



ТЕХНОСФЕРА

Методы распределенной обработки больших объемов данных в Hadoop

Лекция 7: Введение в Pig и Hive

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 высокого уровня *Pig Latin* для обработки данных
 - 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

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
- Среда выполнения
 - Среда, в которой выполняются команды Pig Latin
 - Поддержка режимов выполнения *Local* и *Hadoop*
- Компилятор Pig преобразует код Pig Latin в MapReduce
 - Компилятор оптимизирует процесс выполнения

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 команда *run* или *exec*
- Embedded
 - Выполнять команды Pig используя класс *PigServer*
 - По типу использования JDBC для исполнения SQL
 - 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 отличие от реляционной таблицы)

Простой пример Pig Latin

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

```
a    1  
d    4  
c    9  
k    6
```

Grunt поддерживает
системные команды

Запуск Grunt в режиме MapReduce
по умолчанию

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

```
grunt> DUMP records;
```

```
...
```

```
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer  
.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)
```

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 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

records =

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

USING PigStorage()

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 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 | collection of tuples | [open#apache] |

Pig Latin: средства диагностики

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

Pig Latin: группировка

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

```
grunt> DESCRIBE chars;
```

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

```
grunt> DUMP chars;
```

```
(a)
```

```
(k)
```

```
...
```

```
(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*'

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

```
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)})
```



Inner Bag



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

```
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

- Разбивает строку на токены и возвращает результат в виде bag из токенов
 - Разделители: space, double quote("), coma(,), parenthesis(()), star(*)

```
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)}}
```

```
grunt> DESCRIBE tokenBag;
```

```
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

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

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

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

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

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

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

PigLatin: Joins

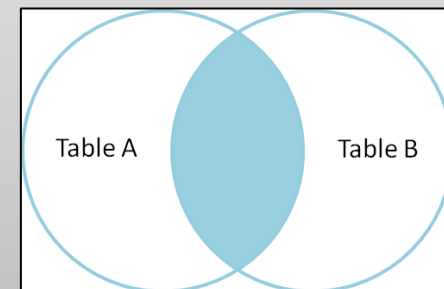
- Pig поддерживает
 - Inner Joins
 - Outer Joins
 - Full Joins
- Фазы join
 - Загрузить записи в bag из input #1
 - Загрузить записи в bag из input #2
 - Сделать join для двух массивов данных (bags) по заданному ключу
- По-умолчанию используется ***Inner Join***
 - Объединяются строки, у которых матчатся ключи
 - Строки, у которых нет совпадений, не включаются в результат

Inner Join, пример

```
-- 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;
```



Inner Join, пример

```
$ 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

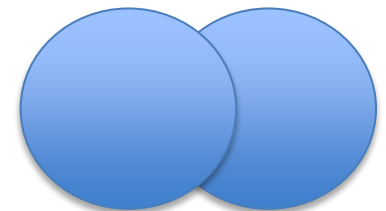
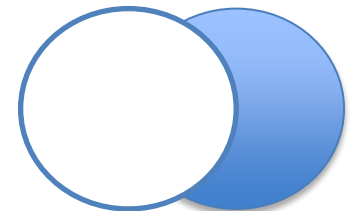
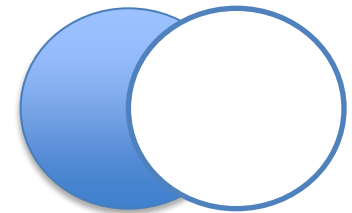
```
grunt> DESCRIBE posts;  
posts: {user: chararray,post: chararray,date: long}
```

```
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

- 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>;

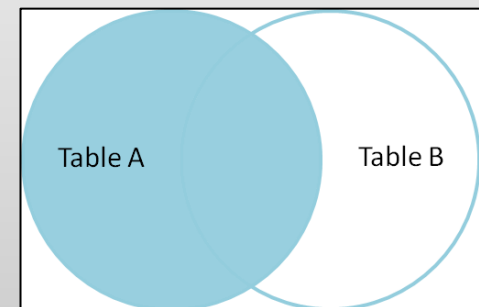


Left Outer Join, пример

```
--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;
```



Left Outer Join, пример

```
$ 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,,)
```

Hive



Hive

- Решение *Data Warehousing*, работающее поверх Hadoop
- Предоставляет SQL-подобный язык, который называется *HiveQL*
 - Минимальный порог вхождения для тех, кто знаком 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

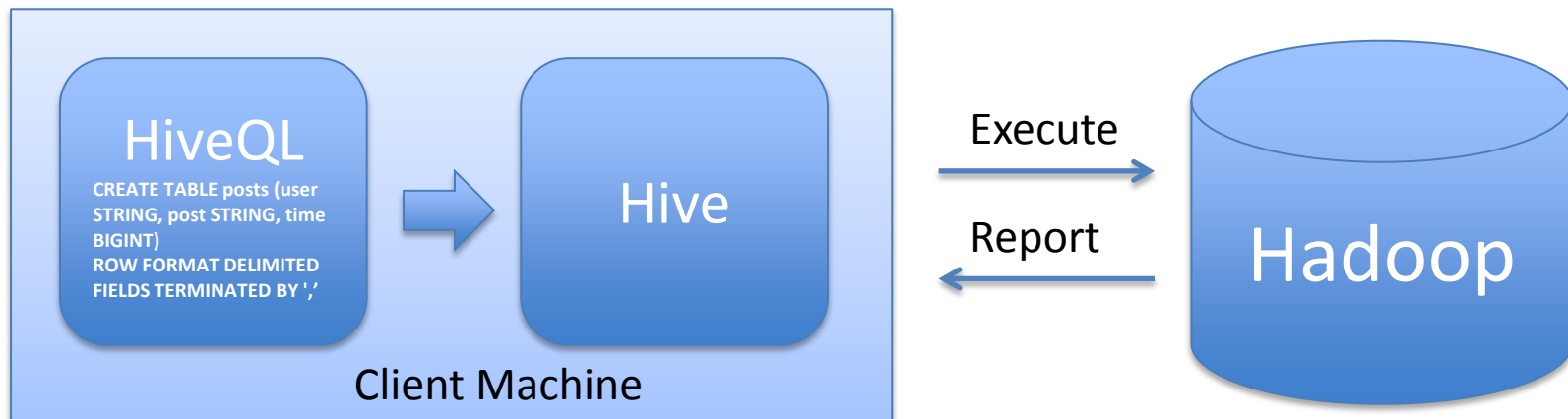
- Hive предоставляет:
 - Возможность структурировать различные форматы данных
 - Простой интерфейс для *ad hoc* запросов, анализа и обобщения больших объемов данных
 - Доступ к данным из различных источников, таких как HDFS и HBase

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

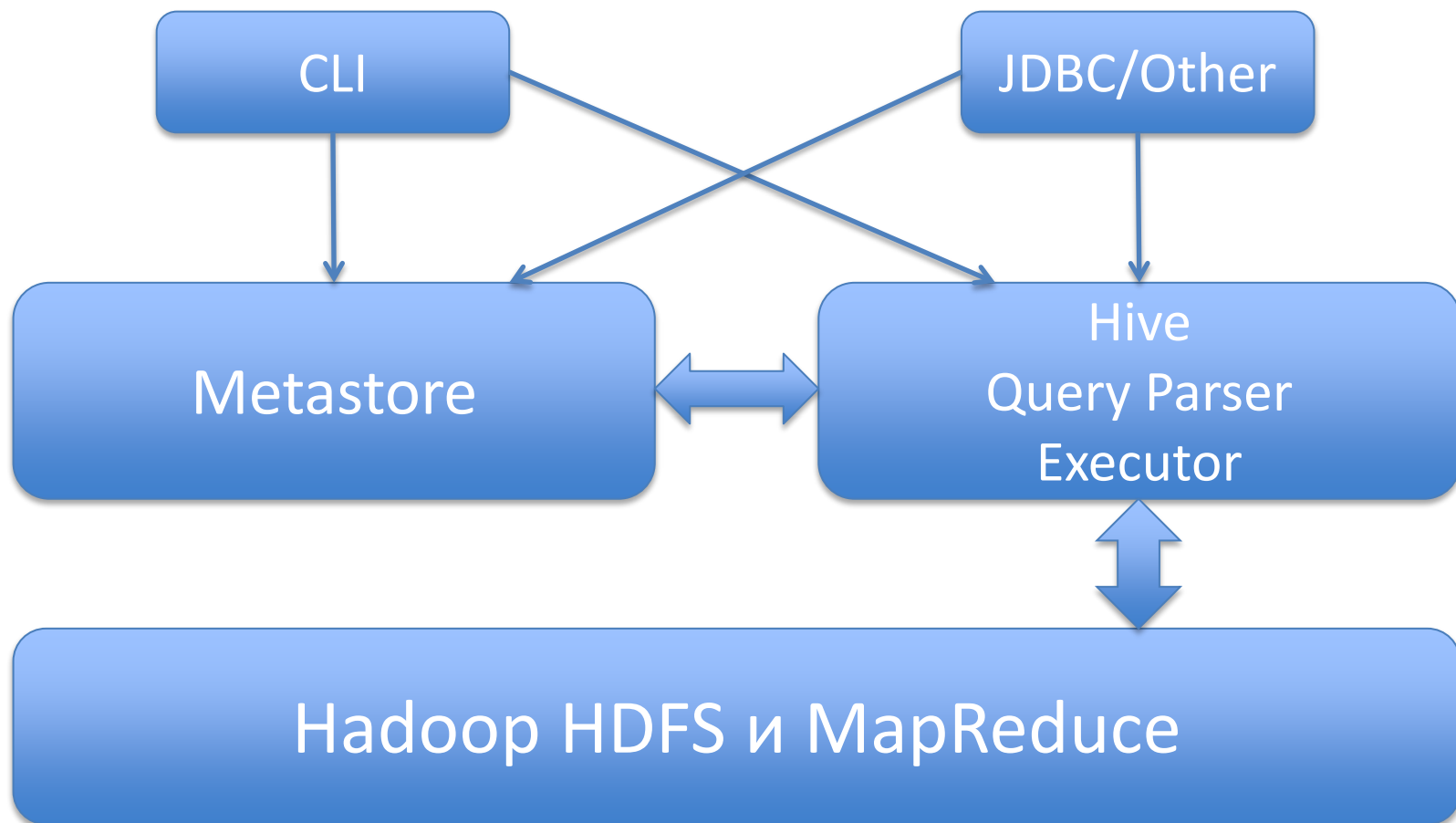
- Транслирует запросы HiveQL в набор MapReduce-задач, которые затем выполняются на кластере Hadoop



Hive

- Для поддержки возможностей типа схемы или партиционирования данных Hive хранит метainформацию 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 другую БД, например, MySQL

Архитектура Hive

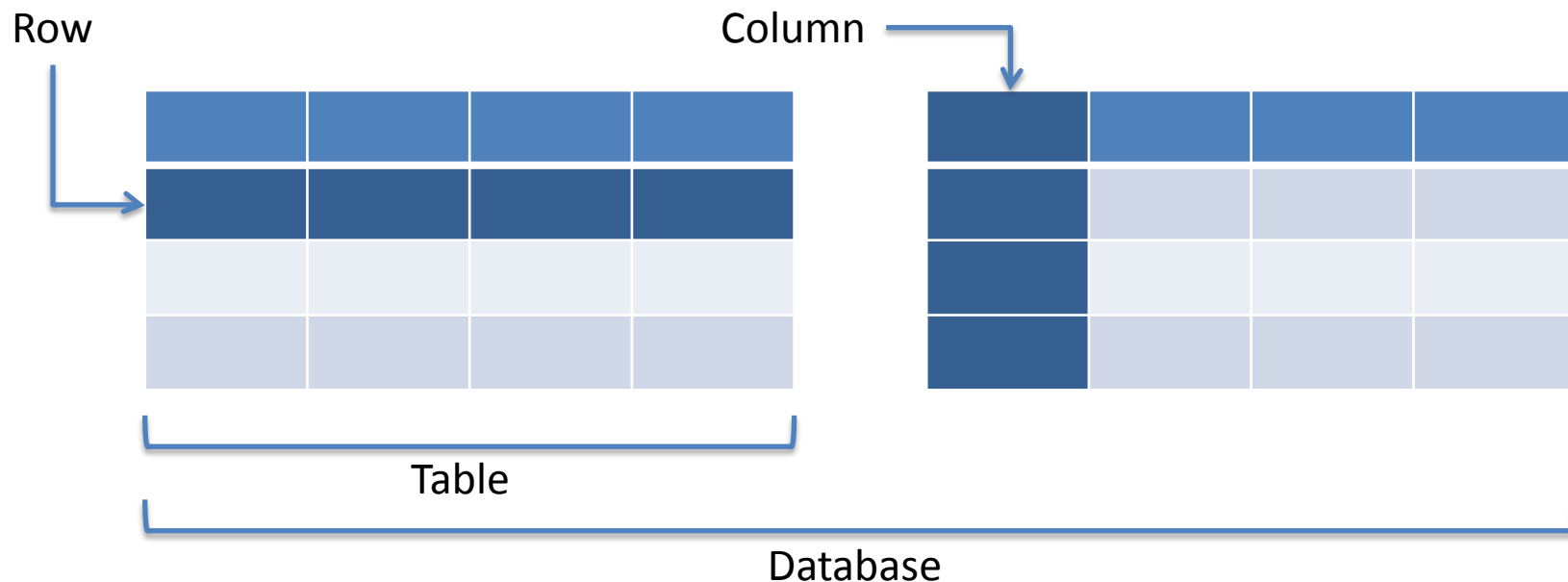


Hive Интерфейс

- 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

- Создать таблицу
- Загрузить данные в таблицу
- Сделать запрос к таблице
- 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

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
```

```
> ROW FORMAT DELIMITED
```

```
> FIELDS TERMINATED BY ','
```

```
> STORED AS TEXTFILE;
```

```
OK
```

```
Time taken: 10.606 seconds
```

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

```
hive> show tables;
```

```
OK
```

```
posts
```

```
Time taken: 0.221 seconds
```

```
hive> describe posts;
```

```
OK
```

```
user string
```

```
post string
```

```
time bigint
```

```
Time taken: 0.212 seconds
```

Показать схему таблицы

Hive: Загрузка данных

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'  
> OVERWRITE INTO TABLE posts;  
Copying data from file: data/user-posts.txt  
Copying file: file: data/user-posts.txt  
Loading data to table default.posts  
Deleted /user/hive/warehouse/posts  
OK  
Time taken: 5.818 seconds  
hive>
```

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

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

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

Hive: Выполнение запроса

```
hive> select count (1) from posts;
```

```
Total MapReduce jobs = 1
```

```
Launching Job 1 out of 1
```

```
...
```

```
Starting Job = job_1343957512459_0004, Tracking URL =
```

```
http://localhost:8088/proxy/application_1343957512459_0004/
```

```
Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
```

```
job_1343957512459_0004
```

```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
```

```
2014-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
```

```
2014-08-02 22:37:31,577 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
```

```
2014-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.64 sec
```

```
MapReduce Total cumulative CPU time: 2 seconds 640 msec
```

```
Ended Job = job_1343957512459_0004
```

```
MapReduce Jobs Launched:
```

```
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS Write: 0
```

```
SUCCESS
```

```
Total MapReduce CPU Time Spent: 2 seconds 640 msec
```

```
OK
```

```
4
```

```
Time taken: 14.204 seconds
```

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

Hive: Выполнение запроса

```
hive> select * from posts where user="user2";
```

```
...
```

Выбрать записи пользователя 'user2'

```
...
```

```
OK
```

```
user2 Cool Deal 1343182133839
```

```
Time taken: 12.184 seconds
```

```
hive> select * from posts where time<=1343182133839 limit 2;
```

```
...
```

```
...
```

```
OK
```

```
user1 Funny Story 1343182026191
```

```
user2 Cool Deal 1343182133839
```

```
Time taken: 12.003 seconds
```

```
hive>
```

- Фильтр по timestamp
- Ограничиваем число записей в результате

Hive: Удаление таблицы

```
hive> DROP TABLE posts;  
OK  
Time taken: 1.234 seconds  
hive> exit;
```

```
$ hdfs dfs -ls /user/hive/warehouse/  
$
```

Hive: Нарушение схемы

```
hive> !cat data/user-posts-error.txt;
user1,Funny Story,1343182026191
user2,Cool Deal,2012-01-05
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394
hive> describe posts;
OK
user string
post string
time bigint
Time taken: 0.289 seconds
```

Hive: Нарушение схемы

```
hive> LOAD DATA LOCAL INPATH  
      > 'data/user-posts-error.txt '  
      > OVERWRITE INTO TABLE posts;
```

OK

Time taken: 0.612 seconds

```
hive> select * from posts;
```

OK

user1 Funny Story 1343182026191

user2 Cool Deal **NULL**

user4 Interesting Post 1343182154633

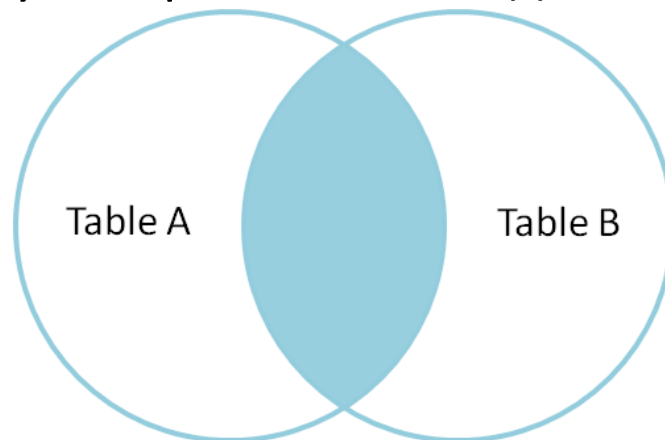
user5 Yet Another Blog 13431839394

Time taken: 0.136 seconds

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

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

Hive: Inner Join

```
hive> INSERT OVERWRITE TABLE posts_likes  
      > SELECT p.user, p.post, l.count  
      > FROM posts p JOIN likes l ON (p.user = l.user);
```

OK

Time taken: 17.901 seconds

```
hive> select * from posts_likes limit 10;
```

OK

user1 Funny Story 12

user2 Cool Deal 7

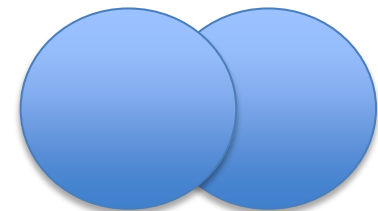
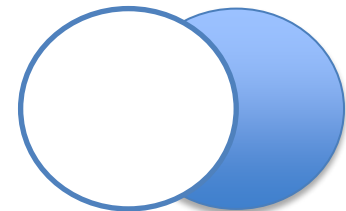
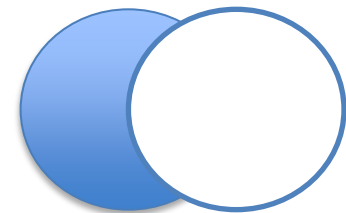
user4 Interesting Post 50

Time taken: 0.082 seconds

hive>

Hive: Outer Join

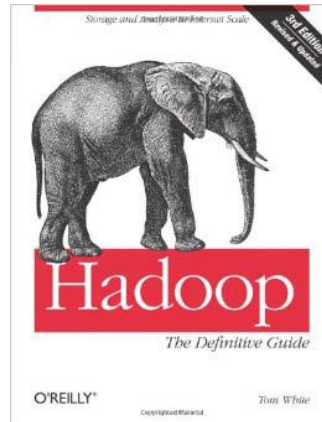
- Строки, которые не джойнятся с “другой” таблицей, все равно будут включены в результат
- **Left Outer**
SELECT p.*, l.*
FROM posts p **LEFT OUTER JOIN** likes l ON (p.user = l.user)
limit 10;
- **RIGHT Outer**
SELECT p.*, l.*
FROM posts p **RIGHT OUTER JOIN** likes l ON (p.user = l.user)
limit 10;
- **FULL Outer**
SELECT p.*, l.*
FROM posts p **FULL OUTER JOIN** likes l ON (p.user = l.user)
limit 10;



Hive: WordCount

```
CREATE TABLE docs (line STRING);  
  
LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;  
  
CREATE TABLE word_counts AS  
SELECT word, count(1) AS count FROM  
    (SELECT explode(split(line, '\s')) AS word FROM docs) w  
GROUP BY word  
ORDER BY word;
```

Ресурсы



Hadoop: The Definitive Guide

Tom White (Author)

O'Reilly Media; 3rd Edition

Chapter 11 Pig

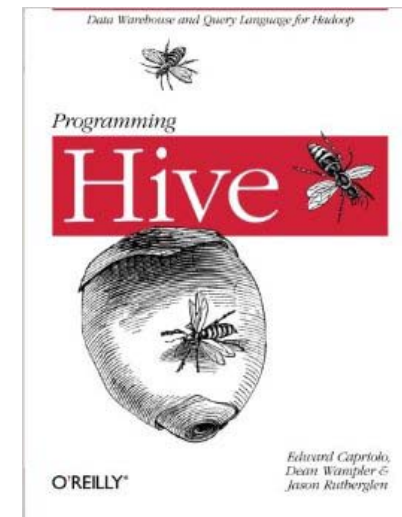
Chapter 12 Hive

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



Вопросы?

