Documentação do Projeto de NoSQL

Contexto e aplicação

A aplicação desenvolvida tem o objetivo de coletar as postagens mais recentes do Twitter (www.twitter.com) sobre um tópico de livre escolha, armazená-las localmente, carregá-las no banco de dados NoSQL MongoDB (www.mongodb.com) e, finalmente, produzir análises utilizando os dados disponíveis.

A tecnologia utilizada para esta implementação foi a linguagem de programação Java juntamente com as interfaces (*APIs*) para interconexão com o Twitter (TwitterJ) e o MongoDB (MonogoDB Driver). Para que a implementação mantivesse a independência de qualquer ambiente de desenvolvimento (*IDE*) como, por exemplo, Eclipse ou NetBeans, toda a compilação e execução é realizada utilizando-se a linha de comando sendo necessário realizar apenas a instalação do ambiente padrão de desenvolvimento para Java (*JDK*) disponível em java.com e referenciar as *APIs* citadas conforme descrito no tópico "Metodologia".

A metodologia utilizada para captura dos dados foi a leitura das postagens mais recentes com a utilização de um filtro definido como assunto relevante para estudo e exploração.

O filtro pode ser definido como qualquer palavra-chave que represente um tema, um local, uma pessoa ou qualquer outro assunto de interesse.

A aplicação desenvolvida pode ser utilizada em vários setores com objetivos diversos. Dependendo do filtro e da análise estatística aplicados, a aplicação pode ser explorada por:

- empresas de turismo com o objetivo de avaliação de locais de visita com atividades atrativas utilizando análise de sentimento;
- empresas de mídia e jornalistas para elaboração de reportagens utilizando repercursão de acontecimentos recentes;
- setores de segurança pública para tentativa de identificação do porte de eventos não-oficiais que necessitem de especial atenção através do número de usuários citando determinado evento ou compartilhando determinada *hashtag*.

Para o objetivo proposto pela disciplina foi realizado levantamento utilizando-se como filtro, a palavra "Curitiba" que originalmente representa um local - neste caso a cidade que é a capital do estado do Paraná no Brasil - mas, no atual momento, tem sido utilizado metaforicamente para representar um destino incômodo para políticos corruptos que estão sendo julgados e encarcerados naquela cidade.

Metodologia

A aplicação desenvolvida foi dividida em módulos para melhor entendimento e aprimoramentos futuros sendo assim possível a substituição de um (ou mais) módulo(s), da forma de exibição, do banco de dados ou até mesmo da linguagem de programação.

Os módulos foram divididos em:

1. *Crawler* - módulo que recupera os dados do Twitter e os armazena em um arquivo de *log*.
2. *ETL* - módulo de extração (*Extract*), transformação (*Transform*) e carga (*Load*) dos dados do *log* no banco de dados.
3. *Report* - módulo que extrai do banco de dados as informações e relatórios conforme a necessidade apresentada.

Os títulos abaixo descrevem em detalhes o funcionamento e execução de cada um dos módulos

Crawler

A utilização deste módulo depende de prévia configuração do arquivo twitter4j.properties com as seguintes diretivas:

- debug - diretiva para preenchimento com *true* ou *false* caso deseje ou não ativar o detalhamento das atividades que estão sendo realizadas durante a execução.
- oauth.consumerKey, oauth.consumerSecret, oauth.accessToken e oauth.accessTokenSecret - diretivas que devem ser mantidas em sigilo e são obtidas após registro da aplicação em http://twitter.com/oauth_clients/new

A compilação deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório crawler):

Windows:

```
javac -cp .;..;../lib/twitter/twitter4j-
core-4.0.4.jar;../lib/twitter/twitter4j-stream-4.0.4.jar TwitterCrawler.java
```

Linux:

```
javac -cp .:..:../lib/twitter/twitter4j-
core-4.0.4.jar:../lib/twitter/twitter4j-stream-4.0.4.jar TwitterCrawler.java
```

onde:

- *javac* é o comando de compilação padrão do Java.
- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).
- twitter4j-core-4.0.4.jar e twitter4j-stream-4.0.4.jar são os conjuntos de classes da *API* do TwitterJ.
- TwitterCrawler.java é o nome da classe implementada para realizar a captação dos *tweets*.

A execução deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório crawler):

Windows:

```
java -cp .;..;../lib/twitter/twitter4j-core-4.0.4.jar;../lib/twitter/twitter4j-stream-4.0.4.jar TwitterCrawler Curitiba
```

Linux:

```
java -cp .:..:../lib/twitter/twitter4j-core-4.0.4.jar:../lib/twitter/twitter4j-stream-4.0.4.jar TwitterCrawler Curitiba
```

onde:

- *java* é o comando de execução padrão do Java.
- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).
- twitter4j-core-4.0.4.jar e twitter4j-stream-4.0.4.jar são os conjuntos de classes da *API* do TwitterJ.
- TwitterCrawler é o nome da classe implementada para realizar a captação dos *tweets*.
- Curitiba foi o parâmetro indicado como filtro para a busca dos *tweets*. Nomes compostos ou parâmetros mais complexos devem ser delimitados por aspas duplas como, por exemplo, "Belo Horizonte".

O resultado da execução deste módulo é a geração do arquivo twitter.log no subdiretório "data" em formato *JSON* contendo os seguintes campos:

- "time" - momento da postagem do *tweet*.
- "user_id" - identificação do *login* do usuário.
- "user" - nome de exibição escolhido pelo usuário.
- "twit" - texto publicado.

ETL

A utilização deste módulo depende de prévia inicialização do banco de dados MongoDB que, a partir do diretório *bin* na instalação padrão, pode ser realizada pelo comando:

```
mongod -dbpath ../data/db
```

A compilação deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório ETL):

Windows:

```
javac -cp .;..;../lib/mongo/bson-3.0.2.jar;../lib/mongo/mongodb-driver-3.0.2.jar;../lib/mongo/mongodb-driver-core-3.0.2.jar MongoETL.java
```

Linux:

```
javac -cp .:..:../lib/mongo/bson-3.0.2.jar:../lib/mongo/mongodb-driver-3.0.2.jar:../lib/mongo/mongodb-driver-core-3.0.2.jar MongoETL.java
```

onde:

- *javac* é o comando de compilação padrão do Java.
- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).

- bson-3.0.2.jar, mongodb-driver-3.0.2.jar e mongodb-driver-core-3.0.2.jar são os conjuntos de classes da *API* do MongoDB Driver.
- MongoETL.java é o nome da classe implementada para realizar a carga dos *tweets* no MongoDB.

A execução deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório ETL):

Windows:

```
java -cp .;..;../lib/mongo/bson-3.0.2.jar;../lib/mongo/mongodb-
driver-3.0.2.jar;../lib/mongo/mongodb-driver-core-3.0.2.jar MongoETL
```

Linux:

```
java -cp .:..:../lib/mongo/bson-3.0.2.jar:../lib/mongo/mongodb-
driver-3.0.2.jar:../lib/mongo/mongodb-driver-core-3.0.2.jar MongoETL
```

onde:

- *java* é o comando de execução padrão do Java.
- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).
- bson-3.0.2.jar, mongodb-driver-3.0.2.jar e mongodb-driver-core-3.0.2.jar são os conjuntos de classes da *API* do MongoDB Driver.
- MongoETL é o nome da classe implementada para realizar a carga dos *tweets* no MongoDB.

O resultado da execução deste módulo é a carga do arquivo twitter.log do subdiretório "data" na *collection* "tweets" do banco de dados "tweet_db" do MongoDB previamente inicializado no *localhost*.

Report

A utilização deste módulo depende de prévia inicialização do banco de dados MongoDB que, a partir do diretório *bin* na instalação padrão, pode ser realizada pelo comando:

```
mongod -dbpath ../data/db
```

A compilação deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório report):

Windows:

```
javac -cp .;..;../lib/mongo/bson-3.0.2.jar;../lib/mongo/mongodb-
driver-3.0.2.jar;../lib/mongo/mongodb-driver-core-3.0.2.jar MongoReport.java
```

Linux:

```
javac -cp .:..:../lib/mongo/bson-3.0.2.jar:../lib/mongo/mongodb-
driver-3.0.2.jar:../lib/mongo/mongodb-driver-core-3.0.2.jar MongoReport.java
```

onde:

- *javac* é o comando de compilação padrão do Java.

- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).
- bson-3.0.2.jar, mongodb-driver-3.0.2.jar e mongodb-driver-core-3.0.2.jar são os conjuntos de classes da *API* do MongoDB Driver.
- MongoReport.java é o nome da classe implementada para executar os relatórios desejados no MongoDB.

A execução deste módulo pode ser realizada a partir da linha de comando utilizando a seguinte instrução (a partir do subdiretório report):

Windows:

```
java -cp .;..;../lib/mongo/bson-3.0.2.jar;../lib/mongo/mongodb-
driver-3.0.2.jar;../lib/mongo/mongodb-driver-core-3.0.2.jar MongoReport
```

Linux:

```
java -cp .:..:../lib/mongo/bson-3.0.2.jar:../lib/mongo/mongodb-
driver-3.0.2.jar:../lib/mongo/mongodb-driver-core-3.0.2.jar MongoReport
```

onde:

- *java* é o comando de execução padrão do Java.
- -cp é a diretiva que aponta a localização das *APIs* necessárias para a compilação (*CLASSPATH*).
- bson-3.0.2.jar, mongodb-driver-3.0.2.jar e mongodb-driver-core-3.0.2.jar são os conjuntos de classes da *API* do MongoDB Driver.
- MongoReport.java é o nome da classe implementada para executar os relatórios desejados no MongoDB.

O resultado da execução deste módulo é o menu abaixo cujos resultados estão dispostos na próxima seção:

```
Selecione o número do relatório desejado ou 0 (zero) para finalizar:
[1] Total de Tweets
[2] Volume de Tweets por dia
[3] Volume de Tweets por hora do dia
[4] Termos mais frequentes
```

Resultados

Total de *tweets* coletados: 14999
Tempo total da consulta: 224 milissegundos

Volume de *tweets* por dia:
12/12/2016: 3175
13/12/2016: 8688
14/12/2016: 3136
Tempo total da consulta: 5669 milissegundos

Volume de *tweets* por hora do dia:
12/12/2016 16h00 ~ 17h00: 85
12/12/2016 17h00 ~ 18h00: 103
12/12/2016 18h00 ~ 19h00: 478
12/12/2016 19h00 ~ 20h00: 512
12/12/2016 20h00 ~ 21h00: 546
12/12/2016 21h00 ~ 22h00: 467
12/12/2016 22h00 ~ 23h00: 495
12/12/2016 23h00 ~ 24h00: 489
13/12/2016 0h00 ~ 1h00: 320
13/12/2016 1h00 ~ 2h00: 465
13/12/2016 2h00 ~ 3h00: 305
13/12/2016 3h00 ~ 4h00: 131
13/12/2016 4h00 ~ 5h00: 78
13/12/2016 5h00 ~ 6h00: 75
13/12/2016 6h00 ~ 7h00: 143
13/12/2016 7h00 ~ 8h00: 223
13/12/2016 8h00 ~ 9h00: 318
13/12/2016 9h00 ~ 10h00: 471
13/12/2016 10h00 ~ 11h00: 478
13/12/2016 11h00 ~ 12h00: 506
13/12/2016 12h00 ~ 13h00: 502
13/12/2016 13h00 ~ 14h00: 521
13/12/2016 14h00 ~ 15h00: 485
13/12/2016 15h00 ~ 16h00: 428
13/12/2016 16h00 ~ 17h00: 452
13/12/2016 17h00 ~ 18h00: 406
13/12/2016 18h00 ~ 19h00: 399
13/12/2016 19h00 ~ 20h00: 467
13/12/2016 20h00 ~ 21h00: 463
13/12/2016 21h00 ~ 22h00: 245
13/12/2016 22h00 ~ 23h00: 461
13/12/2016 23h00 ~ 24h00: 346
14/12/2016 0h00 ~ 1h00: 241
14/12/2016 1h00 ~ 2h00: 223
14/12/2016 2h00 ~ 3h00: 105
14/12/2016 3h00 ~ 4h00: 62
14/12/2016 4h00 ~ 5h00: 43
14/12/2016 5h00 ~ 6h00: 27
14/12/2016 6h00 ~ 7h00: 65
14/12/2016 7h00 ~ 8h00: 113
14/12/2016 8h00 ~ 9h00: 180

14/12/2016 9h00 ~ 10h00: 204
14/12/2016 10h00 ~ 11h00: 332
14/12/2016 11h00 ~ 12h00: 348
14/12/2016 12h00 ~ 13h00: 345
14/12/2016 13h00 ~ 14h00: 317
14/12/2016 14h00 ~ 15h00: 436
14/12/2016 15h00 ~ 16h00: 95
Tempo total da consulta: 866 milissegundos

Termos mais frequentes (limitados em 50 itens com, no mínimo, 2 caracteres):
Curitiba ==> 10110.0
https ==> 8671.0
co ==> 7958.0
de ==> 7339.0
RT ==> 7330.0
em ==> 3846.0
curitiba ==> 3253.0
Brazil ==> 2971.0
que ==> 2670.0
24K_7S ==> 1846.0
belohorizonte ==> 1688.0
riodejaneiro ==> 1688.0
do ==> 1601.0
24kNoBrasil ==> 1364.0
Goodbye ==> 1346.0
pra ==> 1335.0
in ==> 1301.0
um ==> 1208.0
PR ==> 1189.0
at ==> 1111.0
com ==> 1084.0
da ==> 1074.0
no ==> 1070.0
brazil ==> 918.0
para ==> 890.0
ao ==> 885.0
eu ==> 821.0
na ==> 817.0
seu ==> 727.0
CURITIBA ==> 695.0
povo ==> 674.0
PEC55SIM ==> 656.0
se ==> 611.0
pessoal ==> 605.0
alfafa ==> 598.0
brasileiro ==> 586.0
comedor ==> 581.0
mandando ==> 577.0
Recadinho ==> 574.0
representando ==> 573.0
24K ==> 569.0
AnaPaulaDlamari ==> 559.0
tem ==> 534.0

24U ==> 505.0
BeloHorizonte ==> 505.0
RiodeJaneiro ==> 502.0
por ==> 489.0
SaoPaulo ==> 484.0
24kinbrazil ==> 481.0
24K__OFFICIAL ==> 480.0
Tempo total da consulta: 32995 milissegundos

Termos mais frequentes (limitados em 50 itens com, no mínimo, 3 caracteres):
Curitiba ==> 10110.0
https ==> 8671.0
curitiba ==> 3253.0
Brazil ==> 2971.0
que ==> 2670.0
24K_7S ==> 1846.0
belohorizonte ==> 1688.0
riodejaneiro ==> 1688.0
24kNoBrasil ==> 1364.0
Goodbye ==> 1346.0
pra ==> 1335.0
com ==> 1084.0
brazil ==> 918.0
para ==> 890.0
seu ==> 727.0
CURITIBA ==> 695.0
povo ==> 674.0
PEC55SIM ==> 656.0
pessoal ==> 605.0
alfafa ==> 598.0
brasileiro ==> 586.0
comedor ==> 581.0
mandando ==> 577.0
Recadinho ==> 574.0
representando ==> 573.0
24K ==> 569.0
AnaPaulaDlamari ==> 559.0
tem ==> 534.0
24U ==> 505.0
BeloHorizonte ==> 505.0
RiodeJaneiro ==> 502.0
por ==> 489.0
SaoPaulo ==> 484.0
24kinbrazil ==> 481.0
24K__OFFICIAL ==> 480.0
mais ==> 467.0
est ==> 457.0
Moro ==> 455.0
uma ==> 446.0
vai ==> 445.0
ser ==> 417.0
dia ==> 411.0
via ==> 366.0

voc ==> 357.0
YouTube ==> 353.0
aqui ==> 350.0
obrigado ==> 341.0
Paran ==> 336.0
saopaulo ==> 326.0
GkTiZ3NPxH ==> 324.0
Tempo total da consulta: 33986 milissegundos

Conclusões

O MongoDB, banco de dados utilizado para elaboração deste projeto, apresentou comportamento bastante satisfatório para o objetivo proposto de análise de dados de redes sociais. O desempenho para execução das consultas mostrou-se bem semelhante aos bancos de dados relacionais mais populares com a vantagem de apresentar um consumo de memória menos significativo.

A implementação dos módulos utilizando a linguagem Java nas interações com o banco de dados demandou mais esforço que seria necessário para outros bancos mais populares - que dispõem de mais usuários ativos e diversidade de exemplos. Provavelmente há outras linguagens mais efetivas para esta implementação ou futuramente surjam novas formas de interação para possibilitar mais facilidade aos desenvolvedores desta linguagem.

Trabalhos Futuros

Um aprimoramento imediato da aplicação para um ambiente de produção poderia ser a substituição do *log* em arquivo texto para um barramento de serviços. Com esta adaptação, o módulo de captação redirecionaria os dados para uma fila de consumo/processamento pelo módulo de *ETL*. O módulo de *ETL*, por sua vez, ao finalizar a execução, alimentaria outra fila que seria imediatamente consumida pelo módulo de relatórios promovendo atualizações em tempo real.