Análisis de datos en HDFS con Apache Pig

Documentación oficial:

[Apache PIG]

Tutorial breve:

https://www.tutorialspoint.com/apache_pig/index.htm

- Una introducción a Apache Pig
- Fundamentos de la programación de Pig
- Cómo cargar datos en Pig
- Cómo utilizar las funciones incorporadas en Pig

A medida que la plataforma Hadoop fue ganando terreno, empezaron a surgir alternativas a la programación MapReduce en Java. Una de estas alternativas fue el proyecto Pig. Apache Pig es un lenguaje de flujo de datos de alto nivel utilizado para analizar datos en HDFS. El uso de Pig aprovecha el entorno de programación distribuido de Hadoop, permitiendo el desarrollo rápido de aplicaciones y velocidad en la extracción de datos.

Presentación de Pig

En 2006, se inició un nuevo proyecto de investigación en Yahoo! llamado Pig. La premisa detrás de Pig era proporcionar una interfaz de lenguaje alternativa a la programación en Java de MapReduce. Pig se considera una abstracción de MapReduce ya que abstrae al desarrollador o analista los entresijos del código MapReduce subyacente, permitiendo al codificar en un lenguaje de flujo de datos interpretado, que luego se convierte en una serie de operaciones de MapReduce.

El lenguaje de flujo de datos utilizado por Pig se llama Pig Latin. Un intérprete que se ejecuta en una máquina cliente toma las instrucciones de Pig Latin y las convierte en una serie de trabajos de MapReduce, enviando estos trabajos al clúster, supervisando su progreso y devolviendo los resultados a la consola o guardando los resultados en archivos en HDFS. Este proceso se muestra en la Figura 1.

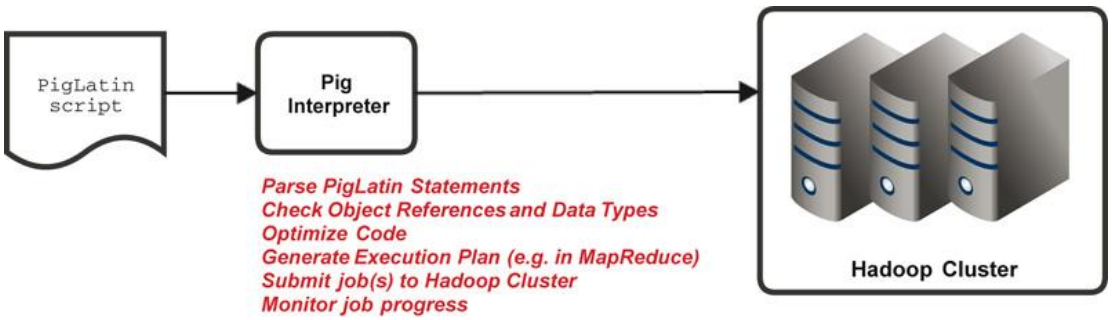


Figura 1 Flujo de Pig.

En el Listado 1 se muestra un ejemplo de aplicación Pig. Como se puede ver el código es muy intuitivo (pronto discutiremos las sentencias en detalle).

Listado 1 Programa Pig

```
A = LOAD 'clientes' USING PigStorage() AS (nombre:chararray, edad:int);
B = FOREACH A GENERATE nombre;
```

Beneficios de Pig:

- Los desarrolladores no necesitan saber programar en Java.
- El código de Pig no necesita ser compilado y empaquetado, a diferencia de las aplicaciones MapReduce de Java.
- Pig no obliga a los desarrolladores a describir objetos, lo que facilita el descubrimiento de datos.

En 2008, Pig se convirtió en un proyecto de la Apache Software Foundation con la adopción y contribución de empresas como LinkedIn, Twitter y muchas otras. Pig sigue siendo un proyecto activo de primer nivel de Apache con extensiones en Spark como motor de ejecución, y proyectos como Spork-Pig en Spark.

Conceptos básicos de Pig Latin

Para instalar Pig simplemente se necesita un sistema con un cliente Hadoop y una configuración, que, por supuesto, requerirá que haya un motor de ejecución Java.

Modos de Pig

Pig, al igual que MapReduce, tiene diferentes modos en los que puede ejecutarse, y son los siguientes:

- MapReduce (el predeterminado o se lanza con `pig -x mapreduce`)
- Local (se lanza con `pig -x local`)
- Tez (p.e. en *clusters* Hortonworks o donde Tez esté disponible).
- Spark (en algunos sistemas)

Estos modos determinan cómo se ejecutarán las instrucciones de Pig en el cliente o clúster. Por lo general, se utiliza el modo local para el desarrollo y las pruebas y uno de los modos *mapreduce*, *tez* o *spark* para las aplicaciones de producción.

Grunt: El Shell de Pig

El *shell* interactivo de programación de Pig se llama **`grunt`** (como ya habrás notado, el proyecto Pig está lleno de referencias porcinas). Usando el **`shell grunt`**, puedes introducir sentencias Pig Latin de forma interactiva. Cada sentencia es analizada e interpretada a medida que se introduce, y la ejecución comienza cuando se solicita la salida, ya sea a la consola o a un directorio de salida. Este proceso se llama evaluación perezosa.

Grunt es útil para el desarrollo, ya que permite muestrear e inspeccionar los datos mientras se desarrolla una rutina. La figura 2 muestra un ejemplo de `shell grunt`.

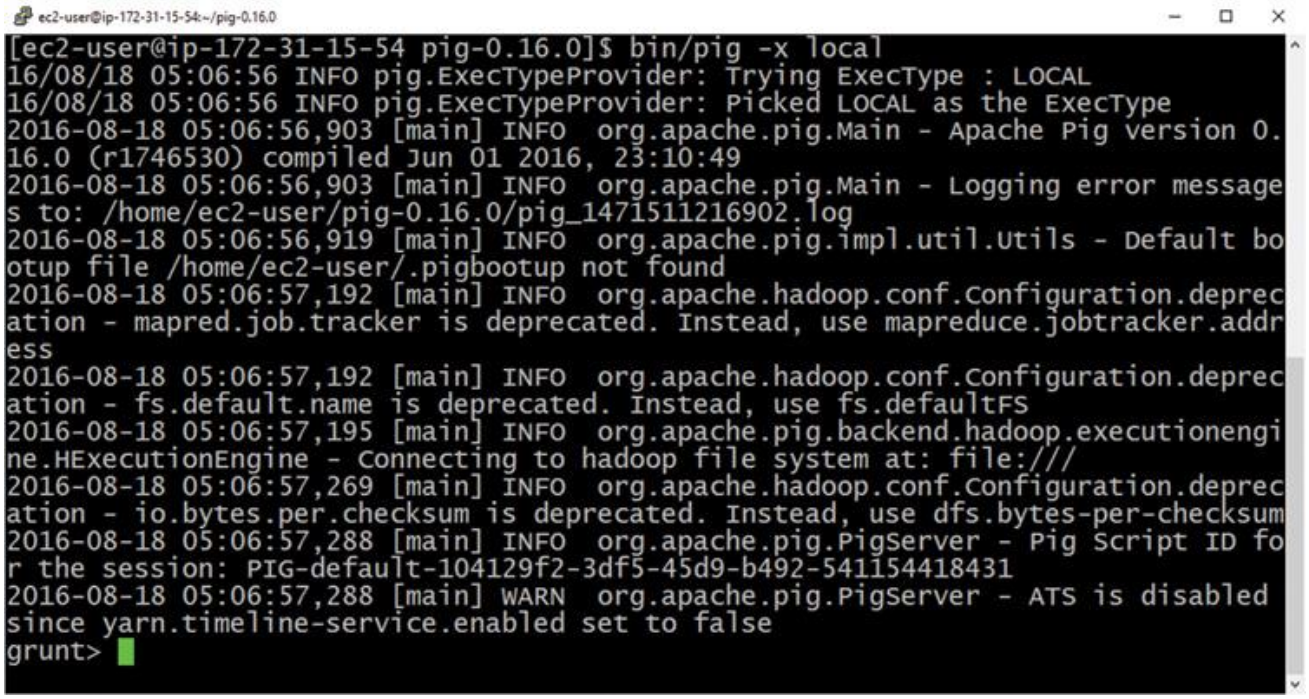


Figura 2 Shell de Grunt.

Pig también puede ejecutarse en modo no interactivo o por lotes, es decir, escribiendo las sentencias en un archivo y lanzándolo con el intérprete posteriormente.

Conceptos básicos de Pig Latin

Como ya hemos dicho, Pig Latin es un lenguaje de flujo de datos; no es un lenguaje SQL ni un lenguaje orientado a objetos. Los programas de Pig Latin son una secuencia de sentencias que siguen el siguiente patrón:

1. Cargar los datos en un conjunto de datos con nombre.

2. Crear un nuevo conjunto de datos manipulando el conjunto de datos de entrada.

3. Repita el paso 2 n veces, creando un conjunto de datos cada vez (estos conjuntos de datos intermedios no son necesariamente materializados).

4. Enviar el conjunto de datos final a la pantalla o a un directorio de salida.

Estructuras de datos de Pig

Los conjuntos de datos utilizados por Pig se denominan relaciones o bolsas. Las bolsas contienen registros llamados tuplas. Las tuplas contienen campos, y los campos pueden contener estructuras de datos como otras bolsas o tuplas, o datos atómicos llamados átomos. Las estructuras de datos de Pig se describen en la Tabla 1.

Type	Description	Example
`tuple`	ordered set of fields	('Jeff',47)
`bag`	unordered collection of tuples	{('Jeff',47),('Paul',44)}
`map`	set of key value pairs	[name#Jeff]

Tabla 1 Estructuras de datos de Pig

Los *bags*, tuplas y mapas tienen una notación distintiva como se ve en el ejemplo. Esta notación se mostrará en la consola cuando se devuelvan los datos.

Identificadores de objetos

Los *bags*, los mapas, las tuplas y los campos dentro de las tuplas suelen tener identificadores con nombre. Un ejemplo de esto se muestra en el Listado 2, donde se crea un *bag* a partir de un conjunto de datos de entrada, se le asigna un identificador a la bolsa (estudiantes) y se aplica un esquema a las tuplas en la bolsa con cada campo asignado un identificador (nombre, edad, gpa).

```
estudiantes = LOAD 'estudiante' USING PigStorage() AS (nombre:
chararray, edad: int, nota: float);
```

Listado 2 Identificadores en Pig

Los nombres de los identificadores distinguen entre mayúsculas y minúsculas y no pueden coincidir con ninguna de las palabras clave del lenguaje Pig Latin

Aparte de no usar palabras clave para nombrar objetos, las otras reglas para los identificadores son

- Debe empezar por una letra
- Sólo puede incluir letras, números y guiones bajos

Listado 4 Identificadores válidos e inválidos Pig.

```
/* ¡identificadores inválidos! */

$nombres = LOAD 'nombres';

1nonmbres = LOAD 'nombres';

/* identificadores válidos */

Nombres_estudiantes = LOAD 'nombres';

nombres_1 = LOAD 'nombres';
```

Tipos de datos simples de Pig

Los tipos de datos simples utilizados en el modelo de programación de Pig se enumeran en la Tabla 2.

Type	Description	Example
int	32-bit signed integer	20
long	64-bit signed integer	20L
float	32-bit floating point	20.5F
double	64-bit floating point	20.5
chararray	UTF-8 string	'Jeff'
bytearray	uninterpreted byte array	*<BLOB>*
boolean	True or False	true
datetime	datetime	1970-01-01T00:00:00.000+00:00
biginteger	Java BigInteger	200000000000
bigdecimal	Java BigDecimal	33.456783321323441233442

Tabla 2 Tipos de datos simples de Pig

Pig soporta operadores de igualdad y relacionales, similares a muchos lenguajes de programación comunes, que se utilizan principalmente en operaciones de filtrado. Estos operadores se describen en la Tabla 3.

Operator	Notation
equal	==
not equal	!=
less than	<
greater than	>
less than or equal to	<=
greater than or equal to	>=
pattern matching	matches

Tabla 3 Operadores relacionales de Pig

Además, como es de esperar, Pig soporta el conjunto estándar de operadores matemáticos que se describen en la Tabla 4.

Operator	Notation
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo	%

Tabla 4 Operadores matemáticos de Pig

Declaraciones, comentarios y convenciones en Pig

Hay algunos requisitos diferentes y convenciones comunes cuando se trata de la programación de Pig. Primero, las sentencias deben terminar con un punto y coma, aunque pueden abarcar varias líneas y pueden incluir sangría. En segundo lugar, las sentencias siempre comienzan asignando un conjunto de datos a un *bag*, ya sea mediante el proceso de carga de datos o la manipulación de un *bag* previamente definido. La única excepción es cuando se requiere una salida, que suele ser la última línea de un programa Pig, utilizando una sentencia DUMP o STORE.

Las palabras clave (como LOAD, STORE, FOREACH, etc.) se escriben en mayúsculas por convención, pero por lo demás no distinguen entre mayúsculas y minúsculas (recuerde que, sin embargo, no pueden utilizarse como identificadores). También tenga en cuenta que muchas de las funciones incorporadas en Pig (como COUNT o SUM) son sensibles a las mayúsculas y minúsculas.

Pig soporta comentarios en línea usando el doble guión (--) así como comentarios en bloque usando la notación /* */. Un ejemplo de esto se muestra en el Listado 5.

Listado 5 Comentarios en Pig

```
-- Este es un comentario en línea
/*
Esto es un comentario en bloque
*/
```

Carga de datos en Pig

La carga de datos en un *bag* es el primer paso de cualquier programa Pig. Pig utiliza funciones de carga que determinan cómo se extrae un esquema de los datos. Algunas de las funciones de carga clave en Pig se resumen en la Tabla 5.

Function	Syntax	Description
PigStorage (default)	`PigStorage([delim])`	Loads and stores data as structured text files
TextLoader	`TextLoader()`	Loads unstructured data in UTF-8 format
JsonLoader	`JsonLoader([schema])`	Loads JSON data

Tabla 5 Funciones de carga comunes de Pig

La función de carga se suministra utilizando la cláusula USING. Si se omite la cláusula USING, se asume que los datos son datos de texto delimitados por tabulaciones.

En el Listado 6 se muestra un ejemplo de sentencia LOAD para un nuevo conjunto de datos delimitado por comas utilizando la función PigStorage.

Listado 6 Función LOAD de PigStorage

```
stations = LOAD 'stations' USING PigStorage(',') AS
            station_id,
            name,
            lat,
            long,
            dockcount,
            landmark,
            installation;
```

Hay muchas otras funciones de carga adicionales en Pig, incluyendo HBaseStorage, AvroStorage, AccumuloStorage, OrcStorage, y más, utilizadas para archivos de almacenamiento en columnas o almacenamiento de datos NoSQL. Nosotros utilizaremos principalmente PigStorage, el predeterminado, para todos nuestros ejemplos.

Hay muchas permutaciones para cargar datos inicialmente como las siguientes:

- Carga de datos sin un esquema definido.
- Carga de datos con un esquema (campos con nombre) pero sin tipos de datos asignados.
- Carga de datos con un esquema y tipos de datos asignados.

Esto se demuestra en el Listado 7.

Listado 7 Suministro de esquemas para bolsas en Pig

```
--Cargar datos sin esquema definido
stations = LOAD 'stations' USING PigStorage(',');
/*
Los elementos son accesibles usando su posición relativa, p.e. $0
Todos los campos son del tipo bytearray
*/
-- Cargar datos nominados sin tipo definido
stations = LOAD 'stations' USING PigStorage(',') AS
           (station_id,name,lat,long,dockcount,landmark,installation);
/*
Los elementos son accesibles usando su identificador, p.e. name
*/


--Carga por nombre y tipada
```

```
stations = LOAD 'stations' USING PigStorage(',') AS
           (station_id:int, name:chararray, lat:float,
           long:float, dockcount:int, landmark:chararray,
           installation:chararray);
/*

Los elementos son accesibles usando su identificador, p.e. name

*/
```

Cargar datos sin tener que proporcionar un esquema o tipos de datos puede ser útil cuando se explora un conjunto de datos por primera vez. Sin embargo, en general se recomienda suministrar esta información si se conoce, ya que permitirá una mejor optimización del programa.

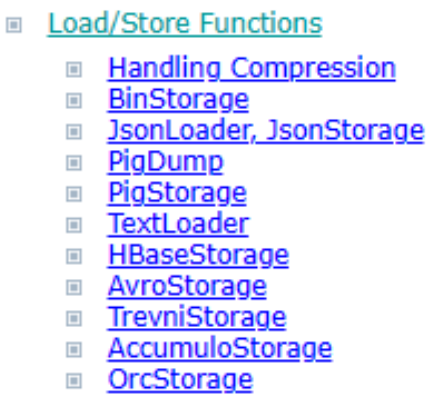
Almacenar

El operador Store se utiliza para almacenar el conjunto de resultados de Pig Latin en Hadoop HDFS o en un sistema de archivos local.

Sintaxis:

`STORE alias INTO 'directory' [USING function];`

- `alias` Representa el nombre de la relación.
- `INTO` 'directory' Representa el nombre del directorio de almacenamiento donde se copiará el conjunto de resultados.
- `USING` function Se pueden utilizar funciones Store como BinStorage para el formato legible por máquina, JsonLoader para datos JSON, si no se utilizan estas funciones se utilizará la función Store de PigStorage como función por defecto. Algunos de ellos son los de la imagen:



Más información en sobre formatos de almacenamiento:
https://pig.apache.org/docs/r0.17.0/func.html#load-store-functions

Por ejemplo, si cargamos el archivo "employee.txt" desde HDFS a Pig y luego almacenaremos el conjunto de resultados de Pig Latin en la ubicación de HDFS '/pigexample/'.

Veamos este proceso usando los siguientes pasos.

Cargamos el fichero y lo mostramos por pantalla:

```
employees = LOAD '/pigexample/employee.txt' USING PigStorage(',') as (empid:i
nt,firstname:chararray,lastname:chararray,city:chararray,county:chararray );


dump employees
```

Almacenamos el resultado separando los campos por comas en la carpeta **/pigexample/employeedata**:

```
STORE employees INTO '/pigexample/employeedata' USING PigStorage (',');
```

Filtrar, proyectar y ordenar datos con Pig

Una vez que se tiene un *bag* "cargado" con datos (virtualmente, no físicamente, por supuesto), generalmente se comienza a manipular. Veamos algunas operaciones simples con Pig.

Sentencia FILTER

La operación más básica es simplemente filtrar datos. En este caso, el esquema no se modifica: las tuplas que coinciden con un criterio determinado se conservan en el *bag* de destino, mientras que las tuplas que no satisfacen la condición de filtrado no se conservan. El filtrado de tuplas en Pig se realiza mediante la sentencia FILTER. Un ejemplo sencillo se muestra en el Listado 8.

Listado 8 Sentencia FILTER

```
sj_stations = FILTER stations BY landmark == 'San Jose';
```

Proyección y transformación de campos con FOREACH

Proyectar o transformar campos dentro de tuplas en bolsas en Pig se logra usando la sentencia FOREACH. FOREACH puede considerarse como el equivalente en Pig de una operación *map* en un trabajo MapReduce; de hecho, estas operaciones se ejecutarían inevitablemente utilizando funciones *map*. Cada tupla de una bolsa dada se itera realizando la operación solicitada, que podría incluir lo siguiente

- Eliminar un campo o campos; por ejemplo, proyectar 3 campos seleccionados de una bolsa de entrada con 4 campos.
- Añadir un campo o campos; por ejemplo, añadir un campo calculado basado en los datos de la bolsa de entrada.
- Transformar un campo o campos; por ejemplo, realizar una manipulación en el lugar de un campo, como convertir un campo de matriz de caracteres en minúsculas.
- Realización de una función agregada en un campo complejo; por ejemplo, una operación COUNT contra un campo que es en sí mismo una bolsa.

El listado 9 muestra un ejemplo de una operación FOREACH para proyectar campos específicos en una bolsa de destino en Pig.

Listado 9 Declaración FOREACH

```
station_ids_names = FOREACH stations GENERATE station_id, name;
```

Sentencia ORDER

Ordenar las tuplas por un campo determinado es otro requisito común del procesamiento y análisis de datos de Pig. Sabiendo lo que se sabe sobre MapReduce, se puede deducir que esto aprovechará la función Shuffle and Sort del framework MapReduce a nivel físico.

La ordenación en Pig se realiza mediante la sentencia ORDER BY. En el Listado 10 se muestra un ejemplo de ordenación de una bolsa por un campo determinado utilizando este operador.

Listado 10 Sentencia ORDER

```
ordered = ORDER station_ids_names BY name;
```

Sentencia DESCRIBE

Una vez creadas y manipuladas las bolsas en un programa Pig, a menudo es útil inspeccionar el esquema de una bolsa. Una simple inspección de una bolsa en cualquier etapa de un programa Pig puede realizarse utilizando la sentencia DESCRIBE. DESCRIBE mostrará los campos y sus tipos de una relación dada (bolsa). Esto no requiere ni invoca la ejecución. En el Listado 11 se muestra un ejemplo de la sentencia DESCRIBE.

Listado 11 Sentencia DESCRIBE

```
DESCRIBE stations;

stations: {station_id: int,name: chararray,lat: float,long: float,

dockcount: int,landmark: chararray,installation: chararray}
```

Funciones integradas en Pig

Pig incluye varias funciones integradas comunes; éstas típicamente operan contra un campo y se usan con el operador FOREACH. En el Listado 13 se muestra un ejemplo de uso de una función incorporada.

Listado 13 Funciones incorporadas en Pig

```
lcase_stations = FOREACH stations GENERATE station_id, LOWER(name),

                 lat, long, landmark;
```

Hay funciones integradas para las operaciones más comunes sobre cadenas de texto, funciones de fecha y funciones numéricas. El listado 14 enumera algunas de las principales funciones incorporadas incluidas en Pig. Muchas de ellas son análogas a las funciones incluidas en la mayoría de los dialectos de SQL. Hay funciones adicionales disponibles en las bibliotecas de la comunidad, o pueden escribirse funciones personalizadas.

Listado 14 Algunas funciones comunes integradas en Pig

```
-Funciones de evaluación
AVG, COUNT, MAX, MIN, SIZE, SUM, TOKENIZE
--Funciones matemáticas
ABS, CEIL, EXP, FLOOR, LOG, RANDOM, ROUND
--Funciones de cadena
STARTSWITH, ENDSWITH, LOWER, UPPER, LTRIM, RTRIM, TRIM, REGEX_EXTRACT
--Funciones de fecha y hora
CurrentTime, DaysBetween, GetDay, GetHour, GetMinute, ToDate
```