

Pig и Hive



Pig

Что такое Pig?

- Pig – высокоуровневая платформа поверх Hadoop
 - Язык программирования высокого уровня *Pig Latin*
 - Код программы преобразуется в MapReduce задачи
- Разработан в Yahoo! в 2006 году
- Top Level Apache Project
 - <http://pig.apache.org>

Для чего нужен Pig?

- Для написания задач MapReduce требуются программисты
 - Которые должны уметь думать в стиле “map & reduce”
 - Скорее всего должны знать язык Java
- Pig предоставляет язык, который могут использовать:
 - Аналитики
 - Data Scientists
 - Статистики

```

public class WordCountJob extends Configured implements Tool{
    static public class WordCountMapper
        extends Mapper<LongWritable, Text, Text, IntWritable> {

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

        @Override
        protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
            StringTokenizer tokenizer = new StringTokenizer(value.toString());
            while (tokenizer.hasMoreTokens()) {

text.set(tokenizer.nextToken());

                context.write(text, one);
            }
        }
        static public class WordCountReducer
            extends Reducer<Text, IntWritable, Text, IntWritable> {

            @Override
            protected void reduce(Text key, Iterable<IntWritable> values,
                Context context)
                throws IOException, InterruptedException {
                int sum = 0;
                for (IntWritable value : values) {
                    sum += value.get();
                }
                context.write(key, new IntWritable(sum));
            }
        }
        @Override
        public int run(String[] args) throws Exception {
            Job job = Job.getInstance(getConf(), "WordCount");
            job.setJarByClass(getClass());

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

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

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

            return job.waitForCompletion(true) ? 0 : 1;
        }
        public static void main(String[] args) throws Exception {
            int exitCode = ToolRunner.run(
                new WordCountJob(), args);
            System.exit(exitCode);
        }
    }
}

```

Java: ~50 строк

```

#!/usr/bin/python
import sys

```

```

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

```

```

#!/usr/bin/python
import sys

```

Python: 17 строк

```

(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)

```

Pig: 5 строк

1. input_lines = LOAD 'copy-of-all-pages-on-internet' AS (line:chararray);
2. words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
3. word_groups = GROUP words BY word;
4. word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
5. STORE word_count INTO 'number-of-words-on-internet';

Основные возможности Pig

- Join Datasets
- Sort Datasets
- Filter
- Data Types
- Group By
- Пользовательские функции

Компоненты Pig

- Pig Latin
- Компилятор Pig
- Среда выполнения

Режимы выполнения

- Local
 - Запускается в рамках одной JVM
 - Работает исключительно с локальной файловой системой
 - **`$pig -x local`**
- Hadoop (MapReduce)
 - Pig преобразует программу Pig Latin в задачи MapReduce и выполняет их на кластере
 - **`$pig -x mapreduce`**

Запуск Pig

- Скрипт
 - Выполняются команды из файла
 - `$pig script.pig`
- Grunt
 - Интерактивная оболочка для выполнения команд Pig
 - Можно запускать скрипты из *Grunt* командой *run* или *exec*
- Embedded
 - Можно выполнять команды Pig, используя класс *PigServer*
 - Имеется программный доступ к Grunt через класс *PigRunner*

Pig Latin

Строительные блоки

- *Field* (поле) – часть данных
- *Tuple* (кортеж) – упорядоченный набор полей, заключенный в скобки “(” и “)”

```
(10.4, 5, word, 4, field1)
```

- *Bag* (мешок) – коллекция кортежей, заключенная в скобки “{” и “}”

```
{ (10.4, 5, word, 4, field1), (this, 1, blah) }
```

Pig Latin

Схожесть с реляционными БД:

- *Bag* – это таблица в БД
- *Tuple* – это строка в таблице
- Но: Bag не требует, чтобы все tuples содержали одно и то же число полей

```
$ pig
```

```
grunt> cat /path/to/file/a.txt
```

```
a          1
```

```
d          4
```

```
c          9
```

```
k          6
```

```
grunt> records = LOAD '/path/to/file/a.txt' as (letter:chararray, count:int);
```

```
grunt> DUMP records;
```

```
...
```

```
org.apache.pig.backend.hadoop.executionengine.MapReduceLauncher - 50% complete
```

```
org.apache.pig.backend.hadoop.executionengine.MapReduceLauncher - 100% complete
```

```
...
```

```
(a,1)
```

```
(d,4)
```

```
(c,9)
```

```
(k,6)
```

Операции DUMP и STORE

DUMP – выводит результат на экран

STORE – сохраняет результат (обычно в файл)

```
grunt> records = LOAD '/path/to/file/a.txt' as (letter:chararray, count:int);
```

```
...
```

```
...
```

```
...
```

```
...
```

```
grunt> DUMP records;
```



Ничего не выполняется,
только оптимизация скрипта

Большой объем данных

Можно ограничить объем выводимых данных

```
grunt> records = LOAD '/path/to/file/big.txt' as (letter:chararray, count:int);  
grunt> toPrint = LIMIT records 5;  
grunt> DUMP toPrint;
```



Показывать только 5 записей

```
LOAD 'data' [USING function] [AS schema];
```

data – имя директории или файла

USING – определяет функцию для загрузки

AS – назначает схему входным данным

```
records =
```

```
    LOAD '/path/to/file/some.log'
```

```
    USING PigStorage(';')
```

```
    AS (userId:chararray, timestamp:long, query:chararray);
```

Тип	Описание	Пример
Простые		
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L или 10l
float	32-bit floating point	10.5F или 10.5f
double	64-bit floating point	10.5 или 10.5e2 или 10.5E2
Массивы		
chararray	Массив символов (string) в Unicode UTF-8	hello world
bytearray	Byte array (blob)	
Комплексные		
tuple	ordered set of fields	(19,2)
bag	collection of tuples	{(19,2), (18,1)}
map	set of key value pairs	[open#apache]

Pig Latin: средства диагностики

- Отобразить структуру Bag
 - *grunt> DESCRIBE <bag_name>;*
- Отобразить план выполнения (*Execution Plan*)
 - *grunt> EXPLAIN <bag_name>;*
- Показать, как Pig преобразует данные
 - *grunt> ILLUSTRATE <bag_name>;*

```
grunt> chars = LOAD '/path/to/file/b.txt' AS (c:chararray);
```

```
grunt> DESCRIBE chars;
```

```
chars: {c: chararray}
```

```
grunt> DUMP chars;
```

```
(a)
```


```
(k)
```

```
...
```

```
(c)
```

```
(k)
```

Создать новый bag с
полями *group* и *chars*



```
grunt> charGroup = GROUP chars by c;
```

```
grunt> DESCRIBE charGroup;
```

```
charGroup: {group: chararray, chars: {(c: chararray)}}
```

```
grunt> dump charGroup;
```

```
(a, {(a), (a), (a)})
```


```
(c, {(c), (c)})
```

```
(i, {(i), (i), (i)})
```

```
(k, {(k), (k), (k), (k)})
```

```
(l, {(l), (l)})
```

'chars' – это bag, который содержит
все tuples из bag 'chars', которые
матчат значение из 'c'



```
grunt> chars = LOAD '/path/to/file/b.txt' AS (c:chararray);
grunt> charGroup = GROUP chars by c;
grunt> ILLUSTRATE charGroup;
```

| chars | c:chararray |

		c	
		c	

```
| charGroup | group:chararray | chars:bag{tuple(c:chararray)} |
```

	c	$\{(c), (c)\}$
--	---	----------------

```
grunt> chars = LOAD '/path/to/file/b.txt' AS (c:chararray);
```

```
grunt> charGroup = GROUP chars by c;
```

```
grunt> dump charGroup;
```

```
(a, { (a), (a), (a) })
```

```
(c, { (c), (c) })
```

```
(i, { (i), (i), (i) })
```

```
(k, { (k), (k), (k), (k) })
```

```
(l, { (l), (l) })
```

FOREACH <bag> GENERATE <data>

```
grunt> records = LOAD '/path/to/file/a.txt' AS (c:chararray, i:int);
```

```
grunt> DUMP records;
```

```
(a, 1)
```

```
(d, 4)
```

```
(c, 9)
```

```
(k, 6)
```

```
grunt> counts = FOREACH records GENERATE i;
```


```
grunt> DUMP counts;
```

```
(1)
```

```
(4)
```

```
(9)
```

```
(6)
```




Для каждой строки
вывести поле 'i'

PigLatin: FOREACH с функцией

FOREACH B GENERATE group, **FUNCTION**(A);

- Встроенные функции Pig:
 - *COUNT, FLATTEN, CONCAT* и т.д.
- Можно реализовать свою функцию
 - Java, Python, JavaScript, Ruby или Groovy

```
grunt> chars = LOAD 'data/b.txt' AS (c:chararray);
grunt> charGroup = GROUP chars BY c;
grunt> DUMP charGroup;
(a, {(a),(a),(a)})
(c, {(c),(c)})
(i, {(i),(i),(i)})
(k, {(k),(k),(k),(k)})
(l, {(l),(l)})
grunt> DESCRIBE charGroup;
charGroup: {group: chararray, chars: {(c: chararray)}}
grunt> counts = FOREACH charGroup GENERATE group, COUNT(chars);
grunt> DUMP counts;
(a, 3)
(c, 2)
(i, 3)
(k, 4)
(l, 2)
```



Для каждой строки в '*charGroup*'
вывести поле '*group*' и число
элементов в '*chars*'

PigLatin: функция TOKENIZE

```
tokenBag = FOREACH Text GENERATE TOKENIZE(line);
```

- Разбиение строки на токены
- Результат в виде bag из токенов
- Разделители:
 - пробел
 - кавычки: “
 - запятая: ,
 - скобки: ()
 - звездочка: *


```
grunt> linesOfText = LOAD 'data/c.txt' AS (line:chararray);
grunt> DUMP linesOfText;
(this is a line of text)
(yet another line of text)
(third line of words)
```

```
grunt> tokenBag = FOREACH linesOfText GENERATE TOKENIZE(line);
```

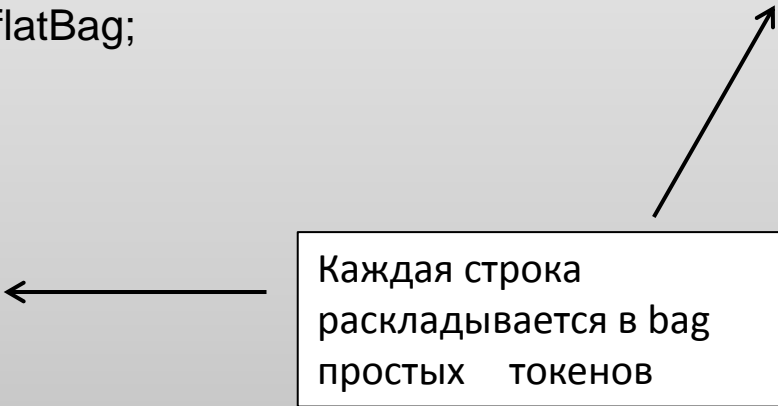
```
grunt> DUMP tokenBag;
({ (this), (is), (a), (line), (of), (text) })
({ (yet), (another), (line), (of), (text) })
({ (third), (line), (of), (words) })
```



Разбить каждую строку по пробелам и вернуть *bag of tokens*

```
grunt> DESCRIBE tokenBag;
tokenBag: {bag_of_tokenTuples: {tuple_of_tokens: (token:
chararray) }}
```

```
grunt> DUMP tokenBag;
({(this), (is), (a), (line), (of), (text))}
({(yet), (another), (line), (of), (text))}
({(third), (line), (of), (words))}
grunt> flatBag = FOREACH tokenBag GENERATE flatten($0);
grunt> DUMP flatBag;
(this)
(is)
(a)
...
...
(text)
(third)
(line)
(of)
(words)
```



Каждая строка
раскладывается в bag
простых токенов

PigLatin: WordCount

```
input_lines = LOAD 'file-with-text' AS (line:chararray);

words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

filtered_words = FILTER words BY word MATCHES '\\w+';

word_groups = GROUP filtered_words BY word;

word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count,
group AS word;

ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO 'number-of-words-file';
```

PigLatin: Joins

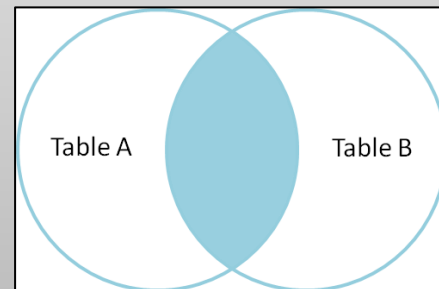
- Pig поддерживает
 - Inner Joins (по-умолчанию)
 - Outer Joins
 - Full Joins
- Фазы join
 - Загрузить записи в bag из input #1
 - Загрузить записи в bag из input #2
 - Сделать join для двух массивов данных (bags) по заданному ключу

Inner Join, пример

```
-- InnerJoin.pig
-- Загрузить записи в bag #1
posts = LOAD 'data/user-posts.txt' USING PigStorage(',') AS
(user:chararray,post:chararray,date:long);

-- Загрузить записи в bag #2
likes = LOAD 'data/user-likes.txt' USING PigStorage(',') AS
(user:chararray,likes:int,date:long);

userInfo = JOIN posts BY user, likes BY user;
DUMP userInfo;
```



```
$ hdfs dfs -cat data/user-posts.txt
```

```
user1,Funny Story,1343182026191  
user2,Cool Deal,1343182133839  
user4,Interesting Post,1343182154633  
user5,Yet Another Blog,13431839394
```

```
$ hdfs dfs -cat data/user-likes.txt
```

```
user1,12,1343182026191  
user2,7,1343182139394  
user3,0,1343182154633  
user4,50,1343182147364
```

```
$ pig InnerJoin.pig
```

```
(user1,Funny Story,1343182026191,user1,12,1343182026191)  
(user2,Cool Deal,1343182133839,user2,7,1343182139394)  
(user4,Interesting Post,1343182154633,user4,50,1343182147364)
```

Inner Join, пример

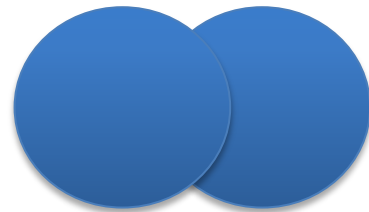
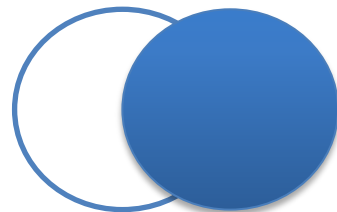
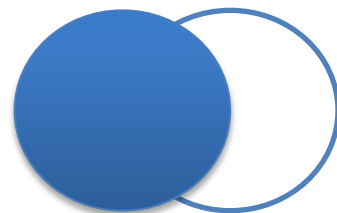
```
grunt> DESCRIBE posts;  
posts: {user: chararray,post: chararray,date: long}
```

```
grunt> DESCRIBE likes;  
likes: {user: chararray,likes: int,date: long}
```

```
grunt> DESCRIBE userInfo;  
UserInfo: {  
    posts::user: chararray,  
    posts::post: chararray,  
    posts::date: long,  
    likes::user: chararray,  
    likes::likes: int,  
    likes::date: long}
```

PigLatin: Outer Join

- JOIN <bag #1> BY <join field>
LEFT OUTER, <bag #2> **BY** <join field>;
- JOIN <bag #1> BY <join field>
RIGHT OUTER, <bag #2> **BY** <join field>;
- JOIN <bag #1> BY <join field>
FULL OUTER, <bag #2> **BY** <join field>;

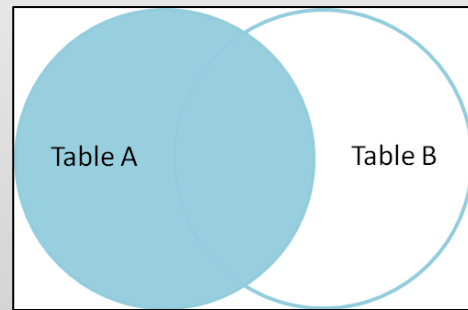


Left Outer Join, пример

```
--LeftOuterJoin.pig
posts = LOAD 'data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);

likes = LOAD 'data/user-likes.txt'
        USING PigStorage(',')
        AS (user:chararray,likes:int,date:long);

userInfo = JOIN posts BY user LEFT OUTER, likes BY user;
DUMP userInfo;
```



```
$ hdfs dfs -cat data/user-posts.txt  
user1,Funny Story,1343182026191  
user2,Cool Deal,1343182133839  
user4,Interesting Post,1343182154633  
user5,Yet Another Blog,13431839394
```

```
$ hdfs dfs -cat data/user-likes.txt  
user1,12,1343182026191  
user2,7,1343182139394  
user3,0,1343182154633  
user4,50,1343182147364
```

```
$ pig LeftOuterJoin.pig  
(user1,Funny Story,1343182026191,user1,12,1343182026191)  
(user2,Cool Deal,1343182133839,user2,7,1343182139394)  
(user4,Interesting Post,1343182154633,user4,50,1343182147364)  
(user5,Yet Another Blog,13431839394,,)
```