Linguagens de programação para ciência de dados (*Python* com *Spark*)

WBA0753 v1.0

Autoria do Desafio Profissional: Danilo Rodrigues Pereira

Leitor Crítico: Adriano Thomaz

Proposta de Resolução

A quantidade de dados criados e armazenados globalmente continua crescendo a cada ano. Essa constante criação de dados pelas mídias sociais, aplicativos de negócios e telecomunicações e vários outros domínios está levando à formação de Big Data (CHAMBERS, 2018). Entretanto, não é a quantidade de dados disponíveis que importa, mas sim o que vamos fazer com eles para conseguir extrair informações importantes (conhecimentos).

A resolução deste desafio profissional consiste em propor um pipeline para extrair informações dos arquivos de logs Apache.

PIPELINE PROPOSTO

1. **Análise dos arquivos de *logs***

   O objetivo dessa etapa é analisar e entender quais arquivos um servidor *Apache* disponibiliza; são eles:

   - *Security Warning* – Contém informações sobre segurança e permissões de usuários que podem gravar ou acessar algum diretório no servidor.
   - *Error Log* – É o arquivo de log mais importante do servidor *Apache*. Nesse arquivo são armazenadas informações de diagnóstico, e serão registrados os erros encontrados no processamento de solicitações.
   - *Access Log* – Registra todas as solicitações processadas pelo servidor. Uma configuração típica para o *log* de acesso é o formato Common Log Format (CLF).
   - *Script Log* – Para ajudar na depuração, a diretiva ScriptLog permite gravar a entrada e a saída dos scripts CGI. Esse arquivo é útil apenas em ambientes de testes.

2. Leitura dos arquivos

O objetivo dessa etapa é fazer a leitura dos arquivos de logs usando a biblioteca PySpark da API *Spark Apache*. Segue um exemplo de código:

Quadro 1 | Exemplo de leitura de arquivo de *log Apache* usando *PySpark*.

```
from pyspark import SparkContext
from pyspark.sql import SQLContext


sc = SparkContext()

sqlContext= SQLContext(sc)


logs = sqlContext.read.text("<caminho_do_diretorio_dos_logs>")


# Outra opção
logs = sc.textFile(<caminho_do_diretorio_dos_logs>')
```

3. Função para extração de informações de um arquivo de *log* do *Apache* do tipo *Acess Log*, usando expressão regular (REF)

Quadro 2 | Exemplo de uma função para extrair informações de arquivo de *log Apache* (*Acess log*)

```
import re
from pyspark.sql import Row

from pyspark import SparkContext
from pyspark.sql import SQLContext


sc = SparkContext()

sqlContext= SQLContext(sc)

# Retorna um dicionário contendo as informações do arquivo de log Apache (Access Log)

def parse_apache_acess_log_line(linha_log, pattern):

        busca = re.search(pattern, linha_log)

        if busca is None:

            # Exibindo mensagem de erro, pois não foi encontrado na linha nenhuma valores       #
    definido na expressão regular

            raise Exception("Linha Inválida. Verifica se o arquivo de log é correto: %s" % linha_log)

        else:

            ip_address    = busca.group(1)

            id_client = busca.group(2)
            id_user   = busca.group(3)

            date_time   =  busca.group(4)

            method      =  busca.group(5),

            end_point   =  busca.group(6)

            protocol    =  busca.group(7)

            response_code = int(busca.group(8))

            content_size  = long(match.group(9))
```

```
    return Row(ip_address, id_client, id_user , time_date, method, end_point , protocol,
response_code, content_size )


arquivos = "<caminho_do_diretorio_dos_logs>"

dados_arquivos = sc.textFile(arquivos)

dados_arquivos.count()

# Expressao regular para filtrar as informações do arquivo de log Apache (Acess Log)

LOG_ APACHE_ACCESS_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3})
(\d+)'


linhas = parse_apache_acess_log_line(dados_arquivos, LOG_ APACHE_ACCESS_PATTERN)

parsed = dados_arquivos.map(linhas)

# Criando uma Data Frame temporario com as informações de cada linha do log

parsed.toDF().registerTempTable("informacao_log")
```

4. Gerando relatórios utilizando análise estatística

Quadro 3 | Exemplo de código em *Python* para agrupar e contabilizar as informações do arquivo de *logs Apache* (*Acess Log*)

```
import re
from pyspark.sql import Row

from pyspark import SparkContext
from pyspark.sql import SQLContext


sc = SparkContext()

sqlContext= SQLContext(sc)

# Retorna um dicionário contendo as informações do arquivo de log Apache (Access Log)

def parse_apache_acess_log_line(linha_log, pattern):

        busca = re.search(pattern, linha_log)

        if busca is None:

            # Exibindo mensagem de erro, pois não foi encontrado na linha nenhuma valores
    # definido na expressão regular

            raise Exception("Linha Inválida. Verifica se o arquivo de log é correto: %s" %
    linha_log)

        else:

            ip_address    = busca.group(1)

            id_client = busca.group(2)
            id_user   = busca.group(3)

            date_time   =  busca.group(4)

            method     =  busca.group(5),

            end_point   =  busca.group(6)

            protocol    =  busca.group(7)

            response_code = int(busca.group(8))
```

```
        content_size  = long(match.group(9))


        return Row(ip_address, id_client, id_user , time_date, method, end_point , protocol,
    response_code, content_size )


    arquivos = "<caminho_do_diretorio_dos_logs>"

    dados_arquivos = sc.textFile(arquivos)

    dados_arquivos.count()

# Expressao regular para filtrar as informações do arquivo de log Apache (Acess Log)

LOG_ APACHE_ACCESS_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)"
(\d{3}) (\d+)'


linhas = parse_apache_acess_log_line(dados_arquivos, LOG_ APACHE_ACCESS_PATTERN)

parsed = dados_arquivos.map(linhas)


# Criando uma Data Frame temporario com as informações de cada linha do log

df_formatado = parsed.toDF().registerTempTable("informacao_log")

print "Agrupando as informações por host (IP)" +
df_formatado.groupBy('ip_address').count().filter('count = 1').select(' ip_address').show()


print "Quantificando os erros do tipo 404 foram gravados no log" +
df_formatado.groupBy('response_code').count().filter('response_code = "404"').show()


print "Quantificando o total de bytes no arquivo de log" +
```

Referências

CHAMBERS, B.; ZAHARIA, M. **Spark: The Definitive Guide:** Big Data Processing Made Simple. San Francisco: O'Reilly Media, 2018.

Bons estudos!