Curso de Graduação em Ciência da Computação
Disciplina: Sistemas Distribuídos
Trabalho Final
Professor: Everthon Valadão

Relatório Trabalho Final

Alunos:	1- Eduardo Gabriel Reis Miranda	Nº 0026495
	2 - Flávia Santos Ribeiro	Nº 0022651
	3 - Jonathan Arantes Pinto	Nº 0021625
	4 - Luís Fernando da Silva Corrêa	Nº 0022644

Formiga, 29 junho de 2019

Objetivo:

Projetar um sistema distribuído (SD) que proveja um Serviço de e-Marketplace. A infraestrutura do SD deverá ser provida pelo middleware JGroups, com a comunicação entre os componentes — membros do(s) cluster(s) — realizada através de conectores JChannel, do componente MessageDispatcher (multicast, anycast, unicast), demais building blocks e protocolos que se fizerem necessários.

Arquitetura do Sistema:

A comunicação funciona da seguinte maneira, como mostrada logo abaixo na Figura 1. Foram inseridas camadas na distribuição vertical, que são Modelo, Controle e Visão. São necessárias no mínimo uma réplica do Modelo e uma do Controle em cada na distribuição horizontal.

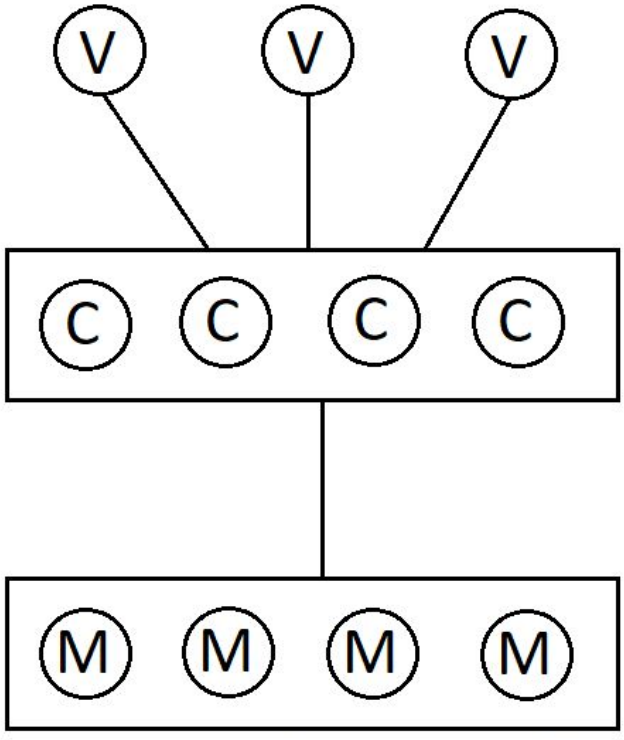

Figura 1 - Modelo Comunicação

Vamos resumir brevemente como funciona cada camada de distribuição.

Como funciona o Modelo:

O '*Modelo*' ou a parte do sistema responsável por manter a persistência dos dados, utiliza uma estrutura simples para o armazenamento das informações, em arquivos no formato *JSON*.

Os dados mantidos em memória estão armazenados dentro de objetos do tipo *JSONObject*, que implementa a interface *Map*, facilitando o acesso aos dados.

A comunicação entre o *modelo* e o *controle* é feita através de um canal único exclusivo. No formato atual do sistema, todos os *modelos* são distintos e não interagem entre si.

Como funciona o Controle:

O controle é a parte do sistema responsável por fazer o intermédio entre a visão e o modelo. Ele recebe as mensagens da visão, verifica se está tudo correto e só então manda uma mensagem para que seja salva a informação no modelo caso seja uma operação de escrita.

Como funciona a Visão:

A visão é quem controla a parte das telas do nosso sistema. O início dela é a tela de *Cadastro* e/ou *Login,* onde o usuário precisa realizar o cadastro e após realizar o *Login*. Com o Login realizado você tem acesso ao sistema dependendo do seu tipo de conta. Nela podemos controlar funções de usuários e/ou de vendedor.

Principais Decisões:

- Existem dois canais de comunicação um CV (que é do Controle com a Visão) e outro CM (que é do Controle com o Modelo).
- Utilização do arquivo Json para fazer a persistência dos dados.
- Padronização das mensagens através de uma classe "*Comunication*" que utiliza enumerações.

Pontos fortes:

- Nossa arquitetura está bem modularizada, então pode-se adaptar novas funcionalidades facilmente.

Pontos fracos:

- Não há um balanceamento de carga nas requisições que o controle faz pro modelo.

Pilha de Protocolos:

A seguir está a configuração de cada protocolo utilizado bem como justificativas:

O protocolo de transporte utilizado é o UDP:

- *ip_mcast* : habilita o *multicast* pro *IP*.
- *mcast_send_buf_size* : define o tamanho do *buffer* de envio.
- *mcast_recv_buf_size* : define o tamanho do *buffer* de recebimento.

```
<UDP
           ip_mcast="true"
            mcast_send_buf_size="150000"
            mcast_recv_buf_size="150000"/>
```

PING é utilizado na detecção de membros e também para descobrir quem é o coordenador. Ele envia um ping para endereço *multicast IP*

```
<PING/>
```

Para detecção de falhas é utilizado FD_SOCK com FD_ALL

- *FD_SOCK* : verifica se algum membro do grupo está 'morto' para que o consenso não seja travado.
- *FD_ALL* : utilizado para verificar o consenso.

- *timeout* : Utiliza-se timeout para detectar se um membro está morto.
- *interval* : Intervalo no qual uma *HEARTBEAT* é enviada para o *cluster*.
- *timeout_check_interval* : Intervalo no qual os tempos limite do *HEARTBEAT* são verificados.

```
<FD_SOCK/>
<FD_ALL
           timeout="10029"
           interval="3529"
           timeout_check_interval="1629"/>
```

VERIFY SUSPECT é utilizado na verificação quando se suspeita de um membro estar morto, é necessário enviar um ping antes de excluí-lo do grupo.

- *timeout*: Ao se suspeitar de um membro é necessário um número (em milissegundos) para aguardar sua resposta.

```
<VERIFY_SUSPECT timeout="823"/>
```

NAKACK2 e *UNICAST3* são utilizados na confirmação de recebimento das mensagens, também são usados para realizar e garantir a entrega sem perdas. São *Multicast* e *Unicast*. respectivamente.

```
<pbcast.NAKACK2/>
<UNICAST3/>
```

TOTAL ORDERED ANYCAST é utilizado quando o *anycast* faz ordenação total das mensagens.

```
<tom.TOA/>
```

STABLE é utilizado para excluir mensagens que foram vistas por todos os membros do grupo e coleta lixo de mensagens.

```
<pbcast.STABLE/>
```

O *GMS* é quem controla a entrada e saída de membros no cluster, nova visão.

- *join_timeout* : Limite de tempo para juntar-se.
- *view_ack_collection_timeout* : Tempo em milissegundos para esperar por todas as *VIEW acks*.

```
<pbcast.GMS
        join_timeout = "2829"
        view_ack_collection_timeout = "3429"/>
```

SEQUENCER é quem mantém a ordem total do provedor para a mensagem *multicast*, ele estabelece ordem global.

```
<SEQUENCER />
```

Resultados e Conclusões:

Podemos notar que a utilização de um MiddleWare facilita em muito, só de não ter que tratar coisas pequenas nos economizou bastante tempo.

A realização deste trabalho nos proporcionou um conhecimento extra e em relação à prática do mesmo fez com que pudéssemos colocar em prática o que foi visto em sala de aula.

Bibliografia:

[1] Reliable group communication with JGroups. Acesso em 29 junho de 2019. Disponível em <http://www.jgroups.org/manual/index.html>.