Интеллектуальные информационные системы

Apache Pig

Кафедра управления и интеллектуальных технологий НИУ «МЭИ»

2023 г.

Apache Pig

Pig - высокоуровневая платформа для создания программ, работающих на Apache Hadoop.

Позволяет реализовывать MapReduce программы без глубоких навыков программирования и понимания парадигмы MapReduce.

Компоненты:

- Pig Latin
- Компилятор
- Среда выполнения

Apache Pig

https://pig.apache.org/

Word Count

```
import sys
for line in sys.stdin:
   line = line.strip()
   words = line.split()
   for word in words:
      print (word, 1, sep = '\t')
```

```
input_lines = LOAD 'file.txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
word_groups = GROUP words BY word;
word_count = FOREACH word_groups GENERATE COUNT(words) AS count, group AS word;
STORE word_count INTO 'count_words';
```

Оптимизация кода. Выполнения только в на последней строке.

```
import sys
(last_key, count) = (None, 0)

for line in sys.stdin:
        (key, value) =  line.strip().split("\t")
        if last_key != key and last_key:
                print (last_key, str(count), sep="\t")
                (last_key, count) = (key, int(value))
        else:
                (last_key, count) = (key, count +  int(value))

if last_key:
        print (last_key, str(count), sep="\t")
```

Основные возможности

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

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 программы

Скрипт из файла
$pig script.pig

Grunt – Интерактивная оболочка для выполнения команд Pig

Embedded – из кода на Java

Pig Latin

Pig – типизированный язык.
Единица данных – **field (поле)** – содержит единицу данных – число, строку,…
Tuple (кортеж) – упорядоченный набор полей. Оформляется в круглые скобки:

(123, word, 2.34, field2)

Bag (мешок) - коллекция кортежей. Оформляется в фигурные скобки:

{ (123, word, 2.34, field2), (Pig, 3.14, foo) }

Аналогия с реляционными БД:
Bag – таблица
Tuple – строка в таблице
Отличие от реляционных БД – не обязательно совпадение полей

Pig Latin

```
$pig –x local
grunt> cat file.txt;
grunt> records = LOAD 'file2.txt' AS (site:chararray, time:int);
grunt> limit_records = LIMIT records 5
grunt> DUMP limit_records
…
SUCCESS
…
(yandex.ru,10)
(vk.com,120)
(mail.ru,45)
(yandex.ru,25)
(yandex.ru,30)
```

DUMP и STORE

DUMP – выводит результат на экран.
STORE – сохраняет результат в файл.

Реального выполнения MapReduce задач не происходит, пока не дойдем до этих операторов – отложенный запуск.

Только оптимизация скрипта, отслеживание, что операторы записаны корректно.

LOAD

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);
```

Типы данных

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]

Сколько MapReduce задач запустится?

1. posts = LOAD 'data/user-posts.txt' AS user:chararray,post:chararray,date:long);

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

3. toPrintPosts = LIMIT posts 100;

4. DUMP toPrintPosts;

5. toSaveLikes = LIMIT likes 50;

6. STORE toSaveLikes INTO 'likes.txt'

Сколько MapReduce задач запустится?

```
grunt> chars = LOAD '/path/to/file/b.txt' AS (c:chararray);
grunt> DESCRIBE chars;
chars: {c: chararray}
grunt> DUMP chars;
(a)
(k)
...
(c)
(k)
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)})
```

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

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

Group

grunt> chars = LOAD 'file3.txt' AS (c:chararray);
grunt> DESCRIBE chars
chars: {c: chararray}

grunt> charGroup = GROUP chars BY c;
grunt> DESCRIBE charGroup
charGroup: {group: chararray,chars: {(c: chararray)}}

grunt> DUMP charGroup
(a,{(a),(a),(a),(a)})
(b,{(b),(b),(b),(b)})
(c,{(c),(c),(c)})
(d,{(d),(d)})

Illustrate

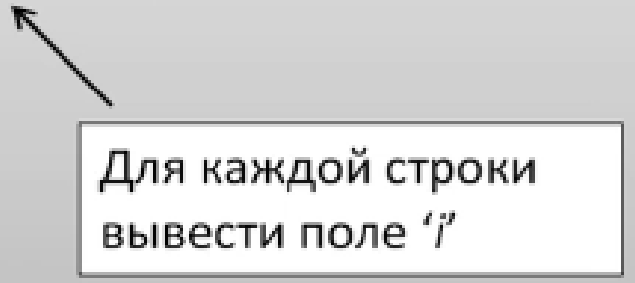
```
grunt> ILLUSTRATE charGroup;

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

FOREACH

FOREACH <bag> GENERATE <data_or_function>

```
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)
```



FOREACH

FOREACH <bag> GENERATE <data_or_function>

- **COUNT** – подсчет повторений данных
- **FLATTEN** – «разворачивание» bag в набор tuple
- **TOKENIZE** – разбиение на слова
- **CONCAT** – конкатенация

И др.

FOREACH

```
grunt> chars = LOAD 'file3.txt' AS (c:chararray);
grunt> charGroup = GROUP chars BY c;
grunt> DUMP charGroup
grunt> counts = FOREACH charGroup  GENERATE  group, COUNT(chars);
grunt> DUMP counts
(a,4)
(b,4)
(c,3)
(d,2)
```

TOKENIZE

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

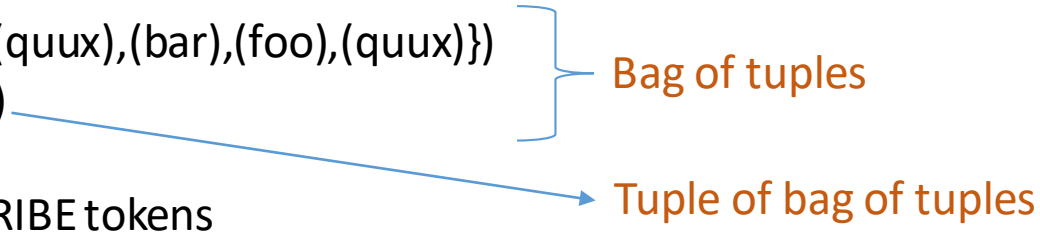
- Разбиение строки на токены
- Результат в виде bag из токенов
- Разделители:
  - пробел
  - кавычки: “
  - запятая: ,
  - скобки: ()
  - звездочка: *

TOKENIZE, FLATTEN

```
grunt> textLines = LOAD 'file.txt' AS (line:chararray);
grunt> DUMP textLines
(foo foo quux bar foo quux)
(foo bar)

grunt> tokens = FOREACH textLines GENERATE TOKENIZE (line)
grunt> DUMP tokens
({(foo),(foo),(quux),(bar),(foo),(quux)})
({(foo),(bar)})

grunt> DESCRIBE tokens
tokens: {bag_of_tokenTuples_from_line: {tuple_of_tokens: (token: chararray)}}
```



TOKENIZE, FLATTEN

grunt> tokens = FOREACH textLines GENERATE FLATTEN(TOKENIZE(line));

(foo)
(foo)
(quux)
(bar)
(foo)
(quux)
(foo)
(bar)

Bag of tuples of fields

grunt> DESCRIBE tokens

tokens: {bag_of_tokenTuples_from_line::token: chararray}