FIAP GRADUAÇÃO

BANCO DE DADOS

ARQUITETURA DE BI E BIG DATA

PROF. MILTON

Versão 2.1 – <abril de 22>

FIAP

PIG

FIAP

Apache PIG

- Pig começou originalmente como um projeto de pesquisa no Yahoo! em 2006 com o objetivo de simplificar a criação e execução de tarefas **MapReduce** para processamento de grande volume de dados.

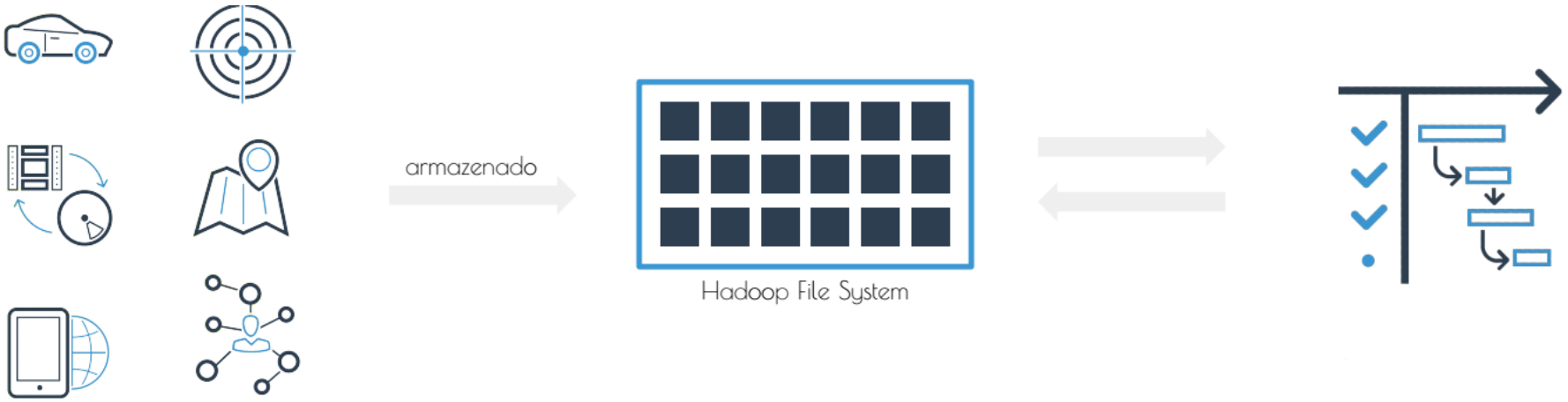
- Em 2007 o Pig foi incubado pela Apache e o primeiro release foi liberado em setembro de 2008. Um ano depois o Pig passou a ser um subprojeto do Apache Hadoop.

- No final de 2009 metade das tarefas em execução no Yahoo! eram tarefas Pig. Em 2010, o Pig se tornou um projeto top-level da Apache.

Onde PIG pode ser usado?

- Apache Pig é comumente utilizado:
  - Extração, Transformação e Carga (ETL)
  - Pesquisa nos dados brutos
  - Processamento iterativo

Onde PIG pode ser usado?



- Na maioria dos casos, o dado original precisa ser extraido, transformado e carregado (ETL) de volta no HDFS.
- Isso inclui processo de normalização dos dados como por exemplo, dividir uma string, definir delimitadores, agregar dados e etc.
- Este processo pode ser automatizado através de uma série de scripts Pig.

FIAP

Filosofia do Apache PIG

- Porcos comem qualquer coisa
  - Pig pode operar sobre quase qualquer tipo de dado com ou sem metadados. Os dados podem ser não estruturados, estruturados ou aninhados. Pode operar não só em arquivos, mas em bancos de dados.
- Porcos vivem em qualquer lugar
  - Pig foi criado para processamento paralelo dos dados. Ele foi inicialmente projetado para rodar sobre o Hadoop. Mas essa não precisa ser a única opção.

FIAP

Filosofia do Apache PIG

- Porcos são animais domésticos
  - Pig foi projetado para ser facilmente controlado e modificado. Pig é muito versátil e suporta uma grande variedade de funções definidas pelos usuários. Além disso, ele pode ser integrado com Java ou Python.
- Porcos voam
  - Pig processa os dados rapidamente.

FIAP

Apache PIG

- Map Reduce é muito poderoso mas
  - Exige bom conhecimento de linguagens de programação como, por exemplo, Java.
  - O usário tem que que reinventar funcionalidades como JOIN, FILTER, GROUP, SORT.
- Pig fornece as operações padrão de processamento de dados
  - Pig Latin oferece as operações: Join, Filter, Group By, Order By, Union, etc. implementadas e testadas. Algumas delas podem ser obtidas do MapReduce direta ou indiretamente. No entanto, algumas delas precisam ser implementadas pelos usuários.

FIAP

Apache PIG

- Apache Pig é uma abstração sobre MapReduce.
- É uma ferramenta / plataforma que é usada para analisar grande conjuntos de dados.
- Normalmente usado com Hadoop.
- Podemos executar todas as operações de manipulação de dados no Hadoop usando o Apache Pig.

FIAP

Apache PIG

- Pig Latin é a linguagem de alto nível do Apache Pig.
- Essa linguagem fornece vários operadores que permitem que o desenvolvedor crie suas próprias funções para leitura, escrita e processamento de dados.

Apache PIG

- Os desenvolvedores criam scripts em Pig Latin.
- O Apache Pig internamente converte esses scritps em tarefas de Map e Reduce.
- O componente Pig Engine lê os scripts em Pig Latin e os converte em Jobs MapReduce.

FIAP

PIG Latin

- Sendo uma linguagem de alto nível, Pig Latin, aumenta a produtividade:
  - 10 linhas em Pig Latin ≅ 200 linhas em Java.
  - 4 horas de desenvolvimento JAVA podem ser reduzidas para 15 minutos em Pig Latin.
- Abre o ambiente para programadores non-Java
- Fornece operações como JOIN, GROUP, FILTER e SORT.

PIG Latin

- É uma linguagem de fluxo de dados (data flow language)
  - Não é procedural
  - Não é declarativa
- Binários e códigos dos usuário podem ser incluídos em quase qualquer lugar.
- Metadados não são necessários mas são usados caso existam
- Suporta tipo de dados aninhados
- Trabalha com arquivos em HDFS.

PIG Latin

- As declarações PIG Latin geralmente são organizadas da seguinte maneira:
  - Uma instrução LOAD lê dados do sistema de arquivos.
  - Uma série de declarações de "transformação" processam os dados.
  - Uma instrução STORE grava saída no sistema de arquivos; ou, uma declaração DUMP exibe saída para a tela.

PIG Latin

FIAP

```
visits       = load '/data/visits' as (user, url, time);
gVisits      = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo      = load '/data/urlInfo' as (url, category, pRank);
visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

PIG Engine

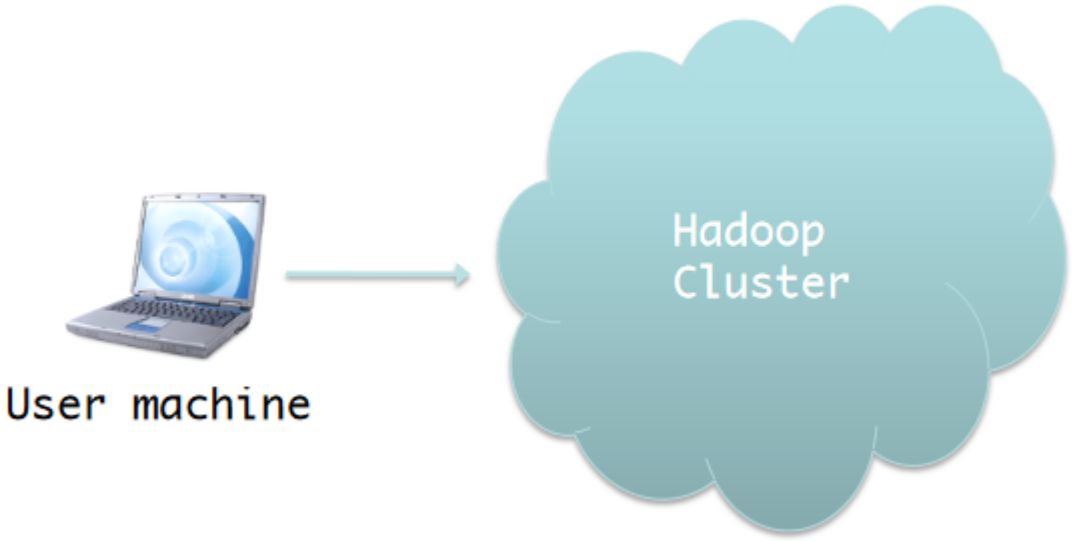
- Apache Pig fornece um Engine que roda sobre o Hadoop.
  - O usuário não precisa ajustar o Haddop para suas necessidades.
  - Isola o usuário das alterações na interface do Hadoop

Acessando PIG

- Submeta o script diretamente.
- Use PIG Shell (Grunt)
- Use a interface JDBC através da classe Java PigServer.
- Use PigPen, um plugin do Eclipse
  - Permite criação de scripts usando as ferramentas gráficas ou texto.
  - Permite amostrar dados e exibir fluxo de dados.

Componentes

- Pig fica na máquina do usário.
- O job é executa do Cluster
- Não é preciso instalar componentes no Cluster Hadoop.



Como Funciona

```
A = LOAD 'population.csv'
    USING PigStorage (',');
B = FILTER A BY $0 > 1990;
C = GROUP B BY $1;
D = FOREACH C GENERATE group
    AS idade: int,
    COUNT(B.$3) AS total: int;
STORE D INTO 'saida';
```

pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

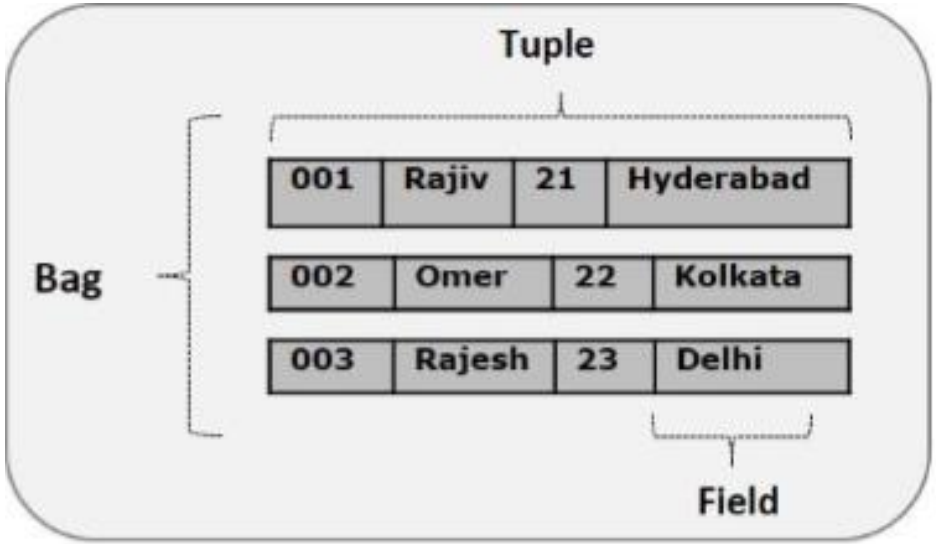
Execution Plan

Map:
Filter

Reduce:
Count

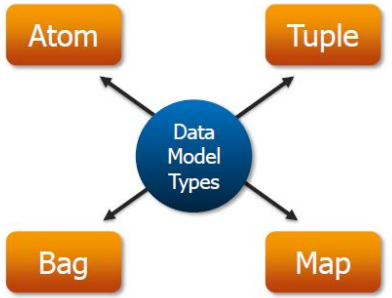
Modelo de Dados

- O modelo de dados do Apache Pig é totalmente encadeado (*fully nested*) e permite tipos de dados não-atômicos como **map** e **tupla**.



Modelo de Dados

- Átomo – qualquer valor único, independente do tipo de dados
  - Exemplo: ‘30’ ou ‘Escola’
- Tupla – um conjunto ordenado de campos
  - Exemplo: (Escola, 30)
- Bag – um conjunto não-ordenado de tuplas
  - Exemplo: {(Escola, 30),(Posto, 20)}
- Map – um conjunto de chave-valor
  - Exemplo: [Predio#Escola, Numero#30]



Tipo de Dados Simples

FIAP

Tipo de dado	Descrição	Exemplo
int	inteiro de 32 bits	10
long	inteiro de 64 bits	10L ou 10l
float	ponto flutuante de 32 bits	10.5F 10.5f 10.5e2f 10.5E2F
double	ponto flutuante de 64 bits	10.5 10.5e2 10.5E2
chararray	array de caracteres	olá
bytearray	blob ou um array de bytes	

FIAP

Tipo de Dados Complexos

Tipo de dado	Descrição	Exemplo
map	Um conjunto de chaves e valores	['name'#'bob', 'age'#55]
tuple	Um conjunto de campos ordenados	('bob', 55)
bag	uma coleção de tuplas	{('bob', 55), ('sally', 52), ('john', 25)}

FIAP

Esquemas

- Pig não exige a definição explícita de um esquema. Isso faz parte da filosofia do Pig em que Porcos comem qualquer coisa.
- Se for definido um esquema o Pig utilizará. Caso contrário, o Pig processará os dados e fará uma “adivinhação” do tipo do dado.

Esquema

- Leitura de dados sem a definição do esquema

```
pop = LOAD 'population.csv';
```

- Definição explícita do esquema sem a definição dos tipos de dados

```
pop = LOAD 'population.csv' AS (ano, idade,
sexo, qt);
```

- Definição explícita do esquema com a definição dos tipos de dados

```
pop = LOAD 'population.csv' AS (ano:int,
idade:int, sexo:int, qt:int);
```

FIAP

Sintaxe na definição do esquema

Tipo de dado	Sintaxe	Exemplo
int	int	as (a:int)
long	long	as (a:long)
float	float	as (a:float)
double	double	as (a:double)
chararray	chararray	as (a:chararray)
bytearray	bytearray	as (a:bytearray)
map	map[] ou map[*tipo*]	as (a:map[], b:map[int])
tuple	tuple() ou tuple(*lista de campos*)	as (a:tuple(), b:tuple(x:int,y:int))
bag	bag{} ou bag{ t:(*lista de campos*) }. A tuple dentro da bag deve ter um nome (no exemplo: t) mesmo você nunca podendo acessar esta tuple diretamente.	(a:bag{}, b:bag{t:(x:int,y:int)})

Censo

- Os dados destes exemplos são do Censo do EUA

1850	0	1	1483789
1850	0	2	1450376
1850	5	1	1411067
1850	5	2	1359668
1850	10	1	1260099

- A primeira coluna indica o ano do censo.
- A segunda coluna indica a idade do cidadão
- A terceira o seu sexo
- A última indica o número de pessoas.
- A linha um indica que, em 1850, existiam 1.483.789 pessoas do sexo masculino com idade inferior a cinco anos.

Relação

- Uma relação no Pig é uma coleção de tuplas.
- Em uma relação no Pig, cada tupla corresponde a uma linha de uma tabela relacional.
- No entanto, uma relação no Pig não exige que todas as tuplas tenham o mesmo número de campos nem que os campos na mesma posição sejam do mesmo tipo.
- Além disso, relações no Pig não são ordenadas.
- Não existe garantia de que as tuplas serão processadas em alguma ordem particular.

FIAP

Pig latin

- Cada processamento resulta em uma nova relação. Apesar do nome das relações parecerem variáveis, elas não são variáveis.
- Além de nome para as relações, o Pig também possui nome para os campos (colunas). Tanto o nome das relações quanto dos campos devem ser iniciados com um caractere do alfabeto.
- Palavras reservadas no Pig não são case-sensitive. Por exemplo, UPPER é equivalente a upper. Porém, o nome das relações e campos são case-sensitive. Assim, o campo **pop** é diferente de **Pop**.

Comando Shell básicos do GRUNT

- Help
  - `pig -h`
- Iniciando o pig localmente
  - `pig -x local`
- Pig suporta comandos HDFS
  - `grunt> pwd`
  - put, get, cp, ls, mkdir, rm, mv, etc.

LOAD

- Lendo dados a partir de uma tabela no HBase

```
pop = LOAD ‘population' USING HBaseStorage();
```

- Lendo dados textuais utilizando a vírgula como separador

```
pop = LOAD 'population.csv' USING PigStorage(',')
AS (ano:int, idade:int, sexo:int, qt:int);
```

STORE

- Armazena a relação **pop** no HBase

```
STORE pop INTO 'pop' USING HBaseStorage();
```

- Armazena a relação **pop**. Por padrão, o Pig armazena os dados no HDFS em um arquivo delimitado por tab
- `STORE pop INTO 'popu.txt';`

Describe e Illustrate

- DESCRIBE mostra o esquema e o tipo inferido do dado

```
DESCRIBE pop;
```

- O comando ILLUSTRATE imprime o esquema da relação e um exemplo do dado

```
ILLUSTRATE pop;
```

Operadores Relacionais

- FOREACH
  - Aplica expressões para cada registro em uma **bag**
- FILTER
  - Filtra por expressão
- GROUP
  - Agrupa registros com a mesma chave
- ORDER BY
  - Classifica dados
- DISTINCT
  - Remove duplicidades

FOREACH . . .GENERATE

- FOREACH pega um conjunto de expressões e as aplica em todos os registros
- Sintaxe Básica:

```
alias2 = FOREACH alias1 GENERATE expression;
```

- Examplo:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)

alias2 = FOREACH alias1 GENERATE col1, col2;

DUMP alias2;
(1,2) (4,2) (8,3) (4,3) (7,2) (8,4)
```

FILTER. . .BY

- Use o operador FILTER para restringir tuplas or linhas de dados
- Sintaxe Básica:

```
alias2 = FILTER alias1 BY expression;
```

- Examplo:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)
alias2 = FILTER alias1 BY (col1 == 8) OR (NOT
   (col2+col3 > col1));
DUMP alias2;
(4,2,1) (8,3,4) (7,2,5) (8,4,3)
```

GROUP. . .ALL

- Use o operador GROUP…ALL para agrupar dados
  - Use GROUP quando apenas uma relação estiver envolvida
  - Use COGROUP quando múltiplas relaçoes estiverem envolvidas
- Sintaxe Básica :

```
alias2 = GROUP alias1 ALL;
```

- Exemplo:

```
DUMP alias1;
(John,18,4.0F) (Mary,19,3.8F) (Bill,20,3.9F)
   (Joe,18,3.8F)
alias2 = GROUP alias1 BY col2;
DUMP alias2;
(18,{(John,18,4.0F),(Joe,18,3.8F)})
(19,{(Mary,19,3.8F)})
(20,{(Bill,20,3.9F)})
```

ORDER. . .BY

- Use o operador ORDER…BY ordenar uma relação baseada em um ou mais campos
- Sintaxe básica:

```
alias = ORDER alias BY field_alias [ASC|DESC];
```

- Exemplo:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)
alias2 = ORDER alias1 BY col3 DESC;
DUMP alias2;
(7,2,5) (8,3,4) (1,2,3) (4,3,3) (8,4,3) (4,2,1)
```

DISTINCT. . .

- Use o operador DISTINCT para remover tuplas duplicadas em uma relação.
- Sintaxe básica:

```
alias2 = DISTINCT alias1;
```

- Exemplo:

```
DUMP alias1;
(8,3,4) (1,2,3) (4,3,3) (4,3,3) (1,2,3)
alias2= DISTINCT alias1;
DUMP alias2;
(8,3,4) (1,2,3) (4,3,3)
```

INNER JOIN Exemplo

```
DUMP Alias1;
    (1,2,3)
    (4,2,1)
    (8,3,4)
    (4,3,3)
    (7,2,5)
    (8,4,3)
DUMP Alias2;
    (2,4)
    (8,9)
    (1,3)
    (2,7)
    (2,9)
    (4,6)
    (4,9)
```

```
Join Alias1 by Col1 to
Alias2 by Col1
    Alias3 = JOIN Alias1 BY
    Col1, Alias2 BY Col1;

Dump Alias3;
    (1,2,3,1,3)
    (4,2,1,4,6)
    (4,3,3,4,6)
    (4,2,1,4,9)
    (4,3,3,4,9)
    (8,3,4,8,9)
    (8,4,3,8,9)
```

Exemplo Prático

```
tweets = LOAD 'united_airlines_tweets.tsv' USING PigStorage('\t')
    AS (id_str:chararray, tweet_url:chararray,
created_at:chararray,
    text:chararray, lang:chararray, retweet_count:int,
favorite_count:int,
    screen_name:chararray);

dictionary = LOAD 'dictionary.tsv' USING PigStorage('\t') AS
(word:chararray,
score:int);


 english_tweets = FILTER tweets BY lang == 'en';
```

Exemplo Prático

```
tokenized = FOREACH english_tweets GENERATE id_str,
 FLATTEN( TOKENIZE(text) ) AS word;
clean_tokens = FOREACH tokenized GENERATE id_str,
 LOWER(REGEX_EXTRACT(word, '[#@]{0,1}(.*)', 1)) AS word;
```

```
token_sentiment = JOIN clean_tokens BY word, dictionary BY word;
```

```
sentiment_group = GROUP token_sentiment BY id_str;
```

```
sentiment_score = FOREACH sentiment_group GENERATE group AS id,
SUM(token_sentiment.score) AS final;
```

A função TOKENIZE separa o texto em palavras.
A função FLATTEN extrai o resultado de uma coleção de tuplas em uma coleção simples

Exemplo Prático

```
classified = FOREACH sentiment_score GENERATE id,
( (final >= 0)? 'POSITIVE' : 'NEGATIVE' )
AS classification, final AS score;


 final = ORDER classified BY score DESC;


 STORE final INTO 'sentiment_analysis';
```