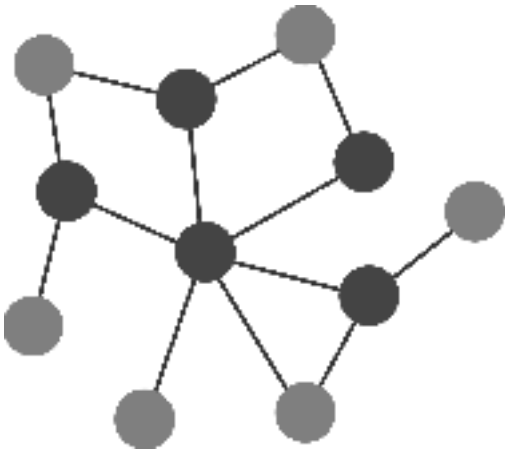
Megadados

Projeto 3 - PySpark e GraphFrames

Frederico Curti e Raphael Costa



Insper
São Paulo - 27 de Novembro de 2018

Introdução **2**

Tutorial **3**

Criando uma Instância 3

Criando o Snapshot 3

Criando e anexando um Volume 4

Criando um Bucket 5

Subindo o cluster 6

Adicionando as dependências do GraphFrames ao Cluster 6

Configurando o Zeppelin 7

Preparando os dados 7

Transformações para a criação dos nós 8

Introdução

O Objetivo desse projeto foi a adoção de uma tecnologia nova de processamento de dados em larga escala. Foi definido pelo grupo que a base de dados utilizada seria dados de páginas da Wikipédia. Tal decisão foi inspirada por uma brincadeira que o grupo vinha participando, no qual os participantes desta partem de uma página aleatória da enciclopédia digital, e através dos links entre artigos, todos devem alcançar uma página definida de antemão.

O dataset utilizado é o dataset Wikipedia Traffic Statistics Dataset, compilado pela Data Wrangling, LLC em 2009. Foi possível obtê-lo através do sistema de Public Snapshots do EC2, no qual ele se chama Wikipedia Page Traffic Statistics (Linux) - *snap-753dfc1c*. Nós o associamos à um volume, que pode ser montado por uma instância e acessado.

Ele é um dataset dividido em vários blocos, sendo estes:

- wikistats, que contém ~1TB de dados sobre acessos à wikipedia durante 7 meses,
- wikilinks, que contém um dataset compactado de ~1.1GB que indicam os links entre páginas
- wikidump, que contém ~30GB de dados com o conteúdo das páginas

Decidimos usar o wikilinks, que trazia os dados em .txt possíveis de serem transformados em um grafo, no qual os nós são as páginas e as arestas são links

presentes em cada artigo que referenciam outros artigos. O processamento desses dados permite descobertas interessantes como medidas de centralidade e relevância em redes, além de permitir encontrar o menor caminho entre dois nós, um achado relevante para a brincadeira que originou a ideia.

Tomada a decisão do escopo do projeto, buscamos por soluções que implementasse uma pipeline de processamento de dados para larga escala que fosse situada para grafos, e encontramos duas soluções compatíveis com o Spark, a ferramenta que tivemos contato durante o curso. A primeira que encontramos foi o GraphX, a biblioteca padrão do Spark para lidar com grafos. No entanto, ela não possui bindings para Python, ou seja, não funciona com o PySpark, só em Scala nativamente. Como alternativa, encontramos recomendações para o GraphFrames, uma dependência externa que funciona com python. O tutorial abaixo mostra as etapas que aplicamos para transformar os dados brutos em um grafo, e a partir deste, calcular métricas relevantes.

Tutorial

Criando uma Instância

Primeiro, é necessário criar uma instância no EC2 da AWS. Para este projeto, foi criada uma instância do tipo m4.large, porém, como foi utilizado um Snapshot e um Volume a parte (explicaremos depois), qualquer instância, inclusive a t2.micro que é de graça inicialmente funciona.

Modelo	vCPU*	Mem (GiB)	Armazenamento	Largura de banda dedicada do EBS (Mbps)	Performance de rede
m4.large	2	8	Somente EBS	450	Moderada

Tipo de Instância utilizada.

Criando o Snapshot

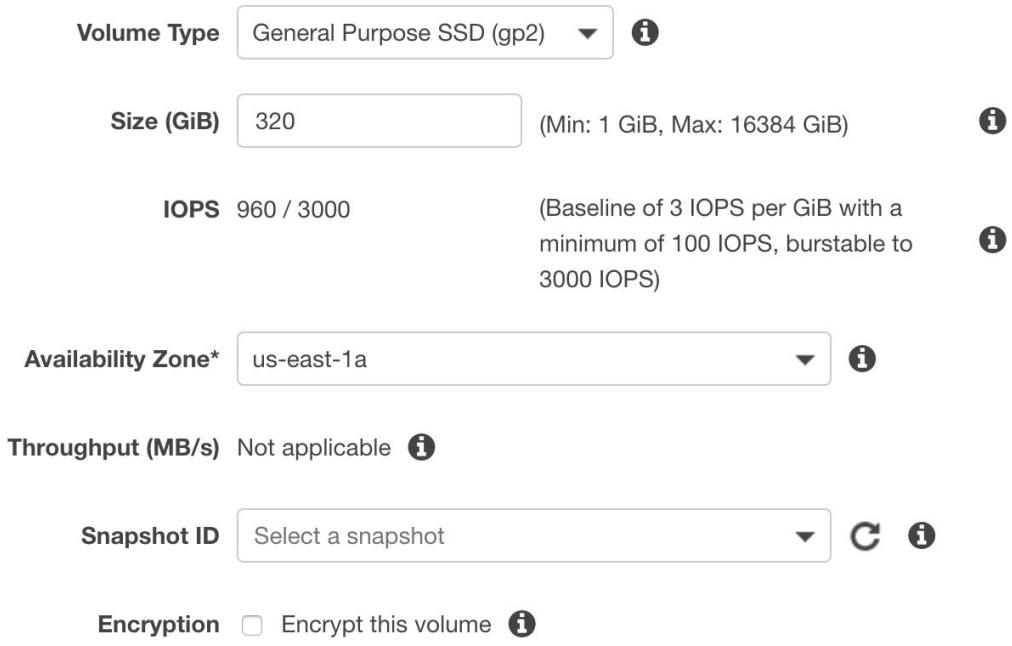
Após isto, é necessário criar um Volume da aws a partir de um Snapshot público. Isto é, "criar"um HD com o dataset a ser utilizado, disponibilizado pela própria amazon.

Vá até a aba de Snapshots e pesquise por Wikipedia Page Traffic Statistics, selecione a opção Wikipedia Page Traffic Statistics (Linux), de tamanho 320GB, e Crie um Snapshot.

Snapshot utilizado

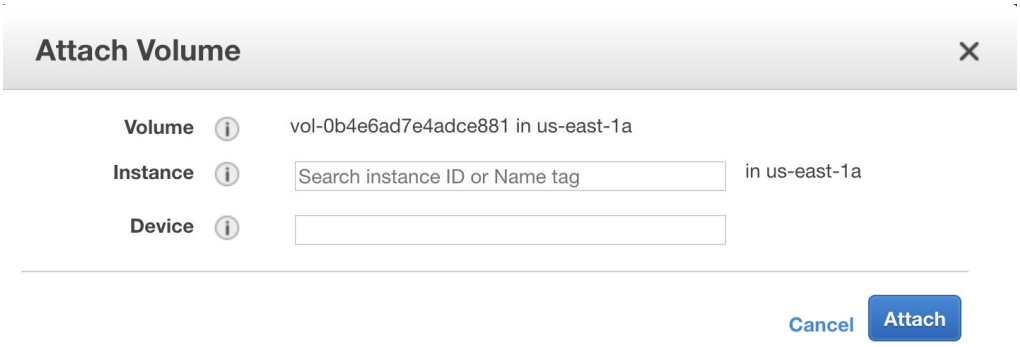
Criando e anexando um Volume

Assim, é necessário associar o snapshot a um Volume que será montado em sua instância. Basta criar um Volume pela EC2 e associar o snapshot criado no passo anterior com esse Volume, não esquecendo de utilizar a mesma região que o Snapshot e o tamanho necessário para que o Volume não apresente falhas.



Configurações do Volume e apresentação dos campos citados

Após a criação do Volume através de um Snapshot é necessário anexar este novo HD a sua instância. Assim, selecione o volume, clique em Actions e selecione Attach Volume, configurando para ser anexado á instância criada anteriormente.



Campo de anexação do Volume

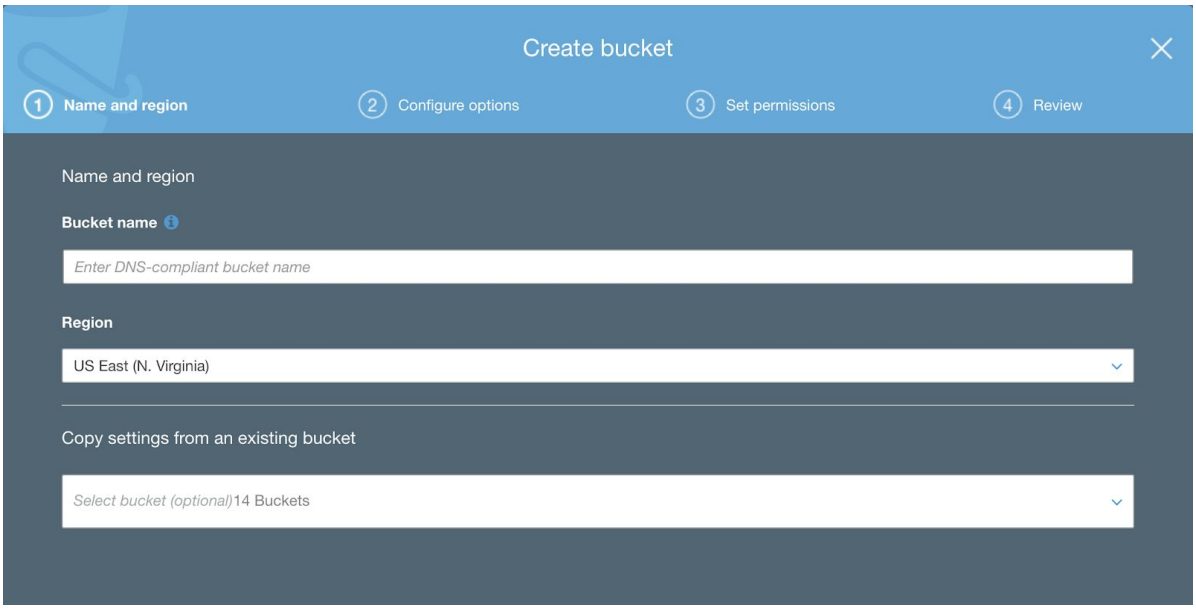
Acesse sua instância por SSH e execute o seguinte comando:

```
$ sudo mount /dev/xvdf /mnt
```

Após este comando, o dataset da Wikipedia estará disponível no seu path /mnt.

Criando um Bucket

O bucket da S3 servirá para armazenar os arquivos txts necessários para a análise, tornando o acesso pelo Zeppelin possível. Crie um Bucket na aba S3 do dashboard da amazon.



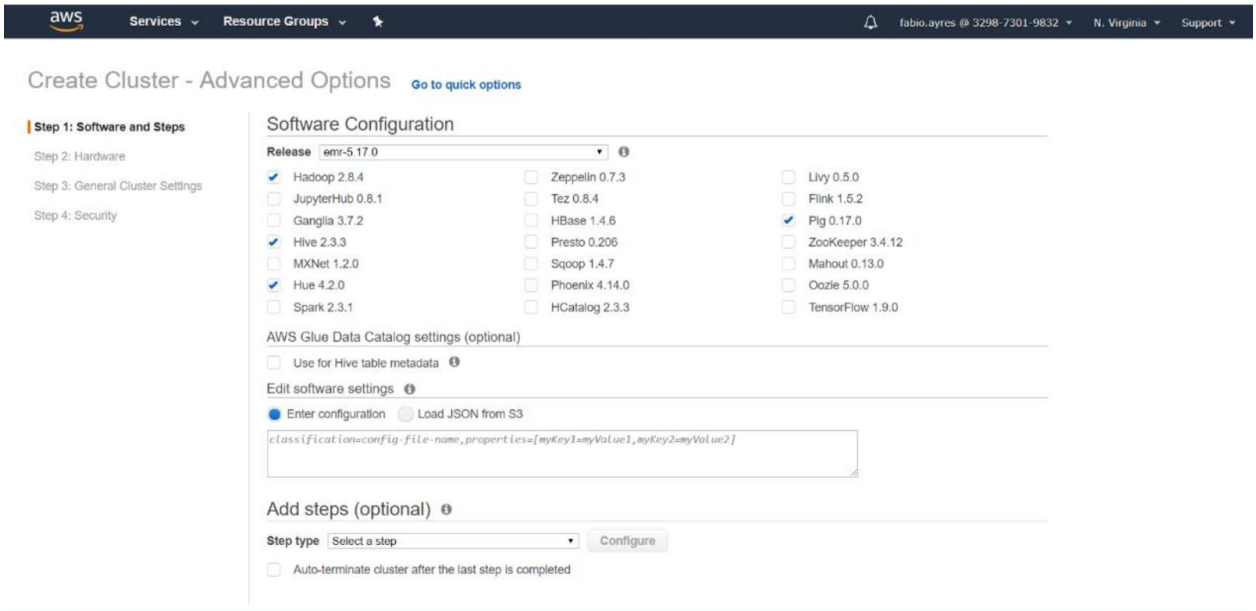
Criando um Bucket

Para transferir os txts, é necessário conectar a sua instância por ssh e transferir os arquivos via aws client. Configure o aws client com suas keys e execute os comandos abaixo:

```
$ aws configure
$ aws s3 cp ./titles-sorted.txt s3://nomedobucket
$ aws s3 cp ./links-simple-sorted.txt s3://nomedobucket
```

Subindo o cluster

Para criar um cluster, basta acessar o campo EMR do Dashboard e cria-lo. Atente-se as dependências necessárias para rodar este projeto: Spark, Zeppelin, Hadoop e Ganglia.



Criando o Cluster

Adicionando as dependências do GraphFrames ao Cluster

-> Conecte-se via **SSH** à máquina **master** do cluster, seguindo as instruções do EMR

```
$ sudo nano /usr/lib/zeppelin/conf/zeppelin-env.sh
 -> adicionar no final da última linha:
      --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11

$ sudo stop zeppelin; sudo start zeppelin
```

Feito isso, basta acessar o Zeppelin através do **FoxyProxy** ou um **SSH Tunnel**

Configurando o Zeppelin

Para executar as células do zeppelin, é necessário configurar alguns campos do interpretador do Spark para que este possa utilizar a biblioteca do GraphFrames. Acesse a página de interpretadores do zeppelin e modifique o campo master para local[*] e zeppelin.pyspark.python para python3.

Preparando os dados

Os dados disponíveis estavam contidos em dois arquivos. Um deles contém os nomes das páginas, e o segundo os links de cada página para outros artigos, como ilustram os exemplos à seguir:

```
Eh
Quarta
Feira
Meus
Bacanos
Eae
Galera
Desse
Brasil
```

Exemplo arquivo de títulos

```
1: 2 3 1
2: 1 3 4 2
3: 1 2 2 3
5: 3 5
6: 2 8 6
7: 1 2 3 7
9: 3 9
```

Exemplo arquivo de arestas

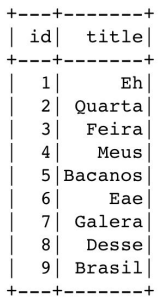
Após testar os dados remotamente pela primeira vez, nos deparamos com um pequeno empecilho: o número de títulos das páginas não batia com o número de nós presentes no arquivo bruto, ou seja, as páginas que não possuíam nenhum link para outro artigo eram ‘puladas’ da lista.

Pode-se observar que os nós 4 e 8 foram pulados da segunda lista, pois eles não referenciavam nenhum outro artigo. Para contornar esse problema, criamos o RDD dos nós a partir do arquivo de títulos, com a função `zipWithIndex` e um `map`, como mostra abaixo:

```
nodes = titles.zipWithIndex().map(lambda x: (x[1] + 1, x[0]))
```

Transformações para a criação dos nós

Isso permitiu criar um DataFrame como o seguinte:



id	title
1	Eh
2	Quarta
3	Feira
4	Meus
5	Bacanos
6	Eae
7	Galera
8	Desse
9	Brasil

Exemplo de DataFrame de Nós

Em seguida, para as arestas, usamos uma subtração de regex para trocar todos os caracteres não numéricos por espaços vazios, e por fim um split, obtendo uma array no qual o primeiro elemento é o nó de origem, e o segundo é outra array com todos os nós que a origem indica. Com a função `flatMapValues` é possível 'quebrar' essas arrays *nested* em um RDD com a configuração desejada

```
def edge_map(x):
    p = non_decimal.sub(' ', x).split(' ')
    return (p[0], p[1:])

edges = txt.map(edge_map).flatMapValues(lambda x: x)
```

Transformações para a criação das arestas

Isso permitiu a obtenção de uma lista de vértices como a seguinte, facilmente transformável em um DataFrame no qual a primeira coluna é o nó de origem e a segunda o nó de destino:

src	dst
1	2
1	3
2	1
2	3
2	4
3	1
3	2
3	2
5	3
6	2
6	8
7	1
7	2
7	3
9	3

Exemplo de DataFrame de Arestas

Para criar o objeto `GraphFrames` é necessário fornecer dois `DataFrames`: o primeiro deve conter os nós e o segundo as arestas. Ambos podem ser obtidos convertendo o RDD resultante das transformações com a função `toDf`, como mostra o exemplo abaixo:

```
nodesDf = nodes.toDF(['id', 'title'])

edgesDf = edges.toDF(['src', 'dst'])
```

Feito isso, é possível criar o Grafo:

```
g = GraphFrame(nodesDf, edgesDf)
```

À partir do grafo, é possível usar todas as funções disponíveis na biblioteca GraphFrames, capazes de calcular várias métricas relevantes para grafos, como caminhos mínimos, indegree e outdegree, PageRank e outros. A documentação completa está em https://graphframes.github.io/api/python/graphframes.html

O exemplo que testamos foi o de menor caminho entre dois nós. A função **bfs** da classe GraphFrames possui a seguinte assinatura:

bfs(*fromExpr*, *toExpr*, *edgeFilter=None*, *maxPathLength=10*)[source]

Os argumentos `fromExpr` e `toExpr` esperam uma expressão de filtro, que será aplicada aos nós do `GraphFrame` e com esse resultado será executada a função. Por exemplo, para calcular o menor caminho entre o nós de id 1 e 3, basta executar:

```
g.bfs('id=1', 'id=3')
```

Obs: Essa função pode levar um bom tempo para executar. Em nossa base de dados, com aproximados 5.000.000 de nós e 130.000.000 arestas, encontrar esse caminho levou aproximadamente 22 minutos, o que consideramos um tempo bastante razoável dado o tamanho do grafo, mas isso pode variar em função da capacidade do cluster designado.

Encontramos que o menor caminho entre os nós 1 e 3 era através do nó 1664968!

```
+--------------------+-----------+--------------------+-----------+--------------------+
|                from|         e0|                  v1|         e1|                  to|
+--------------------+-----------+--------------------+-----------+--------------------+
|[1, !O!ung_language]|[1, 1664968]|[1664968, Fred_Mo...|[1664968, 3]|[3, !Oh_Gloria_In...|
+--------------------+-----------+--------------------+-----------+--------------------+
```

Took 21 min 55 sec. Last updated by anonymous at November 26 2018, 7:31:30 PM. (outdated)

Também calculamos o indegree dos nós, o que levou aproximadamente 10 minutos.

```
%pyspark
g.inDegrees.show()
```

```
+-------+--------+
|     id|inDegree|
+-------+--------+
|2640902|   10580|
|2821237|     192|
|5062062|      34|
|  64985|    1074|
|5308413|   11765|
|4332245|      12|
|4491104|      33|
|2281655|     573|
|5198037|      51|
|2111342|      46|
| 234737|       5|
|3743924|      49|
| 298403|      22|
|    691|      47|
|4925229|     462|
```