Relatório Trabalho Sistemas Distribuídos

Alecsander Pasqualli Gesser, Vinicius Ferri, Loren Mattana Viegas
Engenharia de Computação – Departamento de Computação – Universidade Federal de Santa Catarina
pasqualli.alecs@gmail.com

Relatório elaborado para o trabalho final da disciplina de Sistemas Distribuídos, ministrada pelo Professor Jim Lau do Departamento de Computação da Universidade Federal de Santa Catarina.

Este documento tem por objetivo a explicação do trabalho prático realizado para a matéria em questão, desde seus objetivos, ferramentes e algoritmos utilizados, até o funcionamento da ferramenta final.

INTRODUÇÃO

Ao decorrer do documento será explicado passo a passo a realização de uma ferramenta distribuída para visualização de dados capturadas do Instituto Nacional de Meteorologia (INMET) e a aplicação de algoritmos de mineração de dados sobre estes dados.

1 Lista de Bibliotecas e Ferramentas necessárias

Para o desenvolvimento foram utilizados algumas ferramentas principais e suas dependências:

1.1 Python (3.6)

- Flask: servidor para comunicação com nodejs
- Sqlite3: banco de dados não relacional
- Pandas: biblioteca de ciência de dados com diversas estruturas de dados para facilitar a manipulação dos mesmos
- Sklearn: bblioteca com algoritmos para pre-processamento e processamento de dados
- Numpy: biblioteca padrão Python para estruturas dados e métodos já implementados

1.2 Nodejs (12.13.1)

- Express: comunicação NODEJS com clientes(requisição POST e GET)
- Axios: comunicação com servidor Python
- Body-parser: usado para analisar dados JSON enviados pelo cliente e servidor Python

1.3 HTML

- JQuery Ajax Google: Biblioteca de comunicação para o cliente.
- Plot ly: Biblioteca para visualização de dados através de gráficos interativos utilizando React Native

2 Obtenção de Dados

Para a captura de dados foi utilizado um Crawler no site INMET disponível para *clone* em GitHub link

2.1 Utilizando Crawler

Para utilização do Crawler, já com o Git instalado, siga par ao seguintes comandos:

2.1.1 Clone o repositório crawler-inmet

```
$ git clone https://github.com/dedeco/crawler-inmet.git
```

2.1.2 Instale pacotres necessários

```
$ cd crawler-inmet
$ pip install -r requirements
```

2.1.3 Configure Usuário e Senha em config.py

```
USER = 'email@sample.com'
PASS = '123456'
```

2.1.4 Configurar data de aquisição de dados em crawler_data.py

```
start = "01/01/2017"
end = "01/09/2017"
```

2.1.5 Execute Crawler e exporte dados para clima.db

```
$ python3 crawler_data.py
$ python3 export_data.py
```

3 Cliente e Servidor

3.1 Arquitetura da Rede

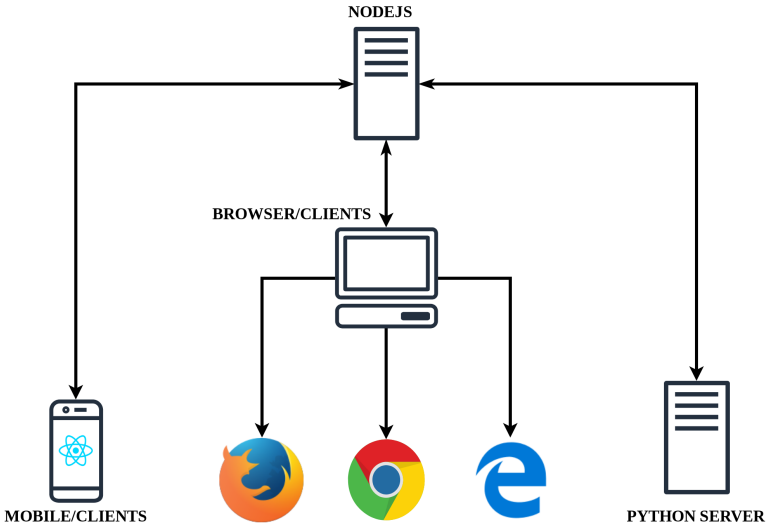

Figura 1: *Arquitetura proposta.*

3.2 Cliente

Para a ferramente de visualização no lado cliente utilizamos HTML e requisições GET e POST para todos os seguintes procedimentos:

- *Connect*: Utilizado para testar conexão cliente servidor, enviando o nome do cliente e esperando resposta *'sucess'*
- *GetCols*: Requisição de colunas disponíveis
- *Visualise*: Requisitar dados selecionados para visualização dos mesmos.
- *RandomForest*: Requisitar Processamento ao servidor *NODEJS*

3.2.1 Implementação HTML

Abaixo podemos visualizar todo html,

- HTML
  - Caixa de Texto:

```
<p>IP: <input type="TEXT" id="ip" size="40"><br></p>
```

  - Botão:

```
<input type="button" id="select" value="Select">
```

  - Data:

```
<input type="date" id="start" value="StartDate">
```

  - Seleção de Argumentos:

```
<p>Predict Parameter Y: <select id="Para1"></select></p>
```

- JavaScript
  - Incluir dependências para comunicação e visualização de dados:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js">
    </script>
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
```

  - Corpo Básico de Requisição POST(connect) ao Servidor:

```
$("#connect").click(function()
{
    user=$("#user").val();      pass=$("#password").val();
    let msg = {username: user,password: pass}; // mensagem enviada para servidor
    $.post("http://"+$("#ip").val()+":3000/connect", msg ,
        function(data) // data = mensagem recebido do servidor
        {
            data = JSON.parse(data);
            console.log(data.data);
        }
    );
});
```

  - Visualização de dados:

```
        $("#visualise").click(function () {
        user = $("#user").val();
        pass = $("#password").val();
        let msg = { parameter: $('#ParaY').val(), city: $("#city").val(), start:
            $("#startDate").val(), end: $("#endDate").val() };
        $.post("http://" + $("#ip").val() + ":3000/visualise", msg,
          function (data) {
            data = JSON.parse(data);
            console.log(data);
            x = [];
            y = [];
            for (let i = 0; i < data.data.length; i++) {
              x.push(data.data[i].measure_date);
              y.push(data.data[i][$('#ParaY').val()]);
            }
            var data = [
              {
                x: x,
                y: y,
                type: 'line'
              }];
            //----------------PLOT VISUALISATION--------------------
            Plotly.newPlot('PLOTDIV', data);
          });
      });
```

- Requisição para o Servidor:

```
        $("#server").click(function () {
        //--------------------SERVER---------------------------------------

        user = $("#user").val();
        pass = $("#password").val();
        let msg = { username: user, parameter: $('#Para1').val(), start: $('#
            startDate').val(), end: $('#endDate').val(), city: $('#Para2').val()
            };
        $.post("http://" + $("#ip").val() + ":3000/randomForest",
          msg,
          function (data) {
            data = JSON.parse(data);
            let dropdown = $('#waiting');
            dropdown.empty();
            for (let i = 0; i < data.list.length; i++) {
              dropdown.append($('<p></p>').text(data.list[i]));
            }

            if (data.type != `waiting`) {
              var dados = [{
                type: 'bar',
                x: data.x,
                y: data.label,
                orientation: 'h'
              }];


              //------------------------PLOT
                  DATA------------------------------------------------------------
              Plotly.newPlot('PLOTDIV', dados);
            }

          });
      });
```

- Requisição de Colunas Disponíveis:

```
$("#getcols").click(function () {
//-------------------GET COLUNAS-----------------------------------------------
    $.post("http://" + $("#ip").val() + ":3000/getcols",
    function (data) {
        let dropdown = $('#ParaY');
        data = JSON.parse(data)
        colunas = data;
        console.log(data)
        dropdown.empty();
        dropdown.append('<option selected="true" disabled>Choose Colunas</option>
            ');
        dropdown.prop('selectedIndex', 0);
        for (let i = 0; i < data.columns.length; i++) {
          dropdown.append($('<option></option>').attr('value', data.columns[i]).
              text(data.columns[i]));
        }
        let dropdown2 = $('#city');
        dropdown2.append('<option selected="true" disabled>Choose Cidade</option>
            ');
        dropdown2.prop('selectedIndex', 0);
        for (let i = 0; i < data.capitais.length; i++) {
          dropdown2.append($('<option></option>').attr('value', data.capitais[i])
              .text(data.capitais[i]));
        }
    });
});
```

- Implementação Servidor NODEJS:
  - Resposta de requisição básica(Connect);

```
app.post('/connect', function (req, res) {
  console.log("POST CONNECT");
  console.log("User name =  " + req.body.username);
  let msg = {
    type: 'connect',
    data: 'sucess'
  };
  //-----------------ANSWER TO CLIENT---------------------------------
  res.send(JSON.stringify(msg));
});
```

  - Resposta requisição de visualização:

```
app.post('/visualise', function (req, res) {
  console.log("POST VISUALISE");
  // ---------------------------------DATABSE CONNECTION --------------------
  var db = new sqlite3.Database('../clima.db', sqlite3.OPEN_READONLY, (err) => {
    if (err) {
      console.error(err.message);
      return;
    } else {
      console.log('Connected to the in-disk SQlite database.');
    }
  });
  let start = req.body.start;
  let end = req.body.end;
  let parameter = req.body.parameter;
  var date1 = ' "' + req.body.start + '" ';
  var date2 = ' "' + req.body.end + '" ';
  var city = req.body.city;
  var query = `SELECT ` + parameter + `, measure_date FROM measurements_daily
      INNER JOIN weather_stations
         ON measurements_daily.weather_station_id = weather_stations.id
         WHERE weather_stations.name LIKE '%`+ city + `%' and
             measurements_daily.measure_date>` + date1 + ` and
             measurements_daily.measure_date<` + date2;
  console.log(query)
  // ------------------------DATABSE QUERY -------------------------------------

  db.all(query, [], (err, rows) => {
    if (err) {
      console.log(err.message);
    } else {
      //console.log(rows)
      let msg = {
        type: 'visualise',
        data: rows,
      };
      //-----------------------ANSWER TO CLIENT--------------------------------
      res.send(JSON.stringify(msg));
    }
  });
});
```

- Resposta requisição de Random Forest(procesamento):

```
app.post('/randomForest', function (req, res) {
  console.log("POST RANDOMFOREST");
  //console.log(req.body);
  if (server_in_use == 0) {
    lock();
    //---------LOCK SERVER----------------------

    //----------------- PYTHON CALL----------------------

    axios.post('http://localhost:5000/server', {
      data: req.body
    })
      .then((response) => {
        let columns = Object.keys(response.data);
        let x = [];
        let label = [];

        for (let i = 1; i < columns.length; i++) {
          x.push(response.data[columns[i]]);
          label.push(columns[i]);
        }

        let msg = {
          x: x,
          label: label,
          list: waiting_list
        }

        waiting_list = [];
        numero_clients = 0;

        //---------------UNLOCK SERVER------------------------------
        unlock();
        res.send(JSON.stringify(msg))
      })
      .catch((error) => {
        console.error(error)
      })
  } else {
    // ----------------WAITING LIST MANAGER---------------------------
    var cont = -1;
    for (let i = 0; i < waiting_list.length; i++) {
      if (waiting_list[i] === req.body.username) {
        cont = i;
      }
    }
    if (cont == -1) {
      waiting_list.push(req.body.username);
      numero_clients++;
    }
    if (waiting_list.length === 0) {
      waiting_list.push(req.body.username);
      numero_clients++;
    }
    let aa = {
      type: 'waiting',
      list: waiting_list,
    };
    //--------------------ANSWER TO CLIENT-----------------------------------
    res.send(JSON.stringify(aa));
  }
});
```

- Implementação Servidor Python:
  - Criando Servidor Flask:

```
app = Flask(__name__)
if __name__ == "__main__":
    app.run(debug=True)
```

  - Servidor Flask e requisição POST:

```
@app.route('/server', methods=['GET','POST'])
def test():
    if request.method == "POST":
        df = pre_process_data( request.json['data']['city'], request.json['data']['start'], request.json['data']['end'], request.json['data']['parameter'] )
        print(df.head())
        a = run_RandomForest(df, request.json['data']['parameter'])
        return a
```

  - Pre processamento para o random forest:

```
def pre_process_data( a,b,c,d ):
    time.sleep(2) # TO DEBUG TIME
    #----------GETTING DATA FROM DATABASE--------------------------------

    conn = sq.connect('../clima.db')
    cursor = conn.cursor()

    city  = a
    date1 = b
    date2 = c
    predict =' "'+ d+'" '

    query = " SELECT * "
    query +=" FROM measurements_daily INNER JOIN weather_stations ON measurements_daily.weather_station_id = weather_stations.id "
    query += ' WHERE weather_stations.name LIKE "'+ city + '"'
    query += ' and measurements_daily.measure_date > "'+ date1 +'"'
    query += ' and measurements_daily.measure_date < "'+ date2+'"'


    df = pd.read_sql_query(query, conn)

    #--------------PREPROCESSING DATA-------------------------

    #--------------SPLITING DATE INTO NEW COLUMNS-------------

    df['year']=[d.split('-')[0] for d in df.measure_date]
    df['month']=[d.split('-')[1] for d in df.measure_date]
    df['day']=[d.split('-')[2] for d in df.measure_date]
    df['utf_hour'] = [ datetime.datetime.strptime(d,  '%Y-%m-%d %H:%M:%S.%f').time().hour for d in df.utf_hour ]

    #---------------DROPING UNUSEFUL DATA----------------------------
    d = list(set(df.columns.values) & set(['measure_date', 'weather_station_id', 'id', 'name', 'province', 'omm', 'inmet_id', 'measure_date_complete']))
    df = df.drop(d, axis=1)

    #----------------FILLING NULL DATA---------------------------

    df = df.groupby(df.columns, axis = 1).transform(lambda x: x.fillna(x.mean()))
    df = df.fillna(0)


    #----------------CONVERT DATA---------------------------
    df = df.transform( lambda x: pd.to_numeric(x, downcast='float'))

    return df
```

- Processamento Random Forest:

```
def run_RandomForest(df, predict):
    y = df[predict]
    X = df.drop(predict, axis=1)

    rotulos = X.columns.values
    #---------------SPLITING DATA--------------------------------

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% TO TRAINNING 30% TO TEST

    lab_enc = preprocessing.LabelEncoder()
    y_train = lab_enc.fit_transform(y_train)
    y_test = lab_enc.fit_transform(y_test)


    clf=RandomForestClassifier(n_estimators=100)
    clf.fit(X_train,y_train)
    y_pred=clf.predict(X_test)
    #-----------------SHOWING RESULTS-----------------------------

    print("Accuracy RandomForest:",metrics.accuracy_score(y_test, y_pred))
    print(len(rotulos))

    #-------------------CREATING OUTPUT DATA-----------------------
    feature_imp = pd.Series(clf.feature_importances_, index = rotulos).sort_values(ascending=False).to_json()

    print(feature_imp)
    return feature_imp
```

4 Cliente(Navegador)

Abaixo podemos visualizar a interface web para visualização e processamento dos dados.

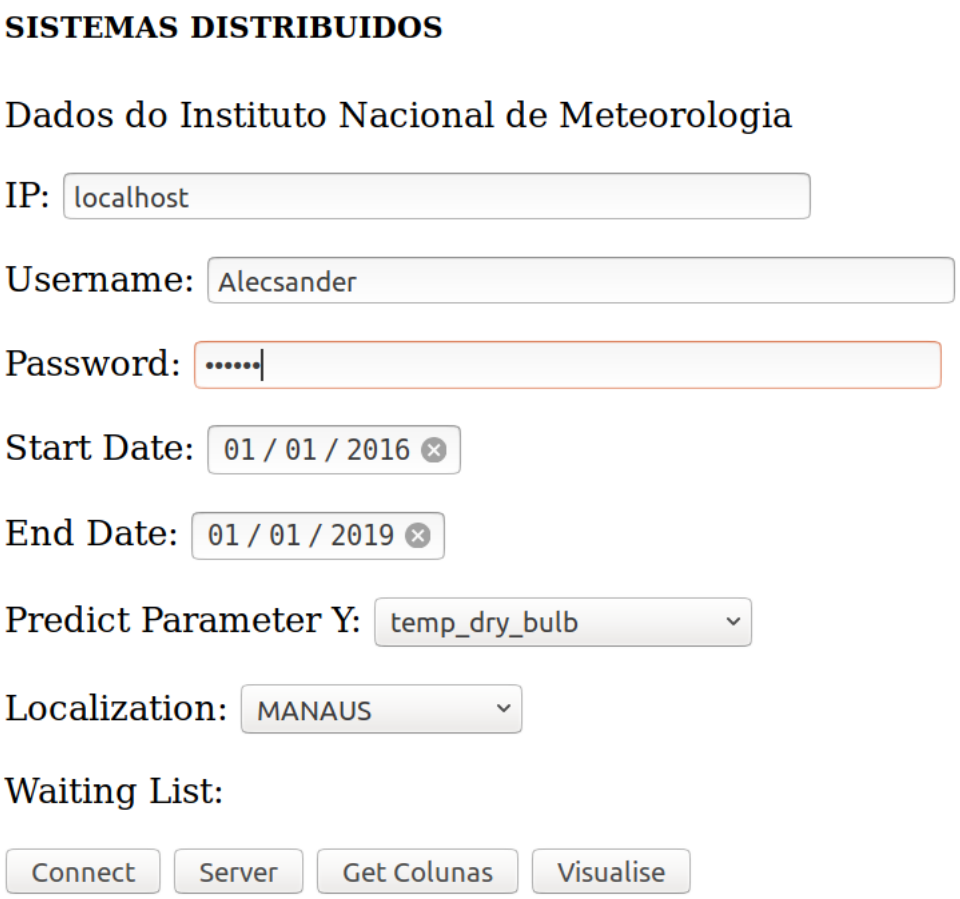

Figura 2: *Interface WEB.*

Assim como o gráfico com resultados:

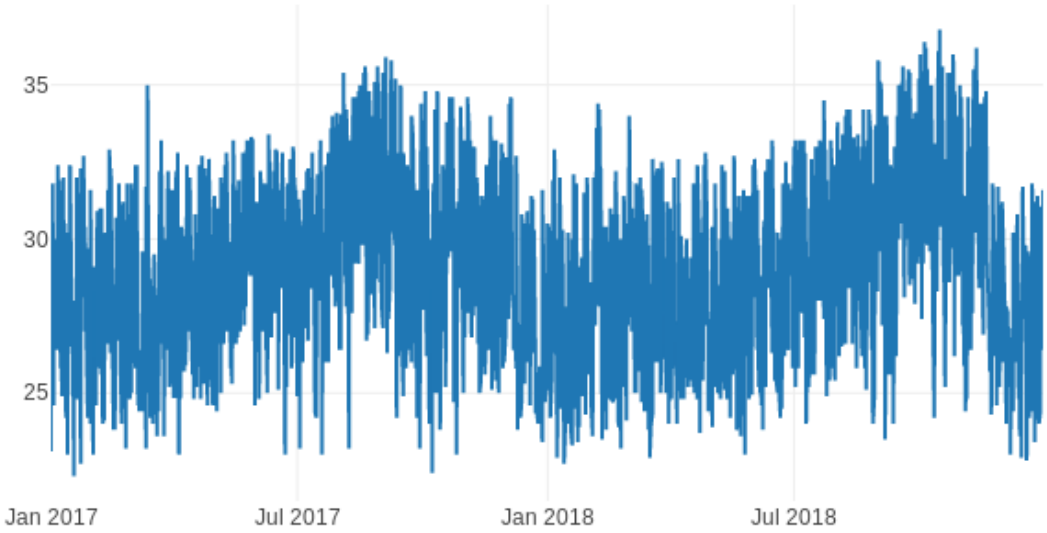

Figura 3: *Gráfico.*

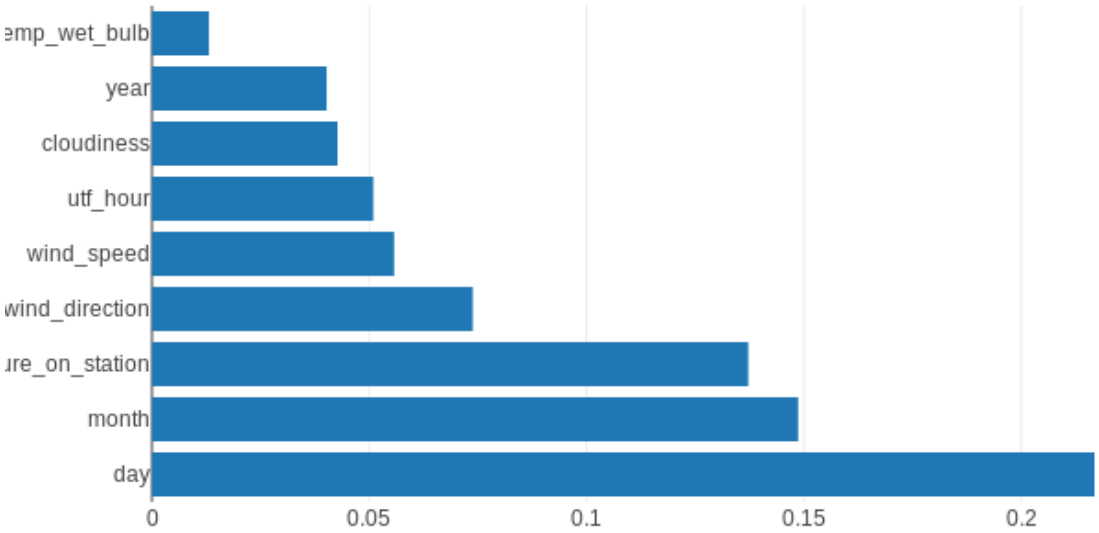

Figura 4: *Importância para cada coluna.*

5 Utilização

5.1 Servidores

Em um terminal execute o seguinte comando:

```
$ node server.js
```

Em outro terminal execute outro comando

```
$ python3 python_server.py
```

5.2 Navegador

Em qualquer navegador, digite o número do IP do servidor a porta :3000/debug. Como na figura abaixo:

Figura 5: *Link*