HA3 (Programação em Cluster e Multicore)

A nível geral, existem 2 tipos de actores e 10 tipos de messages.

BinaryTreeActor é um actor transforma a mensagem do actor cliente em mensagem que a tree conhece e vice versa mas com uma particularidade: tem uma PriorityQueue que vai armazenando as mensagens que estão fora de ordem e em cada mensagem que recebe da tree, vai vendo se envia para o cliente e existem mais para mandar, ou guarda-a na queue para mais tarde enviar.

NodeActor representa um nó da árvore. Tem uma lista com os 0, 1 ou 2 filhos e o seu índex (caso haja), tem uma flag para verificar se está a ser apagado, e o seu conteúdo enquanto node de uma árvore.
Para qualquer operação o cliente manda uma *RequestMessageClient* para a *BinaryTreeActor,* a mesma converte para uma *RequestMessage,* incrementa o contador de id's e manda a mensagem para a raiz da tree.

Na operação *insert* procede-se quase como uma BST normal. Caso o conteúdo da mensagem seja igual ao do nó, manda uma *ResponseMessage* com o resultado para o *BinaryTreeActor.* Caso não seja igual, recorrendo ao less, ao greater e à lista de filhos, verifica se é para inserir um filho (e enviar uma *ResponseMessage*) ou passar a mensagem a algum filho já existente.

Na operação *contains,* se o conteúdo da mensagem é igual ao conteúdo do nó, é enviada uma *ResponseMessage.* Se não, recorrendo ao less, ao greator e à lista de filhos, manda a mesma mensagem para o filho se existir. Se não existir, manda uma *ResponseMessage* com a devida informação.

Por fim na operação *delete,* a mensagem propaga-se na mesma forma que na operação *contains.* Mas quando o conteúdo da mensagem é igual ao do nó, a flag *isDeleted* é tornada *true* para as mensagens anteriores poderem saber que foi "eliminado" um nó é feito uma verificação para aferir se o nó é folha, tem um filho ou tem 2 filhos. Se for um nó folha, a flag *isDeleted* é tornada *true* e tem que enviar uma mensagem (*KillMeMessage*) ao *supervisor* para o matar porque as variáveis *greater* ou *less* tem de ser atualizadas para -1 (não tem filho). Se for um nó com um filho (quer seja *greater* ou *less*), o ator tem de mandar uma mensagem (*PleaseHaveMyGreaterSon* ou *PleaseHaveMyLessSon*) ao *supervisor*, no supervisor é enviada uma menssagem de volta ao nó para apenas de suicidar e não matar os filhos (*JustSuicide*) e é substituída a referência. Em suma, se o nó tiver 2 filhos a flag *isDeleted* é tornada *true* é enviada uma mensagem (*GiveMeLessSonMessage*) a pedir o filho mais pequeno da árvore maior (greater). Quando é encontrado, é enviada uma mensagem (*CanSwapMessage*) ao nó a ser eliminado com o conteúdo do filho mais pequeno para substituir e sem a eliminar(pois as mensagens que estão entre o nó a ser eliminado e o filho mais pequeno, não sabem da substituição).É substituído, a flag *isDeleted* tornada *false* e emitido uma operação *delete* oculta para o cliente com o objetivo de apagar o filho mais pequeno visto que foi substituído. Com isto, temos em conta que quando a flag *isDeleted* está a *true*, todas as mensagens são reenviadas de volta para o *sender* para mais tarde serem reenviadas de volta já com o nó atualizado e com o *isDeleted* a *false* para poder continuar. É como um lock. Também temos em conta que as mensagens têm todas a referência do ator cliente e do ator *BinaryTreeActor.*

André Nicolau
47880