PROGRAMAÇÃO DISTRIBUÍDA ORIENTADA A EVENTOS

Aluno: Ricardo Gomes Leal Costa
Orientadora: Noemi Rodriguez

Introdução

Visando a facilitar a programação de aplicações geograficamente distribuídas, a biblioteca DALua foi desenvolvida, baseada na linguagem de programação Lua [2] e no sistema ALua [4]. O paradigma de orientação a eventos, com chamadas de função assíncronas, é utilizado por ser adequado às necessidades específicas desse tipo de aplicação.

O DALua oferece abstrações de programação para o gerenciamento da entrada e saída de processos na aplicação, envio de mensagens por multicast, tratamento de eventos de erro e facilidades para testes automatizados e depuração. Para testar a biblioteca, desenvolvemos módulos de suporte a algoritmos distribuídos, provendo exclusão mútua e envio de mensagens com ordenação causal e total.

Objetivos

Desenvolver uma biblioteca para facilitar a programação de aplicações distribuídas. Sua interface deve ser simples e coerente com os algoritmos distribuídos descritos na literatura. A tolerância a falhas é importante, devido aos problemas inerentes a sistemas geograficamente distribuídos. O desenvolvimento de testes automatizados também é essencial para garantir o correto funcionamento da biblioteca sob diversas condições de funcionamento.

Metodologia

O sistema ALua, desenvolvido na PUC-Rio, adiciona suporte à programação distribuída orientada a eventos em Lua. Uma aplicação ALua é composta por processos executando em várias máquinas interligadas em rede. A comunicação entre os processos é feita de maneira assíncrona através da primitiva *send* para envio de mensagens. Não existe uma primitiva de recebimento; a chegada da mensagem é tratada como um evento que implica na execução da mensagem, assumindo-se que esta é um trecho de código Lua.

A biblioteca DALua cria uma abstração sobre o modelo do ALua para gerenciar automaticamente a entrada e saída de processos na aplicação, o envio e recebimento de mensagens com suporte a *multicast* e o tratamento de erros. Dessa forma, torna-se mais fácil aplicar os algoritmos clássicos encontrados na literatura de sistemas distribuídos, evitando os detalhes de baixo nível da implementação. Há ainda uma série de métodos para a criação de testes automatizados, incluindo a simulação de atrasos e erros na troca de mensagens.

A entrada e saída de processos na aplicação, através dos métodos *dalua.enter* e *dalua.leave*, gera eventos DALua em todos os outros processos da aplicação. Os eventos são tratados automaticamente por padrão, mas o usuário tem a opção de definir seu próprio tratador de eventos em cada processo.

Para facilitar a criação de processos, o método *dalua.spawn* foi adicionado. São especificados o número de processos que se deseja criar na aplicação e o código-fonte a ser carregado em cada processo. Os processos são distribuídos inteligentemente pelo número de máquinas disponível na aplicação, balanceando a quantidade de processos em cada máquina.

O envio de mensagens é efetuado pelo método *dalua.send*. Uma mensagem é composta de um nome de função e os argumentos passados para essa função. Quando a mensagem é

recebida, a chamada da função é feita localmente. O destinatário da mensagem pode ser um único processo ou uma lista de processos (multicast).

O tratamento de erros do DALua funciona pelo mesmo sistema de eventos utilizado para notificar a entrada e saída de processos na aplicação. Cada evento tem um motivo associado, que indica se o evento ocorreu em condições normais ou se foi devido a um erro, como, por exemplo, um *timeout* no envio da mensagem. O tratamento padrão no caso de erro é imprimir na tela a mensagem de erro com sua descrição, mas o usuário pode definir um tratamento especial para tornar sua aplicação tolerante a falhas.

Para facilitar a criação de testes automatizados, desenvolvemos um sistema de agendamento de tarefas cronometrado. Com ele, é possível agendar chamadas aos métodos do DALua, especificando o número de vezes que devem ser chamados e o intervalo de tempo entre cada chamada. Este recurso permite simular várias condições de atraso no envio de mensagens para garantir o correto funcionamento da aplicação.

Com o intuito de testar o modelo do DALua, criamos três módulos com algoritmos para auxiliar o desenvolvimento de aplicações distribuídas. O de exclusão mútua garante que apenas um processo da aplicação tem acesso a um determinado recurso. Para implementá-lo foi utilizado o algoritmo de Ricart e Agrawala [3], que utiliza multicast e relógios lógicos para controlar a entrada e saída da região crítica.

Os outros dois módulos abordam o problema da entrega ordenada de mensagens [1]. Na ordenação causal, uma mensagem M2 que foi causada em resposta a uma mensagem M1 só é entregue após a mensagem M1. Na ordenação total, a relação causal das mensagens não é importante, desde que todos os processos recebam as mensagens na mesma ordem.

O DALua foi utilizado em 06.1 em disciplina de pós-graduação do Depto de Informática da PUC-Rio como ferramenta de aprendizado, permitindo experiência prática com algoritmos distribuídos e arquiteturas peer-2-peer.

Conclusões

A biblioteca DALua mostrou-se útil no desenvolvimento de aplicações geograficamente distribuídas. Com o modelo de orientação a eventos, o usuário pode facilmente tratar os erros para tornar sua aplicação tolerante a falhas. Os testes automatizados podem simular uma série de problemas na troca de mensagens que são comuns nesse tipo de aplicação.

O desenvolvimento do DALua deu a oportunidade de estudar vários modelos de sistemas distribuídos tolerantes a falhas para a definição do modelo adotado na nossa biblioteca. Permitiu também um melhor entendimento dos algoritmos distribuídos clássicos, que foram implementados e testados para verificar a funcionalidade do DALua.

Finalmente, o desenvolvimento da biblioteca nos permitiu um teste bastante exaustivo da própria funcionalidade do ALua, tendo acarretado na identificação e correção de alguns problemas no sistema.

Referências

1 - G. COULOURIS, J. DOLLIMORE e T. KINDBERG. **Distributed Systems: Concepts and Design**. Addison-Wesley, 2001.

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

3 - M. RAYNAL. **Distributed Algorithms and Protocols**. Wiley, 1988.

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