Implementação de um Serviço utilizando Comunicação Inter-processos através de Remote Method Invocation - RMI

Prof. Adriano Fiorese

1 Caracterização do Serviço

O serviço a ser desenvolvido é o de *echo* de mensagem remota com replicação das mensagens entre servidores e tolerância a falhas. Nesse trabalho, o(s) cliente(s) deve(m) executar as seguintes operações:

- *Echo*: Permite a invocação do método *echo* no servidor de *echo*. Tal operação passa como argumento a mensagem a ser devolvida como um eco, e que também será replicada no(s) servidor(es) clone(s) réplica. Na Figura 1 essa operação está representada como envio da msg (operação *echo*).
- Obter a Lista de Mensagens já ecoadas: Essa operação permite invocar o método getListOfMsg no(s) servidor(es) de *echo* de forma a obter a lista completa de mensagens já enviadas anteriormente pelos clientes. Assim, quando o cliente invocar esse método no servidor, o retorno do método será a lista de mensagens que representa o histórico de mensagens já enviadas pelo(s) cliente(s).

O sistema funciona de forma replicada, como forma de garantir tolerância a falhas. Isso significa que as operações realizadas por uma entidade (servidor) devem ser replicadas em todas as demais réplicas (servidores). Ou seja, além das operações que podem ser executadas pelos clientes, o servidor também deverá executar (invocar) uma operação de replicação no(s) outro(s) servidor(es) réplica. É possível utilizar o paradigma publish/subscribe para realizar a replicação das mensagens apenas, ao invés de ser necessário executar uma operação de replicação nos servidores clones (réplicas). Para tanto será possível utilizar um Broker MQTT que notificará os servidores clones enviando a msg recebida na operação echo, por meio de sua publicação (operação Publish(Msg,topic)). Na Figura 1 essa operação é ilustrada como notification(Msg, topic). Portanto, para que isso aconteça, em primeiro lugar, assim que inicializados, os servidores clones deverão assinar o(s) tópico(s) que representa(m) o encaminhamento da mensagem para replicação. Assim, esses servidores farão o papel de subscribers e na Figura 1, tal operação é ilustrada como subscribe(topic).

Portanto, nesse caso, será possível que quando o cliente invoque o método *echo* do servidor mestre, este servidor durante a execução desse método execute a operação publish (passando a msg como argumento) no Broker MQTT. O Broker MQTT, por sua, vez notificará aqueles que previamente assinaram ao topico que representa a replicação da mensagem. Atente para o fato de que pode haver n servidores réplica. Ou seja, os servidores réplica deverão assinar o(s) tópico(s) para receberem as mensagens enviadas ao servidor mestre. Tanto o servidor mestre (o servidor que atualmente atende o cliente) quanto o(s) servidor(es) replica deverão manter todas as mensagens recebidas através da invocação do método *echo* em uma lista de mensagens que poderá ser acessada quando da invocação do método getListOfMsg por parte do cliente.

Importante notar que quando o servidor mestre não estiver ativo (for derrubado, desligado, etc...), a invocação dos métodos echo(...), e getListOfMsg(...) deverá ser feita transparentemente para o próximo servidor réplica (clone) que for eleito dentre os atualmente ativos, de forma que a falha de omissão do servidor mestre seja tolerada, de forma transparente para o usuário. Naturalmente que para tal funcionar de acordo, os servidores devem ser instanciados antes que o(s) cliente(s). Ou seja, o código fonte do servidor é único. A quantidade de servidores réplica será equivalente a quantidade de vezes que o executável desse código for executada diretamente na linha de comando, por exemplo. Também é possível utilizar um objeto remoto de controle para manter

uma lista de nomes de objetos remotos que representem os servidores replica, de forma que seja possível aos clientes/(servidores replica) eliminarem a primeira posição da lista e buscarem sempre o que se torna primeiro nome dessa lista no serviço de nomes RMI, quando da falha do servidor mestre, tornando-o o servidor mestre. Esse é um exemplo de algoritmo de eleição centralizado, mas o ideal é que a eleição ocorra de forma distribuída. Da mesma forma, quando um novo servidor for instanciado com um nome de objeto remoto específico, ele deve ser adicionado ao final dessa lista. Uma forma de possibilitar que a eleição inicie é o monitoramento por parte das réplicas do servidor mestre. Isso pode ser feito, executando método remoto nele que simplesmente indique que ele está ativo, por exemplo.

Novos servidores entrantes no sistema deverão subscrever a replicação das mensagens junto ao Broker MQTT (assinar o(s) tópico(s) para receber notificações da operação de replicação de mensagem). Além disso, deverão, de alguma forma, obter as mensagens já ecoadas de forma que o sistema como um todo continue coerente. Ainda, deverão participar do processo de eleição da mesma forma que os demais, caso o mestre seja considerado inativo. Para tanto, é possível que cada clone (réplica) monitore o funcionamento do mestre e inicie uma eleição assim que detectar a falha de omissão do mesmo. Observando isso, será necessário que o cliente obtenha a referência remota de objeto remoto do novo servidor mestre. Todo esse processo deverá ser transparente para o usuário final.

Quando ocorrer a eleição de um novo servidor mestre entre os servidores réplica (clones), este servidor réplica (clone) que passará a ser o mestre deverá fazer o unsubscribe relativo ao tópico de replicação de mensagem junto ao Broker MQTT. Isso é necessário pois todo clone deve ao inicializar inscrever-se (fazer o subscribe) no referido tópico; assim caso ele não se desinscreva, acabará sendo notificado também quando fizer a publicação (publish) de uma nova mensagem recebida de um cliente.

2 Implementação

A implementação de tal serviço deverá ser realizada utilizando-se comunicação Inter-Processos por meio de RMI. Recomenda-se utilizar o middleware RMI Java. Além da linguagem java com sua biblioteca RMI apenas a linguagem python, preferencialmente python3, com a biblioteca Pyro4.0 será aceita. O modelo arquitetural a ser utilizado deverá ser o cliente/servidor, na variação múltiplos servidores.

Uma máquina virtual (VM) Ubuntu 20.04 com o Broker MQTT mosquitto instalado, bem como o framework Eclipse Paho Java, para desenvolvimento de clientes MQTT, está disponível na Seção Material Extra do moodle da disciplina.

Na Figura 1, podemos observar as invocações de métodos da aplicação necessárias para a perfeita execução do serviço solicitado.

3 Prazos

A entrega ocorrerá via moodle, envolvendo executáveis e código fonte, de maneira individual. na forma de um arquivo zipado ou de maneira a ser possível fazer o download dos executáveis e fonte. Documentação acerca de como criar um projeto com o código fonte (e bibliotecas utilizadas), preferencialmente no IDE Intellij IDEA no Ubuntu, deverá acompanhar os arquivos fontes. A entrega deverá ser no dia 28/06/2023, sem possibilidades de adiamento.

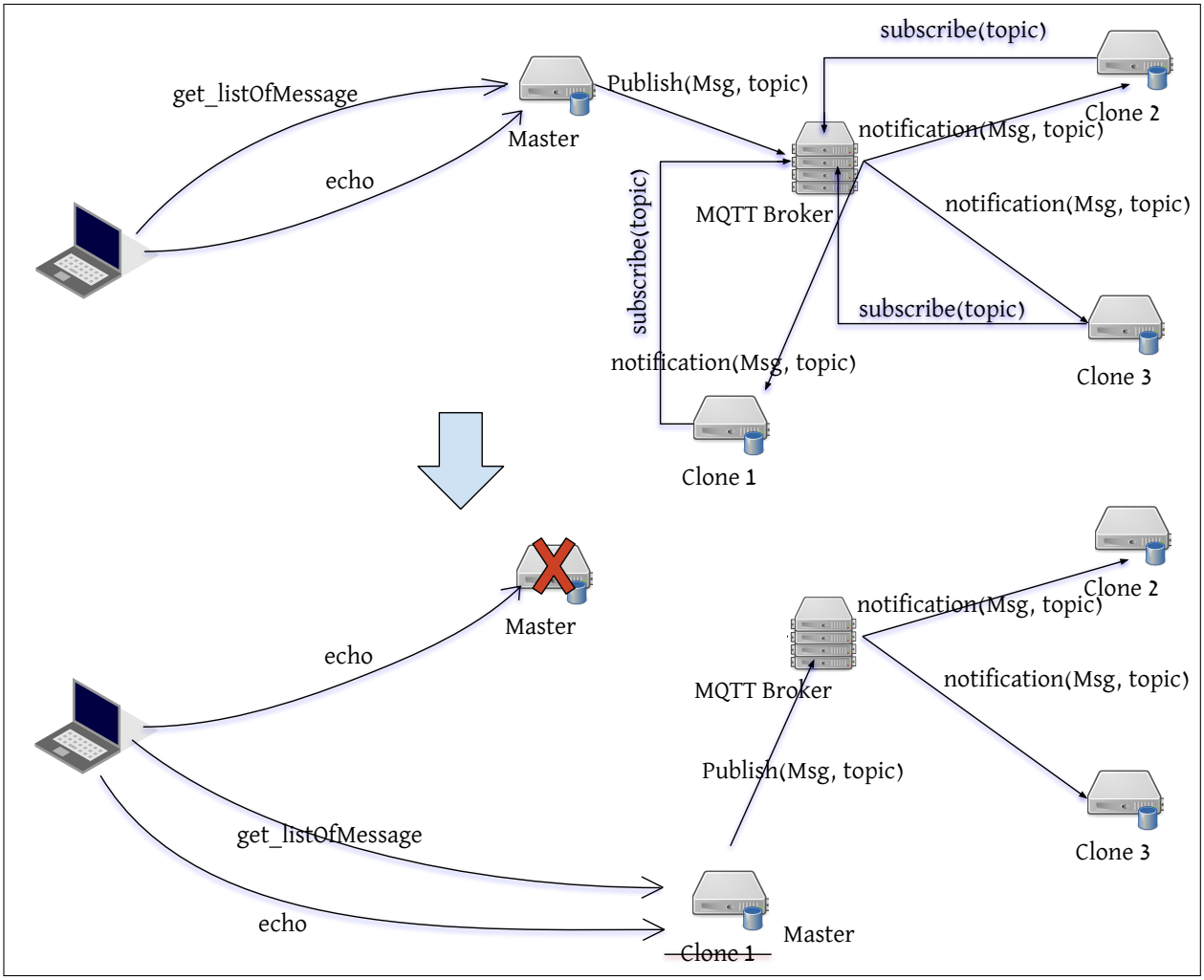

Figure 1: Serviço de *Echo* de Mensagens com replicação tolerante a falhas