, Text, DocumentCounterTuple>{
    private DocumentCounterTuple dct = new DocumentCounterTuple();
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
        String[] line = value.toString().split("\t");
        word.set(line[0]);
        dct.set(line[1], Integer.parseInt(line[2]));
        context.write(word, dct);
    }
}
```

Рисунок 25 — Фаза Map этапа 3

Фаза Reduce. В этой фазе завершается вычисление метрики TF-IDF: $(word, [(document, wordcount)]) \rightarrow ((word, document), float\ TF-IDF\ value)$.

При этом завершается вычисление TF: числитель wordcount делится на длину документа document, вычисленную на этапе 2. Результат работы этапа 2 достаточно компактен для размещения в оперативной памяти, поскольку его наполнение зависит только от количества документов в коллекции. Поэтому целесообразно создать lookup-таблицу при инициализации Reducer (паттерн replicated join). В реализации на Java это будет ассоциативный массив docLengths, отображающий имена документов в их длины (Рисунок 26). Из пользовательского свойства docLengths объекта конфигурации извлекается выходной каталог этапа 2. Функция readLines считывает содержимое всех файлов, кроме служебных (например, игнорируется файл-метка _SUCCESS), и формирует единый список строк. При этом учитывается возможность применения кодеков сжатия. Подробная реализация функции readLines приведена в приложении 3. Каждая строка разделяется на термы (имя документа, длина документа), и в ассоциативном массиве docLengths создается новый элемент после преобразования типа String в Integer для длины документа.

```
public static class P3Reducer extends Reducer<Text, DocumentCounterTuple,
WordDocumentTuple, DoubleWritable> {
    public HashMap<String, Integer> docLengths = new HashMap<String, Integer>();

    public void setup(Context context) throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        for (String line : readLines(new Path(conf.get("docLengths", "<invalid>")),
conf)) {
            String[] dl = line.split("\t");
            docLengths.put(dl[0], Integer.parseInt(dl[1]));
        }
    }
}
```

Рисунок 26 — Инициализация lookup-таблицы

Результат объединения по принципу совпадения ключа word в фазе Shuffle&Sort (word, [(document, wordcount)]) перебирается с помощью итерируемого объекта values (Рисунок 27). Линейный список dcta наполняется клонами кортежей (document, wordcount) с тем, чтобы по ним можно было бы в дальнейшем совершить второй проход. Для объекта values это невозможно. Сохранять ссылку на элемент val в цикле for each также не представляется возможным, поскольку при итерировании объект val остается одним и тем же, но его информационное наполнение меняется. При наполнении списка dcta высчитывается его длина в переменной cnt, которая имеет семантику количества документов в коллекции, в которых встретилось слово word (параметр Text key) — значение знаменателя IDF. Числитель формулы IDF определяется тривиально — длина списка docLengths. При итерировании списка dcta для каждого кортежа (document, wordcount) выполняется следующее:

1. Формируется кортеж (key, document), завернутый в тип WordDocumentTuple (переменная wdt).
2. Рассчитывается метрика TF как частное wordcount и длины документа document.
3. Рассчитывается метрика IDF как частное длины списка docLengths и cnt. Логарифмирование не применяется в целях упрощения интерпретации результатов.
4. Рассчитывается метрика TF-IDF как произведение TF и IDF.
5. В контекст записывается пара ключ-значение, где в качестве ключа выступает кортеж wdt, а в качестве значения — метрика TF-IDF, завернутая в тип DoubleWritable.

```
public static class P3Reducer extends Reducer<Text, DocumentCounterTuple,
WordDocumentTuple, DoubleWritable> { ...
    private WordDocumentTuple wdt = new WordDocumentTuple();
    private DoubleWritable tfidf = new DoubleWritable();

    public void reduce(Text key, Iterable<DocumentCounterTuple> values, Context context)
throws IOException, InterruptedException {
    ArrayList<DocumentCounterTuple> dcta = new ArrayList<DocumentCounterTuple>();
    int cnt = 0;
    for (DocumentCounterTuple val : values) {
        cnt++;
        dcta.add(new DocumentCounterTuple(val.document, val.counter));
    }
    for (DocumentCounterTuple val : dcta) { // count TFIDF
        wdt.set(key.toString(), val.document);
        double tf = ((double) val.counter) / docLengths.get(val.document);
        double idf = ((double) docLengths.size()) / cnt;
        tfidf.set(tf * idf);
        context.write(wdt, tfidf);
    }
}
}
```

Рисунок 27 — Фаза Reduce этапа 3

Запуск вычислений. На рисунке (Рисунок 28) приведен фрагмент функции main, отвечающий за запуск задания MapReduce третьего этапа. В пользовательское поле docLengths объекта конфигурации помещается выходной каталог этапа 2.

```
// phase 3

conf.set("docLengths", outputPath);

String job3Name = "tfidf-phase3-tfidf";
inputPath = args[1] + "/" + job1Name;
outputPath = args[1] + "/" + job3Name;

job = Job.getInstance(conf, job3Name);
job.setJarByClass(TFIDF.class);

job.setMapperClass(P3Mapper.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(DocumentCounterTuple.class);

job.setReducerClass(P3Reducer.class);
job.setOutputKeyClass(WordDocumentTuple.class);
job.setOutputValueClass(DoubleWritable.class);

FileInputFormat.setInputPaths(job, new Path(inputPath));
FileOutputFormat.setOutputPath(job, new
Path(outputPath));

if (!job.waitForCompletion(true))
System.exit(3);

System.exit(0);
```

Рисунок 28 — Запуск задания MapReduce этапа 3

Результат работы задания tfidf-phase3-tfidf представлен на рисунке (Рисунок 29).

Goodbye	file02	0.25
Goodbye	file01	0.25
Hadoop	file02	1.0
Hello	file02	0.25
Hello	file01	0.25
World	file01	1.0

Рисунок 29 — Результат работы этапа 3 для набора данных WordCount

Задание. Распакуйте код примера из архива tfidf.tar.gz. Откройте проект в среде IntelliJ IDEA и обратите внимание на его настройки (рисунки Рисунок 30, Рисунок 31). Объясните наполнение списка внешних зависимостей, пользуясь документацией Hadoop. Запустить расчет можно выполнив команду ./run.sh в терминале IDE (Рисунок 32).

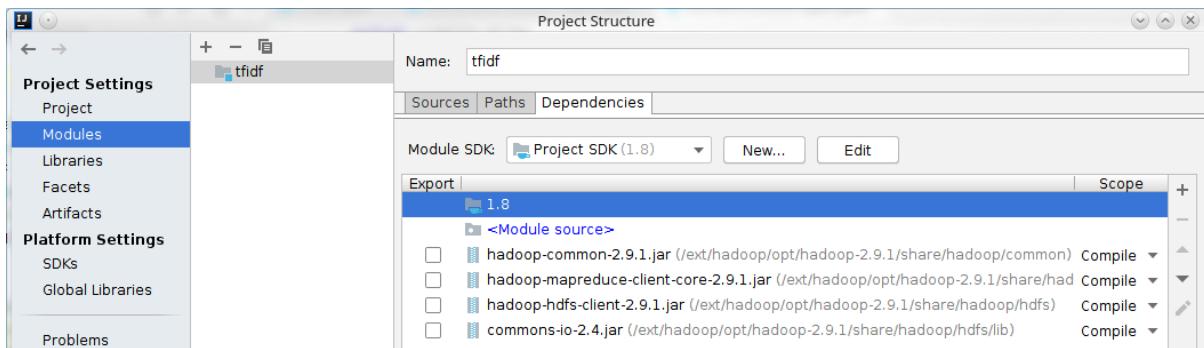


Рисунок 30 — Внешние зависимости проекта tfidf

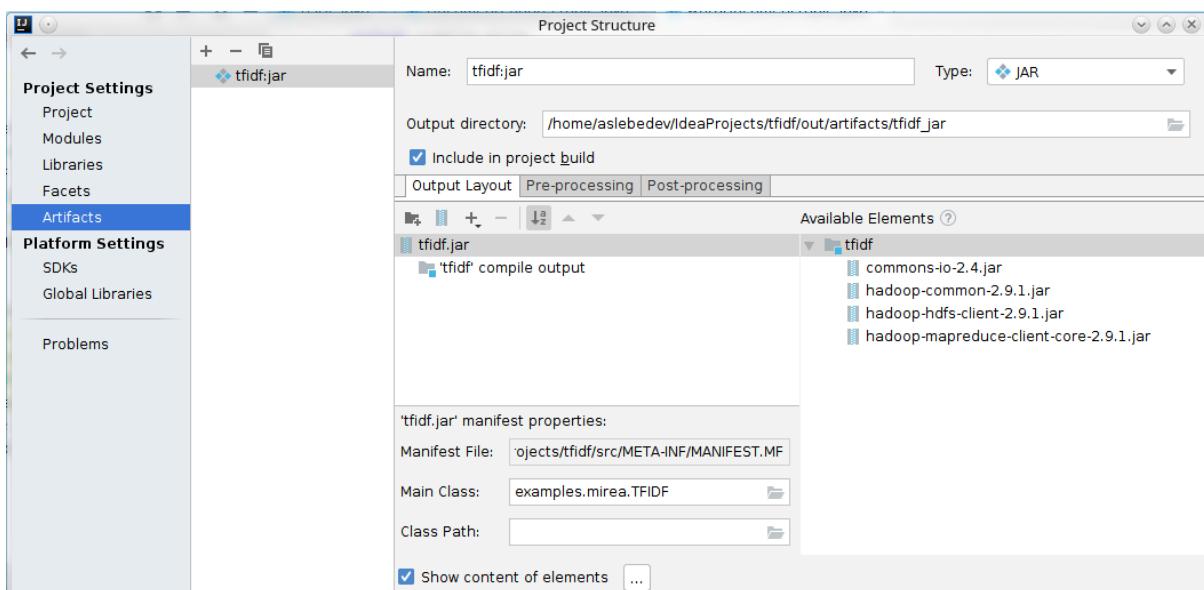


Рисунок 31 — Сборка артефакта tfidf.jar

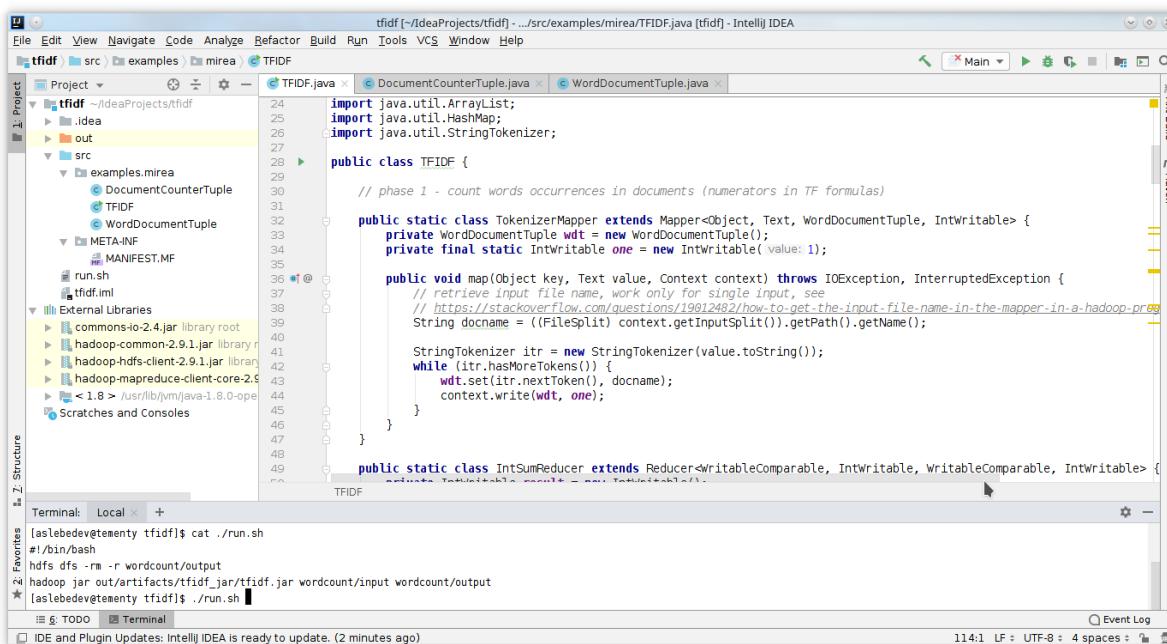


Рисунок 32 — Проект tfidf в среде IDEA и скрипт запуска run.sh

Технология Hadoop Streaming

Hadoop предоставляет API для MapReduce, позволяющий записывать функции отображения и свертки на других языках, кроме Java. Технология Hadoop Streaming использует стандартный поток Unix для организации взаимодействия Hadoop с программами, так что при написании программ MapReduce можно использовать любой язык, поддерживающий чтение из стандартного входного потока (`stdin`) и запись в стандартный выходной поток (`stdout`).

Streaming обычно подходит для обработки текста. Входные данные отображения проходят через стандартный ввод в функцию отображения, которая обрабатывает строку за строкой и записывает строки в стандартный вывод. Пара «ключ-значение» вывода отображения записывается как разделенная табуляцией строка. Ввод в функцию свертки того же формата — разделенная пара «ключ-значение» — проходит через стандартный ввод. Функция свертки читает строки из стандартного ввода (которые, как гарантирует инфраструктура, отсортированы по ключу) и записывает свои результаты в стандартный вывод.

Пример реализации байесовской фильтрации спама на Python

Описание модели классификации

Байесовская фильтрация спама — метод для фильтрации спама, основанный на применении наивного байесовского классификатора, опирающегося на прямое использование теоремы Байеса.

При обучении фильтра для каждого встреченного в письмах слова высчитывается и сохраняется его «вес» — оценка вероятности того, что письмо с этим словом — спам. При проверке вновь пришедшего письма вероятность

письма быть спамом вычисляется по формуле: $P(B) = \sum_{i=1}^N P(A_i)P(B|A_i)$. В данном

случае «гипотезы» — это слова, и для каждого слова «достоверность гипотезы» $P(A_i) = N_{word_i}/N_{words_total}$ — доля этого слова в письме, а «зависимость события от гипотезы» $P(B|A_i)$ — вычисленный ранее «вес» слова. То есть «вес» письма в данном случае — усредненный «вес» всех его слов. Отнесение письма к «спаму» или «не-спаму» производится по тому, превышает ли его «вес» некую планку, заданную пользователем.

Вычисление вероятности того, что сообщение, содержащее данное слово, является спамом

Предположим, что подозреваемое сообщение содержит слово «replica». Большинство людей, которые привыкли получать электронное письмо, знает,

что это сообщение, скорее всего, будет спамом, а точнее предложением продать поддельные копии часов известных брендов. Программа обнаружения спама, однако, не «знает» такие факты, все, что она может сделать — вычислить вероятности:

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)} = \frac{P(W|S)P(S)}{P(W|S)P(S) + P(W|H)P(H)}. \quad (2.4)$$

$P(S|W)$ — условная вероятность того, что сообщение — спам, при условии, что слово «replica» находится в нем; $P(S)$ — полная вероятность того, что произвольное сообщение — спам; $P(W|S)$ — условная вероятность того, что слово «replica» появляется в сообщениях, если они являются спамом; $P(H)$ — полная вероятность того, что произвольное сообщение не спам (то есть «ham»); $P(W|H)$ — условная вероятность того, что слово «replica» появляется в сообщениях, если они являются «ham».

Недавние статистические исследования показали, что на сегодняшний день вероятность любого сообщения быть спамом составляет по меньшей мере 80%: $P(S) = 0.8$, $P(H) = 0.2$. Однако, большинство байесовских программ обнаружения спама делают предположение об отсутствии априорных предпочтений у сообщения быть «spam», а не «ham», и полагает, что у обоих случаев есть равные вероятности 50%: $P(S) = P(H) = 0.5$. О фильтрах, которые используют эту гипотезу, говорят, как о фильтрах «без предубеждений». Это означает, что у них нет никакого предубеждения относительно входящей электронной почты. Это предположение позволяет упрощать общую формулу до:

$$P(S|W) = \frac{P(W|S)}{P(W|S) + P(W|H)}. \quad (2.5)$$

Значение $P(S|W)$ называют «спамовостью» слова W , при этом $P(W|S)$ приближенно равно относительной частоте сообщений, содержащих слово W , в сообщениях, идентифицированных как спам во время фазы обучения, то есть:

$$P(W_i|S) = \frac{\left| \{M \in S | W_i \in M\} \right|}{\sum_j \left| \{M \in S | W_j \in M\} \right|}. \quad (2.6)$$

Точно так же $P(W|H)$ приближенно равно относительной частоте сообщений, содержащих слово W , в сообщениях, идентифицированных как «ham» во время фазы обучения:

$$P(W_i | H) = \frac{|\{M \in H | W_i \in M\}|}{\sum_j |\{M \in H | W_j \in M\}|}. \quad (2.7)$$

Для того, чтобы эти приближения имели смысл, набор обучающих сообщений должен быть большим и достаточно представительным. Также желательно, чтобы набор обучающих сообщений соответствовал гипотезе о 50% перераспределении между спамом и «ham», то есть чтобы наборы сообщений «spam» и «ham» имели один и тот же размер.

Объединение индивидуальных вероятностей

Программные спам-фильтры, построенные на принципах наивного байесовского классификатора, делают «наивное» предположение о том, что события, соответствующие наличию того или иного слова в электронном письме или сообщении, являются независимыми по отношению друг к другу. Это упрощение в общем случае является неверным для естественных языков — таких, как английский, где вероятность обнаружения прилагательного повышается при наличии, к примеру, существительного. Исходя из такого «наивного» предположения, для решения задачи классификации сообщений лишь на 2 класса: S (спам) и H («хэм», то есть не спам) из теоремы Байеса можно вывести следующую формулу оценки вероятности «спамовости» всего сообщения, содержащего слова W_1, W_2, \dots, W_N :

$$P(S | W_1, W_2, \dots, W_N) = \frac{\prod_i P(S | W_i)}{\prod_i P(S | W_i) + \left(\frac{P(H)}{P(S)}\right)^{1-N} \prod_i P(H | W_i)}. \quad (2.8)$$

Для классификатора, работающего по схеме «без предубеждения», множитель $\left(\frac{P(H)}{P(S)}\right)^{1-N}$ тождественно равен 1. Результат, полученный по формуле (2.8), обычно сравнивают с некоторым порогом (например, 0.5), чтобы решить, является ли сообщение спамом или нет. Если значение ниже, чем порог, сообщение рассматривают как вероятный «ham», иначе его рассматривают как вероятный спам.

Описание набора данных

Набор данных электронных сообщений utf8_spamdb.csv представлен в формате CSV. Он содержит заголовок на первой строке (“class, text,,”), за которым следуют строки, содержащие метку класса сообщения (“spam” или “ham”), текст самого сообщения, и три пустых поля. Разделителем является запятая (Рисунок 33).

```

[aslebedev@tementy SpamBayes]$ head utf8_spamdb.csv | grep -E '^spam|^ham'
"Go until jurong point, crazy.. Available only in bugis n great world la e b
,Ok lar... Joking wif u oni....,
,Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA
t rate)T&C's apply 08452810075over18's...
,U dun say so early hor... U c already then say....,
,"Nah I don't think he goes to usf, he lives around here though",
,"FreeMsg Hey there darling it's been 3 week's now and no word back! I'd lik
std chgs to send, £1.50 to rcv",
,Even my brother is not like to speak with me. They treat me like aids patent
,As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has b
. Press *9 to copy your friends Callertune,
,WINNER!! As a valued network customer you have been selected to receivea £
1. Claim code KL341. Valid 12 hours only....,
[aslebedev@tementy SpamBayes]$

```

Рисунок 33 — Набор данных utf8_spamdb.csv

Подготовка данных

Узнать количество строк в файле можно открыв его в текстовом редакторе, или воспользовавшись утилитой wc:

```
$ cat utf8_spamdb.csv | wc -l
5575
```

Вы можете сформировать обучающую выборку с помощью утилиты head, выбрав указанное количество строк с начала файла, например, 4000:

```
$ head -n 4000 utf8_spamdb.csv > utf8_spamdb_train.csv
```

Из оставшихся строк можно сформировать тестовую выборку с помощью утилиты tail:

```
$ tail -n 1575 utf8_spamdb.csv > utf8_spamdb_test.csv
```

Тренировочная и тестовая выборки должны быть размещены в разных каталогах в HDFS. Скрипт copyInput.sh, организующий рабочий каталог в HDFS, представлен на рисунке (Рисунок 34).

```
#!/bin/bash

hdfs dfs -mkdir spamdb
hdfs dfs -mkdir spamdb/input
hdfs dfs -mkdir spamdb/input/train
hdfs dfs -mkdir spamdb/input/test

hdfs dfs -put utf8_spamdb_train.csv spamdb/input/train
hdfs dfs -put utf8_spamdb_test.csv spamdb/input/test
```

Рисунок 34 — Размещение данных в HDFS

Этап 1. Вычисление TF

Фаза Map. Модуль wordDocumentMap.py (Рисунок 35) получает на вход строки тренировочной выборки. Для каждой из них метка класса сохраняется в переменной messageClass, а текст сообщения в переменной messageText. Для каждого сообщения вычисляется его хеш messageId по алгоритму MD5, служащий в дальнейшем его уникальным идентификатором.

Каждое сообщение разбивается на слова. Интерес представляют только те из них, которые состоят из символов латиницы в количестве не менее 3 (регулярное выражение в переменной wordPattern). Для каждого слова определяется его основа с применением реализации PorterStemmer из библиотеки nltk.stem. Это значение, сохраняемое в переменной token, используется при формировании выходных пар ключ-значение:

(messageClass, messageText) → [((token, messageId), 1)].

Единица в выходной паре ключ-значение фиксирует факт того, что слово с основой token встретилось в сообщении с идентификатором messageId.

Для каждого сообщения в распределенное хранилище Redis записывается пара ключ-значение: (messageId, {"L": tokenCnt, "C": messageClass}). Значением выступает строковое представление ассоциативного массива, в котором поле L содержит количество значимых токенов tokenCnt в сообщении, а поле C содержит метку класса сообщения messageClass из набора данных.

```
#!/usr/bin/python3

import sys
import re
import hashlib
import redis
from nltk.stem import PorterStemmer

wordPattern = re.compile('^[a-zA-Z]{3,}$')
messageClasses = ["spam", "ham"]
r = redis.Redis(host='localhost', port=6379)

ps = PorterStemmer()

for line in sys.stdin:
    line = line.strip()
    borderIdx = line.index(",")
    messageClass = line[0 : borderIdx]
    messageText = line[borderIdx + 1 :].lower()
    if messageClass in messageClasses:
        messageBytes = messageText.encode('utf-8')
        messageId = hashlib.md5(messageBytes).hexdigest()
        tokenCnt = 0
        for token in messageText.split(" "):
            if wordPattern.match(token):
                token = ps.stem(token)
                tokenCnt += 1
                print("{}\t{}\t{}\n".format(token, messageId))
        r.hmset(messageId, {"L": tokenCnt, "C": messageClass})
```

Рисунок 35 — Исходный текст wordDocumentMap.py

Фаза Reduce. На этой фазе реализуется отображение:

$((\text{token}, \text{messageId}), [\text{integer value}]) \rightarrow ((\text{token}, \text{messageId}), \text{float TF value}).$

При этом выполняется вычисление числителей TF (количество вхождений слова в сообщение): для тех пар ключ-значение, у которых ключ совпадает, выполняется суммирование значений:

$((\text{token}, \text{messageId}), [\text{integer value}]) \rightarrow ((\text{token}, \text{messageId}), \text{sum}).$

Результирующее значение TF получается делением числителя sum на знаменатель, определяемый длиной сообщения documentLen. Она извлекается из распределенного хранилища Redis по ключу-идентификатору document, который вычисленается из составного ключа, пришедшего с фазы Map (Combine).

Исходный текст модуля tfReduce.py представлен на рисунке (Рисунок 36).

```
#!/usr/bin/python3
import sys
import redis

r = redis.Redis(host='localhost', port=6379)

(lastKey, sum)=(None, 0)

▼ for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    ▼ if lastKey and lastKey != key:
        (word, document) = lastKey.split(",")
        documentLen = int(r.hget(document, "L").decode("utf-8"))
        print("{0}\t{1:f}".format(lastKey, sum / documentLen))
        (lastKey, sum) = (key, int(value))
    ▼ else:
        (lastKey, sum) = (key, sum + int(value))

▼ if lastKey:
    (word, document) = lastKey.split(",")
    documentLen = int(r.hget(document, "L").decode("utf-8"))
    print("{0}\t{1:f}".format(lastKey, sum / documentLen))
```

Рисунок 36 — Исходный текст tfReduce.py

Фаза Combine. На этой фазе полезно сократить объем данных, поступающих на фазу Reduce, посчитав частичные суммы для числителей TF (количество вхождений слова в сообщение). Итоговые суммы будут вычислены в фазе Reduce. Редукцию частичных результатов, полученных от экземпляров map, можно выполнить универсальным алгоритмом (Рисунок 37), складывающим целочисленные значения для тех пар ключ-значение, у которых ключ совпадает:

$((\text{token}, \text{messageId}), [1]) \rightarrow ((\text{token}, \text{messageId}), \text{sum}).$

```

#!/usr/bin/python3
import sys

(lastKey, sum)=(None, 0)

for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    if lastKey and lastKey != key:
        print("{0}\t{1:d}".format(lastKey, sum))
        (lastKey, sum) = (key, int(value))
    else:
        (lastKey, sum) = (key, sum + int(value))

if lastKey:
    print("{0}\t{1:d}".format(lastKey, sum))

```

Рисунок 37 — Исходный текст intSumCombine.py

Запуск вычислений. Перед запуском задания MapReduce необходимо очистить используемую БД Redis:

```
$ redis-cli -h 'localhost' -p 6379 flushall
```

Если выходной каталог уже существует, его необходимо удалить:

```
$ hdfs dfs -rm -r spamdb/tf
```

Запуск задания с помощью yarn:

```
$ yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="TF job via streaming" \
-files `pwd`/wordDocumentMap.py,`pwd`/tfReduce.py,`pwd`/intSumCombine.py \
-input spamdb/input/train \
-output spamdb/tf/ \
-mapper `pwd`/wordDocumentMap.py \
-combiner `pwd`/intSumCombine.py \
-reducer `pwd`/tfReduce.py
```

Этап 2. Вычисление $P(W_i|S)$ и $P(W_i|H)$ (только числители)

Разделим набор данных для обучения на две коллекции документов: S — множество спам-сообщений, H — множество легальных сообщений.

Рассмотрим формулу (2.2) вычисления характеристики IDF для слова t и коллекции документов D . Заметим, что при подстановке аргументов $IDF(W_i, S)$ или $IDF(W_i, H)$ знаменатель в формуле (2.2) становится идентичным числителю в формуле (2.6) или (2.7), и имеет смысл количества спам- или легальных сообщений, в которых встретилось заданное слово W_i , соответственно. Вычислить эти значения можно с помощью следующего задания MapReduce.

Фаза Map. Модуль wordMap.py (Рисунок 38) получает строки, полученные в результате работы задания MapReduce на этапе 1, и выполняет реконструирование пар ключ-значение согласно следующему отображению:

$$((\text{token}, \text{messageId}), \text{TF}) \rightarrow (\text{token}, (\text{messageId}, \text{TF})).$$

```

#!/usr/bin/python3

import sys

▼ for line in sys.stdin:
    (key, tf) = line.strip().split("\t")
    (word, document) = key.split(",")
    print("{}\t{},{}".format(word, document, tf))

```

Рисунок 38 — Исходный текст wordMap.py

Фаза Reduce. Модуль idfReduce.py получает кортежи вида (token, (messageId, TF)) в строковом представлении. В фазе Shuffle&Sort они были объединены в группы по признаку совпадающего ключа. Все кортежи для заданного ключа token гарантированно будут получены только одним экземпляром reduce. На этой фазе реализуется отображение:

(token, [(messageId, TF)]) → (token, {"spam": integer value, "ham": integer value}).

Выходным значением является ассоциативный массив, в поле “spam” которого хранится количество вхождений слова token в спам-сообщения, а в поле “ham” — количество вхождений слова token в легальные сообщения.

Во время потоковой обработки входных кортежей (Рисунок 39) класс сообщения извлекается из распределенного хранилища Redis по его идентификатору messageId. Соответственное поле ассоциативного массива documentsCnt, аккумулирующего выходное значение, инкрементируется.

```

#!/usr/bin/python3
import sys
import redis

r = redis.Redis(host='localhost', port=6379)

(lastKey, documentsCnt)=(None, {"spam" : 0, "ham" : 0})

▼ for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    (document, tf) = value.split(",")
    ▼ if lastKey and lastKey != key:
        print("{}\t{}".format(lastKey, str(documentsCnt)))
        for k in documentsCnt.keys():
            documentsCnt[k] = 0
        documentsCnt[r.hget(document, "C").decode("utf-8")] += 1
        lastKey = key
    ▼ if lastKey:
        print("{}\t{}".format(lastKey, str(documentsCnt)))

```

Рисунок 39 — Исходный текст idfReduce.py

Запуск вычислений. Если выходной каталог уже существует, его необходимо удалить:

```
$ hdfs dfs -rm -r spamdb/idf_denominators
```

Запуск задания с помощью yarn:

```
$ yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="IDF denominators job via streaming" \
-files `pwd`/wordMap.py,`pwd`/idfReduce.py \
-input spamdb/tf/ \
-output spamdb/idf_denominators/ \
-mapper `pwd`/wordMap.py \
-reducer `pwd`/idfReduce.py
```

Этап 3. Вычисление $P(W_i|S)$ и $P(W_i|H)$ (только знаменатели)

Знаменатель в каждой из формул (2.6) и (2.7) вычисляется простым суммированием всех возможных значений соответствующего числителя.

Достаточно выполнить поэлементное суммирование всех ассоциативных массивов, полученных в результате работы задания MapReduce на этапе 2. Для этого можно использовать одну программу (Рисунок 40) на всех фазах: Map, Combine, Reduce.

```
#!/usr/bin/python3
import sys

documentsCnt = {"spam" : 0, "ham" : 0}

for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    valueDict = eval(value)
    for k in documentsCnt.keys():
        documentsCnt[k] += valueDict[k]

print("_total_\t" + str(documentsCnt))
```

Рисунок 40 — Исходный текст dictSum.py

Модуль dictSum.py суммирует поэлементно ассоциативные массивы из всех пар ключ-значение, поступающих на вход. В качестве выходного ключа применяется служебная строка “_total_”, заведомо не подходящая под шаблон рассматриваемых слов.

В фазе map реализуется отображение:

$$[(\text{token}, \{\text{"spam": integer value, "ham": integer value}\})] \rightarrow (\text{"_total_"}, \{\text{"spam": integer value, "ham": integer value}\}).$$

В фазе reduce реализуется отображение:

$$(\text{token}, [\{\text{"spam": integer value, "ham": integer value}\}]) \rightarrow (\text{"_total_"}, \{\text{"spam": integer value, "ham": integer value}\}).$$

В результате работы задания получается одна пара ключ-значение вида

$$(\text{"_total_"}, \{\text{"spam": integer value, "ham": integer value}\}).$$

Запуск вычислений. Если выходной каталог уже существует, его необходимо удалить:

```
$ hdfs dfs -rm -r spamdb/p_wi_denominator
```

Запуск задания с помощью yarn:

```
$ yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="P(Wi|S/H) denominator job via streaming" \
-files `pwd`/dictSum.py \
-input spamdb/idf_denominators/ \
-output spamdb/p_wi_denominator/ \
-mapper `pwd`/dictSum.py \
-combiner `pwd`/dictSum.py \
-reducer `pwd`/dictSum.py
```

Сохранение результата в хранилище Redis. Поскольку в результате получается только одна пара ключ-значение, и количество экземпляров reduce равно 1, извлечь результат работы задания MapReduce в переменную можно следующей командой:

```
$ mr_text_result=`hdfs dfs -cat spamdb/p_wi_denominator/part-00000`
```

В предположении, что ключ от значения отделяется символом табуляции, можно извлечь их в различные переменные следующим образом:

```
$ mr_key=`echo "$mr_text_result" | awk -F '\t' '{print $1}'` \
$ mr_val=`echo "$mr_text_result" | awk -F '\t' '{print $2}'`
```

Добавление пары ключ-значение в хранилище Redis выполняется следующим образом:

```
$ redis-cli -h 'localhost' -p 6379 set "$mr_key" "$mr_val"
```

Ключом будет выступать строка “_total_”, а в качестве значения будет фигурировать строковое представление ассоциативного массива Python с ключами “spam” и “ham”.

Этап 4. Вычисление вероятностей $P(S|W_i)$ и $P(H|W_i)$

Числители формул (2.6) и (2.7) были вычислены на этапе 2, знаменатели — на этапе 3. Вычислить вероятности $P(W_i|S)$ и $P(W_i|H)$, а затем $P(S|W_i)$ и $P(H|W_i)$ можно с помощью задания MapReduce, содержащего только фазу Map (Рисунок 41).

```
#!/usr/bin/python3
import sys
import redis

r = redis.Redis(host='localhost', port=6379)
documentsCntTotal = eval(r.get("_total_").decode("utf-8"))

▼ for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    documentsCnt = eval(value)
    # calculate P(Wi|S) and P(Wi|H)
    pWi = 0
    ▼ for k in documentsCnt.keys():
        documentsCnt[k] /= documentsCntTotal[k]
        pWi += documentsCnt[k]
    # calculate P(S|Wi) and P(H|Wi)
    ▼ for k in documentsCnt.keys():
        documentsCnt[k] /= pWi
    #print("{}\t{}".format(key, str(documentsCnt)))
    r.hmset(key, documentsCnt)
```

Рисунок 41 — Исходный текст pWiMap.py

В фазе Map реализуется отображение:

$$(token, \{“spam”: integer value, “ham”: integer value\}) \rightarrow (token, \{“spam”: P(S|token), “ham”: P(H|token)\}).$$

Модуль pWiMap.py обрабатывает результат работы этапа 2. Ассоциативный массив с числителями формул (2.6) и (2.7) для каждого слова сохраняется в переменной documentsCnt. Результат работы этапа 3 извлекается из распределенного хранилища Redis в ассоциативный массив documentsCntTotal. Деление элемента documentsCnt на соответственный элемент documentsCntTotal дает $P(key|S)$ для ключа “spam” и $P(key|H)$ для ключа “ham”. В переменной pWi накапливается сумма $P(key|S) + P(key|H)$.

Далее $P(S|key)$ вычисляется как $P(key|S)/pWi$ и $P(H|key)$ вычисляется как $P(key|H)/pWi$ в согласии с формулой для классификатора без предубеждения (2.5). Результат вычислений сохраняется в распределенном хранилище Redis для быстрого доступа к значениям вероятностей в оперативной памяти. В контекст MapReduce данные не записываются.

Запуск вычислений. Если выходной каталог уже существует, его необходимо удалить:

```
$ hdfs dfs -rm -r spamdb/p_i
```

Запуск задания с помощью yarn:

```
$ yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="P(S/H|Wi) job via streaming" \
-files `pwd`/pWiMap.py \
-input spamdb/idf_denominators/ \
-output spamdb/p_i/ \
-mapper `pwd`/pWiMap.py
```

Этап 5. Классификация сообщений тестовой выборки

Фаза Map. Модуль classifyMap.py (Рисунок 42) получает на вход строки тестовой выборки. Для каждой из них метка класса сохраняется в переменной messageClass, а текст сообщения в переменной messageText. Каждое сообщение разбивается на слова. Интерес представляют только те из них, которые состоят из символов латиницы в количестве не менее 3 (регулярное выражение в переменной wordPattern). Для каждого слова определяется его основа с применением реализации PorterStemmer из библиотеки nltk.stem. Это значение, сохраняемое в переменной token, используется при запросе вероятностей $P(S|token)$ и $P(H|token)$ из распределенного хранилища Redis. Если такая основа в обучающей выборке не встречалась, и по запросу ничего не найдено, основа игнорируется. В противном случае в переменных ms и mh накапливаются произведения $P(S|token)$ и $P(H|token)$ для различных значений token, соответственно.

Если $ms > mh$, сообщение определяется классификатором как спам, в противном случае — как легальное сообщение (в согласии с формулой 2.5 для схемы классификатора без предубеждения). Ассоциативный массив stats с ключами “spam” и “ham”, определяющий выходное значение, будет иметь 1 в поле, соответствующем результату классификации, и 0 — в другом поле.

Фаза Map реализует отображение:

$(\text{messageClass}, \text{messageText}) \rightarrow (\text{messageClass}, \{\text{"spam": 1 or 0, "ham": 1 or 0}\})$.

Ключ messageClass будет в дальнейшем интерпретироваться как экспертная оценка сообщения при вычислении характеристик классификатора в фазах Combine и Reduce.

```
#!/usr/bin/python3

import sys
import re
import redis
from nltk.stem import PorterStemmer

wordPattern = re.compile('^[a-zA-Z]{3,}$')
messageClasses = ["spam", "ham"]
r = redis.Redis(host='localhost', port=6379)

ps = PorterStemmer()

▼ for line in sys.stdin:
    line = line.strip()
    borderIdx = line.index(",")
    messageClass = line[0 : borderIdx]
    messageText = line[borderIdx + 1 :].lower()
    if messageClass in messageClasses:
        messageBytes = messageText.encode('utf-8')
        ms = 1.0
        mh = 1.0
        tokenCnt = 0
        for token in messageText.split(" "):
            if wordPattern.match(token):
                token = ps.stem(token)
                p_dict = r.hgetall(token)
                if p_dict:
                    tokenCnt += 1
                    ms *= float(p_dict[b"spam"].decode("utf-8"))
                    mh *= float(p_dict[b"ham"].decode("utf-8"))
        stats = dict.fromkeys(messageClasses, 0)
        stats["spam" if ms > mh else "ham"] = 1
        print("{}\t{}".format(messageClass, str(stats)))
```

Рисунок 42 — Исходный текст classifyMap.py

Фаза Reduce. На этой фазе сначала формируется таблица сопряженности (контингентности):

$(\text{messageClass}, [\{\text{"spam": integer value, "ham": integer value}\}]) \rightarrow [(\text{"spam"}, [\{\text{"spam": TP, "ham": FN}\}]), (\text{"ham"}, [\{\text{"spam": FP, "ham": TN}\}])]$.

Это достигается простым поэлементным суммированием ассоциативных массивов тех входящих пар ключ-значение, у которых ключ (интерпретируемый как экспертная оценка сообщения) совпадает (Рисунок 43).

```
#!/usr/bin/python3
import sys

(lastKey, partialStats)=(None, {"spam" : 0, "ham" : 0})

def calcRatio(d):
    s = sum(d.values())
    return {k: v / s for k, v in d.items()}

for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    valueDict = eval(value)
    if lastKey and lastKey != key:
        print("{}\t{}".format(lastKey, str(calcRatio(partialStats))))
        partialStats = valueDict
    else:
        for k in partialStats.keys():
            partialStats[k] += valueDict[k]
    lastKey = key

if lastKey:
    print("{}\t{}".format(lastKey, str(calcRatio(partialStats))))
```

Рисунок 43 — Исходный текст classifyReduce.py

Перед выводом в контекст значения нормируются в функции calcRatio:

$$(\text{"spam"}, [\{\text{"spam": TP, "ham": FN}\}]) \rightarrow \\ (\text{"spam"}, [\{\text{"spam": TP/(TP+FN), "ham": FN/(TP+FN)}\}]).$$

Истинно положительная пропорция $\text{TP}/(\text{TP+FN})$ называется чувствительностью и отражает долю спам-сообщений, которые правильно идентифицированы как таковые.

$$(\text{"ham"}, [\{\text{"spam": FP, "ham": TN}\}]) \rightarrow \\ (\text{"ham"}, [\{\text{"spam": FP/(FP+TN), "ham": TN/(FP+TN)}\}]).$$

Истинно отрицательная пропорция $\text{TN}/(\text{FP+TN})$ называется специфичностью и отражает долю легальных сообщений, которые правильно идентифицированы как таковые.

Фаза Combine. На этой фазе полезно сократить объем данных, поступающих на фазу Reduce, посчитав частичные суммы для характеристик TP, FN, FP, TN. Итоговые суммы будут вычислены в фазе Reduce. Редукцию частичных результатов, полученных от экземпляров map, можно выполнить универсальным алгоритмом (Рисунок 44), складывающим поэлементно ассоциативные массивы для тех пар ключ-значение, у которых ключ совпадает:

$$(\text{messageClass}, [\{\text{"spam": 1 or 0, "ham": 1 or 0}\}]) \rightarrow \\ (\text{messageClass}, [\{\text{"spam": integer value, "ham": integer value}\}]).$$

```

#!/usr/bin/python3
import sys

(lastKey, partialStats)=(None, {"spam" : 0, "ham" : 0})

▼ for line in sys.stdin:
    (key, value) = line.strip().split("\t")
    valueDict = eval(value)
    ▼ if lastKey and lastKey != key:
        print("{}\t{}".format(lastKey, str(partialStats)))
        partialStats = valueDict
    else:
        ▼ for k in partialStats.keys():
            partialStats[k] += valueDict[k]
        lastKey = key

▼ if lastKey:
    print("{}\t{}".format(lastKey, str(partialStats)))

```

Рисунок 44 — Исходный текст classifyCombine.py

Запуск вычислений. Если выходной каталог уже существует, его необходимо удалить:

```
$ hdfs dfs -rm -r spamdb/stats
```

Запуск задания с помощью yarn:

```
$ yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-D mapreduce.job.name="Classify job via streaming" \
-files `pwd`/classifyMap.py,`pwd`/classifyCombine.py,`pwd`/classifyReduce.py \
-input spamdb/input/test/ \
-output spamdb/stats/ \
-mapper `pwd`/classifyMap.py \
-combiner `pwd`/classifyCombine.py \
-reducer `pwd`/classifyReduce.py
```

Вывод результата на терминал:

```
$ hdfs dfs -cat spamdb/stats/part-00000
```

Задание. Доработайте классификатор, чтобы использовалась схема с предубеждением. Проанализируйте поток данных. Возможно ли уменьшить количество заданий MapReduce для обучения классификатора? Если возможно, доработайте реализацию.

Задание. Алгоритм кросс-корреляции

Необходимо решить задачу формирования списка рекомендованных товаров для пользователя интернет-магазина с применением алгоритма кросс-корреляции (имя множество кортежей объектов, для каждой возможной пары объектов посчитать число кортежей, где они встречаются вместе).

1. Реализовать два алгоритма на MapReduce:
 - Cross Correlation Pairs (Рисунок 45).

```

class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            for all item j in [i1, i2,...]
                Emit(pair [i j], count 1)

class Reducer
    method Reduce(pair [i j], counts [c1, c2,...])
        s = sum([c1, c2,...])
        Emit(pair[i j], count s)

```

Рисунок 45 — Псевдокод алгоритма Cross Correlation Pairs

- Cross Correlation Stripes (Рисунок 46).

```

class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            H = new AssociativeArray : item -> counter
            for all item j in [i1, i2,...]
                H{j} = H{j} + 1
            Emit(item i, stripe H)

class Reducer
    method Reduce(item i, stripes [H1, H2,...])
        H = new AssociativeArray : item -> counter
        H = merge-sum( [H1, H2,...] )
        for all item j in H.keys()
            Emit(pair [i j], H{j})

```

Рисунок 46 — Псевдокод алгоритма Cross Correlation Stripes

2. Написать генератор базы данных интернет-заказов (или взять готовую). Учесть, что заказ состоит из произвольного количества позиций (товаров).
3. Обработать алгоритмом кросс-корреляции базу данных заказов (подсчитать количество вхождений каждой пары товаров).
4. Реализовать компонент советника, не применяя паттерн MapReduce. Вводится название товара. Выводятся 10 товаров, которые чаще всего покупают с заданным товаром. Чтение результатов работы алгоритма кросс-корреляции из HDFS реализовать вручную (для Java — с помощью FileSystem API, для Python — с помощью библиотеки pyhdfs).

3. Обработка реляционных данных в экосистеме Hadoop

Технология Hive

Технология Hive выросла из потребности в управлении и извлечении информации из огромных объемов данных, ежедневно производимых Facebook в стремительно растущей социальной сети. Опробовав несколько разных систем, группа разработки выбрала Hadoop для хранения и обработки информации, так как эта технология была экономичной и удовлетворяла их потребности в масштабировании.

Система Hive создавалась для того, чтобы аналитики, хорошо владеющие SQL (но слабо разбирающиеся в программировании на Java), могли выполнять запросы к гигантским объемам данных, хранимых Facebook в HDFS. Сегодня Hive — успешный проект Apache, используемый во многих организациях как универсальная и масштабируемая платформа обработки данных.

Взаимодействие с Hive в основном происходит через программную оболочку, в которой вводятся команды на HiveQL — языке запросов Hive, диалекте SQL. Хотя Hive во многих отношениях напоминает традиционные базы данных, из тесной связи этой технологии с HDFS и MapReduce вытекает целый ряд архитектурных различий, которые напрямую влияют на функциональность Hive — которая, в свою очередь, влияет на возможное применение Hive.

В традиционных базах данных схема таблицы проверяется во время загрузки данных. Если загружаемые данные не соответствуют схеме, они отвергаются. Такой метод называется проверкой схемы при записи. С другой стороны, Hive проверяет данные не при загрузке, а при выдаче запроса. Это называется проверкой схемы при чтении.

У каждого из двух методов есть свои достоинства и недостатки. Проверка схемы при чтении заметно ускоряет начальную загрузку, поскольку данные не нужно читать, разбирать и сериализовать на диск во внутреннем формате базы данных. Операция загрузки сводится к копированию или перемещению файла. Кроме того, улучшается гибкость обработки данных: возможно использовать две схемы для одного набора данных в зависимости от выполняемого анализа.

Проверка схемы при записи ускоряет выполнение запросов, потому что база данных может проиндексировать столбцы и выполнить сжатие данных. Вместе с тем процедура загрузки данных в базу выполняется медленнее. Кроме того, схема часто неизвестна на стадии загрузки. В таком случае индексирование невозможно, потому что запросы еще не были сформулированы. В таких ситуациях достоинства Hive проявляются особенно ярко.

Установка и настройка Hive на Hadoop

Перейдите в домашний каталог, если Вы не там:

```
$ cd ~
```

1. Скачать Hive и распаковать архив.

```
$ wget http://mirror.linux-ia64.org/apache/hive/hive-2.3.6/apache-hive-2.3.6-bin.tar.gz  
$ tar -xvf apache-hive-2.3.6-bin.tar.gz
```

2. Конфигурация Hive.

Перейдите в каталог с конфигурационными файлами и создайте там конфигурацию для своего пользователя:

```
$ cd apache-hive-2.3.6-bin/conf  
$ cat <<EOT >> hive-site.xml  
<configuration>  
  <property>  
    <name>hive.metastore.warehouse.dir</name>  
    <value>/user/`whoami`/warehouse</value>  
    <description>location of default database for the  
warehouse</description>  
  </property>  
</configuration>  
EOT
```

Это многострочная команда, вставляйте ее в терминал целиком.

3. Конфигурация Hadoop.

Настройте и перезапустите Hadoop (если он запущен и работает).

Добавьте два свойства в файл конфигурации \$HADOOP_HOME/etc/hadoop/core-site.xml:

```
<property>  
  <name>hadoop.proxyuser.ИМЯ_ПОЛЬЗОВАТЕЛЯ.groups</name>  
  <value>*</value>  
</property>  
<property>  
  <name>hadoop.proxyuser.ИМЯ_ПОЛЬЗОВАТЕЛЯ.hosts</name>  
  <value>*</value>  
</property>
```

В качестве значения ИМЯ_ПОЛЬЗОВАТЕЛЯ подставьте вывод whoami или id -nu.

Перезапустите Hadoop, если он запущен:

```
$ hdpStop.sh  
$ hdpStart.sh
```

Запустите Hadoop, если он не был запущен:

```
$ hdpFormat.sh # если файловая система создается впервые, или Вы просто желаете  
ее отформатировать  
$ hdpStart.sh
```

Задание. Ознакомьтесь с документацией Hadoop и Hive и объясните назначение каждого конфигурационного параметра.

4. Настройте переменные окружения.

Убедитесь, что файл `~/.bash_profile` (или `~/.bashrc`) имеет следующий текст в конце:

```
# User specific environment and startup programs

JAVA_HOME=$HOME/jdk1.8.0_221
HADOOP_HOME=$HOME/hadoop-2.9.2
HIVE_HOME=$HOME/apache-hive-2.3.6-bin
PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin

export JAVA_HOME
export HADOOP_HOME
export HIVE_HOME
export PATH
```

Внесите изменения в имеющуюся конфигурацию. Не забудьте дописать новый путь в `PATH`.

Перезапустите сеанс пользователя. Перейдите в домашний каталог, если Вы не там.

5. Установите вспомогательные скрипты. Они должны лежать в `~/bin`.

- `hiveMeta.sh` — подготовка хранилища метаданных для Hive. Используется встроенная СУБД derby.
- `hiveStart.sh` — запуск Hive Server как демона.

6. Подготовьте хранилище метаданных для Hive.

```
$ hiveMeta.sh
```

7. Запустите Hive и подключитесь к терминалу screen.

```
$ hiveStart.sh
$ screen -r hiveserver2
```

Отобразится журнал (Рисунок 47). Можно оставить этот терминал и наблюдать за событиями.

Если нажать `Ctrl+C`, Hive Server остановится, и терминал `screen` закроется.

Если нажать `Ctrl+A`, а затем `D`, то вы отключитесь от терминала `screen`, и Hive Server будет продолжать работать как демон.

8. Подключитесь к Hive Server и проверьте работоспособность Hive.

Подключитесь к серверу Hive следующей командой (Рисунок 48):

```
$ beeline -u jdbc:hive2://localhost:10000 -n `whoami`
```

Поэкспериментируйте с таблицей:

```
create table testtable(id int, name string);
insert into testtable(id, name) values(1,"usr");
select * from testtable;
show tables;
```

Выход по `Ctrl+D`.

Посмотрите на файловую систему:

```

$ hdfs dfs -ls warehouse
$ hdfs dfs -cat warehouse/testtable/000000_0
user@user-VirtualBox: ~

Файл Правка Вид Поиск Терминал Справка
2019-10-09T13:44:36,059  INFO [main] HiveMetaStore.audit: ugi=user      ip=unkno
wn-ip-addr      cmd=get_multi_table : db=default tbls=
2019-10-09T13:44:36,271  INFO [main] server.Server: jetty-7.6.0.v20120127
2019-10-09T13:44:36,361  INFO [main] webapp.WebInfConfiguration: Extract jar:fil
e:/home/user/apache-hive-2.3.6-bin/lib/hive-service-2.3.6.jar!/hive-webapps/hive
server2/ to /tmp/jetty-0.0.0.0-10002-hiveserver2--any-/webapp
2019-10-09T13:44:36,379  INFO [Thread-10] thrift.ThriftCLIService: Starting Thri
ftBinaryCLIService on port 10000 with 5...500 worker threads
2019-10-09T13:44:36,565  INFO [main] handler.ContextHandler: started o.e.j.w.Web
AppContext{/ ,file:/tmp/jetty-0.0.0.0-10002-hiveserver2--any-/webapp/},jar:file:
/home/user/apache-hive-2.3.6-bin/lib/hive-service-2.3.6.jar!/hive-webapps/hives
erver2
2019-10-09T13:44:36,630  INFO [main] handler.ContextHandler: started o.e.j.s.Ser
vletContextHandler{/static,jar:file:/home/user/apache-hive-2.3.6-bin/lib/hive-se
rvice-2.3.6.jar!/hive-webapps/static}
2019-10-09T13:44:36,653  INFO [main] server.AbstractConnector: Started SelectCha
nnelConnector@0.0.0.0:10002
2019-10-09T13:44:36,658  INFO [main] http.HttpServer: Started HttpServer[hiveser
ver2] on port 10002

```

Рисунок 47 — Журнал HiveServer2

```

user@user-VirtualBox: ~

Файл Правка Вид Поиск Терминал Справка
user@user-VirtualBox:~$ beeline -u jdbc:hive2://localhost:10000 -n `whoami`
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/user/apache-hive-2.3.6-bin/lib/log4j-slf
4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/user/hadoop-2.9.2/share/hadoop/common/li
b/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 2.3.6)
Driver: Hive JDBC (version 2.3.6)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.6 by Apache Hive
0: jdbc:hive2://localhost:10000> 

```

Рисунок 48 — Оболочки beeline

Примеры работы с Hive

Пример разбиения таблицы на разделы (partitions)

Hive создает в таблицах разделы (partitions); этот механизм позволяет разделить таблицу на блоки по значению некоторого столбца (например, даты). Разделы ускоряют выполнение запросов.

Представьте себе журнальный файл, в котором каждая запись содержит временную метку. Если разделить данные по дате, то записи с одинаковой датой

будут находиться в одном разделе. Преимущество такой схемы состоит в том, что запросы, относящиеся к определенной дате или набору дат, будут обрабатываться значительно эффективнее — поиск будет осуществляться только в файлах разделов, к которым относится запрос. При этом разделы не препятствуют выполнению более общих запросов ко всему набору данных по многим разделам.

Разделы могут определяться для таблицы по нескольким «измерениям». Например, в каждом разделе по дате также могут определяться дополнительные подразделы по странам, повышающие эффективность запросов.

Разделы определяются в момент создания таблицы при помощи условия PARTITIONED BY со списком определений столбцов. В примере с журнальными файлами можно определить таблицу с записями, состоящими из временной метки и строки сообщения.

Создайте таблицу logs, имеющую следующие информационные поля: ts BIGINT (целочисленный идентификатор), line STRING (произвольное строковое содержимое), dt STRING (строковая запись даты), country STRING (страна). Для полей dt и country должно выполняться разбиение на разделы (partitioning). Конструкция для создания таблицы проиллюстрирована на рисунке (Рисунок 49).

```
0: jdbc:hive2://localhost:10000> CREATE TABLE logs (ts BIGINT, line STRING) PARTITIONED BY (dt STRING, country STRING);
No rows affected (2,414 seconds)
0: jdbc:hive2://localhost:10000>
```

Рисунок 49 — Создание таблицы logs с разделами (partitions)

На рисунке (Рисунок 50) показано, как можно добавить запись в раздел с помощью конструкции INSERT INTO <table_name> PARTITION (<partition_field1_name> = <value>, <partition_field2_name> = <value>, ...) VALUES (...)

```
0: jdbc:hive2://localhost:10000> INSERT INTO logs PARTITION (dt = "2019-10-08",
country = "Russia") values (5, "Moscow");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (28,75 seconds)
0: jdbc:hive2://localhost:10000>
```

Рисунок 50 — Добавление записи в раздел

Добавьте еще несколько записей (Рисунок 51).

```
user@user-VirtualBox: ~
Файл Правка Вид Поиск Терминал Справка
country = "Russia") values (5, "Moscow");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (28,75 seconds)
0: jdbc:hive2://localhost:10000> INSERT INTO logs PARTITION (dt = "2014-02-01",
country = "Russia") values (6, "Sochi");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (20,736 seconds)
0: jdbc:hive2://localhost:10000> INSERT INTO logs PARTITION (dt = "2019-10-08",
country = "Ukraine") values (8, "Kiev");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (19,987 seconds)
0: jdbc:hive2://localhost:10000> INSERT INTO logs PARTITION (dt = "2019-10-08",
country = "Russia") values (9, "Krasnodar");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (18,679 seconds)
0: jdbc:hive2://localhost:10000> █
```

Рисунок 51 — Заполнение таблицы logs

Выборка из таблицы logs должна выглядеть так, как представлено на рисунке (Рисунок 52).

Задание. Запрос для вставки записи с полем country, содержащим строку USSR, сконструируйте самостоятельно.

```
0: jdbc:hive2://localhost:10000> select * from logs;
+-----+-----+-----+-----+
| logs.ts | logs.line | logs.dt   | logs.country |
+-----+-----+-----+-----+
| 10      | Leningrad | 1969-11-07 | USSR          |
| 6       | Sochi     | 2014-02-01  | Russia        |
| 5       | Moscow    | 2019-10-08  | Russia        |
| 9       | Krasnodar | 2019-10-08  | Russia        |
| 8       | Kiev      | 2019-10-08  | Ukraine       |
+-----+-----+-----+-----+
5 rows selected (0,723 seconds)
0: jdbc:hive2://localhost:10000> █
```

Рисунок 52 — Содержимое таблицы logs

Откройте в браузере просмотр файловой системы HDFS. Изучите структуру разделов таблицы logs (рисунки Рисунок 53 — Рисунок 60).

Browse Directory

/user/user/warehouse/logs									Go!			
Show 25 entries		Search:										
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	drwxrwxr-x	user	supergroup	0 B	Oct 09 14:26	0	0 B	dt=1969-11-07				
<input type="checkbox"/>	drwxrwxr-x	user	supergroup	0 B	Oct 09 14:21	0	0 B	dt=2014-02-01				
<input type="checkbox"/>	drwxrwxr-x	user	supergroup	0 B	Oct 09 14:23	0	0 B	dt=2019-10-08				

Showing 1 to 3 of 3 entries

Previous 1 Next

Рисунок 53 — Разделы logs по полю dt

Browse Directory

/user/user/warehouse/logs/dt=1969-11-07									Go!			
Show 25 entries		Search:										
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	0 B	Oct 09 14:27	0	0 B	country=USSR				

Showing 1 to 1 of 1 entries

Previous 1 Next

Рисунок 54 — Разделы logs по полю country внутри раздела dt=1969-11-07

Browse Directory

/user/user/warehouse/logs/dt=1969-11-07/country=USSR									Go!			
Show 25 entries		Search:										
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	13 B	Oct 09 14:27	1	128 MB	000000_0				

Showing 1 to 1 of 1 entries

Previous 1 Next

Рисунок 55 — Структура раздела dt=1969-11-07/country=USSR таблицы logs

The screenshot shows the Hadoop HDFS browser interface. At the top, there are tabs for 'Hadoop', 'Overview', and 'Datanodes'. Below that is a 'Browse Directory' section with a path bar showing '/user/user/warehouse/logs/dt=1969-11-07/country=USSR'. A search bar and a 'Show 25 entries' button are also present. The main content area displays a table with one entry:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name			
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	13 B	Oct 09 14:27	1	128 MB	000000_0			

Below the table, a 'File contents' section shows the text '10月Leningrad'.

Рисунок 56 — Единственная запись в разделе dt=1969-11-07/country=USSR таблицы logs

В отличие от раздела $dt=1969-11-07$, в раздел $dt=2019-10-08$ попало два подраздела по полю country, и, соответственно, большее количество записей (рисунки Рисунок 57 — Рисунок 60).

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxr-x	user	supergroup	0 B	Oct 09 14:25	0	0 B	country=Russia
drwxrwxr-x	user	supergroup	0 B	Oct 09 14:23	0	0 B	country=Ukraine

Рисунок 57 — Разделы logs по полю country внутри раздела $dt=2019-10-08$

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	user	supergroup	9 B	Oct 09 14:19	1	128 MB	000000_0
-rwxrwxr-x	user	supergroup	12 B	Oct 09 14:25	1	128 MB	000000_0_copy_1

Рисунок 58 — Структура раздела $dt=2019-10-08/country=Russia$ таблицы logs

Рисунок 59 — Запись 1 в разделе $dt=2019-10-08/country=Russia$ таблицы logs



Рисунок 60 — Запись 2 в разделе dt=2019-10-08/country=Russia таблицы logs

Следует помнить, что определения столбцов в разделе PARTITIONED BY представляют собой полноценные столбцы, называемые столбцами разделов; однако в файлах данных значения этих столбцов отсутствуют, поскольку они определяются по именам каталогов.

Столбцы разделов могут использоваться в инструкциях SELECT обычным образом. Hive проверяет входные данные и просматривает только нужные разделы.

Задание. Напишите запросы на HiveQL для выборки из каждого раздела.

Пример разбиения таблицы на гнезда (buckets)

Таблицы или разделы могут дополнитель но делиться на гнезда (buckets), обеспечивающие дополнительное структурирование данных для повышения эффективности запросов. Например, деление на гнезда по идентификатору пользователя позволяет быстро выполнить запрос для случайной выборки из общего множества пользователей.

Существует две причины для определения гнезд в таблицах или разделах. Первая причина — повышение эффективности запросов. Гнезда образуют в таблице дополнительную структуру, которая может использоваться при выполнении некоторых запросов. В частности, соединение двух таблиц, в которых определены гнезда по одинаковым столбцам (включающим в себя столбцы соединения), эффективно реализуется в форме соединения на стороне отображения (Map-side Join). Вторая причина — повышение эффективности выборки. В процессе разработки или уточнения запроса бывает очень полезно опробовать его на небольшом подмножестве большого набора данных.

Создайте таблицу bucketed_users, имеющую следующие информационные поля: id INT (целочисленный идентификатор), name STRING (имя). По полю id должно выполняться разбиение на 4 гнезда (bucketing). Конструкция для создания таблицы проиллюстрирована на рисунке (Рисунок 61).

```
0: jdbc:hive2://localhost:10000> CREATE TABLE bucketed_users(id INT, name STRING  
) CLUSTERED BY (id) INTO 4 BUCKETS;  
No rows affected (0,233 seconds)  
0: jdbc:hive2://localhost:10000>
```

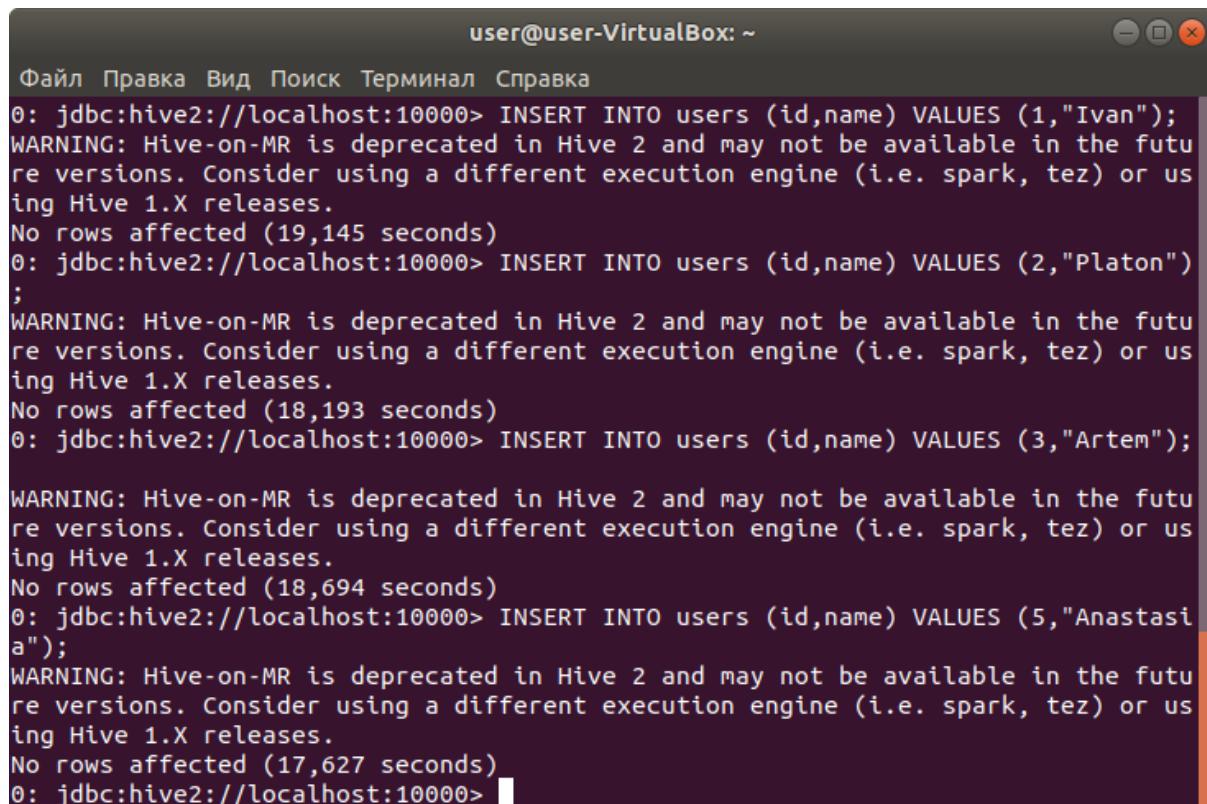
Рисунок 61 — Создание таблицы bucketed_users с гнездами (buckets)

Создайте таблицу users с тем же набором информационных полей, что и bucketed_users, но без разбиения на гнезда (Рисунок 62).

```
0: jdbc:hive2://localhost:10000> CREATE TABLE users(id INT, name STRING);  
No rows affected (0,211 seconds)
```

Рисунок 62 — Создание таблицы users

Добавьте в таблицу users несколько записей (Рисунок 63).



The screenshot shows a terminal window titled "user@user-VirtualBox: ~". The window contains the following text:

```
Файл Правка Вид Поиск Терминал Справка  
0: jdbc:hive2://localhost:10000> INSERT INTO users (id,name) VALUES (1,"Ivan");  
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  
No rows affected (19,145 seconds)  
0: jdbc:hive2://localhost:10000> INSERT INTO users (id,name) VALUES (2,"Platon");  
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  
No rows affected (18,193 seconds)  
0: jdbc:hive2://localhost:10000> INSERT INTO users (id,name) VALUES (3,"Artem");  
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  
No rows affected (18,694 seconds)  
0: jdbc:hive2://localhost:10000> INSERT INTO users (id,name) VALUES (5,"Anastasiya");  
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  
No rows affected (17,627 seconds)  
0: jdbc:hive2://localhost:10000>
```

Рисунок 63 — Заполнение таблицы users

Вставьте содержимое таблицы users в пустую таблицу bucketed_users с помощью конструкции INSERT OVERWRITE, и сделайте выборку из всей таблицы и гнезда 1 (Рисунок 64).

```
user@user-VirtualBox: ~
Файл Правка Вид Поиск Терминал Справка
0: jdbc:hive2://localhost:10000> INSERT OVERWRITE TABLE bucketed_users SELECT * FROM users;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (37,983 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users;
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 5                 | Anastasia      |
| 1                 | Ivan           |
| 2                 | Platon          |
| 3                 | Artem          |
+-----+-----+
4 rows selected (0,224 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users TABLESAMPLE(BUCKET 1 OUT OF 4 ON id);
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
+-----+-----+
No rows selected (0,171 seconds)
0: jdbc:hive2://localhost:10000> □
```

Рисунок 64 — Заполнение и выборка из таблицы bucketed_users

На рисунках (Рисунок 65) и (Рисунок 66) представлены запросы для выборки из гнезд 2–4.

```
user@user-VirtualBox: ~
Файл Правка Вид Поиск Терминал Справка
1 OUT OF 4 ON id;
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
+-----+-----+
No rows selected (0,171 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users TABLESAMPLE(BUCKET 2 OUT OF 4 ON id);
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 5                 | Anastasia      |
| 1                 | Ivan           |
+-----+-----+
2 rows selected (0,162 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users TABLESAMPLE(BUCKET 3 OUT OF 4 ON id);
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 2                 | Platon          |
+-----+-----+
1 row selected (0,155 seconds)
0: jdbc:hive2://localhost:10000> □
```

Рисунок 65 — Выборки из гнезд 2–3 таблицы bucketed_users

```

user@user-VirtualBox: ~
Файл Правка Вид Поиск Терминал Справка
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 5                 | Anastasia          |
| 1                 | Ivan               |
+-----+-----+
2 rows selected (0,162 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users TABLESAMPLE(BUCKET
3 OUT OF 4 ON id);
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 2                 | Platon              |
+-----+-----+
1 row selected (0,155 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM bucketed_users TABLESAMPLE(BUCKET
4 OUT OF 4 ON id);
+-----+-----+
| bucketed_users.id | bucketed_users.name |
+-----+-----+
| 3                 | Artem               |
+-----+-----+
1 row selected (0,164 seconds)
0: jdbc:hive2://localhost:10000> □

```

Рисунок 66 — Выборка из гнезда 4 таблицы bucketed_users

Откройте в браузере просмотр файловой системы HDFS. Изучите структуру разделов таблицы bucketed_users (рисунки Рисунок 67 — Рисунок 71).

Browse Directory

/user/user/warehouse/bucketed_users										Go!			
Show 25 entries										Search: <input type="text"/>			
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name					
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	0 B	Oct 09 15:01	1	128 MB	000000_0					
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	19 B	Oct 09 15:01	1	128 MB	000001_0					
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	9 B	Oct 09 15:01	1	128 MB	000002_0					
<input type="checkbox"/>	-rwxrwxr-x	user	supergroup	8 B	Oct 09 15:01	1	128 MB	000003_0					

Showing 1 to 4 of 4 entries

Previous 1 Next

Рисунок 67 — Гнезда таблицы bucketed_users в HDFS

The screenshot shows a database interface with two main panes. The left pane is titled 'Browse Directory' and lists four entries under the path '/user/user/warehouse/bucketed_users'. The right pane is titled 'File information - 000000_0' and displays a detailed view of a file. It includes options to 'Download', 'Head the file (first 32K)', and 'Tail the file (last 32K)'. A 'File contents' section is empty. Below it is a table of file blocks:

Block Size	Name	
128 MB	000000_0	
128 MB	000001_0	
128 MB	000002_0	
128 MB	000003_0	

Pagination controls at the bottom show 'Previous', '1', and 'Next'.

Рисунок 68 — Выборка из гнезда 1 таблицы bucketed_users

This screenshot is similar to Figure 68, but the 'Block information' dropdown is open, showing details for Block 0: Block ID: 1073741966, Block Pool ID: BP-1034934114-127.0.1.1-1570619494127, Generation Stamp: 1142, Size: 19, and Availability: user-VirtualBox. The rest of the interface is identical to Figure 68.

Рисунок 69 — Выборка из гнезда 2 таблицы bucketed_users

This screenshot is similar to Figures 68 and 69, but the 'File contents' section shows the first few lines of the file: '5 @Anastasia\n1 @Ivan'. The 'Block information' dropdown shows the same details as Figure 69. The rest of the interface is identical.

Рисунок 70 — Выборка из гнезда 3 таблицы bucketed_users

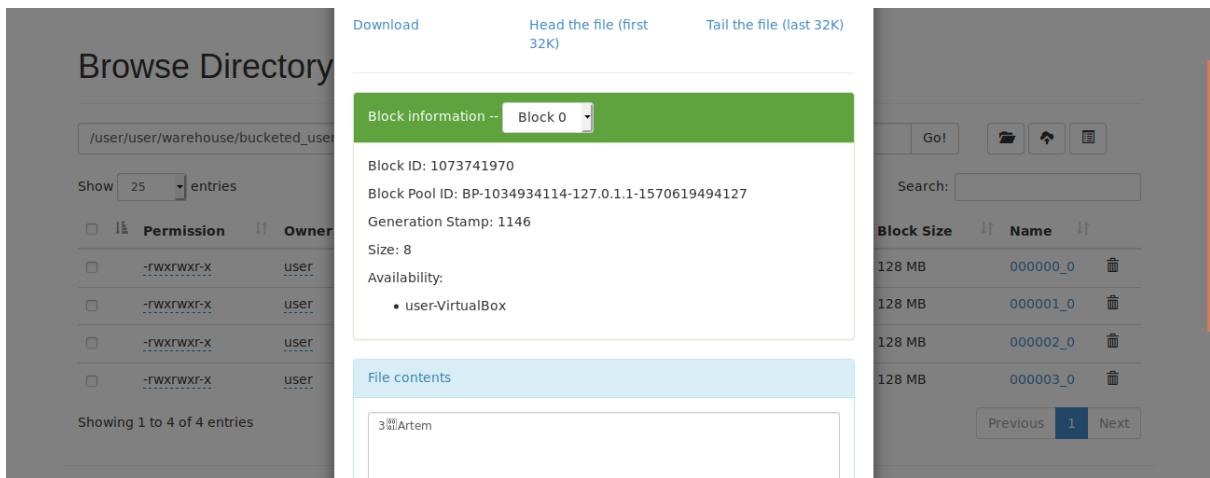


Рисунок 71 — Выборка из гнезда 4 таблицы bucketed_users

Задание. Объясните для каждой записи, почему она попадает в соответствующее гнездо.

Технология Pig

Pig — это скриптовый язык для анализа больших наборов данных. MapReduce критикуют в том, что цикл разработки приложений достаточно продолжительный. Написание классов mapper и reducer на Java, компиляция и упаковка кода, отправка заданий и получение результатов — трудоемкое дело. Даже с технологией Hadoop Streaming, которая исключает этап компиляции и упаковки, опыт отладки программного кода по-прежнему необходим. Привлекательная сторона Pig — способность обрабатывать терабайты данных в ответ на полдюжины строк Pig Latin, введенных с консоли. Pig был создан в Yahoo, чтобы исследователям и инженерам было легче анализировать огромные наборы данных. Pig поддерживает программиста, пишущего запрос, предоставляя команды для анализа структур данных в программе во время ее написания. Еще более полезно то, что Pig позволяет выполнить пробный прогон для представительного подмножества входных данных, чтобы можно было увидеть, есть ли ошибки в обработке, прежде чем запускать ее на полном наборе данных. Pig состоит из двух частей:

- Pig Latin — язык, используемый для описания потоков данных.
- Среда выполнения для запуска программ Pig Latin. В настоящее время существует две среды: локальное выполнение в одной JVM и распределенное выполнение в кластере Hadoop.

Программа на Pig Latin состоит из последовательности операций или преобразований, которые применяются к входным данным для получения выходных данных. В целом, операции описывают поток данных, который среда

выполнения Pig преобразует в исполняемое представление и затем запускает. Pig превращает преобразования в серию задач MapReduce автоматически, что позволяет программисту сосредоточиться на обработке данных, а не на механизме исполнения.

В некоторых случаях Pig работает не так хорошо, как программы, написанные с применением MapReduce, но каждым выпуском его отставание сокращается, поскольку команда Pig реализует сложные алгоритмы для применения реляционных операторов Pig. Если программист не готов вкладывать много усилий в оптимизацию MapReduce, написание запросов на языке Pig Latin может сэкономить его время.

Установка и настройка Pig на Hadoop

Перейдите в домашний каталог, если Вы не там:

```
$ cd ~
```

1. Скачайте Pig и распакуйте архив.

```
$ wget http://apache-mirror.rbc.ru/pub/apache/pig/pig-0.17.0/pig-0.17.0.tar.gz  
$ tar -xvf pig-0.17.0.tar.gz
```

2. Настройте переменные окружения.

Убедитесь, что файл `~/.bash_profile` (или `~/.bashrc`) имеет следующий текст в конце:

```
# User specific environment and startup programs

JAVA_HOME=$HOME/jdk1.8.0_221
HADOOP_HOME=$HOME/hadoop-2.9.2
HIVE_HOME=$HOME/apache-hive-2.3.6-bin
PIG_HOME=$HOME/pig-0.17.0
PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME
/bin:$PIG_HOME/bin

export JAVA_HOME
export HADOOP_HOME
export HIVE_HOME
export PIG_HOME
export PATH
```

Внесите изменения в имеющуюся конфигурацию. Не забудьте дописать новый путь в PATH. Перезапустите сеанс пользователя. Перейдите в домашний каталог, если Вы не там.

Команда должна показать версию Pig:

```
$ pig -version
```

3. Проверьте работоспособность pig.

Создайте каталог `$PIG_HOME/mysources`:

```
$ mkdir $PIG_HOME/mysources
```

Поместите в него файлы `input1`, `cars.txt`, `logs.txt`, `logs.pig` (приложение 4).

Перейдите в него:

```
$ cd $PIG_HOME/mysources
```

Скопируйте файл input1 в HDFS и проверьте содержимое:

```
$ hdfs dfs -put input1 pigInput1 # поместить данные в HDFS  
$ hdfs dfs -cat pigInput1 # просмотреть данные в HDFS
```

Запустите Pig командой pig и выполните команды в оболочке grunt (Рисунок 72).

```
user@user-VirtualBox:~$ pig  
19/10/23 13:41:58 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL  
19/10/23 13:41:58 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE  
19/10/23 13:41:58 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType  
2019-10-23 13:41:59,688 [main] INFO org.apache.pig.Main - Apache Pig version 0.  
17.0 (r1797386) compiled Jun 02 2017, 15:41:58  
2019-10-23 13:41:59,693 [main] INFO org.apache.pig.Main - Logging error messages to: /home/user/pig_1571827319679.log  
2019-10-23 13:41:59,826 [main] INFO org.apache.pig.impl.util.Utils - Default bootstrap file /home/user/.pigbootup not found  
2019-10-23 13:42:02,785 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address  
2019-10-23 13:42:02,792 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://localhost:9000  
2019-10-23 13:42:06,293 [main] INFO org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-b0a3ca97-a43d-4f2b-8719-3e2e59472b20  
2019-10-23 13:42:06,298 [main] WARN org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false  
grunt> []
```

Рисунок 72 — Приветствие оболочки grunt

Выполните загрузку отношения и выведите содержимое на терминал (Рисунок 73):

```
records = LOAD 'pigInput1' using PigStorage(',') AS (id:int, name:chararray,  
department:chararray, city:chararray);  
dump records
```

Задание. Объясните схему данных и назначение параметров операции LOAD.

```
(111,Jogh,Sales,Austin)  
(222,Alex,Marketing,New York)  
(333,Philip,Operation,Sacramento)  
(444,Terry,Sales,New York)  
(555,Jessi,Development,Boston)  
grunt> []
```

Рисунок 73 — Содержимое отношения records

Выполните операцию проекции для загруженного отношения, и выведите содержимое на терминал (Рисунок 74):

```
foreach_records = foreach records generate name,department;  
dump foreach_records  
(Jogh,Sales)  
(Alex,Marketing)  
(Philip,Operation)  
(Terry,Sales)  
(Jessi,Development)  
grunt> []
```

Рисунок 74 — Содержимое проекции foreach_records

Выполните фильтрацию отношения records, наложив условие на значение поля city (Рисунок 75):

```
records_filter = filter records by city == 'Austin';
dump records_filter
(111,Jogh,Sales,Austin)
grunt> █
```

Рисунок 75 — Содержимое отношения records_filter

Протестируйте сортировку по полю id (Рисунок 76):

```
records_order = order records by id desc;
dump records_order
(555,Jessi,Development,Boston)
(444,Terry,Sales>New York)
(333,Philip,Operation,Sacramento)
(222,Alex,Marketing>New York)
(111,Jogh,Sales,Austin)
grunt> █
```

Рисунок 76 — Содержимое отношения records_order

Сохраните содержимое records_order в каталоге pigresult в HDFS, и завершите работу:

```
STORE records_order into 'pigresult';
quit
```

Проверьте содержимое файла в HDFS:

```
$ hdfs dfs -cat pigresult/part-r-00000
```

Pig Latin: отношения, сумки, кортежи, поля

Инструкции Pig Latin работают с отношениями. Отношение можно определить следующим образом:

- Отношение (relation) — это сумка (точнее, сумка верхнего уровня вложенности).
- Сумка (bag) — это набор кортежей.
- Кортеж (tuple) — это упорядоченный набор полей.
- Поле (field) — это элемент данных.

Отношение Pig — это набор кортежей. Отношение Pig похоже на таблицу в реляционной базе данных, где кортежи в сумке соответствуют строкам в таблице. Однако, в отличие от реляционной таблицы, отношения Pig не требуют, чтобы каждый кортеж содержал одинаковое количество полей или чтобы поля в одной позиции (столбце) имели один и тот же тип.

Также обратите внимание, что отношения неупорядочены, что означает, что нет гарантии, что кортежи обрабатываются в каком-либо конкретном

порядке. Кроме того, обработка может быть распараллелена, и в этом случае кортежи не обрабатываются в соответствии с каким-либо линейным порядком.

Рассмотрим работу с отношениями на примере набора данных cars.txt (приложение 4), содержащего сведения об автомобилях. Данные представлены в табличном виде четырьмя колонками: год выпуска, модель, стоимость, пробег.

Скопируйте файл cars.txt в HDFS и проверьте содержимое:

```
$ hdfs dfs -put cars.txt inputCars # поместить данные в HDFS  
$ hdfs dfs -cat inputCars # просмотреть данные в HDFS
```

Запустите оболочку grunt и выполняйте инструкции на языке Pig Latin.

Доступ к отношениям осуществляется по имени. Имена присваиваются в инструкциях Pig Latin. В этом примере имя отношения — records.

```
records = LOAD 'inputCars' USING PigStorage(',') AS  
(date:chararray, name_car:chararray, cost:int, mileage:chararray);
```

Вывод содержимого отношения можно выполнить с помощью dump:

```
dump records
```

Описание схемы данных можно получить с помощью describe:

```
describe records
```

Содержимое и схема отношения records представлены на рисунке (Рисунок 77).

```
(1970,Dodge Challenger,22000,74598)  
(1976,Ford Shellby,45500,23000)  
(1970,Cadillac Eldarado,36000,14023)  
(1970,Caterham 7,55000,1478)  
(1976,Chevrolet Camaro,27000,69963)  
(1976,Chevrolet Panamera,270000,60963)  
grunt> describe records  
records: {date: chararray, name_car: chararray, cost: int, mileage: chararray}  
grunt> █
```

Рисунок 77 — Содержимое и схема отношения records

Доступ к полям осуществляется по именам или с помощью позиционных обозначений. Позиционное обозначение генерируется системой, обозначается знаком доллара и начинается с нуля, например: \$0, \$1, \$2.

Имена назначаются пользователем с помощью схем данных (или, в случае оператора GROUP и некоторых функций, системой). Можно использовать любое имя, которое не является ключевым словом Pig.

Когда назначаются имена полям (сопоставление схемы с помощью AS), возможность использования позиционной нотации остается. Однако для упрощения отладки рекомендуется использовать имена полей. На рисунке (Рисунок 78) показан пример выполнения операции проекции и схема данных результирующего отношения:

```
records_projection = FOREACH records GENERATE name_car,$2;
```

```
(Dodge Challenger,22000)
(Ford Shellby,45500)
(Cadillac Eldarado,36000)
(Caterham 7,55000)
(Chevrolet Camaro,27000)
(Chevrolet Panamera,270000)
grunt> describe records_projection
records_projection: {name_car: chararray,cost: int}
grunt> █
```

Рисунок 78 — Содержимое и схема отношения records_projection

Отношения records и records_projection содержат кортежи, а потому их можно назвать сумками. Поскольку отношения сами по себе не содержатся в каких-либо сумках, их можно считать сумками верхнего уровня вложенности.

Обратите внимание на следующие особенности работы с сумками.

- В сумке могут быть повторяющиеся кортежи.
- В сумке могут быть кортежи с разным количеством полей. Однако, если Pig пытается получить доступ к несуществующему полю, подставляется значение null.
- В сумке могут быть кортежи с полями с разными типами данных. Однако для того, чтобы Pig мог эффективно обрабатывать сумки, схемы кортежей в этих сумках должны быть одинаковыми. Например, если половина кортежей включает поля chararray, а другая половина включает поля float, только половина кортежей будет участвовать в любом виде вычислений, потому что поля chararray будут преобразованы в null.
- Сумки имеют две формы: сумка верхнего уровня вложенности (отношение) и внутренняя сумка.

Сгруппировать автомобили по году выпуска можно оператором GROUP:

```
grouped_records = GROUP records BY date;
```

В этом примере grouped_records — это отношение или набор кортежей. Наборы в отношении grouped_records имеют два поля. Первое поле — это group типа chararray. Второе поле — это records типа сумка (внутренняя). Пример проиллюстрирован на рисунке (Рисунок 79). Схема кортежей в сумке records та же самая, что и в отношении records.

```
(1970,{(1970,Caterham 7,55000,1478),(1970,Cadillac Eldarado,36000,14023),(1970,Dodge Cha
lenger,22000,74598)})
(1976,{(1976,Chevrolet Panamera,270000,60963),(1976,Chevrolet Camaro,27000,69963),(1976,
Ford Shellby,45500,23000)})
grunt> describe grouped_records
grouped_records: {group: chararray,records: {(date: chararray,name_car: chararray,cost:
int,mileage: chararray)}}
```

Рисунок 79 — Содержимое и схема отношения grouped_records

Узнать цену самого дорого автомобиля в каждой группе можно, применив агрегатную функцию MAX к ценам автомобилей внутри каждой сумки:

```
max_cost_grouped = FOREACH grouped_records GENERATE MAX(records.cost);
```

Обратите внимание, что применяется оператор разыменования «точка» для доступа к вложенным полям внутри сумки records.

Пример проиллюстрирован на рисунке (Рисунок 80).

```
(55000)
(270000)
grunt> describe max_cost_grouped
max_cost_grouped: {int}
grunt> █
```

Рисунок 80 — Содержимое и схема отношения max_cost_grouped

Задание. Перепишите конструкцию FOREACH так, чтобы вместе с ценой самого дорогого автомобиля в группе выводился и год его выпуска.

Если требуется применить агрегатную функцию ко всем кортежам отношения, можно поместить их в одну группу (сумку), воспользовавшись ключевым словом ALL.

```
grouped_all = GROUP records ALL;
```

Пример проиллюстрирован на рисунке (Рисунок 81).

```
(all,{(1976,Chevrolet Panamera,270000,60963),(1976,Chevrolet Camaro,27000,69963),(1970,C
aterham 7,55000,1478),(1970,Cadillac Eldarado,36000,14023),(1976,Ford Shellby,45500,2300
0),(1970,Dodge Challenger,22000,74598)})
grunt> describe grouped_all
grouped_all: {group: chararray,records: {(date: chararray,name_car: chararray,cost: int,
mileage: chararray)}}
```

Рисунок 81 — Содержимое и схема отношения grouped_all

Теперь узнать цену самого дорогого автомобиля можно, применив агрегатную функцию MAX к ценам всех автомобилей.

```
max_cost_all = FOREACH grouped_all GENERATE MAX(records.cost);
```

Пример проиллюстрирован на рисунке (Рисунок 82).

```
(270000)
grunt> describe max_cost_all
max_cost_all: {int}
grunt> █
```

Рисунок 82 — Содержимое и схема отношения max_cost_all

Пример обработки журнала веб-сервера с помощью Pig

Пример входных данных — журнал веб-сервера logs.txt (приведен в приложении 4).

Скопируйте файл logs.txt в HDFS и проверьте содержимое:

```
$ hdfs dfs -put logs.txt inputLogs # поместить данные в HDFS
$ hdfs dfs -cat inputLogs           # просмотреть данные в HDFS
```

Выполним обработку событий доступа за определенный день (сутки).

Рассчитаем следующие параметры:

- общее количество запросов;
- количество запросов с каждого уникального IP;
- количество запросов на каждый уникальный URL;
- объем данных, переданных по каждому URL.

Ниже приводится скрипт, с помощью которого решается поставленная задача. Этот скрипт (как и все скрипты в Pig) не выполняется построчно, как в интерпретируемых языках. Компилятор Pig разбирает зависимости и определяет потоки данных. Компилирование скрипта начинается с конца, то есть с команды STORE. Для данных, после обработки которых нет команды сохранения, не будет создано никаких задач, и сами данные не будут даже прочитаны. Это позволяет писать скрипт в достаточно произвольной форме. Всю работу по оптимизации, определению порядка исполнения, распараллеливанию возьмет на себя Pig.

Полный листинг скрипта logs.pig приведен в приложении 4. Он состоит из трех частей: загрузка данных, обработка и сохранение. Такой порядок является общим для большинства задач. В некоторых случаях решение задач может включать и дополнительные этапы — например, генерацию данных (например, структурированных искусственных данных для проверки алгоритма) или сохранение промежуточных результатов вычислений. Рассмотрим каждый этап более подробно.

Загрузка. Оператор LOAD создает отношение records из файлов в HDFS (можно передать как директорию, так и изолированный файл), используя стандартный интерфейс PigStorage (также укажем, что разделителем в файлах является запятая).

```
records = LOAD 'inputLogs' USING PigStorage(',')  
AS (  
    date:chararray,  
    clientip:chararray,  
    clientport:chararray,  
    proto:chararray,  
    statuscode:int,  
    bytes:int,  
    sq:chararray,  
    bq:chararray,  
    request:chararray );
```

Каждая строка из файлов предстанет кортежем в отношении. Секция AS присваивает полям в кортеже типы и имена, по которым нам будет удобнее к ним

обращаться. Рисунок 83 иллюстрирует загруженное содержимое, выведенное операцией dump.

```
(07/Dec/2013:20:04:10,95.153.193.56,37877,http,404,1492,0,0,GET /745dbda3-894e-43aa-9146  
-607f19fe4428.mp3 HTTP/1.1)  
(07/Dec/2013:20:04:33,95.153.193.56,37877,http,206,2048,0,0,GET /login.html HTTP/1.1)  
(07/Dec/2013:20:05:13,95.153.193.56,37877,http,200,1492030,0,0,GET /745dbda3-894e-43aa-9  
146-607f19fe4428.mp3 HTTP/1.1)  
(07/Dec/2013:20:06:44,95.153.193.56,37877,http,200,1492030,0,0,GET /745dbda3-894e-43aa-9  
146-607f19fe4428.mp3 HTTP/1.1)  
(08/Dec/2013:15:00:28,178.88.91.180,13600,http,200,4798,0,0,GET /public/cars/bmw7l/down.  
png HTTP/1.1)  
(08/Dec/2013:15:00:28,188.88.66.183,13644,http,200,4798,0,0,GET /public/cars/bmw7l/down.  
png HTTP/1.1)  
(08/Dec/2013:15:00:29,193.110.115.45,64318,http,200,1594,0,0,GET /K1/img/top-nav-bg-defa  
ult.jpg HTTP/1.1)  
grunt> █
```

Рисунок 83 — Содержимое отношения records

Обработка. Рассчитаем общее количество записей в журнале с помощью оператора COUNT. Перед этим необходимо объединить все строки в records в одну группу операторами FOREACH и GROUP.

```
count_total = FOREACH (GROUP records ALL) GENERATE COUNT(records);
```

Результат представлен на рисунке (Рисунок 84).

```
(7)  
grunt> █
```

Рисунок 84 — Содержимое отношения count_total

Рассчитаем количество запросов с уникальных адресов. В кортежах в отношении records в поле clientip содержатся IP-адреса, с которых выполнялись запросы. Сгруппируем кортежи в records по полю clientip и определим новое отношение, состоящее из двух полей:

1. поле ip, значение которого берется из названия группы в отношении records;
2. количество записей в группе — cnt, посчитанное оператором COUNT, то есть количество записей, соответствующих определенному IP-адресу в поле IP.

```
count_ip = FOREACH (GROUP records BY clientip) GENERATE group AS ip,  
COUNT(records) AS cnt;
```

Результат представлен на рисунке (Рисунок 85).

```
(178.88.91.180,1)  
(188.88.66.183,1)  
(95.153.193.56,4)  
(193.110.115.45,1)  
grunt> █
```

Рисунок 85 — Содержимое отношения count_ip

Далее определяется еще одно отношение top_ip, состоящее из тех же данных, что и count_ip, но отсортированное по полю cnt оператором ORDER. Таким образом, в top_ip у будет список IP-адресов клиентов, с которых чаще всего происходили запросы.

```
top_ip = ORDER count_ip BY cnt DESC;
```

Результат представлен на рисунке (Рисунок 86).

```
(95.153.193.56,4)  
(193.110.115.45,1)  
(188.88.66.183,1)  
(178.88.91.180,1)  
grunt> █
```

Рисунок 86 — Содержимое отношения top_ip

Рассчитаем количество успешных запросов на каждый URL, а также суммарный объем загруженных по каждому URL данных. Для этого сначала воспользуемся оператором фильтрации FILTER, отобрав только успешные запросы с HTTP кодами 200 OK и 206 Partial Content. Этот оператор определяет новое отношение filtered_req из отношения records, отфильтровав его по полю statuscode.

```
filtered_req = FILTER records BY statuscode == 200 OR statuscode == 206;
```

Результат представлен на рисунке (Рисунок 87).

```
(07/Dec/2013:20:04:33,95.153.193.56,37877,http,206,2048,0,0,GET /login.html HTTP/1.1)  
(07/Dec/2013:20:05:13,95.153.193.56,37877,http,200,1492030,0,0,GET /745dbda3-894e-43aa-9  
146-607f19fe4428.mp3 HTTP/1.1)  
(07/Dec/2013:20:06:44,95.153.193.56,37877,http,200,1492030,0,0,GET /745dbda3-894e-43aa-9  
146-607f19fe4428.mp3 HTTP/1.1)  
(08/Dec/2013:15:00:28,178.88.91.180,13600,http,200,4798,0,0,GET /public/cars/bmw7l/down.  
png HTTP/1.1)  
(08/Dec/2013:15:00:28,188.88.66.183,13644,http,200,4798,0,0,GET /public/cars/bmw7l/down.  
png HTTP/1.1)  
(08/Dec/2013:15:00:29,193.110.115.45,64318,http,200,1594,0,0,GET /K1/img/top-nav-bg-defa  
ult.jpg HTTP/1.1)  
grunt> █
```

Рисунок 87 — Содержимое отношения filtered_req

Далее аналогично расчету IP-адресов рассчитаем количество уникальных URL, группируя записи в отношении requests по полю request. Представляет интерес переданный объем данных по каждому URL: его можно рассчитать с помощью оператора SUM, складывающего поля bytes в сгруппированных записях отношения filtered_req.

```
count_req = FOREACH (GROUP filtered_req BY request) GENERATE group AS req,  
COUNT(filtered_req) AS cnt, SUM(filtered_req.bytes) AS bytes;
```

Результат представлен на рисунке (Рисунок 88).

```
(GET /login.html HTTP/1.1,1,2048)
(GET /public/cars/bmw7l/down.png HTTP/1.1,2,9596)
(GET /K1/img/top-nav-bg-default.jpg HTTP/1.1,1,1594)
(GET /745dbda3-894e-43aa-9146-607f19fe4428.mp3 HTTP/1.1,2,2984060)
grunt> █
```

Рисунок 88 — Содержимое отношения count_req

Теперь осуществим сортировку по полю bytes, определяя новое отношение top_req.

```
top_req = ORDER count_req BY bytes DESC;
```

Результат представлен на рисунке (Рисунок 89).

```
(GET /745dbda3-894e-43aa-9146-607f19fe4428.mp3 HTTP/1.1,2,2984060)
(GET /public/cars/bmw7l/down.png HTTP/1.1,2,9596)
(GET /login.html HTTP/1.1,1,2048)
(GET /K1/img/top-nav-bg-default.jpg HTTP/1.1,1,1594)
grunt> █
```

Рисунок 89 — Содержимое отношения top_req

Сохранение результатов. Предпочтительно сохранять результаты каждого выполнения скрипта в отдельную директорию, имя которой включает дату и время исполнения. Для этого можно воспользоваться функцией вызова произвольной shell-команды прямо из Pig-скрипта (ее нужно написать в обратных кавычках). В примере результат команды date заносится в переменную DT, которая затем подставляется в пути сохранения данных. Сохраняются результаты командой STORE: каждое отношение — в свой каталог.

```
%declare DT `date +%y%m%d%H%M`
STORE count_total INTO 'pigresult/$DT/count_total';
STORE top_ip INTO 'pigresult/$DT/top_ip';
STORE top_req INTO 'pigresult/$DT/top_req';
```

Запустить скрипт из терминала Linux можно следующим образом:

```
$ pig logs.pig
```

Проверять результат работы скрипта следует в каталоге HDFS pigresult.

Задание. Обработка реляционных данных с применением Hive, Pig, MapReduce

- Разработать базу данных (не менее 3 сущностей). Разработать не менее 5 запросов к БД, из них не менее 3 с применением JOIN.
- Реализовать БД и запросы на Hive и Pig.
- Реализовать запросы с применением паттерна MapReduce.
- Продемонстрировать, что результаты выполнения запросов на Hive, Pig совпадают с результатами, полученными с помощью собственной реализации MapReduce.

4. Анализ данных с помощью Spark

Технология Spark

Apache Spark — это среда кластерных вычислений для крупномасштабной обработки данных. Spark не использует MapReduce в качестве исполняющего механизма; вместо этого он использует свою собственную распределенную среду выполнения для выполнения заданий на кластере. Тем не менее, Spark имеет много аналогий с MapReduce, с точки зрения API и среды исполнения. Spark тесно связан с Hadoop: он может работать на YARN и работает с форматами файлов Hadoop и хранилищем файлов HDFS.

Spark наиболее известен своей способностью хранить большие рабочие наборы данных в памяти между заданиями. Эта способность позволяет Spark превзойти аналогичный рабочий процесс MapReduce (на порядок или более в некоторых случаях), где наборы данных всегда загружаются с диска. Два типа приложений, которые значительно выигрывают от применения модели обработки Spark:

- итеративные алгоритмы, где функция применяется к набору данных несколько раз до выполнения условия завершения;
- интерактивный анализ, где пользователь отправляет множество целевых исследовательских запросов к набору данных.

Даже если не требуется кэширование в памяти, Spark очень привлекателен по двум другим причинам: по наличию механизма DAG (Directed Acyclic Graph) и удобству использования. В отличие от MapReduce, механизм Spark DAG может обрабатывать произвольные конвейеры операторов и преобразовывать их в одно задание для пользователя.

Удобство использования Spark также не имеет себе равных с богатым набором API для выполнения множества стандартных задач обработки данных, таких как объединения. Spark предоставляет API на трех языках: Scala, Java и Python.

Установка и настройка Spark

Автономный режим

Далее все от имени обычного пользователя. Перейдите в домашний каталог, если Вы не там:

```
$ cd ~
```

1. Скачать Spark и распаковать архив.

```
$ wget http://apache-mirror.rbc.ru/pub/apache/spark/spark-2.3.4/spark-2.3.4-bin-hadoop2.7.tgz
```

```
$ tar -xvf spark-2.3.4-bin-hadoop2.7.tgz
```

Если Вам больше не нужен архив, его можно удалить:

```
$ rm -f tar -xvf spark-2.3.4-bin-hadoop2.7.tgz
```

2. Убедитесь, что файл `~/.bash_profile` (или `~/.bashrc`) имеет следующий текст в конце:

```
# User specific environment and startup programs

JAVA_HOME=$HOME/jdk1.8.0_221
HADOOP_HOME=$HOME/hadoop-2.9.2
HIVE_HOME=$HOME/apache-hive-2.3.6-bin
PIG_HOME=$HOME/pig-0.17.0
SPARK_HOME=$HOME/spark-2.3.4-bin-hadoop2.7
PATH=$PATH:$HOME/.local/bin:$HOME/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$PIG_HOME/bin:$SPARK_HOME/bin

export JAVA_HOME
export HADOOP_HOME
export HIVE_HOME
export PIG_HOME
export SPARK_HOME
export PATH
```

Внесите изменения в имеющуюся конфигурацию. Не забудьте дописать новый путь в PATH.

Перезапустите сеанс пользователя. Перейдите в домашний каталог, если Вы не там.

Примечание. Командой `source <script_name>` Вы можете обработать в открытом shell команды из скрипта `<script_name>`. Например, `source ~/.bash_profile` — временная мера, которая избавит от необходимости перезапускать сеанс пользователя.

3. Запуск главного процесса Spark.

```
$ $SPARK_HOME/sbin/start-master.sh
```

После запуска будет активен веб-интерфейс <http://localhost:8080>

В шапке будет указан URL сервиса, который можно использовать для подключения к нему подчиненных процессов (worker).

4. Запуск подчиненного процесса worker.

```
$ $SPARK_HOME/sbin/start-slave.sh spark://`hostname`:7077
```

Примечание. URL master-процесса, скорее всего, будет совпадать с параметром `spark://`hostname`:7077`

5. Список запущенных java-приложений:

```
$ jps
8327 Worker
8520 Jps
8249 Master
```

6. Запуск тестового приложения.

```
$ spark-submit --class org.apache.spark.examples.SparkPi \
--master spark://`hostname`:7077 \
```

```
--executor-memory 1G \
--total-executor-cores 1 \
$SPARK_HOME/examples/jars/spark-examples_2.11-2.3.4.jar 10
```

Ответом будет строка

```
pi is roughly 3.141347141347141
```

7. Запуск Spark Shell

Scala:

```
$ spark-shell --master spark://`hostname`:7077
```

Python:

```
$ pyspark --master spark://`hostname`:7077
```

8. Проверьте работоспособность Spark с помощью pyspark.

Создайте RDD из файла tweets.csv (приложение 5) и выполните несколько операций.

Вместо /path/to/file/ подставьте путь к файлу на Вашей машине.

```
>>> text = sc.textFile('file:///path/to/file/tweets.csv')
>>> header_text = text.first()
>>> header_list = header_text.split(',')
>>> header_list[12]
'"tweet_text"'
>>> rows = text.filter(lambda row: row != header_text)
>>> rows.count()
9
>>> flt = rows.filter(lambda row: row.split('"',')[12].find(u'их') >= 0)
>>> flt.count()
2
>>> flt.take(2)
```

Выход: Ctrl+D

9. Остановка служб.

Остановить все процессы worker на машине:

```
$ $SPARK_HOME/sbin/stop-slave.sh
```

Остановить процесс master:

```
$ $SPARK_HOME/sbin/stop-master.sh
```

Режим Spark on YARN

1. Совпадает с настройкой Standalone Mode.

2. Настроить переменные окружения согласно Standalone Mode п. 2 и добавить в ~/.bash_profile (или ~/.bashrc) дополнительно следующие записи:

```
HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_CONF_DIR
LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

3. Настроить Spark.

```
$ cd $SPARK_HOME/conf
$ cp spark-defaults.conf.template spark-defaults.conf
$ cat <<EOT >> spark-defaults.conf
```

```
spark.master          yarn
spark.driver.memory   512m
spark.yarn.am.memory 512m
spark.executor.memory 512m
EOT
```

Это многострочная команда, вставляйте ее в терминал целиком.

4. Отключить проверку лимитов памяти для контейнеров YARN.

Добавьте в конфигурацию \$HADOOP_CONF_DIR/yarn-site.xml еще два свойства:

```
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
</property>
```

5. Запустить Hadoop.

6. Запуск тестового приложения в Spark Client Mode.

```
$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn \
--deploy-mode client $SPARK_HOME/examples/jars/spark-examples_2.11-2.3.4.jar 10
```

7. Запуск тестового приложения в Spark Cluster Mode.

```
$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn \
--deploy-mode cluster $SPARK_HOME/examples/jars/spark-examples_2.11-2.3.4.jar 10
```

8. Проверьте работоспособность Spark с помощью pyspark согласно Standalone Mode п. 8.

```
$ pyspark --master yarn
```

Примечание. Параметр --master yarn не является обязательным, так как задан по умолчанию в spark-defaults.conf

9. Загрузите файл tweets.csv в HDFS и выполните п. 8 с новым расположением файла (URL: hdfs://path/to/file/tweets.csv).

Настройка версии Python для pyspark

Пусть /usr/bin/python3 — интерпретатор Python, который Вы хотите использовать вместо умалчиваемого. Задайте две переменные окружения:

```
$ cd $SPARK_HOME/conf
$ cp spark-env.sh.template spark-env.sh
$ cat <<EOT >> spark-env.sh
export PYSPARK_PYTHON=/usr/bin/python3
export PYSPARK_DRIVER_PYTHON=/usr/bin/python3
EOT
```

Это многострочная команда, вставляйте ее в терминал целиком.

Примеры работы с реляционными данными

Максимальная годовая температура

Пусть имеется набор данных, содержащий значения измерений температуры в файле age1.csv (Рисунок 90).

number	month	year	temp	rank
1	01	2000	9999	2
2	02	2000	12	1
3	03	2000	15	4
4	04	2000	34	5
5	05	2000	3	9
6	01	2001	9999	0
7	02	2001	5	2
8	03	2001	32	1
9	04	2001	8	4
10	05	2001	9999	5
11	06	2001	0	5
12	07	2002	8	9
13	08	2002	43	0
14	09	2003	7	1
15	09	2004	6	4

Рисунок 90 — Набор данных температуры

Набор имеет поля: номер измерения, месяц, год, значение температуры, ранг измерения.

Требуется вывести максимальную температуру для каждого года. Есть строки с неправильной температурой — 9999, их надо исключить, а также представляет интерес только ранг измерения, обозначенный 0, 1, 4, 5, 9.

Запустите интерактивную оболочку spark-shell для Scala:

```
$ spark-shell
```

Загрузите данные из HDFS в RDD:

```
val lines = sc.textFile("hdfs:///user/aslebedev/age1.csv")
```

Поскольку заголовок CSV-файла не содержит полезной информации, отфильтруем его, создав новый RDD исключительно с данными:

```
val header = lines.first()
val lines_filtered = lines.filter(row => row != header)
```

Строки необходимо разделить на поля, чтобы в дальнейшем применять операции реляционной алгебры:

```
val records = lines_filtered.map(_.split("\t"))
```

Отфильтруем записи согласно условию задачи — оставим только те, у которых температура имеет корректное значение, и ранг задан допустимым числом:

```
val filtered = records.filter(rec => (rec(3) != "9999" &&
rec(4).matches("[01459]")))
```

Сравнение произведено в рамках строкового типа данных.

Поля года и температуры можно преобразовать в целый тип следующим образом, при этом в одной лямбда-функции будет выполнена и операция проекции, и объединение полей в кортеж:

```
val tuples = filtered.map(rec => (rec(2).toInt, rec(3).toInt))
```

Выполняя редукцию по операции выбора максимума для второго поля (температура) для каждого ключа (первого поля, содержащего год), можно вычислить максимальную температуру для каждого года:

```
val maxTemps = tuples.reduceByKey((a, b) => Math.max(a, b))
```

Сохранение в каталог output:

```
maxTemps.saveAsTextFile("output")
```

Проверьте содержимое этого каталога (Рисунок 91):

```
* keychain 2.8.0 ~ http://www.funtoo.org
* Found existing ssh-agent: 3371

[aslebedev@tementy ~]$ hdfs dfs -cat output/*
(2000,34)
(2002,43)
(2004,6)
(2001,32)
(2003,7)
[aslebedev@tementy ~]$
```

Рисунок 91 — Максимальная годовая температура

Рассмотрим второй вариант решения этой задачи — с помощью API DataFrame и SparkSQL.

Объект DataFrame можно немедленно создать при чтении файла, при этом есть возможность автоматического вывода схемы, автоматической обработки заголовка, и задания разделителя полей:

```
val df = spark.read.options(Map("inferSchema"->"true", "delimiter"->"\t", "header"->"true")).csv("hdfs:///user/aslebedev/input/spark/age1.csv")
```

Просмотреть загруженное содержимое можно конструкцией:

```
df.show()
```

Для применения средств SparkSQL необходимо создать представление над DataFrame:

```
df.createGlobalTempView("age")
```

Можно удостовериться, что данные доступны в корректной форме (Рисунок 92):

```
spark.sql("SELECT * FROM global_temp.age" ).show()
```

The screenshot shows a terminal window titled "aslebedev : bash — Konsole <3>". The command "spark.sql("SELECT * FROM global_temp.age").show()" is run, resulting in the following table output:

number	month	year	temp	rank
1	1	2000	9999	2
2	2	2000	12	1
3	3	2000	15	4
4	4	2000	34	5
5	5	2000	3	9
6	1	2001	9999	0
7	2	2001	5	2
8	3	2001	32	1
9	4	2001	8	4
10	5	2001	9999	5
11	6	2001	0	5
12	7	2002	8	9
13	8	2002	43	0
14	9	2003	7	1
15	9	2004	6	4

Рисунок 92 — Выборка из представления age

Решение задачи полагается на функциональность SQL GROUP BY.

```
spark.sql("SELECT year, max(temp) FROM global_temp.age WHERE temp <> 9999 AND (rank = 0 OR rank = 1 OR rank = 4 OR rank = 5 OR rank = 9) GROUP BY year ORDER BY year").show()
```

Результат проиллюстрирован на рисунке (Рисунок 93).

The screenshot shows a terminal window titled "aslebedev : bash — Konsole <3>". The command "spark.sql("SELECT year, max(temp) FROM global_temp.age WHERE temp <> 9999 AND (rank = 0 OR rank = 1 OR rank = 4 OR rank = 5 OR rank = 9) GROUP BY year ORDER BY year").show()" is run, resulting in the following table output:

year	max(temp)
2000	34
2001	32
2002	43
2003	7
2004	6

Рисунок 93 — Среднегодовая температура

Анализ социальной активности на примере дампа twitter

Студентам предоставляется дамп twitter, приуроченный к политическим событиям в США. Схема данных определяется следующими полями (таблица 1):

Таблица 1 — Описание набора данных дампа twitter

№	Имя поля	Назначение поля
1	tweetid	идентификационный номер твита
2	userid	идентификационный номер пользователя (анонимный для пользователей, у которых на момент блокировки было менее 5000 подписчиков)
3	user_display_name	имя пользователя (кодируется как идентификатор пользователя для анонимных пользователей)
4	user_screen_name	дескриптор пользователя Twitter (кодируется как идентификатор пользователя для анонимных пользователей)
5	user_reported_location	самоотчетное местоположение пользователя
6	user_profile_description	описание профиля пользователя
7	user_profile_url	URL профиля пользователя
8	follower_count	количество аккаунтов, следящих за пользователем
9	following_count	количество учетных записей, за которыми следует пользователь
10	account_creation_date	дата создания учетной записи пользователя
11	account_language	язык учетной записи, выбранный пользователем
12	tweet_language	язык твита
13	tweet_text	текст твита (упоминания анонимных учетных записей заменены анонимными идентификаторами пользователя)
14	tweet_time	время публикации твита (UTC)

15	tweet_client_name	имя клиентского приложения, которое использовалось для публикации твита
16	in_reply_to_tweetid	идентификатор оригинального твита, на который этот твит отвечает (только для ответов)
17	in_reply_to_userid	ИД пользователя исходного твита, на который этот твит отвечает (только для ответов)
18	quoted_tweet_tweetid	идентификатор исходного твита, цитируемого в этом твите (только для цитат)
19	is_retweet	Верно / Неверно, этот твит ретвит
20	retweet_userid	для ретвитов идентификатор пользователя, создавший исходный твит
21	retweet_tweetid	для ретвитов — идентификатор оригинального твита.
22	latitude	географическая широта, если таковая имеется
23	longitude	географическая долгота, если доступно
24	quote_count	количество твитов, цитирующих этот твит
25	reply_count	количество твитов, ответивших на этот твит
26	like_count	количество лайков, полученных этим твитом
27	retweet_count	количество ретвитов, полученных этим твитом
28	hashtags	список хэштегов, используемых в этом твите
29	urls	список URL-адресов, используемых в этом твите
30	user_mentions	список идентификаторов пользователей, упомянутых в этом твите (включая анонимные идентификаторы пользователей)

31	poll_choices	если в твит был включен опрос, в этом поле отображаются варианты опроса, разделенные знаком « »
----	--------------	---

Найдем сообщение англоязычного пользователя, которое набрало больше всего ответов. Решим эту задачу с применением интерактивной оболочки pyspark и языка Python 3.

Запустите интерактивную оболочку следующей командой:

```
$ pyspark
```

Воспользуемся значениями полей reply_count и account_language для осуществления выборки.

Отметим, что не во всех записях поле reply_count содержит корректное числовое значение. Следовательно, необходимо провести предварительную очистку данных.

Напишите функцию безопасной конвертации строки в вещественное значение:

```
def float_convert(s):
    try:
        return float(s)
    except:
        # возвращаем нулевое значение в случае проблем конвертации
        return 0.0
```

Загрузим данные из текстового файла в RDD и пропустим заголовок:

```
text = sc.textFile('hdfs:///user/aslebedev/tweets/ira_tweets_csv_hashed.csv')
text1 = text.filter(lambda row: row.find('tweetid') != 0)
```

Напишите функцию, которая корректирует содержимое поля № 24 (reply_count, нумерация начинается с нуля), и оставляет только 30 полей:

```
def cleaner(s):
    list = s.split(',')
    list[24] = str(float_convert(list[24]))
    return list[0:30]
```

Теперь ее можно применить для построения очищенного RDD:

```
text2 = text1.map(cleaner)
```

Выполните фильтрацию по англоязычным пользователям (account_language, поле №10 при нумерации с нуля), а затем примените редукцию для поиска максимума по полю reply_count:

```
text3 = text2.filter(lambda row: row[10] == 'en')
answer_from_rdd = text3.reduce(lambda a, b: a if float(a[24]) > float(b[24]) else b)
print(answer_from_rdd)
```

Результат проиллюстрирован на рисунке (Рисунок 94).

```

aslebedev : bash — Konsole
File Edit View Bookmarks Settings Help
>>> text = sc.textFile('hdfs:///user/aslebedev/tweets/ira_tweets_csv_hashed.csv')
>>> text1 = text.filter(lambda row: row.find('tweetid') != 0)
>>> def cleaner(s):
...     list = s.split(',')
...     list[24] = str(float_convert(list[24]))
...     return list[0:30]
...
>>> text2 = text1.map(cleaner)
>>> text3 = text2.filter(lambda row: row[10] == 'en')
>>> answer_from_rdd = text3.reduce(lambda a, b: a if float(a[24]) > float(b[24]) else b)
>>> print(answer_from_rdd)
['"892887756296335360"', '4224729994', 'Tennessee', 'TEN GOP', '', 'Unofficial Twitter of Tennessee Republicans. Covering breaking news, national politics, foreign policy and more. #MAGA #2A', '', '147767', '74664', '2015-11-19', 'en', 'en', 'Please RT & help identify these scumbags who viciously beat kids in Quincy, MA. https://t.co/aj0ImTH7cF', '2017-08-02 23:20', 'Twitter Web Client', '', '', 'false', '', '', '', '2976', '3249.0', '7885', '17505', '[]', '[]', '']
>>> 

```

Рисунок 94 — Самый популярный твит

Аналогичный результат (Рисунок 95) можно получить, применив SparkSQL.

Создайте новый DataFrame на основе RDD:

```
newDF = spark.createDataFrame(text2)
```

Создайте новое представление и осуществите запрос:

```

newDF.createOrReplaceTempView("tw")
df = spark.sql("SELECT * from tw where _11 = 'en' order by cast(_25 as decimal(15,2)) desc limit 1")
df.show()

```

Здесь нумерация полей начинается с 1.

```

aslebedev : bash — Konsole
File Edit View Bookmarks Settings Help
>>> df.show()
+-----+-----+-----+-----+-----+-----+-----+
| 9| 10| 11| 12| 13| 14| 15| 16| 17|
+-----+-----+-----+-----+-----+-----+-----+
| 18| 19| 20| 21| 22| 23| 24| 25| 26| 27| 28| 29| 30|
+-----+-----+-----+-----+-----+-----+-----+
| 892887756296335360|4224729994|Tennessee|TEN GOP| Unofficial Twitte...| 147767|74
664|2015-11-19| en| en|Please RT & help identify these scumbags who viciously beat kids in Quincy, MA. https://t.co/aj0ImTH7cF| 2017-08-02 23:20|Twitter Web Client| | |
| false| | | | 2976|3249.0|7885|17505| []| []| |
+-----+-----+-----+-----+-----+-----+-----+
| 892887756296335360|4224729994|Tennessee|TEN GOP| Unofficial Twitte...| 147767|74
664|2015-11-19| en| en|Please RT & help identify these scumbags who viciously beat kids in Quincy, MA. https://t.co/aj0ImTH7cF| 2017-08-02 23:20|Twitter Web Client| | |
| false| | | | 2976|3249.0|7885|17505| []| []| |
+-----+-----+-----+-----+-----+-----+-----+
| 892887756296335360|4224729994|Tennessee|TEN GOP| Unofficial Twitte...| 147767|74
664|2015-11-19| en| en|Please RT & help identify these scumbags who viciously beat kids in Quincy, MA. https://t.co/aj0ImTH7cF| 2017-08-02 23:20|Twitter Web Client| | |
| false| | | | 2976|3249.0|7885|17505| []| []| |
+-----+-----+-----+-----+-----+-----+-----+
>>> 

```

Рисунок 95 — Самый популярный твит

Задание. Анализ социальной активности на примере дампа twitter

Загрузить набор данных из twitter, выбрать и решить один вариант работы:

1. Найти пользователя из РФ, чаще остальных упоминающего фамилии зарубежных политических деятелей (на русском).
2. Найти пользователя из РФ, чаще остальных упоминающего фамилии российских политических деятелей (на русском).
3. Найти пользователя, $n\%$ сообщений которого набирают минимум m ответов.
4. Найти пользователя, сообщения которого чаще заканчиваются беседу, чем начинают или находятся в середине.
5. Найти пользователя, который чаще остальных реагирует на упоминания его в сообщениях.
6. Найти пользователя, имеющего наибольшую скорость написания сообщений.

Литература

1. Ghemawat S., Gobioff H., Leung S. T. The Google file system //Proceedings of the nineteenth ACM symposium on Operating systems principles. — 2003. — С. 29-43.
2. Dean J., Ghemawat S. MapReduce: Simplified data processing on large clusters //Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation. — 2004. — С. 137-150.
3. White T. Hadoop: The definitive guide. — "O'Reilly Media, Inc.", 2015.
4. Уайт Т. Hadoop: подробное руководство //СПб.: Питер. — 2013. — 672 с.
5. Miner D., Shook A. MapReduce design patterns: building effective algorithms and analytics for Hadoop and other systems. — "O'Reilly Media, Inc.", 2012.
6. Lin J., Dyer C. Data-intensive text processing with MapReduce //Synthesis Lectures on Human Language Technologies. — 2010. — Т. 3. — №. 1. — С. 1-177.
7. Holmes A. Hadoop in practice. — New York: Manning, 2012. — Т. 3.
8. Apache Hadoop. <https://hadoop.apache.org/>
9. Apache Hive. <https://hive.apache.org/>
10. Apache Pig. <https://pig.apache.org/>
11. Apache Spark. <https://spark.apache.org/>
12. Redis. <https://redis.io/>
13. SMS Spam Collection Dataset. Collection of SMS messages tagged as spam or legitimate. <https://www.kaggle.com/uciml/sms-spam-collection-dataset>
14. Twitter Elections 2018. <http://tementy.servebeer.com/pub/twitter-elections-integrity-dataset-2018/>

Сведения об авторах

Магомедов Шамиль Гасангусейнович, к.т.н., заведующий кафедрой «Интеллектуальные системы информационной безопасности» Института комплексной безопасности и специального приборостроения Российского технологического университета (МИРЭА).

Лебедев Артем Сергеевич, преподаватель кафедры «Интеллектуальные системы информационной безопасности» Института комплексной безопасности и специального приборостроения Российского технологического университета (МИРЭА).

Приложение 1. Скрипты для управления сервисами Hadoop

hdpFormat.sh — форматирование файловой системы HDFS

```
#!/bin/bash
```

```
hdfs namenode -format
```

hdpStart.sh — запуск сервисов Hadoop

```
#!/bin/bash
```

```
$HADOOP_HOME/sbin/start-dfs.sh
```

```
hdfs dfs -mkdir /user  
hdfs dfs -mkdir /user/`id -nu`
```

```
$HADOOP_HOME/sbin/start-yarn.sh
```

```
$HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

hdpStop.sh — остановка сервисов Hadoop

```
#!/bin/bash
```

```
$HADOOP_HOME/sbin/mr-jobhistory-daemon.sh stop historyserver
```

```
$HADOOP_HOME/sbin/stop-yarn.sh
```

```
$HADOOP_HOME/sbin/stop-dfs.sh
```

hdpClean.sh — удаление всех временных файлов Hadoop и переформатирование HDFS. Можно использовать этот скрипт для восстановления после сбоев, если данные в HDFS не представляют ценности. Необходимо предварительно завершить все процессы Hadoop. Можно воспользоваться командой killall -9 java, если не запущено никаких других приложений Java, кроме сервисов Hadoop.

```
#!/bin/bash
```

```
rm -rf /tmp/Jetty* && \  
rm -rf /tmp/hadoop-`whoami` && \  
hdfs namenode -format
```

hdpCompileJava.sh — сборка простых приложений MapReduce на Java

```
#!/bin/bash
```

```
export HADOOP_JARS=$HADOOP_HOME/share/hadoop  
javac -cp $HADOOP_JARS/common/hadoop-common-2.9.2.jar:\$HADOOP_JARS/mapreduce/hadoop-mapreduce-client-core-2.9.2.jar $1  
bn=`basename $1 .java`  
jar cf $bn.jar $bn*.class
```

hiveMeta.sh — создание хранилища метаданных Hive (derby)

```
#!/bin/bash

username=`whoami` 

hdfs dfs -mkdir /tmp
hdfs dfs -mkdir /user/$username/warehouse
hdfs dfs -chmod g+w /tmp
hdfs dfs -chmod g+w /user/$username/warehouse

metastore_path=/tmp/hive-$username-metastore
rm -rf $metastore_path
mkdir $metastore_path
cd $metastore_path

schematool -dbType derby -initSchema
```

hiveStart.sh — запуск сервера Hive

```
#!/bin/bash

screen -dmS hiveserver2 bash -c 'cd /tmp/hive-`whoami`-metastore && hiveserver2
--hiveconf hive.root.logger=INFO,console'
```

Приложение 2. Исходный текст примера WordCount.java

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    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);
            }
        }
    }

    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);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setNumReduceTasks(2);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Приложение 3. Исходный текст примера TFIDF

TFIDF.java

```
package examples.mirea;

import org.apache.commons.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.CompressionCodecFactory;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.StringTokenizer;

public class TFIDF {

    // phase 1 - count words occurrences in documents (numerators in TF formulas)

    public static class TokenizerMapper extends Mapper<Object, Text, WordDocumentTuple, IntWritable> {
        private WordDocumentTuple wdt = new WordDocumentTuple();
        private final static IntWritable one = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // retrieve input file name, work only for single input, see
            // https://stackoverflow.com/questions/19012482/how-to-get-the-input-file-name-in-the-
            // mapper-in-a-hadoop-program/49502905#49502905
            String docname = ((FileSplit) context.getInputSplit()).getPath().getName();

            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                wdt.set(itr.nextToken(), docname);
                context.write(wdt, one);
            }
        }
    }

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

        public void reduce(WritableComparable 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);
        }
    }

    // phase 2 - count number of words in each document (denominators in TF formulas)

    public static class P2Mapper extends Mapper<Object, Text, Text, IntWritable> {
        private Text document = new Text();
        private IntWritable wordcount = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] line = value.toString().split("\t");
            
```

```

        document.set(line[1]);
        wordcount.set(Integer.parseInt(line[2]));
        context.write(document, wordcount);
    }
}

// phase 3 - count number of documents where each word occurs (denominators of IDF formulas)

// https://www.quora.com/How-can-I-open-and-read-a-text-file-stored-inside-HDFS-prior-to-
// launching-a-MapReduce-job-in-Java
public static ArrayList<String> readLines(Path location, Configuration conf) throws IOException {
    FileSystem fileSystem = FileSystem.get(location.toUri(), conf);
    CompressionCodecFactory factory = new CompressionCodecFactory(conf);
    FileStatus[] items = fileSystem.listStatus(location);
    ArrayList<String> results = new ArrayList<String>();
    if (items == null)
        return results;
    for (FileStatus item: items) {

        // ignoring files like _SUCCESS
        if(item.getPath().getName().startsWith("_"))
            continue;
        }

        CompressionCodec codec = factory.getCodec(item.getPath());
        InputStream stream = null;

        // check if we have a compression codec we need to use
        if (codec != null) {
            stream = codec.createInputStream(fileSystem.open(item.getPath()));
        }
        else {
            stream = fileSystem.open(item.getPath());
        }

        StringWriter writer = new StringWriter();
        IOUtils.copy(stream, writer, "UTF-8");
        String raw = writer.toString();
        String[] resulting = raw.split("\n");
        for(String str: raw.split("\n")) {
            results.add(str);
        }
    }
    return results;
}

public static class P3Mapper extends Mapper<Object, Text, Text, DocumentCounterTuple>{
    private DocumentCounterTuple dct = new DocumentCounterTuple();
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
        String[] line = value.toString().split("\t");
        word.set(line[0]);
        dct.set(line[1], Integer.parseInt(line[2]));
        context.write(word, dct);
    }
}

public static class P3Reducer extends Reducer<Text, DocumentCounterTuple, WordDocumentTuple,
DoubleWritable> {
    private WordDocumentTuple wdt = new WordDocumentTuple();
    private DoubleWritable tfidf = new DoubleWritable();

    public HashMap<String, Integer> docLengths = new HashMap<String, Integer>();

    public void setup(Context context) throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        for (String line : readLines(new Path(conf.get("docLengths", "<invalid>")), conf)) {
            String[] dl = line.split("\t");
            docLengths.put(dl[0], Integer.parseInt(dl[1]));
        }
    }

    public void reduce(Text key, Iterable<DocumentCounterTuple> values, Context context) throws
    IOException, InterruptedException {
        // count number of documents where key word occurs (denominator of IDF formula)
        ArrayList<DocumentCounterTuple> dcta = new ArrayList<DocumentCounterTuple>();
        int cnt = 0;
        for (DocumentCounterTuple val : values) {

```

```

        cnt++;
        // while iterating through "values" expect "val" referring to the same object but with
        different field values
        // on different iterations, so the only way to save information is to clone it,
        dcta.add(val) will not work
        dcta.add(new DocumentCounterTuple(val.document, val.counter));
    }
    // count TFIDF
    for (DocumentCounterTuple val : dcta) {
        wdt.set(key.toString(), val.document);
        double tf = ((double) val.counter) / docLengths.get(val.document);
        double idf = ((double) docLengths.size()) / cnt;
        tfidf.set(tf * idf);
        context.write(wdt, tfidf);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    // phase 1

    String job1Name = "tfidf-phase1-tf-numerators";
    String inputPath = args[0];
    String outputPath = args[1] + "/" + job1Name;

    Job job = Job.getInstance(conf, job1Name);
    job.setJarByClass(TFIDF.class);

    job.setMapperClass(TokenizerMapper.class);
    job.setMapOutputKeyClass(WordDocumentTuple.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(WordDocumentTuple.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(inputPath));
    FileOutputFormat.setOutputPath(job, new Path(outputPath));

    if (!job.waitForCompletion(true))
        System.exit(1);

    // phase 2

    String job2Name = "tfidf-phase2-tf-denominators";
    inputPath = args[1] + "/" + job1Name;
    outputPath = args[1] + "/" + job2Name;

    job = Job.getInstance(conf, job2Name);
    job.setJarByClass(TFIDF.class);

    job.setMapperClass(P2Mapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(inputPath));
    FileOutputFormat.setOutputPath(job, new Path(outputPath));

    if (!job.waitForCompletion(true))
        System.exit(2);

    // phase 3

    conf.set("docLengths", outputPath);

    String job3Name = "tfidf-phase3-tfidf";
    inputPath = args[1] + "/" + job1Name;
    outputPath = args[1] + "/" + job3Name;

    job = Job.getInstance(conf, job3Name);

```

```

        job.setJarByClass(TFIDF.class);

        job.setMapperClass(P3Mapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(DocumentCounterTuple.class);

        job.setReducerClass(P3Reducer.class);
        job.setOutputKeyClass(WordDocumentTuple.class);
        job.setOutputValueClass(DoubleWritable.class);

        FileInputFormat.setInputPaths(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

        if (!job.waitForCompletion(true))
            System.exit(3);

        System.exit(0);
    }
}

```

DocumentCounterTuple.java

```

package examples.mirea;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class DocumentCounterTuple implements WritableComparable<DocumentCounterTuple> {
    public String document;
    public int counter;

    public DocumentCounterTuple() {
    }

    public DocumentCounterTuple(String document, int counter) {
        set(document, counter);
    }

    public void set(String document, int counter) {
        this.document = document;
        this.counter = counter;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        document = in.readUTF();
        counter = in.readInt();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(document);
        out.writeInt(counter);
    }

    @Override
    public int compareTo(DocumentCounterTuple o) {
        int cmpdoc = this.document.compareTo(o.document);
        return (cmpdoc != 0) ? cmpdoc : this.counter - o.counter;
    }

    @Override
    public String toString() {
        return String.format("%s\t%d", document, counter);
    }
}

```

WordDocumentTuple.java

```

package examples.mirea;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;

```

```

import java.io.DataOutput;
import java.io.IOException;

public class WordDocumentTuple implements WritableComparable<WordDocumentTuple> {
    public String word;
    public String document;

    public WordDocumentTuple() {
    }

    public WordDocumentTuple(String word, String document) {
        set(word, document);
    }

    public void set(String word, String document) {
        this.word = word;
        this.document = document;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        word = in.readUTF();
        document = in.readUTF();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(word);
        out.writeUTF(document);
    }

    @Override
    public int compareTo(WordDocumentTuple o) {
        int cmpdoc = this.document.compareTo(o.document);
        return (cmpdoc != 0) ? cmpdoc : this.word.compareTo(o.word);
    }

    @Override
    public String toString() {
        return String.format("%s\t%s", word, document);
    }
}

```

Приложение 4. Тестовые примеры для Pig

input1

```
111,Jogh,Sales,Austin  
222,Alex,Marketing,New York  
333,Philip,Operation,Sacramento  
444,Terry,Sales,New York  
555,Jessi,Development,Boston
```

cars.txt

```
1970,Dodge Challenger,22000,74598  
1976,Ford Shellby,45500,23000  
1970,Cadillac Eldarado,36000,14023  
1970,Caterham 7,55000,1478  
1976,Chevrolet Camaro,27000,69963  
1976,Chevrolet Panamera,270000,60963
```

logs.txt

```
07/Dec/2013:20:04:10,95.153.193.56,37877,http,404,1492,0,0,GET /745dbda3-  
894e-43aa-9146-607f19fe4428.mp3 HTTP/1.1  
07/Dec/2013:20:04:33,95.153.193.56,37877,http,206,2048,0,0,GET  
/login.html HTTP/1.1  
07/Dec/2013:20:05:13,95.153.193.56,37877,http,200,1492030,0,0,GET  
/745dbda3-894e-43aa-9146-607f19fe4428.mp3 HTTP/1.1  
07/Dec/2013:20:06:44,95.153.193.56,37877,http,200,1492030,0,0,GET  
/745dbda3-894e-43aa-9146-607f19fe4428.mp3 HTTP/1.1  
08/Dec/2013:15:00:28,178.88.91.180,13600,http,200,4798,0,0,GET  
/public/cars/bmw71/down.png HTTP/1.1  
08/Dec/2013:15:00:28,188.88.66.183,13644,http,200,4798,0,0,GET  
/public/cars/bmw71/down.png HTTP/1.1  
08/Dec/2013:15:00:29,193.110.115.45,64318,http,200,1594,0,0,GET  
/K1/img/top-nav-bg-default.jpg HTTP/1.1
```

logs.pig

```
records = LOAD 'inputLogs' USING PigStorage(',') AS  
(date:chararray,clientip:chararray,clientport:chararray,proto:chararray,statusco  
de:int,bytes:int,sq:chararray,bq:chararray,request:chararray );  
  
count_total = FOREACH (GROUP records ALL) GENERATE COUNT(records);  
  
count_ip = FOREACH (GROUP records BY clientip) GENERATE group AS ip,  
COUNT(records) AS cnt;  
top_ip = ORDER count_ip BY cnt DESC;  
  
filtered_req = FILTER records BY statuscode == 200 OR statuscode == 206;  
count_req = FOREACH (GROUP filtered_req BY request) GENERATE group AS req,  
COUNT(filtered_req) AS cnt, SUM(filtered_req.bytes) AS bytes;  
top_req = ORDER count_req BY bytes DESC;  
  
%declare DT `date +%y%m%dT%H%M`  
STORE count_total INTO 'pigresult/$DT/count_total';  
STORE top_ip INTO 'pigresult/$DT/top_ip';  
STORE top_req INTO 'pigresult/$DT/top_req';
```

Приложение 5. Тестовый пример для Spark

tweets.csv

"tweetid","userid","user_display_name","user_screen_name","user_reported_location","user_profile_description","user_profile_url","follower_count","following_count","account_creation_date","account_language","tweet_language","twe et_text","tweet_time","tweet_client_name","in_reply_to_tweetid","in_reply_to_userid","quoted_tweet_tweetid","is_ret weet","retweet_userid","retweet_tweetid","latitude","longitude","quote_count","reply_count","like_count","retweet_co unt","hashtags","urls","user_mentions","poll_choices"
"87791995476496385","249064136b1c5cb00a705316ab73dd9b53785748ab757f02df7e7a9876906139","249064136b1c5cb00a705316ab73dd9b53785748ab757f02df7e7a9876906139","249064136b1c5cb00a705316ab73dd9b53785748ab757f02df7e7a9876906139","Москва, Россия","Я примерный семьянин!","","","132","120","2013-12-07","ru","ru","RT @guoprentwit: ⚡ У НАС НОВОЕ ВИДЕО! Американец: ""Если бы не 27 миллионов русских, я бы сейчас говорил по-немецки"" <https://t.co/mAccirn4o1...>","2017-06-22
16:03","TweetDeck","","","","true","2572896396","8779172119416832","","","","0","0","0","0","0","[]","[http://ru-open.livejournal.com/374284.html]","[2572896396]",""
"492388766930444288","0974d5dbe4ca9bd6c3b46d62a5cbdbd5c0d86e196b624dbfc7d18cf17b3eab5","0974d5dbe4ca9bd6c3b46d62a5cbdbd5c0d86e196b624dbfc7d18cf17b3eab5","0974d5dbe4ca9bd6c3b46d62a5cbdbd5c0d86e196b624dbfc7d18cf17b3eab5","Россия","Телефонист.Изучение истории Игра в любительском театре - Воздушные змеи","","","74","8","2014-03-15","en","ru","Серебром отколоколило <http://t.co/Jaa4v4IFPm>","2014-07-24
19:20","generationπ","","","","false","","","","0","0","0","0","","[http://pyupilg33.livejournal.com/11069.html]","","
"719455077589721089","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","Рязань","волны так и плещут фиолетовой волной","","","165","454","2014-04-29","en","bg","@kpru С-300 в Иране <https://t.co/eln3qLUW7>","2016-04-11
09:20","TweetDeck","719439882993545216","40807205","","false","","","","0","0","0","0","0","[]","[https://www.youtube.com/watch?v=9GvpImWxTJc]","[40807205]",""
"536179342423105537","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","bda40f262856eee77c48a332e5eb23bc4f1943d600867d4194d89b1235b17ee0","Рязань","волны так и плещут фиолетовой волной","","","165","454","2014-04-29","en","ru","Предлагаю судить их за поддержку нацизма, т.к. они отказались его осуждать!!
#STOPNazi","2014-11-22 15:28","Twitter Web Client","","","","false","","","","0","0","0","0","[STOPNazi]","","","","
"841410788409630720","a53ed619f1dea6015c7c878bf744b0eefe8f7272dccb340e69363f596ad9551e","a53ed619f1dea6015c7c878bf744b0eefe8f7272dccb340e69363f596ad9551e","","Отвечаю на любой #ВопросПрезиденту","<http://t.co/3CVqbMQFb>","4430","4413","2012-02-25","ru","bg","Предостережение американского дипломата <https://t.co/fKPBVgIoVc>","2017-03-13
22:08","Twitter Web Client","","","","false","","","","0","2","3","4","[]","[https://goo.gl/fBp94X]","","
"834365760776630272","a53ed619f1dea6015c7c878bf744b0eefe8f7272dccb340e69363f596ad9551e","a53ed619f1dea6015c7c878bf744b0eefe8f7272dccb340e69363f596ad9551e","","Отвечаю на любой #ВопросПрезиденту","<http://t.co/3CVqbMQFb>","4430","4413","2012-02-25","ru","ru","Двойная утопия, или Нет у Европы Трампа, кроме Путина <https://t.co/MbxCpuLdDI>","2017-02-22 11:34","Twitter Web Client","","","","false","","","","0","1","3","5","[]","[https://goo.gl/9w5hs0]","","
"577490527299457024","95b3aba6b9140f5dda993148de174ff57d62f4a6e68e886a1927a130b328e14f","95b3aba6b9140f5dda993148de174ff57d62f4a6e68e886a1927a130b328e14f","95b3aba6b9140f5dda993148de174ff57d62f4a6e68e886a1927a130b328e14f","мой мир","my story - TV nerd","","","296","492","2013-12-20","ru","ru","RT @harkovnews: На Харьковщине задержали фуру с тоннами поддельной водки <http://t.co/4FqBwRaaED>
<http://t.co/3NspkvGao1>","2015-03-16
15:24","slovoslav","","","","true","2599775719","577485402124111872","","","","0","0","0","0","[]","[http://nahnews.co m.ua/180774-na-xarkovshhine-zaderzhali-furu-s-tonnami-poddelenoj-vodki/][2599775719]","","
"596522755379560448","5744c546bdf9e81ea0aad223c9db4b702ccba7c81d4c11dc775963f7bcb63ea7","5744c546bdf9e81ea0aad223c9db4b702ccba7c81d4c11dc775963f7bcb63ea7","5744c546bdf9e81ea0aad223c9db4b702ccba7c81d4c11dc775963f7bcb63ea7","Казань","пусть за меня говорят мои твитты","","","375","2067","2014-03-22","ru","ru","RT @NovostiNsk: «Мы все равны в своем праве на город» <http://t.co/SQpuP45mDv>","2015-05-08
03:51","meceslav","","","","true","2518710111","596514222684524545","","","","0","0","0","0","[]","[http://bit.ly/1Rizso9][2518710111]","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-06","en","en","As sun and cloud give way to moon and shadow, the rhythm of the world makes itself apparent, saying, have faith in yourself, you are this.",,"2015-02-16 16:19","Twitter Web Client","","","","false","","","","0","0","0","0","","","","","","
"567357519547207680","2b0d7525bed1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","2b0d7525be1df5119b7956f9be4888b45686172d68006a90634e87952eb45d3","California","Troublemaker","","","696","863","2013-08-