IPV – Instituto Politécnico de Viseu

ESTGV – Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Relatório do Projeto Final

Licenciatura em Engenharia Informática

Realizado em Sistemas Distribuídos por:

Pedro Martins Gomes, nº18739

Ricardo Jorge Esteves Amaral, nº18756

Gonçalo da Costa Marques, nº 17852

Diogo Miguel Conceição Reis, nº 19221

Supervisor: Carlos Cunha

Viseu, 2021

IPV – Instituto Politécnico de Viseu

ESTGV – Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Relatório do Projeto Final

Licenciatura em Engenharia Informática

Ano letivo 2020/2021

3º ano, 1º semestre

Turno 1

Realizado em Sistemas Embebidos por:

Pedro Martins Gomes, nº18739

Ricardo Jorge Esteves Amaral, nº18756

Gonçalo da Costa Marques, nº 17852

Diogo Miguel Conceição Reis, nº 19221

Supervisor: Carlos Cunha

Viseu, 2021

Índice

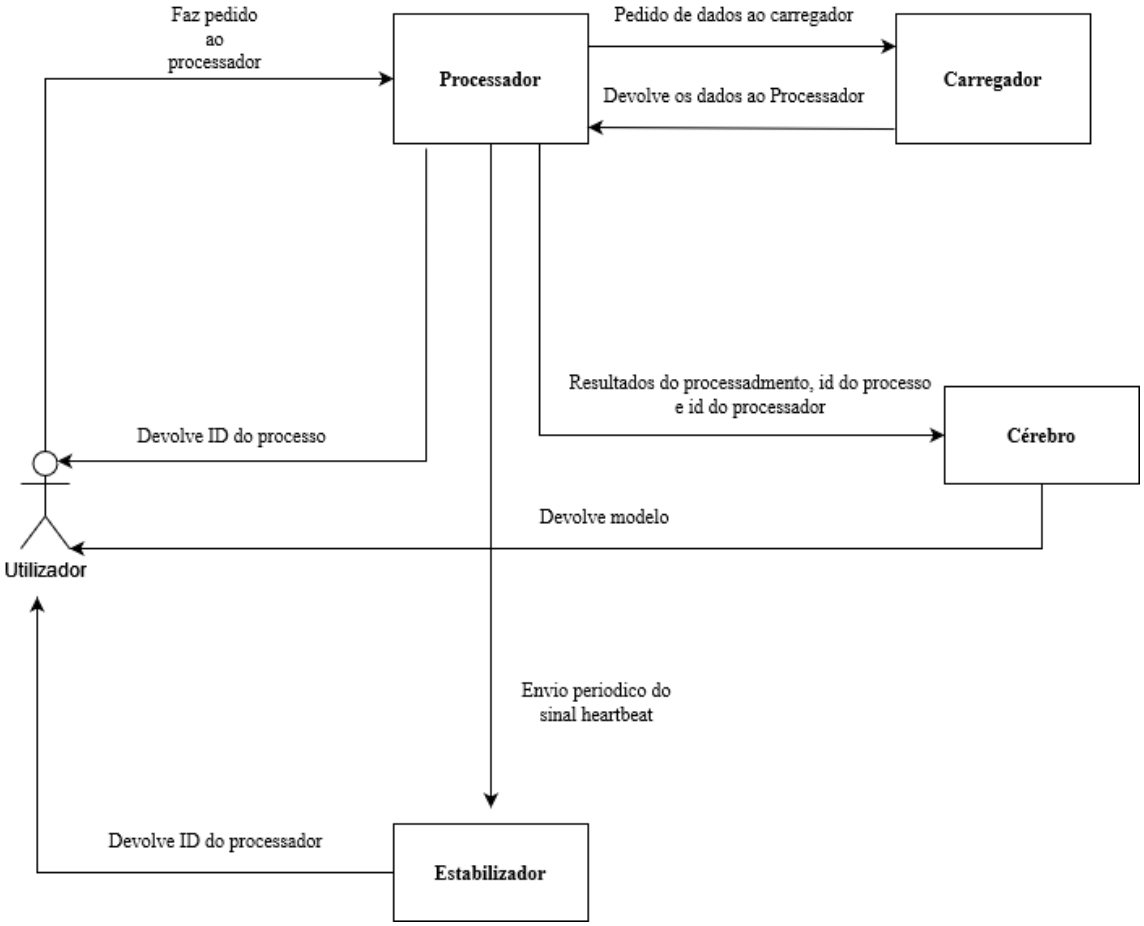
1. Introdução

No âmbito da disciplina de sistemas distribuídos foi proposto o desenvolvimento de uma solução distribuída que permita distribuir os pedidos de computação pelos elementos disponíveis do sistema, de forma a garantir o equilíbrio da carga entre os mesmos. A área de inteligência artificial exige uma grande disponibilidade de recursos computacionais. O CPU é o recurso mais importante nesta área, uma vez que os algoritmos são computacionalmente muito intensivos. Como tal, num sistema distribuído torna-se fundamental rentabilizar a utilização deste recurso nos vários elementos computacionais. Existem vários tipos de elementos:

- Os **processadores** são elementos que executam os algoritmos.
- Os **carregadores** são elementos que armazenam os dados a processar.
- Os **cérebros** são elementos que armazenam os modelos gerados.
- Os **estabilizadores** garantem a distribuição equilibrada da carga pelos processadores.

2. Desenvolvimento

2.1. Arquitetura da solução em UML



2.2. Implementação

No sprint 1 foi pedido a implementação do RF02 onde foram criados um cliente e um processador. O cliente faz um pedido ao processador onde se este tiver recursos ira executar o mesmo, se não é enviado para uma lista de espera. No processador foi criada uma thread onde se vai armazenar os recursos usados pelo CPU numa lista. Quando chega o pedido, o processador vai à lista e faz a media consoante o número de leituras. Se a média for inferior a 40 e enquanto a lista de espera não estiver fazia os pedidos da lista de espera vão ser executados. Se a lista de espera estiver fazia o pedido vai ser executado normalmente. Se a média for superior a 40 o pedido é adicionado à lista de espera. Em qualquer das situações o ID do pedido é devolvido.

```
public String ProcRequest (ProcessRequest pRequest) throws Exception {
    thread1 x;
    x = new thread1();
    int usageCPU = x.sumOfList();

    String pid = UUID.randomUUID().toString();
    pRequest.setpId(pid);

    if(usageCPU < 40) {
        while(!waitL.isEmpty()){
            ExecRequest(pRequest);
            ProcRequestFunc(pRequest);
            waitL.remove( key: 0);
        }
        ExecRequest(pRequest);
        ProcRequestFunc(pRequest);
    }

    else {
        System.out.println("O pedido: " + pRequest.getpScript() + " ficará em lista de espera");
        waitL.put(pid, pRequest);
        System.out.println("A lista de espera tem : ["+waitL.size()+"] scripts por executar");
        // devido à falta de recursos, o pedido fica em lista de espera
    }

    System.out.println("PID: " + pid);
    return pid;
    // geração de id e respetiva devolução
}
```

Sprint 2 (RF03), foi utilizado um programa externo (rebex) que vai fazer o download do ficheiro que se chama no pedido (testfile.txt).

```
ProcessRequest request1 = new ProcessRequest( script: "nslookup sapo.pt",  file: "testfile.txt");
request1.setpId(procManInt.ProcRequest(request1));
showPid(request1);
Thread.sleep( millis: 2000);
```

Excerto da função “getFileRebex()”.

```
        ChannelSftp channelSftp = (ChannelSftp)channel;

        Vector<ChannelSftp.LsEntry> vFiles = channelSftp.ls( path: "/");

        for(ChannelSftp.LsEntry entry : vFiles){
            if(entry.getFilename().equals(file)){
                System.out.println("\nScript: " + entry.getFilename());
                channelSftp.get("/" + entry.getFilename());
                //faz download se o nome do script(2° parâmetro)
                //for igual a um existente na pasta data do Rebex
            }
        }

        //file download
        channelSftp.get( src: "/" + file, dst: "C:\\Users\\Ricardo\\IdeaProjects\\FilesRebex");
                System.out.println("Download feito com sucesso!\n");

        session.disconnect();

    }
    catch (JSchException | SftpException e){
        e.printStackTrace();
    }
}
```

Sprint 3 (RF04 e RF05), o processador atribui ao pedido um cérebro que vai criar um novo modelo com o ID do processador e o ID do processo. Guardando numa ArrayList o ID do processo (para mais tarde ir buscar o modelo) e numa outra ArrayList o modelo completo.

```
public void NewModel(String model1, String procID, String pID) throws RemoteException
{
    System.out.println("-----------Novo modelo-----------");
    System.out.println("modelo gerado: " + model1);
    System.out.println("ID processador: " + procID);
    System.out.println("ID processo: " + pID);
    BrainModel model = new BrainModel(model1, procID, pID);
    ProcIDArray.add(pID);
    //armazena o index do id do processo para ir buscar o modelo ao ModelList
    ModelList.add(model);
}
```

Ao receber um pedido de um modelo do cliente através do ID do processo, o cérebro devolve o modelo associado ao respetivo ID. Isto acontece, pois, o cérebro vai à ArrayList "ProcIDArray" buscar o ID do processo e de seguida procura na "ModelList" o modelo correspondente ao ID encontrado na "ProcIDArray".

```
public String ModelRequest(String pID) throws RemoteException, NumberFormatException
{
    while (true) {
        System.out.println("Modelo gerado do pedido com ID: " + pID);
        String index = ProcIDArray.get(Integer.parseInt(pID));
        BrainModel model;
        model = ModelList.get(Integer.parseInt(index));
        System.out.println("Modelo: " + model.getModel());
        return model.getModel();
    }
}
```

Sprint 4 (RF01) o cliente faz um pedido. O pedido vai para o estabilizador que escolhe o processador com mais recursos computacionais para executar o pedido do cliente.

```
public String[] GetProcessor() throws RemoteException {
    int i = 10000;
    String[] id = null;
    for (Map.Entry<String, triple<Integer, String, LocalDateTime>> entry : map.entrySet()) {
        if (entry.getValue().first < i) {
            assert id != null;
            id[0] = entry.getKey().split( regex: ":")[0];
            id[1] = entry.getKey().split( regex: ":")[1];
        }
    }
    return null;
}
```

Por default é escolhido o processador 1.

```
public static void main(String[] args) {
    try{
        StabilizerManagerInterface stabManInt = (StabilizerManagerInterface) Naming.lookup( name: "rmi://localhost:2025/stabManage

        String idProc, port;
        String[] proc = stabManInt.GetProcessor();
        if(proc == null)
        {
            idProc = "1";
            port = "2022";
        }
        else {
            idProc = proc[0];
            port = proc[1];
        }
```

De 10 em 10 segundos são enviados “heartbeats” pelo processador com a informação do estado (estado = 1 quando processou pelo menos um pedido), informação do CPU, o tamanho da waitlist seguido da descriminação dos pedidos existentes.

```
void SendHeartbeat(int port) throws IOException, InterruptedException {

    int i = 1;

    while(true) {
        thread1 x;
        x = new thread1();
        int CPU = x.Media();

        String mensagem = i + ", ESTADO: " + state + ", CPU: " + CPU + ", WAITLIST: " + waitL.size() + " " + WaitListToString();
        // Se Estado = 1 o processador recebeu pelo menos um pedido
        // Assim dá para identificar o escolhido pelo Estabilizador
        System.out.println("ProcessorID: " + id_processador + " [ MSG N°: " + mensagem + "]");

        DatagramSocket socket = new DatagramSocket();
        InetAddress group = InetAddress.getByName("230.0.0.0");
        byte[] buffer = mensagem.getBytes();
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, port);
        socket.send(packet);
        socket.close();
        Thread.sleep( millis: 10000); //Envia Heartbeat a cada 10 segundos
        i++;
    }
}
```

Sprint 5 (NF03), o estabilizador esta sempre a verificar se os processadores ainda estão ativos através da função “CheckProcessor()”, que passado mais de 30 segundos de inatividade (não mandar heartbeat) , é removido o processador da lista de processadores disponíveis.

```
public void CheckProcessor() throws InterruptedException {
    while (true) {
        for (Map.Entry<String, triple<Integer, String, LocalDateTime>> entry : map.entrySet()) {
            if((Duration.between(entry.getValue().third,LocalDateTime.now())).getSeconds() > 30){
                try {
                    if(!entry.getValue().second.equals("null")) {
                        SendRequest(entry);
                        System.out.println("Removendo o: " + entry.getKey());
                        map.remove(entry.getKey(), entry.getValue());
                    }
                }catch (RemoteException | MalformedURLException | NotBoundException ignored){
                }
            }
        }
        Thread.sleep( millis: 1000);
    }
}
```

Sprint 6 (NF05), se o estabilizador falhar, o estabilizador substituto, através da função "SendRequest()", vai enviar um pedido ao processador "ativo" para obter o seu estado. Recorrendo à função "SaveDataProc()" é possível recuperá-lo.

```java
public void SendRequest(Map.Entry<String, triple<Integer, String, LocalDateTime>> deadProc) throws RemoteException, MalformedURLException, NotBoundException {
    int max = -99;
    String[] proc = null;

    for(Map.Entry<String, triple<Integer, String, LocalDateTime>> entry : map.entrySet())
    {
        if(entry.getValue().first > max) {
            max = entry.getValue().first;
            assert false;
            proc[0] = entry.getKey().split( regex: ":")[0];
            proc[1] = entry.getKey().split( regex: ":")[1];
        }
    }

    if(proc != null)
    {
        String port = proc[1];
        ProcessManagerInterface pmi = (ProcessManagerInterface) Naming.lookup( name: "rmi://localhost:" + port + "/pRequestManager");
        pmi.ResumeProc(deadProc.getValue().second);
    }
    //envia a waitlist ao processador
}
```

```java
void SaveDataProc() throws IOException {
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
    socket.receive(packet);
    String received = new String(packet.getData(), offset: 0, packet.getLength());

    if(received.equals("restore;"))
    {
        socket.receive(packet);
    }
    received = new String(packet.getData(), offset: 0, packet.getLength());
    String[] procs = received.split( regex: "#");

    for (String p : procs) {
        String[] info = p.split( regex: ";");
        map.put(info[0],
                new triple<>(
                        Integer.parseInt(info[1]),
                        info[2].equals("null") ? null : info[2],
                        LocalDateTime.parse(info[3])
                )
        );
    }
}
```

3. Conclusão

Ao longo do trabalho surgiram várias dificuldades, atingindo um resultado não perfeito, mas bastante positivo. Conseguindo então finalizar a realização do trabalho prático. As principais dificuldades foram a implementação do NF03 e NF05, que embora realizadas não estão totalmente funcionais. De forma a talvez resolver estes problemas teríamos de mudar a forma como os heartbeats funcionam e dessa forma seria possível implementar as funções pedidas corretamente.

Este trabalho ajudou também a desenvolver as nossas competências de investigação, de organização enquanto grupo e na discussão entre cada elemento sobre diferentes pontos de vista.