Sistemas Distribuídos - INF 2545 - 2014.1

Performance na execução cruzada de implementações RPC

Alunos:

Juarez Bochi <jbochi@gmail.com>

Marcelo Arza Lobo <marceloalc@gmail.com>

Rogério Carvalho Schneider <stockrt@gmail.com>

Professora:

Noemi Rodriguez <noemi@inf.puc-rio.br>

Objetivo

Comparar o comportamento de performance quando clientes e servidores da biblioteca *luarpc* são intercambiados entre diferentes implementações e em diferentes plataformas.

Plataformas

Foram selecionadas três plataformas para os testes, sendo compostas de três sistemas operacionais e três versões da biblioteca *luarpc*. Os sistemas operacionais Linux, Mac OS X e Windows foram utilizados.

Linux

SO versão: Ubuntu, Kernel 3.2.0-35-generic-pae, 32 bits.
CPU: Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz
RAM: 4 GB
Lua: 5.2.0
Luasocket: 3.0-rc1

Mac OS X

SO versão: Mac OS X Mavericks 10.9.1, Kernel 13.0.0, 64 bits.
CPU: Intel(R) Core(TM) i5 CPU @ 2.50GHz

RAM: 16 GB
Lua: 5.1.5
Luasocket: 3.0-rc1

Windows

SO versão: Windows 7 64 bits
CPU: Intel i5 de 3.2 GHz
RAM: 4 GB
Lua: 5.1.5
Luasocket: 2.0.2

Comparativos

Metodologia

O perfil de teste utilizado consistiu na execução de 5 rodadas consecutivas durante 5 segundos cada. As rodadas com o maior e com o menor número de chamadas foram descartadas. A soma das chamadas realizadas nas rodadas não descartadas serviu de base para o cálculo de requisições por segundo, que é o valor utilizado na tabela para comparativo entre as diferentes implementações. Para esclarecer, são realizadas 5 baterias, 3 delas são aproveitadas e como a duração é de 5 segundos cada, o total de chamadas destas 3 baterias é dividido por 15 segundos para se obter o valor de requisições por segundo.

O código do cliente utilizado no laço de testes foi o seguinte:

```
-- 1 byte de ida + nome do método = 4 bytes
-- 1 byte de volta + nil do result padrão = 4 bytes
-- Total de 8 bytes por chamada.
local calls = 0
local tini = os.time()
while os.difftime(os.time(), tini) < run_seconds do
  local result = proxy.boo(1)
  calls = calls + 1
end
print("boo calls: " .. calls .. " in ".. run_seconds .. " seconds for
server " .. server_port)
```

Todos os testes foram realizados em rede local, na qual uma mesma máquina serviu de cliente e de servidor de uma só vez.

Resultados

Nas tabelas abaixo são apresentados os comparativos da execução do cliente com laço de teste nas diferentes plataformas sugeridas. A execução tomou cuidado para testar o cliente *luarpc* contra o seu próprio servidor e contra os servidores das demais implementações, para se obter uma comparação mais abrangente.

- Linux, requisições por segundo: *close* | *keepalive*

Versão *luarpc*	juarez server	marcelo server	rogerio server
juarez client	NA | 18173	NA | 12854	NA | 13380
marcelo client	NA | 18328	NA | 15131	NA | 13282
rogerio client	6533 | 17094	6253 | 14436	4846 | 14663

* NA para os casos em que o cliente apresentou instabilidade após uma série de reconexões no perfil *close*, perfil no qual o servidor atende a uma requisição e fecha a conexão, obrigando o cliente a se reconectar para a próxima requisição.

- Mac, requisições por segundo: *close* | *keepalive*

Versão *luarpc*	juarez server	marcelo server	rogerio server
juarez client	2605 | 9606	1931 | 8497	1795 | 8179
marcelo client	149 | 9880	150 | 10182	148 | 8891
rogerio client	128 | 9751	132 | 7784	126 | 8503

Na tabela Linux, podemos observar que a maioria dos clientes parece não se comportar bem quanto ao processo de reconexão nesta plataforma. Apenas um cliente funciona corretamente no perfil *close* em Linux, porém o mesmo cliente é o de pior performance no mesmo perfil para a plataforma Mac. Ainda na tabela Linux, para o perfil *keepalive,* todas as implementações tem performance similar, merecendo destaque a implementação de servidor do Juarez combinada com o cliente do Marcelo, por ter sido o conjunto com mais requisições por segundo de todos os testes.

Na tabela Mac é interessante observar como o cliente do Juarez, para o perfil *close*, se destaca dos demais, tendo uma performance muito melhor. A suspeita para tal diferença vem do uso de

cache para métodos ao invés de *lookup* em *metatable* a cada chamada, o não uso da opção tcp-nodelay, e o fato de fazer apenas um *select* juntando servidores e clientes, o que resulta em uma *syscall* a menos do que o usado pelos outros clientes. No perfil *keepalive* os números são parecidos para todas as implementações. Interessante observar que o cliente com menor performance, nesta plataforma e para o perfil *close,* foi o único cliente estável para o mesmo perfil na tabela Linux.

A taxa de erro em todos os perfis foi zero para todas as execuções, exceto nas entradas com valor NA, nas quais a taxa de erro foi alta e os resultados tiveram que ser abandonados.

- Windows, tempo de uma requisição em ms: *close* | *keepalive*

Versão *luarpc*	juarez server	marcelo server	rogerio server
juarez client	NA | NA	3,228 | 1,301	NA | NA
marcelo client	2,968 | 1,476	3,047 | 1,221	2,871 | 1,315
rogerio client	NA | NA	2,297 | 1,386	NA | NA

* NA para os casos em que o perfil não foi executado na plataforma em questão.

Na tabela Windows a execução foi feita entre 2 desktops com a configuração descrita acima, utilizando uma rede Ethernet 100 Mbps. Ao contrário da abordagem do Linux e Mac, no Windows os valores da tabela representam o tempo em milissegundos de uma solicitação do método foo.