Alto desempenho utilizando RMI Assíncrono

Elton Mathias*, Marcelo Pasin

Universidade Federal de Santa Maria - Laboratório de Sistemas de Computação
Faixa de Camobi, Km 9 - CEP 97105-900 - Santa Maria - RS - Fone/Fax: (55) 220 8523
{emathias, pasin}@inf.ufsm.br

Palavras-chaves: RMI-Assíncrono, processamento paralelo e distribuído, paralelismo em Java e comparação de desempenho.

Introdução

Graças à sua simplicidade, segurança e independência de arquitetura a linguagem de programação Java emergiu como uma poderosa ferramenta para a programação em ambientes distribuídos. A fim de facilitar a programação de tais sistemas, a Sun Microsystens lançou o RMI[MIC 2002], uma interface de chamada remota de métodos com protocolos bem definidos, e de fácil utilização. Entretanto, embora RMI seja de uso bastante simples, em determinadas aplicações, ele compromete o desempenho, devido à sua natureza síncrona.

O presente artigo apresenta o RMI assíncrono como uma alternativa para a implementação de aplicações paralelas em Java. Apresentaremos também uma análise comparativa de desempenho para um aplicativo paralelo que busca números primos.

RMI Síncrono

Em 1998, no JDK 1.1 [1] a Sun Microsystems, com o intuito de criar uma interface padrão própria para chamada de método remoto, lança o pacote RMI. Este pacote permite a objetos Java a realização de processamento distribuído e comunicação entre os mesmos através da chamada de métodos remota.

Para a utilização de RMI [MIC 2002], objetos remotos são definidos através de interface remota. Esta interface é registrada em um servidor de nomes, e pode, a partir daí, ser exportada para clientes remotos através de uma classe *stub*, que mantém o controle sobre o protocolo de transferência. Da parte do cliente, basta que ele busque referência a uma determinada interface, junto ao servidor de nomes, para que possa, então, efetuar chamada de método remoto da mesma forma que faz com objetos locais.

Se por um lado o RMI apresenta uma abordagem bastante intuitiva e simples, por outro lado apresenta baixo desempenho em determinadas aplicações. Tal comportamento decorre do fato de haver pouca ou quase nenhuma otimização nos seus mecanismos de serialização e transferência e, principalmente, do seu caráter síncrono. Isto é, após fazer a invocação de um método remoto, o cliente esperará até que a execução, e retorno de resultados desse método seja concluída.

*Apoio CNPQ.

[1] Java Development Kit

Muito se tem pesquisado e produzido para obter solução para o problema do desempenho[MAA 2001]. Os principais esforços concentram-se na busca de implementações mais rápidas que utilizem premissas de serialização otimizadas, código compilado, bem como implementações que permitem execução assíncrona[FAL 99] . Este trabalho, ao contrário, propõe uma interface de invocação assíncrona.

RMI Assíncrono

O RMI assíncrono permite a utilização do tempo de espera do cliente para realizar processamento simultâneo com objetos remotos distribuídos. Com a utilização do tempo de espera em processamento pode-se almejar ganhos no desempenho de aplicações paralelas específicas.

O RMI assíncrono foi construído sobre o RMI original (síncrono). Para implementar a chamada a serviços remotos assíncronos foram criada duas classes, uma que define objetos remotos de invocação assíncrona e outra que define resultados de invocações. A primeira pode ser obtida a partir de qualquer objeto remoto acessível via RMI tradicional e cria um novo fluxo de execução para cada chamada assíncrona efetuada. A segunda cria um objeto a cada chamada e permite o programa verificar se uma dada invocação já terminou e, neste caso, receber o seu resultado.

Análise de Desempenho

A fim de efetuar comparações de desempenho entre o RMI síncrono, assíncrono, e execução sequencial, foi criado um aplicativo que busca todos os números primos até determinado valor. Na sua paralelização foi escolhido o modelo mestre-escravo, no qual o mestre chama remotamente o método que busca números primos para determinada faixa numérica. Esse aplicativo foi escolhido por ser trivialmente paralelizável e permitir, com facilidade, variação dos dados de entrada.

O ambiente utilizado foi um aglomerado de máquinas que possuem, cada uma, dois processadores de 1 GHz Pentium III, 786 Mbytes de memória principal, 20 GB de disco rígido, 512 kbytes de memória cache. As máquinas comunicam-se através de uma rede Gigabit Ethernet, com adaptadores modelo 3Com 3C996-T. O sistema operacional utilizado foi o Red Hat Linux 8.0 com kernel 2.4.22, e Java 2 versão 1.4.2.01.

O gráfico da figura 1 mostra os tempos de execução utilizando RMI Síncrono, assíncrono, e seqüencial [2] para busca de números primos até 10.000.000. Como podemos notar, o uso do RMI Síncrono não acelera em nada a execução do aplicativo, pois além de não explorar o paralelismo inter-nodal, sofre de um acréscimo de tempo em razão da localização e importação das interfaces que definem os objetos remotos e retorno de resultados. Já para o RMI assíncrono pode-se notar que, ao explorar o paralelismo, o aplicativo tem um ganho bastante expressivo,como era esperado.

Para o aplicativo considerado, podemos perceber que até 6 nós há um maior ganho de tempo ao acrescentarmos cada nós. Apartir daí, devido, principalmenteà latência

[2] Apesar do gráfico apresentar numero crescente de máquinas, a execução Sequencial ocorreu em apenas um nó, e foi inserida no gráfico para efeito de comparação.

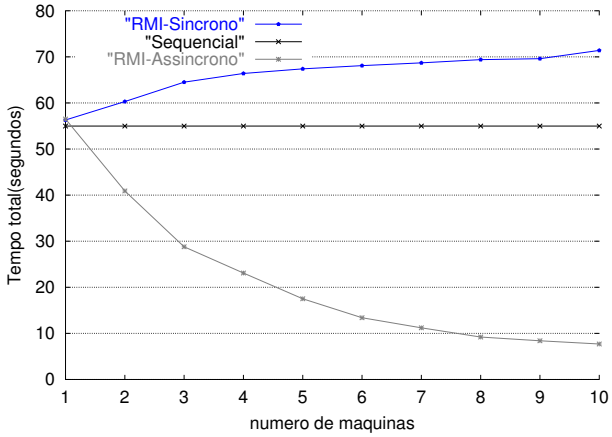

Figura 1: Comparação do tempo de execução do aplicativo.

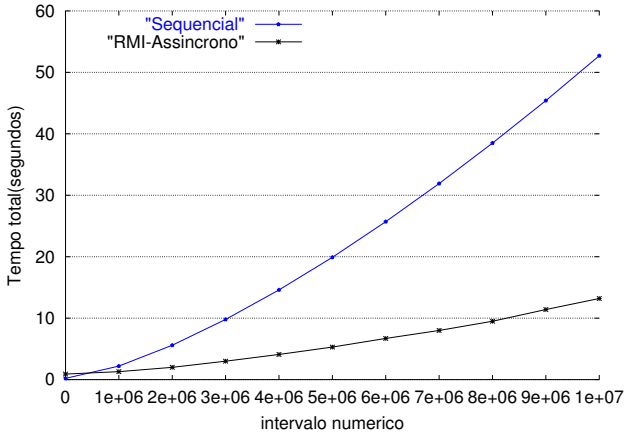

Figura 2: Comparação do tempo de execução do aplicativo para 6 nós

no retorno dos resultados dos nós escravos, o ganho relativo, ao acrescentarmos mais máquinas, diminui, ao ponto de tornar inútil o acréscimo de maismáquinas, o que parece ocorre apartir de 10 nós. Tal decréscimo no ganho relativo de tempo poderia ser adiado com uma melhor sincronização no retorno e captaçãodos resultados.

O gráfico da figura 2 mostra os tempos de execução seqüencial e utilizando RMI assíncrono, em 6 nós, para a busca de números primos com diferentes quantias de dados. Podemos notar que para tarefas relativamente pequenas, o RMI, mesmo assíncrono, não apresenta ganho. Isso deve-se ao fato de o RMI apresentar uma latência inicial correspondente ao tempo que necessita para encontrar e importar as interfaces remotas. Já para tarefas maiores, podemos notar que o RMI assíncrono passa a apresentar ganho signifi-

cativo, pois com tempo de procesamento maior, torna-se menos expressiva a latência, de forma a ser sobreposta por processamento.

A respeito da aplicação devemos também notar que a mesma exige uma pequena transmissão de dados entre os nós, o que favorece o ganho de tempo na paralelização. Testes preliminares, realizados com aplicativos que exigem uma maior transmissão de dados mostram que o RMI assíncrono perde parte de seu ganho, mas mesmo assim mostra-se como uma alternativa viável para a paralelização dos mesmos.

Conclusão e Trabalhos Futuros

O desenvolvimento desse trabalho objetivou uma comparação, em termos de desempenho, entre a versão original, síncrona, de RMI e uma implementação, construída sobre a original, que permite assincronismo na chamada de métodos remotos.

Dada a execução concorrente que obteve-se através do RMI assíncrono comprovou-se que apesar de uma dificuldade maior em termos de uso, houve ganho real e significativo no tempo de execução, principalmente para aplicações de maior porte, nas quais o tempo ganho com o processamento concorrente sobrepõe o tempo gasto com comunicação, importação e exportação de interfaces remotas.

Como trabalhos futuros pretende-se, primeiramente, alterar o funcionamento do RMI assíncrono, a fim de permitir ao usuário a abstração do controle de retorno das chamadas, tornando, dessa forma mais fácil e eficiente o controle das *Threads*. Outro trabalho possível é o teste de versão compilada do RMI assíncrono, a fim de verificar se a mesma é consideravelmente mais vantajosa.

Referências

[FAL 99] FALKNER, K. E. K.; CODDINGTON, P. D.; OUDSHOORN, M. J. **Implementing Asynchronous Remote Method Invocation in Java**. [S.l.]: Distributed High Performance Computing Group, Adelaide University, 1999. (DHPC-072).

[MAA 2001] MAASSEN; NIEUWPOORT van. Efficient java rmi for parallel programming. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, v.23, n.6, p.747–775, 2001.

[MIC 2002] MICROSYSTEMS, S. **Java remote method invocation specification(revision 1.8)**. [S.l.]: Sun, 2002.