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

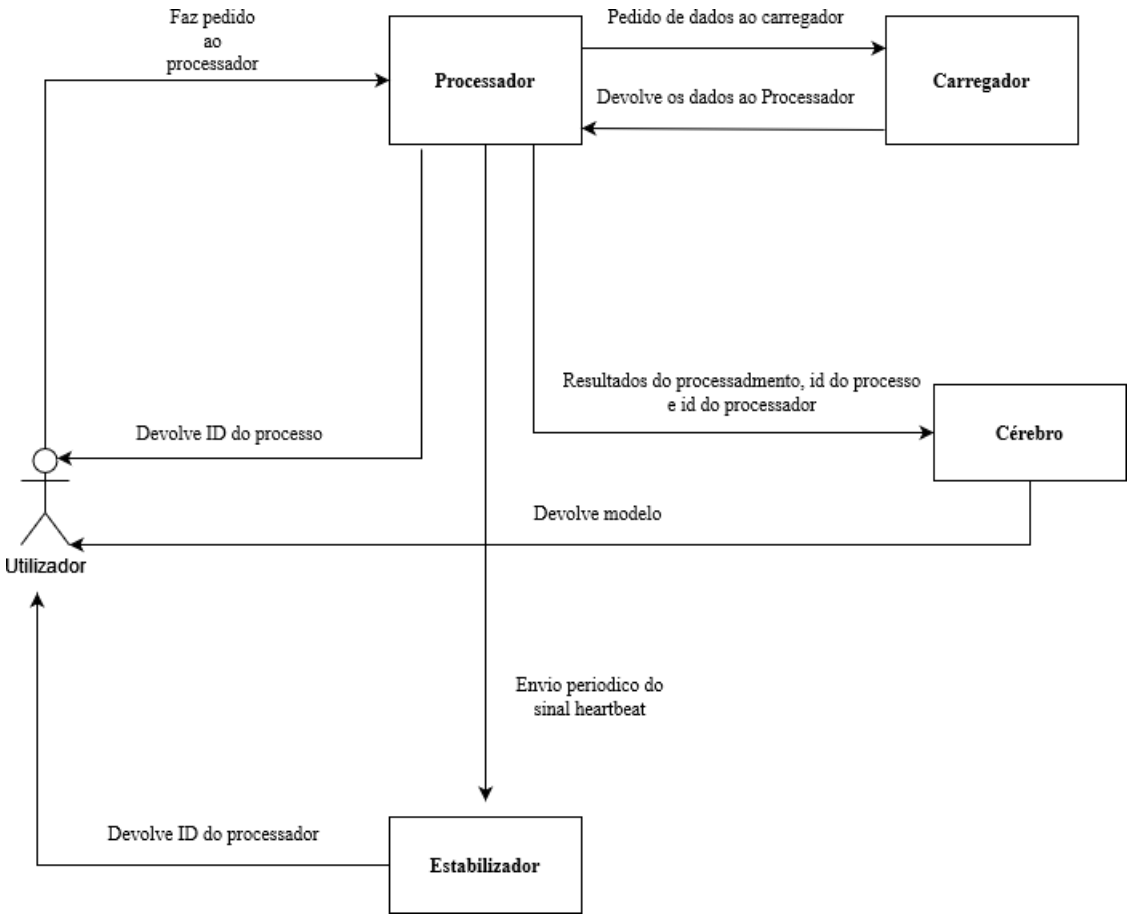
1. Introdução

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 pedido. Guardando num HashMap oID do pedido (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 + "ID processador: " + procID + "ID pedido: " + pID);
    BrainModel model = new BrainModel(model1, procID, pID);
    ReqHMap.put(pID, ModelList.size()); //guarda o pID para depois ir buscar o modelo gerado ao Array
    ModelList.add(model);
}
```

Ao receber um pedido de um modelo do cliente através do ID do pedido, o cérebro devolve o modelo associado ao respetivo ID. Isto acontece, pois, o cérebro vai ao HashMap “ReqHMap” procurar o ID do pedido e de seguida procura na “ModelList” o modelo correspondente ao ID encontrado no “ReqHMap”.

```
public String ModelRequest(String pID) throws RemoteException, NumberFormatException
{
    System.out.println("Modelo gerado do pedido com ID: " + pID);
    if(ReqHMap.containsKey(pID))
    {
        System.out.println("Model Request: OK!");
        int idModel = ReqHMap.get(pID);
        BrainModel model = ModelList.get(idModel);
        System.out.println("Modelo devolvido: " + model.getModel());
        return model.getModel();
    }
    else
    {
        System.out.println("Modelo não gerado!");
        return "NULL";
    }
}
```

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 ChosenOne() throws RemoteException {
    iProcessor p1 = new iProcessor();
    float cpu = 100;
    int queue = 100;
    if(InfoProc.isEmpty())
    {
        System.out.println("Sem processadores disponíveis!");
        return "NULL";
    }
    else
    {
        String theChosenOne = InfoProc.get(IDProc.get(0)).getPid();
        //o processador com mais recursos é considerado o primeiro da fila se nenhum estiver disponível

        for(int i = 0; i < IDProc.size() ; i++)
        {
            p1 = InfoProc.get(IDProc.get(i));
            if(p1.getState() == 0) //verifica apenas processadores que não estão a processar
            {
                if(p1.getCpu() < cpu && p1.getQueue() < queue)
                {
                    cpu = p1.getCpu();
                    queue= p1.getQueue();
                    theChosenOne = p1.getPid();
                }
            }
        }
        System.out.println("Processador com mais recursos: " + theChosenOne);
        return theChosenOne; //devolve id do processador
    }
}
```

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

```
void SendHeartbeat(int port) throws IOException, InterruptedException {
    int i = 1;
    String type;

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

        type = "hb";
        if(setup)
        {
            type = "setup";
            setup = false;
        }
        String mensagem = id_processador + "," + type + "," + state + "," + CPU + "," + waitL.size() + "," + WaitListToString();
        /*String mensagem = i + " " + "TIPO: " + type + ", ESTADO: " + state + ", CPU: " + CPU + ", WAITLIST: "
                + waitL.size() + " -> " + WaitListToString();*/
        // Se Estado = 1 o processador recebeu pelo menos um pedido
        System.out.println("MSG:" + mensagem + "]");
        //System.out.println("ProcessorID: " + id_processador + " [ MSG N°:" + mensagem + "]");
        sendBroadcast(port, mensagem);
        Thread.sleep( millis: 10000); //Envia Heartbeat a cada 10 segundos
        i++;
    }
}
```

Quando um estabilizador recebe um heartbeat do tipo "setup", vindo de um processador que não se encontra na lista, adiciona o processador. O “heartbeat” vem separado por virgulas de forma a podermos separar o mesmo por partes. O estabilizador confirma o tipo, se for “setup” adiciona o processador à lista de id's "IDProc", bem como todas as informações do processador são adiconadas à lista "InfoProc". Se for “hb” verifica se já foi feito “setup”, se não tiver sido feito o heartbeat é ignorado.

```
private void HeartbeatProcess(String msg){
    String[] part = msg.split( regex: ","); //mensagem separada por virgulas
    iProcessor p1 = new iProcessor();

    if (part[1].equals("setup")) { //se heartbeat = "setup", vamos adicionar o processador que o enviou à lista de IDs permitidos
        IDProc.add(part[0]); //Array de IDs de processadores para utiliza posteriormente
        p1.setPid(part[0]);
        p1.setState(Integer.parseInt(part[2]));
        p1.setCpu(Float.parseFloat(part[3]));
        p1.setQueue(Integer.parseInt(part[4]));
        p1.setActive(true); //coloca os processadores como "ativos"
        InfoProc.put(part[0], p1); //HasMap <ID processador, dados processador>
    }

    if(part[1].equals("hb")){ //se heartbeat = "hb" vamos verificar se este processador já fez o setup
        if(IDProc.contains(part[0])) // se fez setup
        {
            p1.setPid(part[0]);
            p1.setState(Integer.parseInt(part[2]));
            p1.setCpu(Float.parseFloat(part[3]));
            p1.setQueue(Integer.parseInt(part[4]));
            InfoProc.put(part[0], p1);
            p1.setActive(true); //coloca os processadores como "ativos"
        }
        else { System.out.println("Processador não fez setup"); } //ignora mensagem
    }
}
```

Sprint 5 (NF03), o estabilizador esta sempre a verificar se os processadores ainda estão ativos através da função “ActivityCheck()”, que passado mais de 30 segundos de inatividade (não mandar heartbeat) , é removido o processador da lista de processadores disponíveis. De seguida, é anunciada a sua falha pela função "AnnounceProcessor()", que vai buscar o endereço do processador que tiver mais recursos ("ChosenOne()"), resumindo os pedidos pendentes do processador que falhou. Isso é feito pela função "ResumeProc()" no processador, que vai verificar a fila do processador que falhou e, para cada pedido da fila, se o modelo para cada pedido não foi gerado, o novo processador adiciona-o à sua própria fila de espera. No caso de o antigo não ter nenhum pedido na fila, o novo prossegue sem problema.

```java
private void ActivityCheck() throws InterruptedException, IOException, NotBoundException {

    iProcessor p1 = new iProcessor();
    String id = null;

    while(true)
    {
        Thread.sleep( millis: 30000); //A cada trinta segundos...
        //System.out.println("A verificar atividade...");

        if(InfoProc.size() == 0) { System.out.println("A aguardar..."); } //se o HasMap ainda estiver vazio
        else{
            for(int i = 0; i < InfoProc.size() ; i++)
            {
                p1 = InfoProc.get(IDProc.get(i));
                if(!p1.getActive()) //se o processador estiver inativo
                {
                    id = IDProc.get(i);
                    System.out.println("Processador com id: [" + id + "] falhou!");
                    System.out.println("A remover...");
                    InfoProc.remove(id);
                    IDProc.remove(id);
                    Thread.sleep( millis: 1000);

                    //anúncio da falha do processador
                    DatagramSocket socket = new DatagramSocket();
                    InetAddress group = InetAddress.getByName("230.0.0.0");
                    byte[] buffer = id.getBytes();
                    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group,  port: 4448);
                    socket.send(packet);
                    socket.close();

                    if(IDProc.isEmpty())
                    {
                        System.out.println("Não existem processadores ativos");
                    }
                    else { AnnounceProcessor(id); } //avisa o processador com mais recursos
                }
                else //se o processador estiver ativo
                {
                    System.out.println("Processador com id: " + p1.getPid() + " está ativo");
                }
            }
            for(int i = 0; i < InfoProc.size() ; i++)
            {
                p1 = InfoProc.get(IDProc.get(i));
                p1.setActive(false); //coloca os processadores como inativos
                InfoProc.put(IDProc.get(i), p1);
            }
        }
    }
}
```

Sprint 6 (NF05), ao iniciar, o estabilizador manda uma mensagem em multicast para saber quais os processadores ativos no momento. A função "AliveProcListener" é responsável pela captação de respostas da parte dos processadores. Se a mensagem recebida for igual à mandada pelo estabilizador ao iniciar, o processador que a recebe, envia o seu porto e a lista de processadores ativos "AliveProc" por broadcast. Se após 5 segundos não obtiver resposta, assume que não existe nenhum processador ativo. Se existir um ativo vai então receber os registos de heartbeats e de filas desse processador. É verificado se o processador fez setup, e em caso positivo regista a fila de espera do mesmo. Isto faz com que se um estabilizador falhar, o estabilizador substituto consiga retomar o registo de heartbeats e de filas dos outros processadores sem estes terem que voltar a fazer setup.

```
private void AliveProcListener(int port) throws IOException {
    MulticastSocket socket1 = null;
    byte[] buffer = new byte[256];
    socket1 = new MulticastSocket(port);
    InetAddress group = InetAddress.getByName("230.0.0.0");
    socket1.joinGroup(group);

    while (true) {
        DatagramPacket packet1 = new DatagramPacket(buffer, buffer.length);
        socket1.receive(packet1);
        String msg = new String(packet1.getData(),  offset: 0, packet1.getLength());
        if(!msg.equals("stab_setup")) //ignora a própria mensagem
        {
            System.out.println("Processador ativo: " + msg);
            if(!IDProc.contains(msg)) //vai receber ID's repetidos dado que todos os processadores vão responder
            {
                IDProc.add(msg);
            }
        }
    }
}
```

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.

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.