PIG Latin

язык для работы с данными в HADOOP

Зачем нам нужен PIG latin ?

- Чем неудобен hadoop ?
  - Большая сложность классического api hadoop обуславливает существенное время разработки.
  - Сложность многоэтапной обработки данных.
  - Сложность перехода от классических требований (join, group, sort) к понятиям map reduce
- PIG предлагает простой язык работы с потоками данных, который компилируется в java код и выполняется в Hadoop

Минусы PIG Latin

- Производительность PIG скриптов меньше чем у аналогичных программ написаных на API Hadoop

На что это похоже

- Загрузка данных

```
records = LOAD 'input/ncdc/micro-tab/sample.txt'
          AS (year:chararray, temperature:int, quality:int);
```

- Очистка

```
filtered_records = FILTER records BY
                   temperature!= 9999 AND quality == 0 ;
```

- Группировка

```
grouped_records = GROUP filtered_records BY year;
```

- Поиск максимумов

```
max_temp = FOREACH grouped_records
           GENERATE group, MAX(filtered_records.temperature);
```

Модель данных

- Примитивы :
  - int, long, float, double
  - bytearray
  - chararray
- Tuple – набор значений имеющий заданный порядок

  (11,'value',(22,'test'))
- Bag – множество tuple, не имеющее заданного порядка

  {(11,22),('test',33)}
- Map – множество пар ключ#значение. Ключ является строковым типом

  ['a'#'value', 'b'#'value2']
- Relation – набор данных над которым производятся действия. Может быть загружен из внешнего источника или получен в результате выполнения операторов (statements).

  relation состоит из записей каждая из которых является Tuple.

  (11,'value',(22,'test'))

  (22,'value2',(33,'test'))

Схема модели данных

- Relation может иметь схему описывающую структору модели данных
- Можно распечатать с помощью команды Describe

```
grunt> systems = LOAD 'systems.txt' using PigStorage('
') AS (code:chararray, name:chararray);;

grunt> DESCRIBE systems;

systems: {code: chararray,name: chararray}
```

- Наличие схемы дает доступ к полям tuple используя их имена.

Выражения(expressions)

Category	Expressions	Description	Examples
Constant	Literal	Constant value (see also literals in Table 11-6)	`1.0`, `'a'`
Field (by position)	`$`*n*	Field in position *n* (zero-based)	`$0`
Field (by name)	*f*	Field named *f*	`year`
Field (disambiguate)	*r*`::`*f*	Field named *f* from relation *r* after grouping or joining	`A::year`
Projection	*c*`.$`*n*, *c*`.`*f*	Field in container *c* (relation, bag, or tuple) by position, by name	`records.$0`, `records.year`
Map lookup	*m*`#`*k*	Value associated with key *k* in map *m*	`items#'Coat'`
Cast	`(`*t*`)` *f*	Cast of field *f* to type *t*	`(int) year`
Arithmetic	*x* `+` *y*, *x* `-` *y*	Addition, subtraction	`$1 + $2`, `$1 - $2`
	x `*` *y*, *x* `/` *y*	Multiplication, division	`$1 * $2`, `$1 / $2`
	x `%` *y*	Modulo, the remainder of *x* divided by *y*	`$1 % $2`
	`+`*x*, `-`*x*	Unary positive, negation	`+1`, `-1`
Conditional	*x* `?` *y* `:` *z*	Bincond/ternary, *y* if *x* evaluates to true, *z* otherwise	`quality == 0 ? 0 : 1`

Выражения(Expressions)

Category	Expressions	Description	Examples
Comparison	*x* == *y*, *x* != *y*	Equals, not equals	quality == 0, temperature != 9999
	x > *y*, *x* < *y*	Greater than, less than	quality > 0, quality < 10
	x >= *y*, *x* <= *y*	Greater than or equal to, less than or equal to	quality >= 1, quality <= 9
	x matches *y*	Pattern matching with regular expression	quality matches '[01459]'
	x is null	Is null	temperature is null
	x is not null	Is not null	temperature is not null
Boolean	*x* or *y*	Logical or	q == 0 or q == 1
	x and *y*	Logical and	q == 0 and r == 0
	not *x*	Logical negation	not q matches '[01459]'
Functional	*fn*(*f1*,*f2*,…)	Invocation of function *fn* on fields *f1*, *f2*, etc.	isGood(quality)
Flatten	FLATTEN(*f*)	Removal of a level of nesting from bags and tuples	FLATTEN(group)

Операторы (statements) Загрузка и сохранение данных

- Load - загрузка данных из внешнего источника, например файла HDFS

  ```
  systems = LOAD 'systems.txt' using PigStorage(' ') AS (code:chararray, name:chararray);
  ```

  using <Storage> указывает на использование модуля отвечающего за интерфейс к источнику данных.

  Примеры Storage :

  - PigStorage загружает текстовый файл используя разделитель.
  - BinStorage использует формат hadoop (объекты writable)
  - TextLoader загружает файл генерируя для каждой строки tuple с одним полем
  - JsonLoader разбирает json формат
- Store – сохраняет relation во внешнюю систему используя Storage

  ```
  store processed into 'processed' using PigStorage(',');
  ```
- Dump – используется для разработки, печатает содержимое relation на экран

Foreach

- Берет все записи из relation и применяет к ним выражения(expressions)
- В результате формируется новый relation

```
A = load 'foo' as (x:chararray, y:int, z:int);
A1 = foreach A generate x, y + z as yz;
```

- Для задания схемы в новом relation используем ключевое слово as

Filter, Group

- Filter - фильтрует записи из relation используя заданное условие (predicate) типа boolean

  startswithcm = filter divs by symbol matches 'CM.*';

- Group – группирует записи имеющее одинаковое заданное поле.
- В результате выполнения group формируется relation такого вида

  (group:<поле группировки>, <имя исходного relation>:{(исходные записи),()})

  пример :

  calls = LOAD 'calls.txt' using PigStorage(' ') AS (id:chararray, systema:chararray, systemb:chararray, msgid:chararray, originalmsgid:chararray, body:chararray);

  calls_by_a = group calls by systema;

  describe calls_by_a

  calls_by_a: {group: chararray,calls: {(id: chararray,systema: chararray,

  systemb: chararray,msgid: chararray,originalmsgid: chararray,body: chararray)}};

Использование Group

- Обычно к Bag применяют функцию агрегирования Count, Max, Min и т.д.

  calls = LOAD 'calls.txt' using PigStorage(' ') AS (id:chararray, systema:chararray, systemb:chararray, msgid:chararray, originalmsgid:chararray, body:chararray);

  calls_by_a = group calls by systema;

  stat_by_a = foreach calls_by_a generate group as system, COUNT(calls) as call_count;

  >>(systema,3)

  >>(systemb,2)

  >>(systemc,11)

Order, Distinct

- Order сортирует relation

  ordered_calls = order calls by systema

  Следует учесть что order в PIG всегда использует reduce и сэмплирует исходные данные

- Distinct отбирает уникальные записи

  uniq_calls = distinct calls;

Join

- При связывании relation требуется указать поля по которым происходит join

```
joined_calls = join calls by systema, systems by  code;
describe joined_calls;
joined_calls: {calls::id: chararray,....,systems::name: chararray}
```

- В генерируемый relation добавляются поля исходных relation следующего вида

  <имя relation>::<имя поля>

- Возможно использование outer join – когда требуется добавлять в итоговый relation записи для которых нет соответствия в другом relation.
- Left outer join – использует в качестве ведушего первый relation (его записи добавляются все – даже те, для которых не соответствия в правом relation)
- Right outer join – использует в качестве основного второй relation

```
joined_calls2 = join calls by systemb left outer, systems by code;
```

Cogroup

- Аналогичен Join, но для каждого ключа генерируется запись, содержащая ключ и два Bag.
- В каждый из Bag добавляются tuple из связываемых relation которые имеют значение ключа равное ключу записи.
  - cogroup_calls = cogroup calls by systema, systems by code;
  - describe cogroup_calls;

    >>cogroup_calls: {group: chararray,calls: {(id: chararray,systema: chararray,systemb: chararray,msgid: chararray,originalmsgid: chararray,body: chararray)},systems: {(code: chararray,name: chararray)}}
  - dump cogroup_calls;

    >>(systema,{(12,systema,systema,msgid4,originalmsgid3,body5),
    (1,systema,systemb,msgid,originalmsgid,body),
    (3,systema,systemc,msgid2,originalmsgid2,body3)},{(systema,mvd)});
    - >>...

Cross

- Производит декартово произведение relation
- В результат добавляются все комбинации записей из первого и второго relation
  - cross_systems = cross systems, systems2;
  - describe cross_systems;

    >>cross_systems: {systems::code: chararray,systems::name: chararray,systems2::code: chararray,systems2::name: chararray}
  - dump cross_systems;

    >>(systema,mvd,systema,mvd)

    >>(systema,mvd,systemb,fms)

    >>(systema,mvd,systemc,minzdrav)

    >>(systema,mvd,systemd,minregion)

    >>(systema,mvd,systeme,vs)

    >>(systemb,fms,systema,mvd)
    -

Union, Split

- Union объединяет два relation в один
- В случае одинаковой схемы – схема остается. В случае если схема была разной, получившийся relation не имеет схемы

  union_all = union systems, calls;
- Split предназначени для разбиения relation

  аналогичен набору выражений с filter
  - split calls into

    fromA if systema == 'systema',

    fromB if systema == 'systemb';

Flatten

- Ключевое слово Flatten предназначено для уменьшения уровня вложенности tuple и bag
- Flatten “вытаскивает” из вложенного tuple поля на более высокий уровень.
- В случае применения flatten к bag – производится декартово произведение содержимого bag и других полей
  - calls_by_a = group calls by systema;
  - unwrapped_calls - foreach calls_by_a generate flatten(calls);
  - >> <копия оригиналного calls>