DALUA: BIBLIOTECA PARA APLICAÇÕES DISTRIBUÍDAS

Aluno: Ricardo Gomes Leal Costa
Orientadora: Noemi de la Rocque Rodriguez

Introdução

A biblioteca DALua [1], fruto do projeto anterior, tem por objetivo oferecer uma série de facilidades para o desenvolvimento de aplicações distribuídas, ou seja, executadas por diversas máquinas, possivelmente em posições geográficas distantes, trabalhando em paralelo.

Para o projeto deste ano, pretendíamos implementar técnicas de migração de processos Lua [3] entre máquinas através da rede, o que abriria uma série de novas possibilidades em relação à tolerância a falhas e escalabilidade. Contudo, no decorrer do período, detectamos a necessidade de realizar mudanças no DALua, frente às requisições de usuários. A biblioteca foi então remodelada sob o paradigma de eventos, aumentando substancialmente sua flexibilidade e facilidade de uso em relação ao modelo anterior de callbacks.

Com o intuito de validar a nova forma de utilização da biblioteca, desenvolvemos um protótipo de jogo online para vários jogadores, com arquitetura peer-to-peer, fazendo uso de todas os recursos oferecidos pelo DALua.

Objetivos

Projetar e implementar um sistema de eventos para simplificar o seqüenciamento lógico de aplicações distribuídas baseadas no modelo de chamadas remotas assíncronas do sistema ALua [2].

Permitir a migração de processos DALua em execução entre máquinas separadas, tornando possível alterar fisicamente a rede sem interrupções no andamento da aplicação.

Planejamento

A biblioteca DALua é construída com a linguagem de programação Lua e o sistema ALua, ambos desenvolvidos na PUC-Rio. Sua versão incial foi utilizada na disciplina de pós-graduação sobre sistemas distribuídos INF2545 na PUC-Rio, onde os alunos tiveram a oportunidade de experimentar a programação de um pequeno jogo peer-to-peer com a biblioteca.

Os alunos forneceram críticas construtivas sobre as dificuldades de uso e possíveis melhorias. Detectamos então a necessidade de uma maneira mais flexível de saber quando as chamadas assíncronas se completam ou, mais genericamente, como tratar eventos de sucesso ou erro que ocorrem nos processos que compõem a aplicação.

Outra mudança diz respeito à forma com que grupos lógicos de processos são gerenciados. A partir da versão 5.0 do ALua, a responsabilidade por esses grupos, chamados de Aplicações, foi removida e, portanto, teve que ser transferida ao DALua. Assim, a biblioteca deve gerenciar a criação de aplicações e a entrada e saída de seus processos.

Também achamos pertinente melhorar a arquitetura da biblioteca, tornando-a mais modular de acordo com suas funcionalidades.

A Nova Arquitetura

O DALua é composto pelos seguintes módulos:

- *dalua*: módulo principal que contém o funcionamento básico da biblioteca.

- *dalua.app*: módulo que suporta grupos lógicos de processos (aplicações).
- *dalua.causal*: módulo que suporta troca de mensagens com ordenação causal.
- *dalua.events*: módulo que permite acessar o sistema de eventos.
- *dalua.mutex*: módulo que suporta exclusão mútua distribuída.
- *dalua.timer*: módulo que oferece um temporizador para tarefas periódicas.
- *dalua.total*: módulo que suporta troca de mensagens com ordenação total.

O Sistema de Eventos

Decorrida a fase inicial de pesquisa, optamos por um modelo de eventos com tratadores. Em contraste com o modelo anterior, em que cada função se ligava a uma única callback, agora é possível ter vários tratadores associados a um evento, ou um tratador para vários eventos.

A vantagem é que diferentes partes do código podem aguardar por um evento e reagir de maneiras distintas no momento em que ele for disparado. Também torna-se opcional a criação de funções callback específicas para cada chamada, já que o tratador de eventos tem uma interface única que recebe todos os argumentos necessários para identificar o tipo e origem do evento ocorrido.

Para receber eventos, é necessário que o processo registre seu interesse no evento desejado, indicando o(s) tratador(es) de evento para cada caso. Isso é feito com uma chamada ao método `dalua.events.monitor`, que recebe uma string que identifica o evento a ser monitorado e uma função Lua que será chamada para tratar o evento quando ele ocorrer. Para registrar mais de um tratador, basta chamar este método mais vezes, passando como argumento o mesmo identificador de evento, porém com diferentes funções tratadoras.

É possível também parar de receber eventos que estão sendo monitorados para um ou mais tratadores. Para tal, deve-se utilizar o método `dalua.events.ignore`, que também recebe a string identificadora do evento e o tratador que desejamos remover. Assim, os processos só recebem eventos quando for realmente necessário, evitando desperdício de processamento.

Adicionamos também a possibilidade de disparar eventos globais que atingem mais de um processo da aplicação. O DALua faz o multicast automaticamente para os processos que devem recebê-los, de acordo com o tipo do evento. O disparo de eventos é feito pelo método `dalua.events.raise`, que recebe o identificador do evento a ser disparado, o(s) processo(s) que receberão o evento e os argumentos a serem passados para o tratador.

Todos os métodos do DALua suportam nativamente o sistema de eventos. Um estilo padrão de evento foi adotado (todos os eventos DALua são prefixados com "dalua_" e os argumentos enviados ao tratador informam sua natureza, seja ela de erro ou o sucesso de uma operação). Desta maneira, a utilização da biblioteca foi drasticamente simplificada.

O Sistema de Aplicações

Uma aplicação é um grupo lógico de processos DALua, identificado por um nome único naquela rede ALua. Inicialmente, um processo deve requisitar a criação de uma nova aplicação, da qual ele fará parte. Para isso, ele deve chamar o método `dalua.app.create`, indicando qual nome identificará esta nova aplicação. Caso o nome já esteja em uso, um evento de erro é gerado.

A partir daí, outros processos poderão entrar nesta aplicação com uma chamada ao `dalua.app.join`. Analogamente, um processo pode sair da aplicação com `dalua.app.leave`. Eventos são disparados para os membros da aplicação sempre que um processo entra ou sai dela.

A grande vantagem desse sistema é que o processo de uma aplicação pode obter a lista de processos que fazem parte do grupo sempre que for necessário, através do método

`dalua.app.processes`, podendo ser usada diretamente em outros métodos do DALua como, por exemplo, o campo de destinatário para envio de uma mensagem. Essa lista é atualizada automaticamente, e condições de corrida são tratadas internamente pela biblioteca com o uso de exclusão mútua distribuída.

É importante notar também que um processo pode participar de diversas aplicações diferentes. Assim, é possível criar grupos com interseções, o que pode ser útil para alguns programas.

Para permitir que processos em diferentes máquinas façam parte de uma mesma aplicação, deve-se criar uma rede ALua entre todas as máquinas envolvidas. O DALua simplifica essa tarefa com o método `dalua.link`, que serve tanto para conectar uma máquina a outra quanto para conectar um grupo inteiro de máquinas a outro grupo. Internamente, o DALua cria uma rede totalmente conectada, o que torna desnecessário o uso de algoritmos complexos de roteamento, porém cria limitações de escalabilidade.

Exemplo

O exemplo a seguir demonstra o funcionamento do novo sistema de eventos em conjunto com o sistema de aplicações. Para efeito de demonstração, são utilizados apenas dois processos, rodando na máquina local. O primeiro processo cria uma nova aplicação e aguarda o evento "dalua_app_join", que indicará a entrada de outros processos nessa aplicação. O segundo processo, executado algum tempo depois, entra na aplicação criada pelo primeiro e envia uma mensagem para todos do grupo. Abaixo do código é exibida a saída do terminal.

Processo 1

```
require("dalua")

function onAppJoin(event, status, app, id)
     print("Processo "..id.." entrou na aplicação "..app)
end

function onAppCreate(event, status, app)
     if status == "success" then
          print("Aplicação '"..app.."' criada com sucesso!")
     else
          print("Erro ao criar aplicação!")
          return
     end
     dalua.events.monitor("dalua_app_join", onAppJoin)
end

dalua.init("127.0.0.1", 4321)
print("Eu sou o processo "..dalua.self())
dalua.events.monitor("dalua_app_create", onAppCreate)
dalua.app.create("MinhaApp")
dalua.loop()
```

```
> lua proc1.lua
Eu sou o processo 127.0.0.1:4321:0
Aplicação 'MinhaApp' criada com sucesso!
Processo 127.0.0.1:4321:1 entrou na aplicação MinhaApp
Olá para todos!
```

Processo 2

```
require("dalua")

function onAppJoin(event, status, app, id)
     if id == dalua.self() and status == "success" then
          dalua.send(dalua.app.processes("MinhaApp"),
"print", "Olá para todos!")
     end
end

dalua.init("127.0.0.1", 4321)
print("Eu sou o processo "..dalua.self())
dalua.events.monitor("dalua_app_join", onAppJoin)
dalua.app.join("MinhaApp")
dalua.loop()
```

```
> lua proc2.lua
Eu sou o processo 127.0.0.1:4321:1
Olá para todos!
```

Testes

Após a implementação das novas funcionalidades, o DALua foi novamente adotado pelos alunos da disciplina de pós-graduação INF2545. O trabalho proposto consistiu em desenvolver um protótipo de jogo online multijogador onde cada jogador corresponde a um processo DALua. Eventos do jogo, como a entrada ou saída de jogadores, são mapeados diretamente como eventos DALua, o que resultou num excelente teste para a biblioteca e provou que a nova arquitetura facilita bastante a criação desse tipo de aplicação.

Migração de Processos

Com a nova versão do DALua funcionando, decidimos voltar ao nosso objetivo inicial e pesquisar maneiras de transportar processos em andamento entre diferentes máquinas, mantendo o seu estado de execução. Assim, resolvemos utilizar a ferramenta Pluto [4] para serialização de processos Lua.

O Pluto é capaz de armazenar em formato texto todo o estado das variáveis de um processo Lua, e então recriar essas variáveis em outro processo a partir daquele texto. A pilha de execução não é mantida, mas para nossos objetivos iniciais basta o estado das variáveis.

Uma dificuldade de implementar isso nos processos DALua é que o estado da conexão não é mantido durante a migração, ou seja, é necessário reconectar os sockets para que a nova máquina faça parte do grupo. Desta forma, os sockets não são serializados, pois eles serão recriados pelo DALua após a migração.

Então, para efetuar a migração, um processo na máquina ‘A’ serializa-se para o formato de texto, que é enviado como uma mensagem à máquina de destino ‘B’. Lá, um processo inicialmente vazio é preenchido com os dados recebidos pela mensagem, e então uma resposta é enviada para que o processo original possa se desligar. A partir daí, o processo da máquina ‘B’ assume o lugar do anterior na aplicação.

Ainda existem problemas a serem resolvidos, como a atualização dos identificadores de processos DALua, que são baseados no endereço IP e porta de onde o processo foi criado. Porém, acreditamos que será possível contornar estes problemas, permitindo que nosso objetivo seja cumprido no futuro próximo.

Conclusões

A revisão da biblioteca DALua para utilizar o paradigma de eventos trouxe benefícios ao desenvolvimento de aplicações distribuídas, como pudemos comprovar durante a programação do protótipo de jogo.

A migração de processos torna possível a modificação da rede física sem interromper o andamento da aplicação, porém existem diversos problemas associados quando se trata de processos conectados em rede. Esperamos resolver esses problemas e cumprir o objetivo proposto enquanto buscamos pelas melhores soluções.

Referências

1 - R. COSTA. DALua. **PUC-Rio**, Rio de Janeiro, jul. 2007. Disponível em: http://alua.inf.puc-rio.br/dalua. Acesso em: 06 jul. 2007.

2 - C. URURAHY, N. RODRIGUEZ e R. IERUSALIMSCHY. ALua: Flexibility for parallel programming. **Computer Languages**, 28(2):155–180, 2002.

3 - R. IERUSALIMSCHY, L. FIGUEIREDO e W. CELES. Lua - an extensible extension language. **Software: Practice and Experience**, 26(6):635–652, 1996.

4 - B. SUNSHINE-HILL. Pluto Library. **Lua-users**, nov. 2005. Disponível em: http://lua-users.org/wiki/PlutoLibrary. Acesso em: 06 jul. 2007.

5 - R.COSTA, P. BASTOS, B. SILVESTRE e N. RODRIGUEZ. **Combining Programming Paradigms in Asynchronous Systems**. Rio de Janeiro, 2007. Monografia em Ciência da Computação - Departamento de Informática, PUC-Rio.