ТЕХНОСФЕРА

Лекция №7

Введение в Pig и Hive

Евгений Чернов

Pig

Pig

Платформа для анализа больших коллекций данных, которая состоит из:

- Языка высокого уровня для написания программ анализа
- Инфраструктуры для запуска этих программ

Pig

- Top Level Apache Project: *http://pig.apache.org*
- Pig – это высокоуровневая платформа поверх Hadoop
  - Предоставляет язык программирования: *Pig Latin*
  - Преобразует код такой программы в MapReduce задачи и выполняет их на кластере Hadoop
- Используется во многих компаниях

Pig и MapReduce

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

Изначально был разработан в компании Yahoo! в 2006 для предоставления аналитикам доступа к данным

WordCount: Java vs Python vs Pig.

Java: ~50 строк

```
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);
                }
}
```

Python: 17 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

(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

- Язык, основанный на командах
- Разработан специально для трансформации данных и последовательно обработки

Компилятор Pig преобразует код Pig Latin в MapReduce

- Компилятор оптимизирует процесс выполнения

Среда выполнения

- Среда, в которой выполняются команды Pig Latin
- Поддержка режимов выполнения *Local* и *Hadoop*

Режимы выполнения

Local

- Запускается в рамках одной JVM
- Работает исключительно с локальной файловой системой
- Отлично подходит для разработки, экспериментов и прототипов
- $pig -x local

Hadoop

- Также известен как режим 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
- Запускается в случае отсутствия скрипта
- Можно запускать скрипты из *Grunt* путем команды *run* или *exec*

Embedded

- 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 “)”
  - Напр. (10.4, 5, word, 4, field1)
- *Bag (мешок)* – коллекция кортежей, заключенная в скобки “{” и “}”
  - Напр. { (10.4, 5, word, 4, field1), (this, 1, blah) }

Схожесть с реляционными БД

- Bag – это таблица в БД
- Tuple – это строка в таблице
- Bag не требует, чтобы все tuples содержали одно и то же число полей (в отличии от реляционной таблицы)

Простой пример Pig Latin

```
$ 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.mapReduce
.MapReduceLauncher - 50% complete
2014-07-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 100% complete
...
(a,1)
(d,4)
(c,9)
(k,6)
```

Запуск Grunt в режиме MapRecue по умолчанию

Grunt поддерживает системные команды

Загрузить данные из текстового файла в таблицу *records*

Вывести записи из таблицы *records на экран*

Операции DUMP и STORE

Никаких действий до тех пор, пока не встретятся команды *DUMP* или *STORE*

- Pig будет парсить, проверять и анализировать выражения
- Но не будет выполнять их

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

Данные в Hadoop, обычно, большого объема и нет смысла их все выводить на экран

Обычно данные находятся в HDFS/Hbase/etc

- Команда *STORE*

Для целей отладки можно выводить только небольшую часть на экран

```
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

Команда LOAD

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

data – имя директории или файла

- Должно быть в одинарных кавычках

USING – определяет используемую функцию для загрузки

- По-умолчанию, используется *PigStorage*, которая парсит каждую строку, используя разделитель
  - По-умолчанию, разделить знак табуляции (‘\t’)
  - Разделить может быть задан, используя регулярные выражения

AS – назначает схему входным данным

- Назначает имена полям
- Объявляет тип полей

Команда LOAD

```
1. records =
2.    LOAD '/path/to/file/some.log'
3.    USING PigStorage(';')
4.    AS (userId:chararray,
   timestamp:long, query:chararray);
```

Типы данных для схемы

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
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>;*

Pig Latin: группировка

```
1.  grunt> chars = LOAD '/path/to/file/b.txt' AS (c:chararray);
2.  grunt> DESCRIBE chars;
3.  chars: {c: chararray}
4.  grunt> DUMP chars;
5.  (a)
6.  (k)
7.  ...
8.  (k)
9.  (c)
10. (k)
11. grunt> charGroup = GROUP chars by c;
12. grunt> DESCRIBE charGroup;
13. charGroup: {group: chararray,chars: {(c: chararray)}}
14. grunt> dump charGroup;
15. (a,{(a),(a),(a)})
16. (c,{(c),(c)})
17. (i,{(i),(i),(i)})
18. (k,{(k),(k),(k),(k)})
19. (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*'

Pig Latin: группировка

```
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)}                     |
-------------------------------------------------------------------------------
```

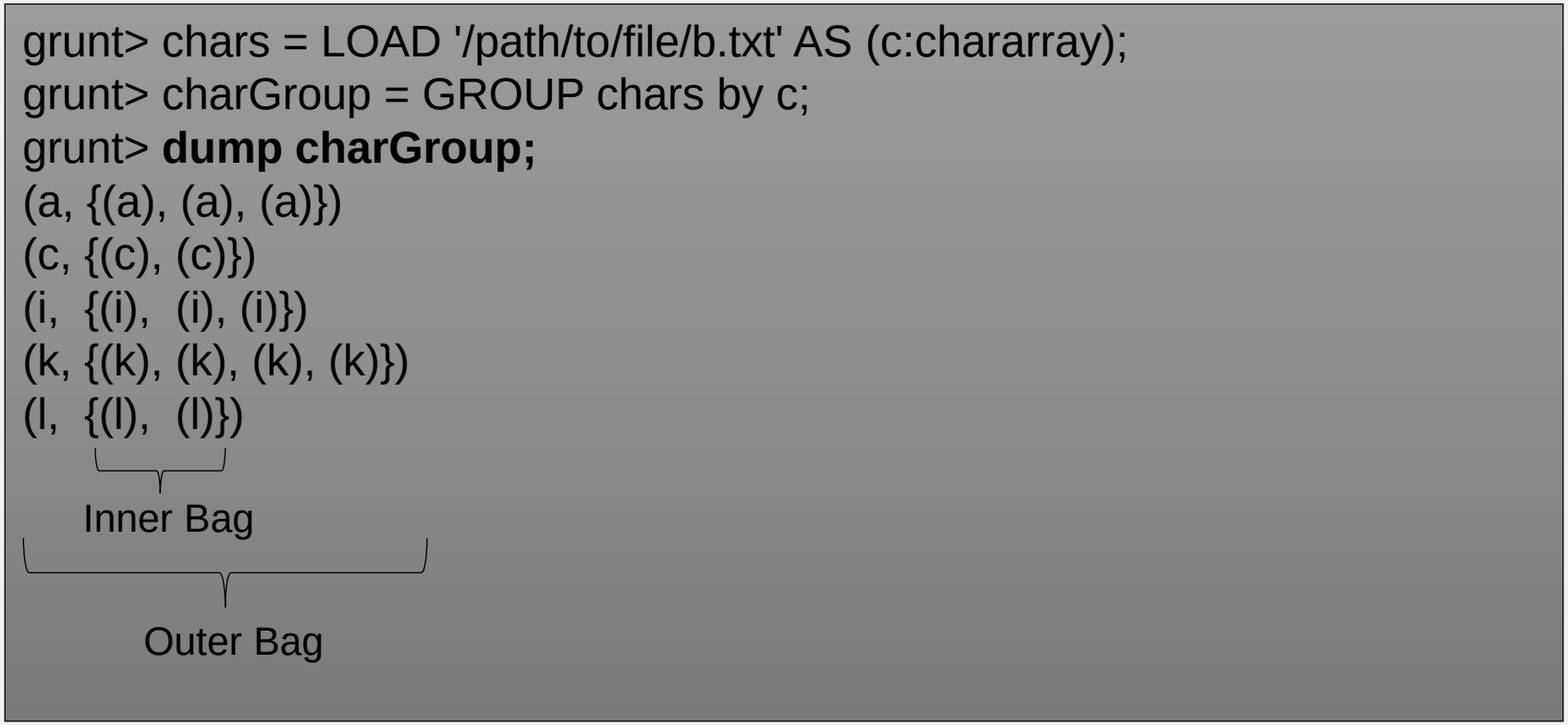
Pig Latin: Inner vs. Outer Bag

```
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)}                     |
-------------------------------------------------------------------------------
```

Inner Bag

Outer Bag

Pig Latin: Inner vs. Outer Bag



PigLatin: FOREACH

FOREACH <bag> GENERATE <data>

- Итерация по каждому элементу в bag и его обработка
- `grunt> result = FOREACH bag GENERATE f1;`

```
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);
```

- Вместе с Pig поставляется множество встроенных функций
  - *COUNT, FLATTEN, CONCAT* и т.д.
- Можно реализовать свою функцию UDF (*User Defined Functions*)
  - Java, Python, JavaScript, Ruby или Groovy

PigLatin: FOREACH с функцией

```
1.  grunt> chars = LOAD 'data/b.txt' AS (c:chararray);
2.  grunt> charGroup = GROUP chars BY c;
3.  grunt> DUMP charGroup;
4.  (a, {(a),(a),(a)})
5.  (c, {(c),(c)})
6.  (i, {(i),(i),(i)})
7.  (k, {(k),(k),(k),(k)})
8.  (l, {(l),(l)})
9.  grunt> DESCRIBE charGroup;
10. charGroup: {group: chararray,chars: {(c: chararray)}}
11. grunt> counts = FOREACH charGroup GENERATE group,
    COUNT(chars);
12. grunt> DUMP counts;
13. (a,3)
14. (c,2)
15. (i,3)
16. (k,4)
17. (l,2)
```

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

PigLatin: функция TOKENIZE

Разбивает строку на токены и возвращает результат в виде bag из токенов

- Разделители: space, double quote("), coma(,), parenthesis(()), star(*)

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

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

7.  grunt> DUMP tokenBag;
8.  ({(this),(is),(a),(line),(of),(text)})
9.  ({(yet),(another),(line),(of),(text)})
10. ({(third),(line),(of),(words)})
11. grunt> DESCRIBE tokenBag;
12. tokenBag: {bag_of_tokenTuples: {tuple_of_tokens: (token:
    chararray)}}
```

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

PigLatin: оператор FLATTEN

- Преобразует вложенные *bag* структуры данных
- FLATTEN – это не функция, это оператор

```
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)
```

Вложенная структура: *bag of bag of words*

Доступ к элементу может быть по индексу

Каждая строка раскладывается в bag простых токенов

PigLatin: WordCount

```
1.  input_lines = LOAD 'copy-of-all-pages-on-internet' AS
    (line:chararray);

2.  -- Extract words from each line and put them into a pig bag
3.  -- datatype, then flatten the bag to get one word on each row
4.  words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS
    word;

5.  -- filter out any words that are just white spaces
6.  filtered_words = FILTER words BY word MATCHES '\\w+';

7.  -- create a group for each word
8.  word_groups = GROUP filtered_words BY word;

9.  -- count the entries in each group
10. word_count = FOREACH word_groups GENERATE COUNT(filtered_words)
    AS count, group AS word;

11. -- order the records by count
12. ordered_word_count = ORDER word_count BY count DESC;
13. STORE ordered_word_count INTO 'number-of-words-on-internet';
```

PigLatin: Joins

Pig поддерживает

- Inner Joins
- Outer Joins
- Full Joins

Фазы join

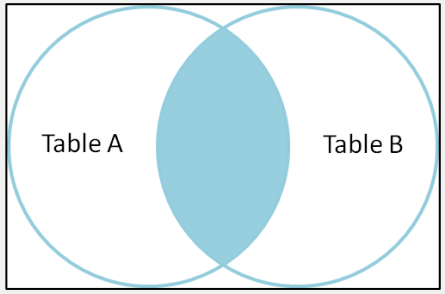
- Загрузить записи в bag из input #1
- Загрузить записи в bag из input #2
- Сделать join для двух массивов данных (bags) по заданному ключу

По-умолчанию используется ***Inner Join***

- Объединяются строки, у которых матчатся ключи
- Строки, у которых нет совпадений, не включаются в результат

Inner Join, пример

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

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

6.  userInfo = JOIN posts BY user, likes BY user;
7.  DUMP userInfo;
```



Inner Join, пример

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

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

11. $ pig InnerJoin.pig
12. (user1,Funny
    Story,1343182026191,user1,12,1343182026191)
13. (user2,Cool
    Deal,1343182133839,user2,7,1343182139394)
14. (user4,Interesting
    Post,1343182154633,user4,50,1343182147364)
```

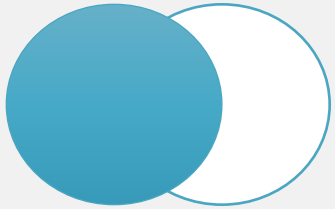
Inner Join, пример

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

3. grunt> DESCRIBE likes;
4. likes: {user: chararray,likes: int,date:
   long}

5. grunt> DESCRIBE userInfo;
6. UserInfo: {
7.     posts::user: chararray,
8.     posts::post: chararray,
9.     posts::date: long,
10.    likes::user: chararray,
11.    likes::likes: int,
12.    likes::date: long}
```

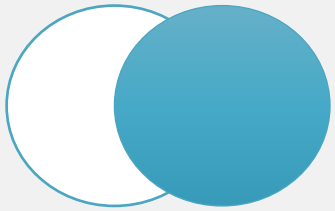
PigLatin: Outer Join

Строки, которые не джойнятся с “другой” таблицей, все равно будут включены в результат

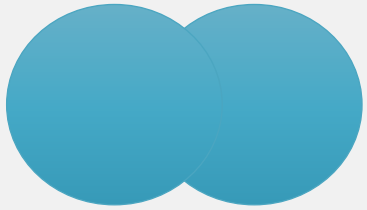
```
JOIN <bag #1> BY <join field>
LEFT OUTER, <bag #2> BY <join field>;
```



```
JOIN <bag #1> BY <join field>
RIGHT OUTER, <bag #2> BY <join field>;
```

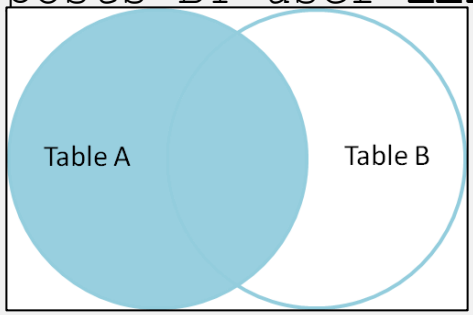


```
JOIN <bag #1> BY <join field>
FULL OUTER, <bag #2> BY <join field>;
```



Left Outer Join, пример

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

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

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



Left Outer Join, пример

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

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

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

Hive

Hive

- Предоставляет SQL-подобный язык, который называется *HiveQL*
  - Минимальный порог вхождения для знакомых с SQL
  - Ориентир на аналитиков данных
- Изначально Hive был разработан в Facebook в 2007
- Сегодня Hive это проект Apache Hadoop
  - http://hive.apache.org

Hive

Hive предоставляет:

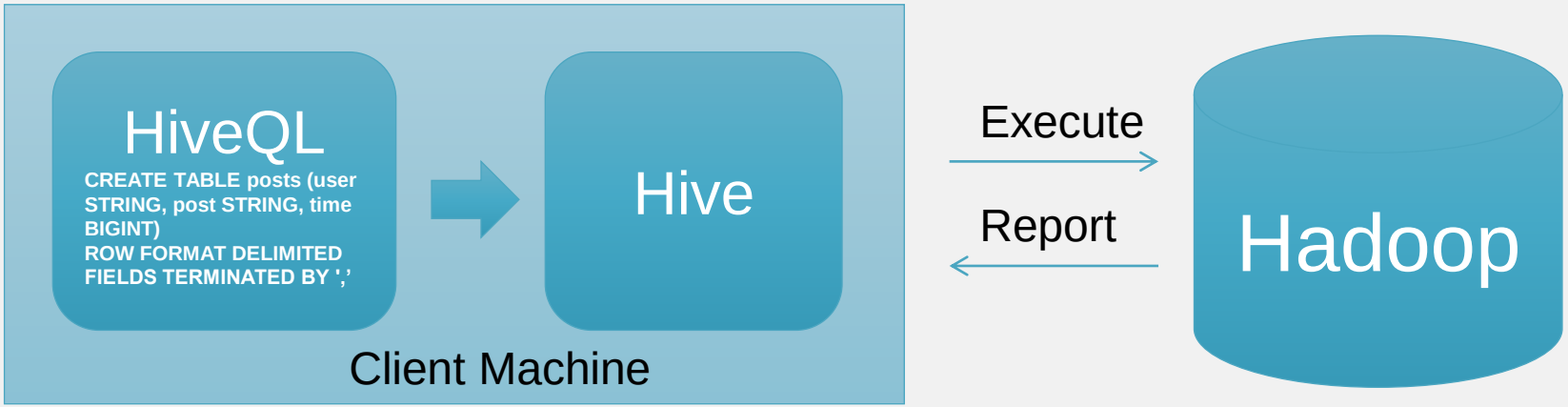
- Возможность структурировать различные форматы данных
- Простой интерфейс для запросов, анализа и обобщения больших объемов данных
- Доступ к данных из различных источников, таких как HDFS и HBase

Hive

- Hive НЕ предоставляет:
  - Low latency или realtime-запросы
- Запрос даже небольших объемов данных может занять минуты
- Разработан с учетом масштабируемости и легкости использования

Hive

- Транслирует запросы HiveQL в набор MapReduce-задач, которые затем выполняются на кластере Hadoop

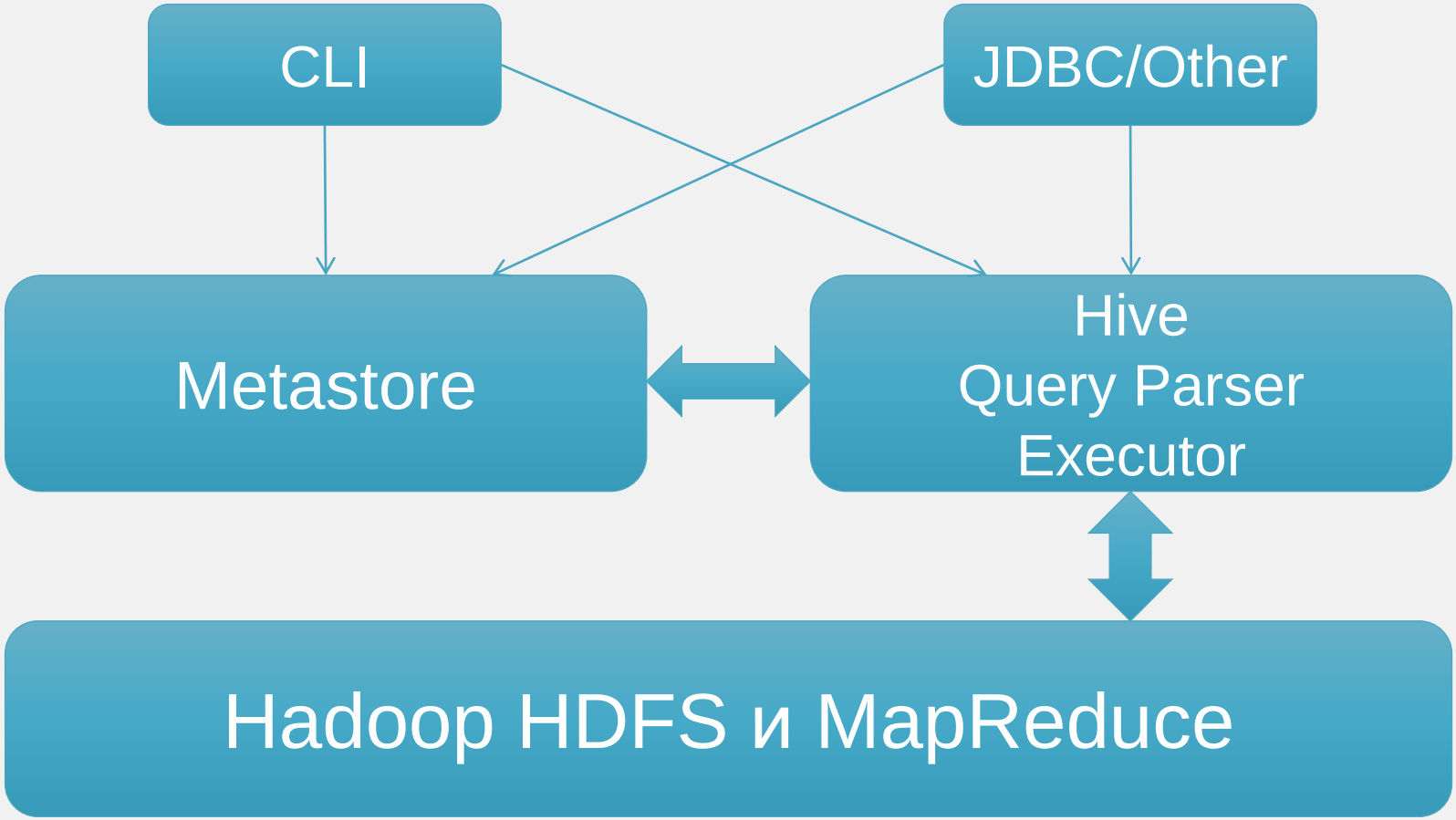


Hive: метаданные

Для поддержки возможностей типа схемы или партиционирования данных Hive хранит мета-информацию в реляционной БД

- Поставляется с Derby, “легковесной” встроенной SQL DB
  - Derby по-умолчанию хорошо подходит для тестирования
  - Схема данных не разделяется между пользователями, т.к. каждый из них имеют свой собственный инстанс Derby
  - Хранит данные в директории *metastore_db,* которая находится в директории, откуда был запущен Hive
- Можно относительно легко переключиться на другу БД, например, MySQL

Архитектура Hive



Hive Интерфейс

Command Line Interface (CLI)

Hive Web Interface

- *https://cwiki.apache.org/confluence/display/Hive/HiveWebInterface*

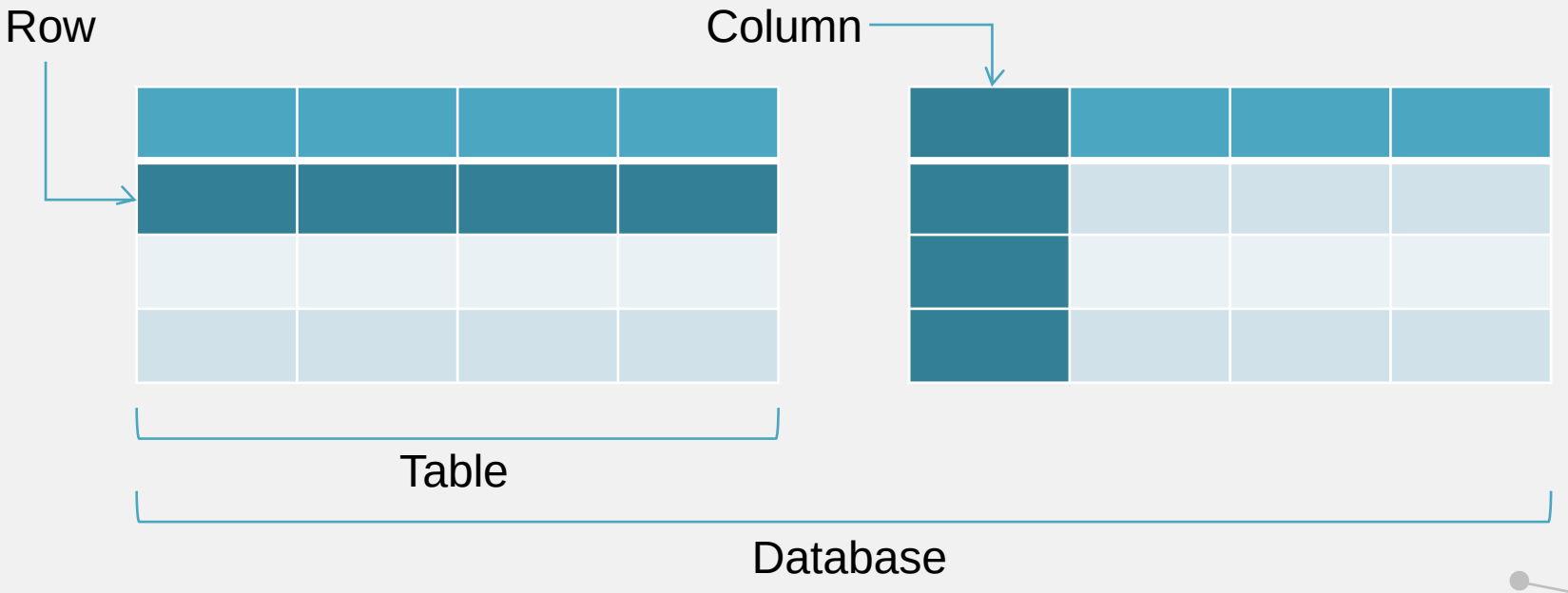
Java Database Connectivity (JDBC)

- *https://cwiki.apache.org/confluence/display/Hive/HiveClient*

Концепция Hive

Позаимствована из реляционных БД

- **Database:** Набор таблиц, для разрешения конфликта имен
- **Table:** Набор строк (*rows*), которые имеют одну схему (для колонок)
- **Row:** Запись, набор колонок
- **Column:** Представляет тип и значение элемента



Hive: пример

1. Создать таблицу
2. Загрузить данные в таблицу
3. Сделать запрос к таблице
4. Удалить таблицу

Hive: Создание таблицы

Загрузим данные из файла *data/user-posts.txt*

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

Выполнения локальных команд в CLI

Hive: Создание таблицы

```
1.  hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
2.      > ROW FORMAT DELIMITED
3.      > FIELDS TERMINATED BY ','
4.      > STORED AS TEXTFILE;
5.  OK
6.  Time taken: 10.606 seconds

7.  hive> show tables;
8.  OK
9.  posts
10. Time taken: 0.221 seconds

11. hive> describe posts;
12. OK
13. user string
14. post string
15. time bigint
16. Time taken: 0.212 seconds
```

-Создать таблицу из 3-х колонок
- Как файл будет парситься
- Как сохранять данные

Показать схему таблицы

Hive: Загрузка данных

```
1.  hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'
2.      > OVERWRITE INTO TABLE posts;
3.  Copying data from file: data/user-posts.txt
4.  Copying file: file: data/user-posts.txt
5.  Loading data to table default.posts
6.  Deleted /user/hive/warehouse/posts
7.  OK
8.  Time taken: 5.818 seconds
9.  hive>
```

Существующие записи в таблице *posts* удаляются
Данные из *data/user-posts.txt* загружены в таблицу *posts*

```
10. $ hdfs dfs -cat /user/hive/warehouse/posts/user-
    posts.txt
11. user1,Funny Story,1343182026191
12. user2,Cool Deal,1343182133839
13. user4,Interesting Post,134318215
14. user5,Yet Another Blog,13431839394
```

По-умолчанию Hive хранит свои таблицы в */user/hive/warehouse*

Hive: Выполнение запроса

```
1.hive> select count(1) from posts;
2.Total MapReduce jobs = 1
3.Launching Job 1 out of 1
4.....
5.Starting Job = job_1343957512459_0004, Tracking URL =
6.http://localhost:8088/proxy/application_1343957512459_0004/
7.Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
8.job_1343957512459_0004
9.Hadoop job information for Stage-1: number of mappers: 1; number of
  reducers: 1
10.2016-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
11.2016-08-02 22:37:31,577 Stage-1 map = 100%, reduce = 0%, Cumulative CPU
  0.87 sec
12.2016-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU
  2.64 sec
13.MapReduce Total cumulative CPU time: 2 seconds 640 msec
14.Ended Job = job_1343957512459_0004
15.MapReduce Jobs Launched:
16.Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS
  Write: 0
17.SUCESS
18.Total MapReduce CPU Time Spent: 2 seconds 640 msec
19.OK
20.4
```

- Считаем кол-во записей в таблице *posts*
- HiveQL преобразуется в 1 MapReduce задачу

Hive: Выполнение запроса

```
1.  hive> select * from posts where user="user2";
2.  ...
3.  ...
4.  OK
5.  user2 Cool Deal 1343182133839
6.  Time taken: 12.184 seconds
7.  hive> select * from posts where time<=1343182133839
    limit 2;
8.  ...
9.  ...
10. OK
11. user1 Funny Story 1343182026191
12. user2 Cool Deal 1343182133839
13. Time taken: 12.003 seconds
14. hive>
```

Выбрать записи пользователя *‘user2’*

- Фильтр по timestamp
- Ограничиваем число записей в результате

Hive: Удаление таблицы

```
1. hive> DROP TABLE posts;
2. OK
3. Time taken: 1.234 seconds
4. hive> exit;


5. $ hdfs dfs -ls /user/hive/warehouse/
6. $
```

Hive: Нарушение схемы

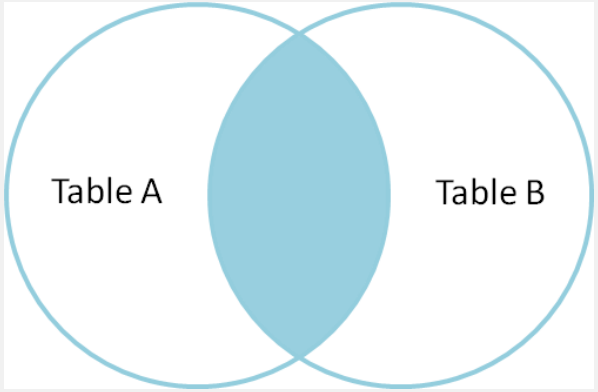
```
1. hive> !cat data/user-posts-
   error.txt;
2. user1,Funny Story,1343182026191
3. user2,Cool Deal,2012-01-05
4. user4,Interesting Post,1343182154633
5. user5,Yet Another Blog,13431839394

6. hive> describe posts;
7. OK
8. user string
9. post string
10.time bigint
11.Time taken: 0.289 seconds
```

Hive: Нарушение схемы

```
1.  hive> LOAD DATA LOCAL INPATH
2.      > 'data/user-posts-error.txt '
3.      > OVERWRITE INTO TABLE posts;
4.  OK
5.  Time taken: 0.612 seconds

6.  hive> select * from posts;
7.  OK
8.  user1 Funny Story 1343182026191
9.  user2 Cool Deal NULL
10. user4 Interesting Post 1343182154633
11. user5 Yet Another Blog 13431839394
12. Time taken: 0.136 seconds
13. hive>
```

Hive: Joins

- Использовать ***joins*** в Hive просто
- Поддержка ***outer joins***
  - left, right и full joins
- Можно объединять множество таблиц
- По-умолчанию используется ***Inner Join***
  - Объединяются строки, у которых матчатся ключи
  - Строки, у которых нет совпадений, не включаются в результат



Hive: Inner Join

```
1.  hive> select * from posts limit 10;
2.  OK
3.  user1 Funny Story 1343182026191
4.  user2 Cool Deal 1343182133839
5.  user4 Interesting Post 1343182154633
6.  user5 Yet Another Blog 1343183939434
7.  hive> select * from likes limit 10;
8.  OK
9.  user1 12 1343182026191
10. user2 7 1343182139394
11. user3 0 1343182154633
12. user4 50 1343182147364
13. Time taken: 0.103 seconds
14. hive> CREATE TABLE posts_likes (user STRING, post
    STRING, likes_count INT);
15. OK
```

Hive: Inner Join

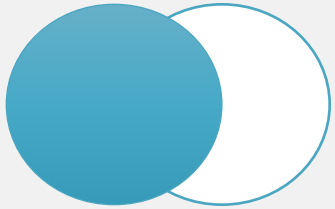
```
1.  hive> INSERT OVERWRITE TABLE posts_likes
2.      > SELECT p.user, p.post, l.count
3.      > FROM posts p JOIN likes l ON (p.user = l.user);
4.  OK
5.  Time taken: 17.901 seconds

6.  hive> select * from posts_likes limit 10;
7.  OK
8.  user1 Funny Story 12
9.  user2 Cool Deal 7
10. user4 Interesting Post 50
11. Time taken: 0.082 seconds
12. hive>
```

Hive: Outer Join

Строки, которые не джойнятся с “другой” таблицей, все равно будут включены в результат

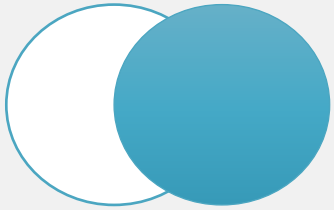
Left Outer

```
SELECT p.*, l.*
FROM posts p LEFT OUTER JOIN likes l ON
(p.user = l.user)
limit 10;
```



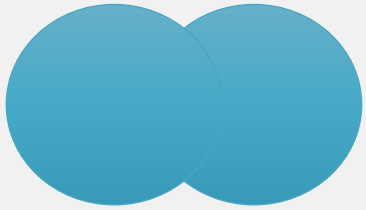
RIGHT Outer

```
SELECT p.*, l.*
FROM posts p RIGHT OUTER JOIN likes l ON
(p.user = l.user)
limit 10;
```



FULL Outer

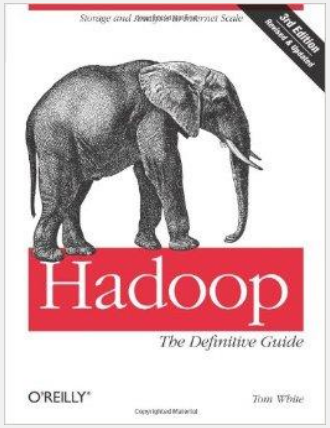
```
SELECT p.*, l.*
FROM posts p FULL OUTER JOIN likes l ON
(p.user = l.user)
limit 10;
```



Hive: WordCount

```
1. CREATE TABLE docs (line STRING);

2. LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

3. CREATE TABLE word_counts AS
4. SELECT word, count(1) AS count FROM
5.     (SELECT explode(split(line, '\s')) AS word
   FROM docs) w
6. GROUP BY word
7. ORDER BY word;
```

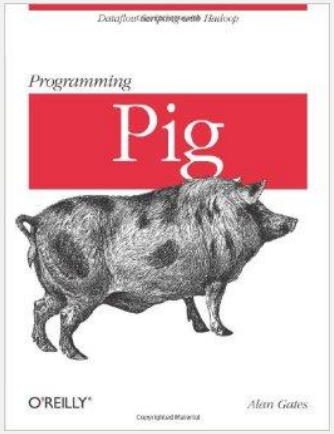
Ресурсы



Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition

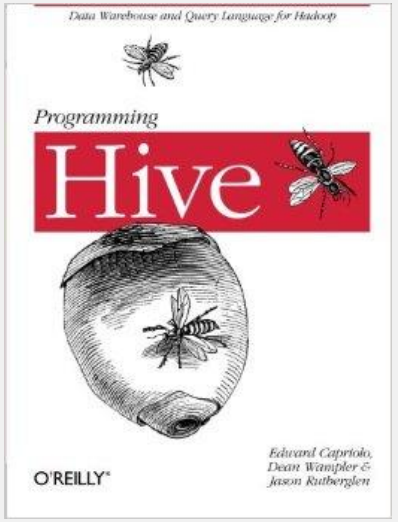
Chapter 11 Pig
Chapter 12 Hive

Ресурсы



Programming Pig
Alan Gates (Author)
O'Reilly Media; 1st Edition

Programming Hive
Edward Capriolo, Dean Wampler,
Jason Rutherglen (Authors)
O'Reilly Media; 1 edition



ТЕХНОСФЕРА

Отмечайтесь и оставляйте отзыв

Спасибо за внимание!

Евгений Чернов

e.chernov@corp.mail.ru