Sistemas Distribuídos

Biblioteca RPC

1. Ambiente de testes

Os testes foram realizados com cliente e servidor em uma única máquina:

- i5-2410M CPU @ 2.30GHz
- GNU/Linux (Arch), kernel 4.4.61-1-MANJARO
- lua-5.3.4
- luasocket-3.0rc1

2. Descrição

O trabalho consiste em uma biblioteca lua (luarpc), 2 clientes, 2 servidores e 2 interfaces. O primeiro grupo cliente-servidor-interface (sufixo example.lua) testa o exemplo mais simples de interface, apresentado no enunciado do trabalho. O segundo grupo (sufixo test.lua) testa a interface **inttests** proposta no final do enunciado.

O script mais importante entre os 7 apresentados é o client_test.lua. Esse arquivo executa os testes abaixo em ordem, e mostra o benchmark em segundos:

- Função *foo* 10000x
- Função *bar* 10000x
- Função *boo* 10000x, com uma string pequena na entrada
- Função *boo* 10000x, com uma string grande na entrada
- Função *serialize_table* 1x
- Função *boo* 10000x, com a tabela serializada no passo anterior

3. Resultados esperados e obtidos

Como a função *foo* lida com mais parâmetros de entrada/saída que a função *boo*, que por sua vez lida com mais parâmetros que a função *bar*, é de se esperar que o tempo de execução também siga nessa ordem. Por outro lado, não se pode ignorar que lua é uma linguagem dinâmica, e o tamanho (em bytes) dos parâmetros não é garantido de antemão, podendo esses também influenciar o tempo de execução da chamada remota.

Segue o output do script client_test.lua:

Execução de foo 10000x Duração: 0.57s Execução de bar 10000x Duração: 0.36s Execução de boo 10000x com uma string pequena Duração: 0.44s Execução de boo 10000x com uma string grande Duração: 0.66s	Serialização de table de 100 doubles Duração: 0.0001378s Execução de boo com tabela de 100 doubles, 10000x Duração: 0.57s

Observe que nos três primeiros testes, a ordem decrescente de execução foi foo > boo > bar, como já era esperado. Além disso, quando boo recebe uma string muito grande, ou uma tabela serializada, seu tempo de execução aumenta, ultrapassando foo, o que mostra que o tamanho (em bytes) dos parâmetros tem um peso maior sobre a chamada remota do que a quantidade de parâmetros.

4. Sobre a biblioteca

LuaRPC se apoia em algoritmos de marshall e unmarshall para serializar e desserializar as chamadas de função, e invocar corretamente os métodos remotos de acordo com a interface.

Como a comunicação TCP é feita através do envio de strings. Assim fica inviável obter um ponteiro para uma posição de memória remota, o que invalida qualquer possibilidade de manipulação direta dos dados remotos.

Entretanto, é possível enviar uma “descrição” da chamada remota, que pode ser entendida por outra máquina desde que usem a mesma interface/protocolo.

Baseado nesse princípio, foram criadas as funções server-side createServant e waitIncoming, e a função cliente createProxy. Todas elas usam funções auxiliares (presentes no mesmo arquivo) para fazer a serialização e desserialização das chamadas enviadas/recebidas.

O código fonte pode ser encontrado em http://github.com/mpolitzer/luarpc.

5. Referências

1) Ierusalimschy, Roberto. Programming in lua / Roberto Ierusalimschy - 2nd ed. Rio de Janeiro, 2006.

2) Reference. LuaSocket. Diego Nehab. 20/04/2006.
http://w3.impa.br/~diego/software/luasocket/reference.html