Variações de Mensagens Ativas para Aglomerados de Computadores

Eduardo Roloff[1], Alexandre da Silva Carissimi[2],
Gerson Geraldo H. Cavalheiro[1]

[1]Universidade do Vale do Rio dos Sinos
Avenida Unisinos 950, São Leopoldo, RS, Brasil
eroloff@terra.com.br, gersonc@exatas.unisinos.br

[2]Universidade Federal do Rio Grande do Sul
Avenida Bento Gonçalves 9500, Porto Alegre, RS, Brasil
asc@inf.ufrgs.br

Introdução

A utilização de ambientes com memória distribuída (aglomerados/clusters) está se tornando uma alternativa atraente na resolução de problemas que necessitam tratar um volume muito grande de dados. Neste tipo de arquitetura, cada nó possui sua própria memória local e utiliza troca de mensagem para comunicação com outros nodos.

Um programa consiste em uma seqüência de tarefas manipulando dados, sendo que relações de dependência entre tarefas explicitam uma troca de dados entre elas. Considerando a execução em um aglomerado de computadores, um dado produzido por um processo em um nó poderá ser consumido por outro em um nó diferente. Como existe a necessidade de processamento de alto desempenho, deseja-se reduzir o tempo entre a produção de um dado e o seu consumo.

Mensagens ativas [EIC 92] são soluções clássicas para o problema de comunicação em ambientes paralelos [LUM 97], oferecendo um mecanismo de troca de mensagem com baixo sobrecusto [WAL 95].

O presente trabalho discute variações para implementação de esquemas de mensagens ativas em um ambiente de aglomerado de computadores.

Mensagens Ativas

Nos modelos de programação tradicionais para arquiteturas com memória distribuída, a troca de dados entre nodos é feita através de primitivas de *send* e *receive*, nesse modelo um processador envia uma mensagem através da chamada *send* e o receptor necessita estar aguardando a mensagem. Em modelos de programação síncronos, o processador que enviou a mensagem fica bloqueado até que o receptor execute. A grande vantagem desse modelo é a simplicidade de sua implementação [EIC 92].

Mensagens ativas é um mecanismo de comunicação assíncrono que tenta explorar toda a flexibilidade e desempenho das modernas redes de computadores [EIC 92]. Cada mensagem contém, em seu cabeçalho, o endereço de uma função a ser executada no receptor e, em seu corpo, os dados que compõe os argumentos [LUM 97]. O mecanismo funciona de forma que a recepção de um mensagem provoca a execução de uma função tendo a mensagem como argumento. Para este mecanismo funcionar é necessário que todos os nodos do aglomerado tenham a mesma imagem do código.

Mensagens ativas são diferentes de mecanismos de RPC, porque o tratamento de mensagens ativas não é para realizar processamento nos dados recebidos e sim para integrá-los no processamento em andamento com pouco volume de trabalho.

O funcionamento do mecanismo de mensagens ativas, pode ser de duas maneiras: por interrupções ou por *polling* (Figura 1).

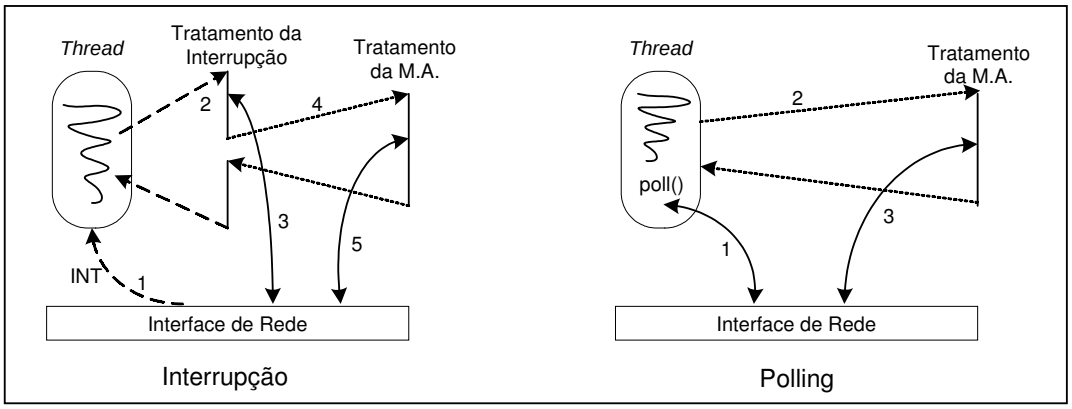

Figura 1: *Funcionamento esquemático do mecanismo de mensagens ativas*

Se estiver sendo utilizado o modelo de interrupções, seu funcionamento ocorre da seguinte maneira: A interface de rede interrompe a *thread* em execução avisando que uma mensagem chegou (1). Então é feito o tratamento dessa interrupção (2), em seguida se extrai o descritor dessa mensagem na interface de rede. O próximo passo é iniciar o tratamento dessa mensagem (4) e finalmente retirar os dados para execução da rede e armazená-los em uma área de dados.

Caso se utilize o modelo de *polling*, a *thread* que está executando necessita, periodicamente, consultar a interface de rede para verificar se chegaram novas mensagens. Caso nesse processo de *polling* tenha chegado uma mensagem, a *thread* retira o descritor dela da rede (1). Então é disparado o tratamento dessa mensagem ativa (2). O passo final é extrair da interface de rede os dados necessários para execução daquela tarefa.

Implementações Alternativas

Upcall

Nesta variação existe uma *thread* especialista que é responsável pelo tratamento de todas as mensagens que são recebidas. Esta *thread* é chamada de *deamon* de comunicação. Neste mecanismo, ao invés da *thread* que está executando parar seu processamento para executar a mensagem, o *deamon* que se encarrega dessa tarefa [CAR 99].

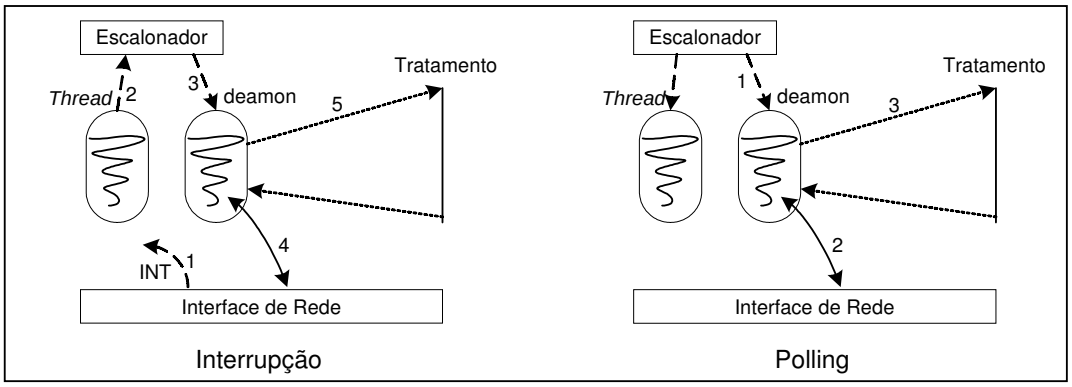

Figura 2: *Funcionamento esquemático do mecanismo de Upcall*

Se a política utilizada fora por interrupções, quando uma mensagem chega, é gerada uma interrupção (1), que provoca o acionamento do escalonador. A *thread* que está executando é preemptada

(2) e o *deamon* passa a executar (3). A mensagem é extraída totalmente da interface de rede (4) e é iniciada a sua execução (5).

Caso não se utilize interrupções, o *deamon* deve periodicamente verificar a chegada de mensagens. O *deamon* deve ganhar o direito de executar (1), para então consultar na interface de rede se existe alguma mensagem e caso exista, ele retira a mensagem da interface (2). A última ação é dar o devido tratamento a mensagem (3).

PopUp

Nesta variação é criada uma nova *thread* para tratar as mensagens recebidas. Isso é feito para que a *thread* que está executando não necessite ter o trabalho de retirar os dados da mensagem da interface de rede [CAR 99].

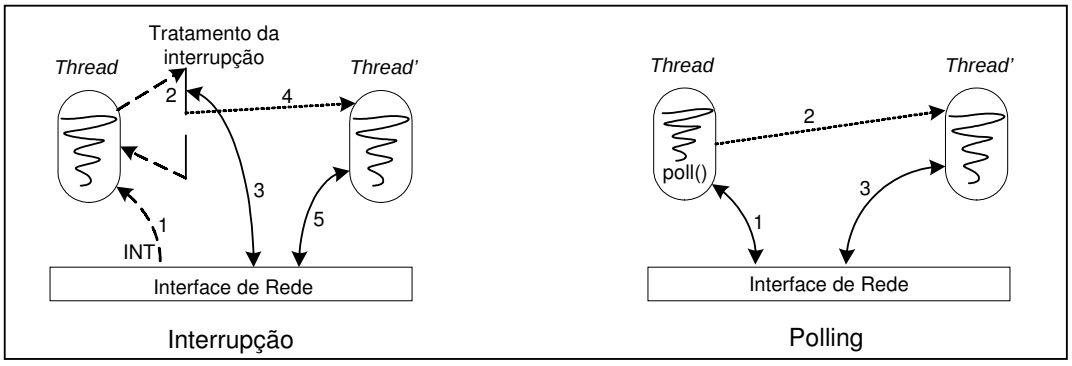

Figura 3: *Funcionamento esquemático do mecanismo de PopUp*

Como os outros mecanismos, este também pode funcionar por interrupções ou por *polling*. Caso esteja se utilizando a estratégia de interrupções, é gerada uma interrupção quando uma nova mensagem chega (1), então a *thread* atual inicia o tratamento dessa interrupção (2). O cabeçalho da mensagem é tirado da interface de rede (3) e é criada uma nova *thread* para trata-la (4). Essa nova *thread* é responsável por retirar da interface de rede os dados que vieram como parâmetros da mensagem (5).

Se a estratégia utilizada for por *polling*, a *thread* que está executando consulta periodicamente a interface de rede para verificar se existem novas mensagens (1), caso alguma mensagem tenha chegado, ela cria uma nova *thread* para tratamento (2), esta nova *thread* é responsável por retirar a mensagem da interface de rede (3) e dar o devido tratamento a ela.

Fila de Execução

Esta variação, que está sendo proposta neste trabalho, reúne parte de Upcall e de PopUp, pois com ela, é possível que se execute mais de um serviço ao mesmo tempo, sem a necessidade de criação de novas *threads* e também pode existir um *deamon* especialista para retirar as mensagens ativas da interface de rede.

Existe um número n de *threads* executando em casa nó. Estas *threads* tem por objetivo executar tarefas definidas pelo programa em execução. Estas tarefas são armazenadas em uma lista no momento de sua criação. Quando uma *thread* encerra a execução de uma tarefa, ela verifica se existe alguma tarefa na lista de tarefas. Caso exista ela executa essa tarefa, caso a fila esteja vazia, ela fica bloqueada esperando até que uma tarefa seja incluída na fila, para então executá-la. Quando uma mensagem é recebida, ela é armazenada nesta lista de tarefas. O mecanismo proposto possibilita a inserção de tarefas na lista de tarefas do nó i por um nó j qualquer.

Esta variação se enquadra no modelo proposto por Anahy [CAV 03], e está sendo desenvolvido para futura integração com ele.

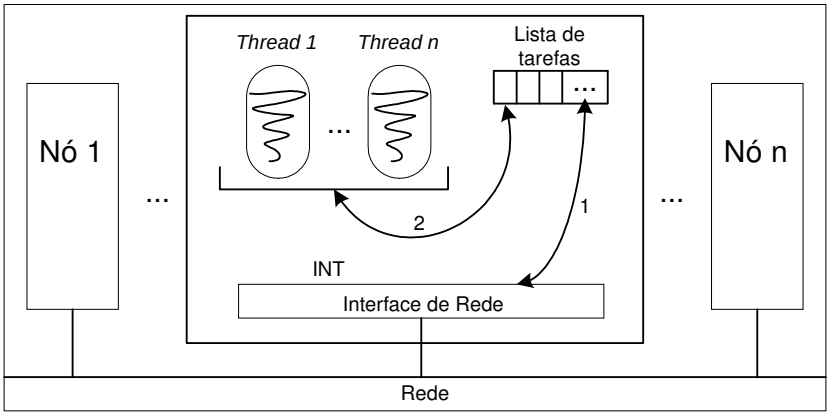

Figura 4: *Funcionamento esquemático da fila de execução*

Uma das vantagens dessa mescla proposta é a criação de um mecanismo simples de ser implementado, com um *deamon* executando *polling* na interface de rede. Outra é que ele permite a criação de um fluxo de execução próprio (no caso, uma *thread* Anahy).

Conclusões e Trabalhos Futuros

O trabalho desenvolvido apresenta-se como uma solução viável para a comunicação em ambientes de aglomerados de computadores.

O próximo passo deste trabalho é a implementação destas variações para que se possa aferir o desempenho de cada uma delas e validar o modelo de fila de execução, como sendo um modelo para comunicação em ambientes altamente paralelos.

Também será feita a integração dessa variação em Anahy.

Referências

[CAR 99] CARISSIMI, A. S. **Athapascan-0: Exploitation de la Multiprogrammation Légère Sur Grappes de Multiprocesseurs**. Tese de Doutorado. Grenoble: ID-IMAG. 1999.

[CAV 03] CAVALHEIRO, G. G. H.; REAL, L. C. V.; DALL'AGNOL, E. C. **Uma Biblioteca de Processos Leves para a Implementação de Aplicações Altamente Paralelas**, in Anais da WSCAD 2003, São Paulo, 2003.

[EIC 92] EICKEN, T. V. *et al*. **Active Messages: a Mechanism for Integrated Communication and Computation**, in Proceedings of the 19th International Symposium on Computer Architecture, Gold Coast, Australia, May 1992.

[LUM 97] LUMETTA, S. S.; MAINWARING, A. M.; CULLER, D. A. **Multi-Protocol Active Messages on a Cluster of SMP's**, in Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM), San Jose, CA, November 1997.

[WAL 95] WALLACH, D. A. *et al*. **Optimistic Active Messages: A Mechanism for Scheduling Communication with Computation**, in Proceedings of the fifth ACM SIGPLAN symposium on Principles & practice of parallel programming, Santa Barbara, CA, August 1995.