Análise da arquitetura VIA como protocolo de baixo nível na implementação da biblioteca de programação DECK

Leonardo Alves de Paula e Silva[1]
Philippe Olivier Alexandre Navaux[2]

Resumo

Com o surgimento de redes que oferecem baixa de latência comunicação e altas taxas de largura de banda, conhecidas como SANs (*System Area Networks*), os protocolos de comunicação integrados ao *kernel* do sistema operacional, como TCP/IP, tiveram de ser substituídos por outros protocolos que pudessem melhor explorar as potencialidades destas redes. Técnicas como cópia zero (*zero-copy*) e sobrepasso do sistema operacional (*operating system bypassing*) foram utilizadas na concepção destes novos protocolos, nomeados pela literatura por protocolos leves (*lightweight protocols*) ou protocolos em nível de usuário (*user-level protocols*). Estes protocolos permitem que a aplicação do usuário acesse o dispositivo de rede sem a necessidade de chamadas ao sistema operacional (*system calls*) e evitam cópias intermediárias do dado a ser comunicado. A utilização de protocolos de comunicação em nível de usuário com estas características é bastante interessante na implementação de abstrações de comunicação de mais alto nível, como o DECK. Através do estudo criterioso da arquitetura VIA, em especial de suas primitivas e de sua semântica, o objetivo deste trabalho é apresentar um estudo de como VIA pode servir como protocolo de comunicação de baixo nível para implementação da camada μDECK da biblioteca de programação DECK.

1 Introdução

O presente trabalho está inserido no Projeto MultiCluster (BARRETO; ÁVILA; NAVAUX, 2000), desenvolvido no GPPD do Instituto de Informática da UFRGS. O MultiCluster provê

[1] lapys@inf.ufrgs.br BolsistaCNPq
[2] Trabalho desenvolvido com apoio do CNPq, FINEP e Dell

uma infraestrutura de integração de *clusters* heterogêneos, onde os nós de cada *cluster* estão interconectados por diferentes tecnologias de rede.

O MultiCluster é capaz de proporcionar ao programador a ilusão de estar utilizando uma única máquina para executar suas aplicações paralelas. Esta transparência se dá pela definição de uma API única para todas as tecnologias de rede. Tal API deve oferecer primitivas que permitam a criação de múltiplos fluxos de execução em um mesmo processo (*threads*), a comunicação e a sincronização dos mesmos. Este conjunto de primitivas é provido pela API da biblioteca de comunicação DECK.

DECK é um ambiente de programação paralela, também desenvolvido no GPPD do Instituto de Informática da UFRGS, que proporciona o desenvolvimento de aplicações paralelas, pela filosofia SPMD, através da sobreposição de multiprogramação (múltiplas *threads*) com comunicação (BARRETO; NAVAUX; RIVIÈRE, 1998; BARRETO, 2000).

DECK permite a criação de *threads* e fornece primitivas de sincronização (semáforos, mutexes e barreiras). A comunicação entre as *threads* de DECK é realizada por intermédio de caixas postais (*mailbox*), pela invocação de primitivas *post* para envio de mensagens, e *retrieve* para recebimento. Cada *thread* cria sua própria caixa postal. *Threads* que desejem enviar mensagens para outra *thread* devem clonar (*clone*) a caixa postal da *thread* destino, ou seja, devem informar-se sobre o endereço da caixa postal da *thread* destino.

2 VIA

Virtual Interface Architecture, ou simplesmente VIA, é um padrão de protocolo de comunicação com todas as características e objetivos de um protocolo de comunicação em nível de usuário. Protocolos de comunicação em nível de usuário proporcionam baixas latências e aumento da largura de banda, sendo bastante interessantes na implementação de abstrações de comunicação de mais alto nível (DUNNING, 1998; BUONADONNA, 1999; BEGEL, 2002). VIA oferece a abstração de **Interface Virtual** (*Virtual Interface*, VI) que dispensa a necessidade de chamadas de sistema para realizar operações de comunicação. O dispositivo de rede pode, assim, ser acessado de forma protegida e diretamente pelo processo usuário através de uma interface virtual. Cada VI representa um ponto de comunicação. Um processo usuário pode ter múltiplas VIs relacionados a um ou mais dispositivos de rede (COMPAQ; INTEL; MICROSOFT, 1997). A figura 1 apresenta o modelo VIA.

Para que a comunicação possa ocorrer, uma conexão entre duas VIs deve ser estabelecida. O dado a ser comunicado deve ser encapsulado em uma estrutura VIA denominada descritor, o qual entre outras funções, aponta para o endereço da região de memória cadastrada onde o dado está localizado, no caso de um descritor de envio, ou onde o dado deve ser recebido, no caso de um descritor de recebimento. Cada VI possui duas filas, uma para envio e outra para recebimento de dados. O remetente deve postar um descritor na fila de envio e o receptor deve postar um descritor na fila de recebimento. Assim que ocorre a transmissão do dado, o

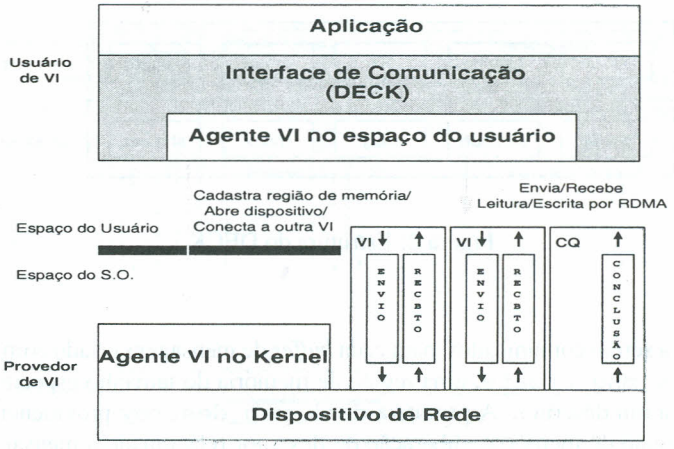


Figura 1: Modelo da arquitetura VIA (COMPAQ; INTEL; MICROSOFT, 1997).

descritor é consumido, ou seja, o estado do descritor é modificado para *done* e o descritor é retirado da fila correspondente. Este processo é denominado de processamento de descritores e existe dois modelos: fila de conclusão e fila de trabalho. O modelo de fila de trabalho, uma *thread* realiza *polling* ou bloqueio na fila de trabalho até que o descritor do início da fila passe para *done*. O modelo de fila de conclusão realiza o mesmo processo, entretanto em múltiplas filas de trabalho. Cada modelo deve ser utilizado dependendo da necessidade da aplicação.

3 DECK/VIA

DECK está dividido em duas camadas distintas como mostrado na figura 2. A camada μDECK é diretamente dependente da tecnologia de rede, sendo os módulos de inicialização, de mensagem (msg) de caixa postal (mbox), os quais serão implementados utilizando as primitivas de VIA.

Para o módulo de inicialização, que engloba as primitivas de preparação dos nós para execução, a primitiva `deck_init` deve ser implementada para executar as seguintes operações de preparação do processo para posterior comunicação com seus pares remotos: 1. ajuste dos atributos do dispositivo de rede; 2. abertura do dispositivo de rede (`VipOpenNic`); 3. criação de uma fila de conclusão para RQ e SQ (`VipCreateCQ`). No momento do encerramento da execução do `deck_done` são executadas as seguintes operações: 1. fechamento das conexões abertas (`VipDisconnect`); 2. destruição da fila de conclusão (`VipDestroyCQ`) 3. fechamento do dispositivo de rede (`VipCloseNic`).

No módulo de mensagens, que lida com as primitivas relacionadas a preparação e aloca-

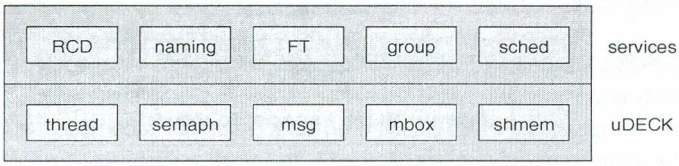


Figura 2: Estrutura do DECK.

ção dos dados a serem comunicados, para cada *buffer* de mensagens criado com a chamada da primitiva `deck_msg_create`, uma região de memória do tamanho especificado é criada e relacionada a um descritor. A primitiva `deck_msg_destroy` providencia o descadastramento da região de memória e liberação do descritor relacionado à mensagem destruída. Valendo-se da possibilidade de um descritor poder ser formado por mais de um segmento de dado, para cada chamada da primitiva `deck_msg_pack`, um novo segmento de dado será adicionado naquela mensagem.

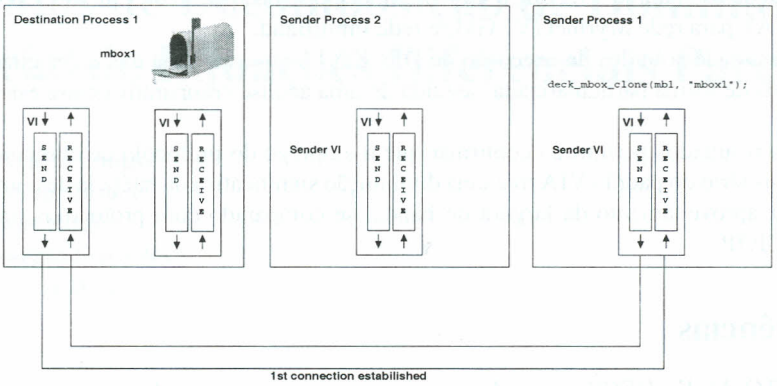
O módulo de caixa postal lida com primitivas relacionada ao canal de comunicação entre os processos e threads DECK, através da abstração de caixas postais. Ao criar uma caixa postal, pela invocação de `deck_mbox_create`, uma interface virtual é criada e colocada em estado de espera de conexão, ou seja, no aguardo de uma clonagem, segundo a semântica de DECK. A cada recebimento de uma requisição de nova clonagem, pela chamada da primitiva `deck_mbox_clone`, uma nova VI é criada e colocada em estado de espera por uma conexão.

No momento em que um processo remetente requisita a clonagem da caixa postal hospedada por um outro processo, uma VI é criada no processo remetente, a qual se ocupará exclusivamente da postagem de mensagens para caixa postal clonada. Neste instante, a VI criada requisita uma conexão a sua VI correspondente no processo destinatário, a conexão é estabelecida e o processo rementente está apto para enviar mensagens para a caixa postal clonada. A figura 3 apresenta graficamente a proposta de implementação de caixas postais.

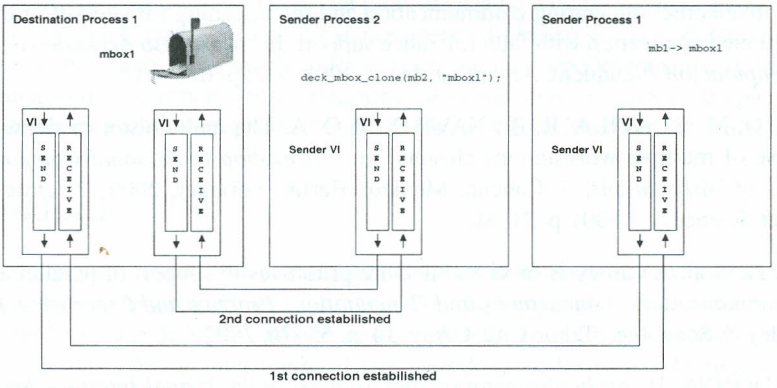
Para o processamento dos descritores, parece mais adequado o uso do modelo de filas de conclusão, onde em cada processo seria criada uma fila de conclusão e todas as interfaces virtuais estariam relegadas a seu controle.

A primitiva `deck_mbox_post` posta o descritor de envio na fila de envio da VI do processo remetente. De forma análoga, a primitiva `deck_mbox_retrv`, posta o descritor de recebimento na fila de recebimento da VI do processo destinatário.

A primitiva `deck_mbox_destroy` providencia a desconexão e a destruição das VIs utilizadas na comunicação entre os processos.



(a) Clonagem da caixa postal e conexão estabelecida com processo remetente 1.



(b) Clonagem da caixa postal e conexão estabelecida com processo remetente 2.

Figura 3: Proposta de implementação de caixas postais.

4 Resultados Esperados

DECK/VIA encontra-se em fase inicial de implementação, não estando disponíveis resultados de sua execução para serem apresentados.

Entretanto, a implementação de DECK sobre VIA poderá ser executada em diversas im-

plementações do padrão VIA que o GPPD tem disponíveis: para rede Ethernet (MVIA, Berkeley VIA), para rede Myrinet (VI-GM) e rede InfiniBand.

Os resultados obtidos da execução de DECK/VIA em cada uma das redes citadas serão analisados de forma particularizada, seguida de uma análise comparativa entre estes resultados.

Estes resultados permitirão confirmar que o emprego do protocolo de comunicação em nível de usuário do padrão VIA traz uma diminuição significativa da latência de comunicação e melhor aproveitamento da largura de banda, se comparado com protocolos tradicionais como TCP/IP.

Referências

BARRETO, M. E. *DECK: um ambiente para programação paralela em agregados de multiprocessadores*. Dissertação (Mestrado) — PPGC/UFRGS, 2000.

BARRETO, M. E.; NAVAUX, P. O. A.; RIVIÈRE, M. P. Deck: a new model for a distributed executive kernel integrating communication and multiheading for support of distributed object oriented application with fault tolerance support. In: *Congreso Argentino de Ciencias de la Computacion*. Neuquém, Argentina: [s.n.], 1998. v. 2, p. 623–637.

BARRETO, M. E.; ÁVILA, R. B.; NAVAUX, P. O. A. The multicluster model to the integrated use of multiple workstations clusters. In: *Workshop of Personal Computer Based Networks of Workstations, 3*. Cancun. México: Berlin, Springer, 2000. (Lecture Notes in Computer Science, v. 1800), p. 71–80.

BEGEL, A. et al. An analysis of vi architecture primitives in support of parallel and distributed communication. *Concurrency and Computation: Practice and Experience*, Hoboken, John Wiley & Sons, Inc., Tahoe City. CA, v. 14, p. 55–76, 2002.

BUONADONNA, P. *An Implementation and Analysis of the Virtual Interface Architecture*. Dissertação (Mestrado) — University of California, Berkeley, 1999.

COMPAQ; INTEL; MICROSOFT. *Virtual Interface Architecture Specification Version 1.0*. 1997. Dezembro 1997. Disponível em `http://developer.intel.com/design/servers/vi/developer/ia_imp_guide.htm`.

DUNNING, D. et al. The virtual interface architecture. *IEEE Micro*, Los Alamitos. CA, v. 18, n. 2, p. 66–76, March/April 1998.