Análise de Dados de Sensores em Tempo Real utilizando Python, Apache Spark e o RabbitMQ

Jobson R. Pereira[1], Lucas F. Silva[1], Lucas Filipe V. Silva[1], Luiz Felipe R. Arruda[1]

[1]Departamento de Estatística e Informática – Universidade Federal Rural de Pernambuco (UFRPE) – CEP: 52.171-900 – Recife, PE – Brasil

{jobson.rocha, lucas.ferreiras, lucas.filipe, luiz.arruda}@ufrpe.br

Abstract. *This paper presents, in details, the process of developing a system of monitoring data generated by motion sensors coupled to the body of each player of a soccer team. The application consists of the integration of open source services in order to promote the distribution of the processes between the systems. Apache Spark was used for the data processing, the RabbitMQ for the distribution of the queues and the MySQL database for the storage of the information. As well as a heat map to identify in real time the position of each participant within the football field during matches.*

Resumo. *Este artigo apresenta, em detalhes, o processo de desenvolvimento de um sistema de monitoramento de dados gerados por sensores de movimento acoplados ao corpo de cada jogador de um time de futebol. A aplicação consiste da integração de serviços de código aberto a fim de promover a distribuição dos processos entre os sistemas. Foram utilizados o Apache Spark para o processamento dos dados, o RabbitMQ para a distribuição das filas e o banco de dados MySQL para o armazenamento das informações. Assim como, um mapa de calor para identificar em tempo real a posição de cada participante dentro do campo de futebol durante as partidas.*

1. Introdução

A tecnologia vestível é um novo termo que está sendo pulverizado em várias áreas e segmentos profissionais. No intuito de obter dados estatisticamente precisos, diversos componentes eletrônicos estão sendo desenvolvidos a fim de serem utilizados em todos os itens de qualquer natureza. Todo este conjunto de itens antigamente estavam inoperantes e não tinham vida tecnológica. Hoje, eles são elementos essenciais da Internet das Coisas, comumente conhecida como ***IoT*** (*Internet of Things*).

Os objetos que compõem a Internet das Coisas são compostos por incontáveis dispositivos que captam eventos e os processam visando criar dados para análise futura. Com base nesses dados, um sistema integrado pode gerar um conteúdo relevante para o usuário e este, por sua vez, pode ser beneficiado em algum momento de tomada de decisão. Neste artigo, um conjunto de pacote de serviços *open source* foram integrados no sentido de captar dados, tratá-los, guardá-los e exibi-los em uma consulta da qual será possível visualizar um mapa de calor, isto é, uma região onde existe a predominância de um mesmo dado para a mesma referência em diversos momentos.

2. Objetivos

O objetivo deste trabalho consiste na reprodução de um material produzido numa das edições do *ACM DEBS 2013 Grand Challenge* [DEBS, 2013]. Nesta edição, um dos projetos criados foi sobre o monitoramento de jogadores de futebol em uma partida realizada na Alemanha. E, para reproduzi-lo, foram definidas algumas atividades importantes para um redesenho mínimo do desafio.

3. Ambiente de Sistema Distribuído

Para realizar a reprodução do desafio é necessário seguir alguns passos fundamentais. Serão necessárias máquinas virtuais que realizarão a distribuição do processamento de dados. Cada uma delas contém uma imagem do sistema operacional Linux, especificamente a versão Mininet que é uma versão *Open Source BSD* distribuída no intuito de facilitar a criação de uma rede virtual instantânea.

Seguindo estritamente o passo-a-passo a seguir será possível obter um ambiente idêntico ao utilizado no desafio deste artigo. A versão de algum pacote pode estar desatualizada no momento da montagem de ambiente deste desafio, por isso, é recomendado utilizar a mesma versão proposta neste artigo. No entanto, nada o impede de utilizar versões mais recentes, porém o resultado final pode não ser tão satisfatório.

3.1. Oracle VM VirtualBox

Para simular um servidor Linux sem afetar o sistema operacional hospedeiro, isto é, o sistema operacional nativo de um computador, é necessária a instalação de um programa de virtualização que permite instalar e executar diferentes sistemas operacionais em um único computador sem afetar diretamente o sistema operacional nativo do computador.

A versão de ambiente utilizada neste desafio foi a 5.1.26 r117224 (Qt5.6.2) para o sistema operacional Windows 10.

3.2. Mininet

Conforme já mencionado, a versão de servidor Linux utilizada é a Mininet que pode ser baixada gratuitamente no site do respectivo. Naturalmente, o servidor Mininet é um emulador de rede que cria uma rede de hosts virtuais, switches, controladores e links. Os hosts Mininet executam um software de rede Linux padrão, e seus switches suportam o OpenFlow para roteamento personalizado altamente flexível e rede definida por software.

A versão de ambiente utilizada neste desafio foi a 2.2.2-170321 no Ubuntu 14.04 LTS – 64 bits.

3.2.1. Pacotes

Vários pacotes de serviços precisam ser obrigatoriamente instalados para assegurar que o servidor virtual atenda aos requisitos mínimos para a reprodução de ambiente do desafio. Os pacotes e suas versões estão listadas nos próximos tópicos.

3.2.1.1. Apache Spark (Versão 1.6.1)

O *Apache Spark* é um sistema de computação em cluster, com ele será possível realizar a execução do processamento de dados. Faça o download do arquivo para o servidor virtual através do seguinte comando:

```
sudo wget https://d3kbcqa49mib13.cloudfront.net/spark-1.6.1-bin-hadoop2.6.tgz
```

Para realizar a extração do arquivo acima, o qual estará localizado no diretório inicial do servidor, utilize o seguinte comando:

```
sudo tar -xvf spark-1.6.1-bin-hadoop2.6.tgz
```

3.2.1.2. RabbitMQ (Versão 3.6.10, Erlang R16B03)

O *RabbitMQ* é um gerenciador de filas de mensagens as quais irão alocar os dados momentaneamente enviados para suas respectivas filas conforme configuradas nos arquivos específicos de produção, recebimento e consumo. Para realizar a instalação do RabbitMQ no servidor virtual realize os seguintes comandos:

```
gpg --keyserver pgpkeys.mit.edu --recv-key 7638D0442B90D010

gpg -a --export 7638D0442B90D010 | sudo apt-key add -
echo 'deb http://ftp.debian.org/debian wheezy-backports main' | sudo tee /etc/apt/sources.list.d/wheezy_backports.list

wget -O- https://packages.erlang-solutions.com/debian/erlang_solutions.asc | sudo apt-key add -

echo 'deb https://packages.erlang-solutions.com/debian wheezy contrib' | sudo tee /etc/apt/sources.list.d/esl.list

sudo apt-get update

sudo apt-get install init-system-helpers socat esl-erlang

wget -O- https://www.rabbitmq.com/rabbitmq-release-signing-key.asc | sudo apt-key add -

echo 'deb http://www.rabbitmq.com/debian/ testing main' | sudo tee /etc/apt/sources.list.d/rabbitmq.list

sudo apt-get update

sudo apt-get install rabbitmq-server
```

Sempre que houver alguma pergunta de confirmação para prosseguir a instalação, pressione a tecla Y a fim de autorizar a instalação ou atualização dos pacotes.

O desafio está configurado para rodar em três servidores distintos, cada um realizando uma determinada tarefa. No servidor que for definido como o de gerenciador de filas de mensagens será necessário configurá-lo para que o mesmo disponibilize o gerenciamento administrativo. Neste caso, execute os comandos a seguir trocando os itens em vermelho. A máquina virtual para o *RabbitMQ* foi definida como a segunda.

```
# Criar o usuario no servidor
sudo rabbitmqctl add_user <user> <passwd>

# Criar o vhost no servidor
sudo rabbitmqctl add_vhost <vhost_name>

# Setar as permissoes no servidor
sudo rabbitmqctl set_permissions -p <vhost_name> <user> ".*" ".*" ".*"

# Configurar interface web
sudo rabbitmq-plugins enable rabbitmq_management

# Dar permissão para usuario acessar a Interface Web
sudo rabbitmqctl set_user_tags <user> management administrator

# Depois, reinicie o servidor rabbitmq:
sudo service rabbimq-server restart

# Acessar a interface web:
<ip-da-vm>:15672

# Para saber o ip da máquina virtual execute o comando:
ifconfig eth0
```

Se o IP da máquina virtual não for um IP válido, tente configurar a placa de rede da máquina virtual para o modo Bridge. Para isso, basta acessar o menu Configurações, clicar na opção Rede, depois em Adaptador 1 e alterar para *Placa em modo Bridge*.

3.2.1.3. Python (Versão 2.7)

A linguagem de programação Python é nativa nos sistemas operacionais Linux, no entanto, para assegurar que aquela esteja instalada ou atualizada, execute os comandos:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

wget https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tgz

tar -xvf Python-2.7.13.tgz

cd Python-2.7.13

./configure

make

sudo checkinstall
```

Sempre que executar alguma instalação de pacotes no servidor virtual, execute o comando para atualizar suas dependências. Este comando pode ser utilizado sempre que necessário. O comando para atualizar as dependências e pacotes é:

```
sudo apt-get -y update
```

3.2.1.4. MySQL (Versão maior que 5.5)

Para armazenar definitivamente os dados é necessária a instalação do pacote de banco de dados MySQL. A versão utilizada pode ser qualquer uma que seja maior que 5.5. Execute os comandos a seguir a fim de instalar e configurar o MySQL:

```
sudo apt-get update

sudo apt-get install mysql-server mysql-client

sudo mysql_secure_installation

sudo mysql_install_db
```

Para acessar o MySQL execute os seguintes comandos:

```
sudo mysql -u root -p


-- Exibir as bases de dados
show databases;

-- Selecoinar uma das bases
use <nome_da_base>

-- Exibir as tabelas da base de dados
show tables;

-- Selecionar os dados da tabela desejada
select * from <nome_da_tabela>
```

Para criar a base de dados e acessá-la, utilize o MySQL Workbench do terceiro servidor virtual via SSH. Após a execução dos passos abaixo, basta executar o script de criação da base de dados (próxima página) para armazenar os dados do desafio.

1 – Abrir o MySQL Workbench;

2 – Criar uma nova conexão (símbolo de mais);

3 – Informe o nome desejado para a conexão;

4 – No método de conexão mude Standard (TCP/IP) para Standard (TCP/IP) over SSH;

5 – Em SSH Hostname informe o endereço IP do terceiro servidor virtual;

6 – Em SSH Username digite mininet;

7 – Em SSH Password informe a senha mininet e salve no cofre (Vault);

8 – Em MySQL Hostname, Server Port, Username mantenha o padrão sugerido;

9 – Em Password digite a senha configurada na hora de instalação do MySQL e salve.

Para criar a base de dados MySQL através do MySQL Workbench acesse o menu File (Arquivo), clique em New Query Tab (Nova Guia de Consulta), cole o script abaixo e execute-o.

```
-- phpMyAdmin SQL Dump
-- version 4.2.7.1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: 21-Ago-2017 às 23:39
-- Versão do servidor: 5.5.39
-- PHP Version: 5.4.31

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

--
-- Database: `dbfutebol`
--
CREATE DATABASE IF NOT EXISTS `dbfutebol` DEFAULT CHARACTER SET latin1
COLLATE latin1_swedish_ci;
USE `dbfutebol`;

-- --------------------------------------------------------
--
-- Estrutura da tabela `grid`
--
CREATE TABLE IF NOT EXISTS `grid` (
`id` int(11) NOT NULL,
  `idsensor` varchar(20) NOT NULL,
  `linha` varchar(20) NOT NULL,
  `coluna` varchar(20) NOT NULL,
  `celula` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Indexes for table `grid`
--
ALTER TABLE `grid`
 ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT for table `grid`
--
ALTER TABLE `grid`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--- Fim do Código
```

3.2.1.5. Spark Streaming MQTT Assembly (Versão 2.11-1.6.1)

Faça o download do diretório que contém o arquivo e extraia-o no diretório home "~/":

```
git clone https://github.com/jualabs/rsi-psd-codes.git
```

Se o pacote Git não estiver instalado, basta executar o comando:

```
sudo apt-get -y install git
```

4. Códigos

Com o ambiente de produção configurado, basta criar os arquivos de cada código responsável por determinadas funções do desafio. Basicamente são três arquivos: producer.py, consumer.py e receiver.py. Todos estão descritos abaixo com seus respectivos códigos.

4.1. Gerador de Eventos (*Producer*)

O producer é um código que será responsável pela leitura do arquivo de texto que contém os dados de leitura dos sensores. Este arquivo estará alocado no servidor virtual que corresponde à primeira máquina. O código do arquivo em python é o que segue:

```
### Começo do código ###


import pika
import sys
import time

ipServidorRabbit = sys.argv[1]

credentials = pika.PlainCredentials('jobson', '123')
connection = pika.BlockingConnection(pika.ConnectionParameters(
               ipServidorRabbit, 5672, 'futebol', credentials))

channel = connection.channel()
channel.exchange_declare(exchange='amq.topic',
                 type='topic', durable=True)

arqent = open(sys.argv[2],"r")
n_salto = int(sys.argv[3])

cont = 1

while True:
        linha1 = arqent.readline()
        if linha1 == "":
                break
        if (cont % n_salto == 0):
            lista1 = linha1.split(",")
        idS = int(lista1[0]);
        ts = int(lista1[1]);
        xx = int(lista1[2]);
        yy = int(lista1[3]);
            message = "%r,%r,%r,%r" % (idS,ts,xx,yy)
            channel.basic_publish(exchange='amq.topic',
                                  routing_key='hello',
                                  body=message)
            print " [x] Sent %r" % message
            print "*******************************"
            time.sleep(1)
        cont += 1

connection.close()


### Fim do código ###
```

4.2. Consumidor de Dados (*Consumer*)

O consumidor de dados será responsável por tratar os dados recebidos do Gerador de Eventos e encaminhará para o receptor de dados. Este código estará localizado no terceiro servidor virtual. Segue seu código:

```
### Começo do código ###


import pika
import time
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.mqtt import MQTTUtils

def calculaMx(d1,d2):
    if (d1 == 0):
        incremento = 1
    elif ( int(100*d1/d2) == 100*int(d1/d2) ):
        incremento = 0
    elif (d1 % d2 == 0):
        incremento = 0
    else:
        incremento = 1
    return int(d1/d2) + incremento


def calculaMy(d1,d2):
    if (d1 == 67925):
        incremento = -1
    else:
        incremento = 0
    return int(d1/d2) + incremento

def mapear(linha):
    lista = linha.split(",")
    cX = x_max/pX
    cY = y_max/pY
    sid = int(lista[0])
    x = int(lista[2])
    y = int(lista[3]) + deslocamentoY
    mX = calculaMx(x, cX)
    mY = calculaMy(y, cY)
    nCelula = mY*pX + mX
    nColuna = mX
    nLinha = mY + 1
    lista2 = [sid,nLinha,nColuna,nCelula]
    enviarDadosGrid(lista2)
    return lista2

def splitLinha(linha):
        return linha.split(",")[0]


def calculaGrid(linha):
        parametros = linha.split(",")
        return parametros[0]
```

```
def enviarDadosGrid(lista):
    credentials = pika.PlainCredentials('jobson', '123')
    connection = pika.BlockingConnection(pika.ConnectionParameters(
        ipServidorRabbit, 5672, 'futebol', credentials))

    channel = connection.channel()

    channel.exchange_declare(exchange='top.futebol',
                             type='topic')#, durable=True)

    message = str(lista[0]) + " " + str(lista[1]) + " " + str(lista[2])
+ " " + str(lista[3])

    channel.basic_publish(exchange='top.futebol',
                          routing_key='hello',
                          body=message)
    print " [x] sent %r" % message
    print "********************"
    #time.sleep(1)

ipServidorRabbit = sys.argv[1]

pX = int(sys.argv[2])

pY = int(sys.argv[3])

x_min = 0

x_max = 52483

deslocamentoY = 33960

y_min = 0

y_max = 67925 # 33965 + 33960

sc = SparkContext()

ssc = StreamingContext(sc, 10)

lines = MQTTUtils.createStream(
    ssc,
    "tcp://%s:1883" % ipServidorRabbit,
    "hello"
)

counts = lines.map(mapear)

counts.pprint()

ssc.start()

ssc.awaitTermination()

ssc.stop()

### Fim do código ###
```

4.3. Receptor de Dados (*Receiver*)

O receptor de dados, por sua vez, será responsável por enviar os dados coletados pelo consumidor de dados e os enviará para a tabela do banco de dados MySQL. O código do receptor de dados também estará alocado no terceiro servidor virtual. Segue o código:

```
### Começo do código ###


import sys
import pika
import mysql.connector

con =
mysql.connector.connect(user='root',password='admin123',host='localhos
t',database='dbfutebol')
cursor = con.cursor()

ipServidorRabbit = sys.argv[1]

credentials = pika.PlainCredentials('jobson', '123')
connection = pika.BlockingConnection(pika.ConnectionParameters(
   ipServidorRabbit, 5672, 'futebol', credentials))

channel = connection.channel()

channel.exchange_declare(exchange='top.futebol',
                         type='topic')

result = channel.queue_declare(exclusive=True)

queue_name = result.method.queue

binding_keys = "*"

channel.queue_bind(exchange='top.futebol',
                   queue=queue_name,
                   routing_key=binding_keys)

print ' [*] Waiting for logs. To exit press CTRL+C'

def callback(ch, method, properties, body):
   body = body.split(" ")
   print (body)
   inserirBanco(cursor, body)

def inserirBanco(cursor, lista):
   cursor.execute("""INSERT INTO grid (id, idsensor, linha, coluna,
celula) VALUES (null,%s,%s,%s,%s)"""%(lista[0], lista[1], lista[2],
lista[3] ) )
   con.commit()

channel.basic_consume(callback, queue=queue_name, no_ack=True)

channel.start_consuming()

### Fim do código ###
```

4.4. Execução do Spark

Execute os seguintes comando no **terceiro servidor virtual** a fim de iniciar o processo de tratamento de dados:

```
cd spark-1.6.1-bin-hadoop2.6/

./bin/spark-submit --master local[2] --jars ../rsi-psd-
codes/psd/rabbitmq-spark-integration/spark_jar/spark-streaming-mqtt-
assembly_2.11-1.6.1.jar ../rsi-psd-codes/psd/rabbitmq-spark-
integration/consumer.py arg1 arg2 arg3
```

Onde:

arg1 é o IP do servidor Rabbit (segundo servidor virtual);

arg2 é o número de linhas do grid (8, ..., 64);

arg3 é o número de colunas do grid (13, ..., 100).

Numa sessão separada, executar os seguintes comandos:

```
cd rsi-psd-codes/psd/rabbitmq-spark-integration/
python receiver.py arg1
```

Onde:

arg1 é o IP do segundo servidor virtual, ou seja, o RabbitMQ.

No **primeiro servidor virtual**, isto é, no gerador de eventos, execute o comando:

```
cd rsi-psd-codes/psd/rabbitmq-spark-integration/
python producer.py arg1 arg2 arg3
```

Onde:

arg1 é o IP do segundo servidor virtual, ou seja, o RabbitMQ;

arg2 é o nome do arquivo de texto com os dados dos sensores;

arg3 é o número de saltos de linhas na leitura do arquivo.

4.5. Mapa de Calor (*HeatMap*)

O mapa de calor é o último recurso a ser utilizado, porém não menos importante. Através dele será possível determinar a permanência de um determinado jogador no campo. Cada jogador estará equipado com dois sensores, um em cada chuteira. No entanto, no desafio deste artigo apenas um dos sensores está sendo utilizado a fim de minimizar a enorme quantidade de dados.

A Figura 1 (abaixo) ilustra metade do campo de um futebol onde os dados serão basicamente capturados pelos sensores. O tamanho das células pode variar conforme o desejado. O mapa pode ser acessado pela máquina hospedeira via navegador web.

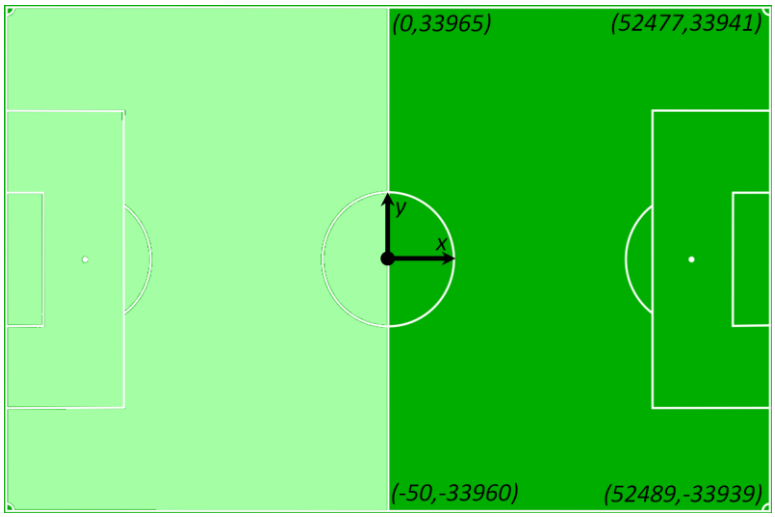

Figura 1. Campo de Jogo e suas dimensões

5. Referências

DEBS (2013) "The DEBS 2013 Grand Challenge – Soccer Monitoring". http://debs.org/debs-2013-grand-challenge-soccer-monitoring/, Agosto 2017.