Um Framework Meta-Programado Para a Implementação de Protocolos Leves de Comunicação*

Thiago Robert C. Santos, Lucas Francisco Wanner,
Augusto Born de Oliveira and Antônio Augusto Fröhlich
Laboratory for Software and Hardware Integration
Federal University of Santa Catarina
PO Box 476 – 88049-900 – Florianópolis, SC, Brazil
{robert, lucas, augusto, guto}@lisha.ufsc.br

Abstract

This article describes a communication system comprised by a meta-programmed framework, responsible for providing mechanisms to select, configure and combine communication protocols according to application requirements, and a basic communication kernel over which the protocols are projected. The basic premise for this communication system is that it is possible to maintain modularity for lightweight protocols, enhancing reusability and, at the same time, supporting highly efficient implementation techniques, using an explicit composition mechanisms instead of layer-based encapsulation.

1 Introdução

Novas tecnologias em infra-estrutura computacional, tais como ambientes de computação de alto desempenho e redes de sensores sem fio, vêm impulsionando a pesquisa nos mais diversos campos da ciência. De fato, uma das mais recentes contribuições da ciência da computação para pesquisadores de outros campos é a *ciência computacional*, que apoiada em ambientes de computação de alto desempenho, busca entender a natureza através do uso e a análise de modelos matemáticos. O cientista computacional, que se alia aos tradicionais teoristas e experimentalistas, utiliza a infra-estrutura computacional disponível como uma ferramenta para construir e processar modelos matemáticos que simulam o comportamento de sistemas complexos, variando de galáxias a moléculas. A ciência computacional é responsável por uma verdadeira revolução na maneira como a pesquisa científica é conduzida em diversas áreas, tais como bioengenharia, nanotecnologia e meteorologia.

No mesmo ritmo em que novas tecnologias em infra-estrutura computacional promovem mudanças na maneira como a pesquisa científica é conduzida, elas também promovem mudanças na maneira como são projetados os ambientes de software que fazem o intermédio entre as aplicações e o hardware. Cientistas da computação têm presenciado uma verdadeira revolução na área de software básico, desencadeada pela constatação de que os requisitos das aplicações e ambientes de hardware modernos invalidam algumas das premissas que guiaram o desenvolvimento de sistemas operacionais amplamente difundidos. Os sistemas de comunicação foram os primeiros a sofrer mudanças devido aos requisitos da nova geração de aplicações distribuídas e as recentes inovações tecnológicas implementadas pelos fabricantes de redes, que moveram o gargalo da performance de comunicação do ambiente de hardware para o ambiente de software. Sistemas de comunicação tradicionais, como por exemplo o Internet Sockets/TCP, colocam todo o processamento relacionado aos serviços de comunicação no núcleo do sistema operacional. Como resultado, o caminho crítico durante o envio e recebimento de mensagens inclui operações caras, tais como trocas de contexto, desencadeadas por chamadas de sistemas, e tratamento de interrupções, além de um excessivo número de cópias. De maneira a prover performance de comunicação próxima ao limite imposto pelo hardware, cientistas da computação tiveram que tirar o sistema operacional do caminho através dos sistemas de comunicação em nível de usuário (ULC), permitindo que protocolos leves acessem diretamente as funcionalidades disponibilizadas pelo hardware de comunicação de maneira otimizada [10, 12, 17].

Sistemas de comunicação modernos têm abandonado a clássica arquitetura em camadas e de natureza monolítica que guiou o desenvolvimento de protocolos bastante difundidos como o Protocolo Internet (TCP/IP) em favor de

*Este trabalho foi parcialmente financiado pelo projeto FINEP (Financiadora de Estudos e Projetos) no. 01.04.0903.00.

arquiteturas mais flexíveis que empregam protocolos leves. Entre as desvantagens associadas aos protocolos de comunicação em camadas podemos destacar as diversas cópias em memória dos dados sendo transmitidos como forma de comunicação entre as camadas do protocolo, tanto no envio quanto no recebimento de mensagens. O principal problema com arquiteturas de comunicação monolíticas é que não existe um único protocolo que seja ótimo para todas as aplicações, pois cada aplicação tem um conjunto específico de requisitos relacionados a comunicação. Ao invés de projetar um único protocolo que atenda as necessidades de uma ampla gama de aplicações, parece ser mais factível desenvolver um componente que permita a seleção, configuração e combinação de protocolos leves previamente implementados. Esse paradigma oferece diversas vantagens, incluindo a habilidade de criar novos serviços de comunicação sob demanda e permitir que aplicações experimentem com diferentes configurações de protocolo de comunicação, coletando métricas para identificar a melhor configuração para as suas necessidades.

Esse artigo discorre a respeito de um sistema de comunicação constituído por um framework meta-programado, responsável por prover mecanismos que permitem selecionar, configurar e combinar protocolos de comunicação de acordo com os requisitos da aplicação, e um núcleo básico de comunicação sobre o qual os protocolos são projetados. A premissa básica desse sistema de comunicação é que seja possível manter a modularidade dos protocolos leves, aumentando a reusabilidade, e mesmo assim suportar técnicas de implementação de alto desempenho utilizando um mecanismo de composição explícito no lugar do encapsulamento em camadas. O sistema de comunicação descrito a seguir foi desenvolvido dentro do contexto do projeto SNOW [4], que tem por objetivo criar um ambiente de software orientado a aplicação para suportar computação de alto desempenho sobre agregados de PCs dedicados de forma eficiente.

As demais seções desse artigo estão divididas da seguinte maneira: a seção 2 descreve o ambiente de programação paralela SNOW e apresenta algumas das características da área para a qual SNOW foi projetado; a seção 3 descreve em detalhes a arquitetura do sistema de comunicação SNOW; a seção 4 discute as decisões de projeto referentes a implementação de três protocolos leves sobre o sistema de comunicação: controle de fluxo, entrega segura e fragmentação; a seção 5 apresenta diversos trabalhos correlatos; a seção 6 conclui esse artigo e discorre a respeito de trabalhos futuros.

2 O Projeto SNOW

Como mencionado anteriormente, novas tecnologias em infra-estrutura computacional vêm promovendo mudanças na maneira como são projetados os ambientes de software que fazem o intermédio entre as aplicações e o hardware. A computação sobre agregados de PCs, a classe de sistemas computacionais de alta performance que mais cresceu nos últimos anos [14], é um dos domínios onde esse problema teve um impacto considerável. Agregados de PCs tem se destacado bastante na área de computação de alto desempenho, sendo considerados a melhor opção em infra-estrutura computacional quando se leva em consideração a relação custo/desempenho. Agregados são sistemas de computação paralelos compostos de uma coleção de nodos de processamento interligados por uma rede de comunicação de alta performance, onde cada um dos nodos é um sistema completo, capaz de operar independentemente dos outros. Agregados podem ser montados utilizando-se nodos de processamentos e redes de comunicação disponíveis comercialmente, o que torna o sistema computacional mais acessível e permite que as inovações tecnológicas implementadas pelos fabricantes de processadores e redes de comunicação sejam rapidamente incorporadas. Os primeiros agregados de PCs, que surgiram a cerca de dez anos, utilizavam esse mesmo conceito de empregar componentes pré-fabricados também para o ambiente de software. Essa classe de agregados recebeu a denominação de Beowulf e pionerizou a utilização do sistema operacional de propósito geral Linux nos nodos de processamento dos agregados. Ainda hoje, agregados de PCs Linux desempenham um importante papel na área de computação de alto desempenho e no setor comercial, sendo considerados o sistema computacional paralelo mais amplamente difundido.

O problema com a estratégia de utilizar software de propósito geral em computação sobre agregados é que os sistemas operacionais de propósito geral, tais como o Unix, Linux e o Windows, foram projetados sem levar em consideração os requisitos específicos das aplicações e do ambiente de hardware paralelos, o que afeta o desempenho, a usabilidade e a escalabilidade do sistema computacional. O grande número de projetos de pesquisa na área de computação paralela sobre agregados de PCs que propõe camadas *middleware* sobre o sistema operacional, destacando-se entre eles as implementações de soluções para troca de mensagens (message passing) [3, 7] e para simulação de ambientes de memória compartilhada (shared memory) [1, 8], serve como fundamento à suposição de que ambientes de software comuns não são adequados para suportar computação de alto desempenho sobre agregados. Se os sistemas operacionais de propósito geral utilizados pela maioria dos agregados em funcionamento fossem capazes de atender as necessidades das aplicações paralelas, ou ao menos disponibilizassem arquiteturas extensíveis que possibilitassem a adição dos serviços requeridos por essas aplicações de forma eficiente, muitos desses projetos

de pesquisa não seriam necessários.

Além disso, os sistemas de comunicação disponibilizados por sistemas operacionais de propósito geral não foram projetados para funcionar eficientemente em ambientes paralelos e sua performance é degradada por trocas de contexto, desencadeadas por chamadas de sistemas, e o excessivo número de cópias durante o envio e recebimento de mensagens. Devido ao fato de que a performance de um agregado de PCs depende de mecanismos eficientes de comunicação entre seus nodos de processamento, cientistas da computação tiveram que tirar o sistema operacional do caminho com o objetivo de aumentar a performance da comunicação em agregados que utilizam ambientes de software de propósito geral. Os sistemas de comunicação em nível de usuário são amplamente empregados na área de computação de alto desempenho sobre agregados de PCs o que pode ser considerado a mais significativa evidência de que a utilização de software de propósito geral não é tão efetiva quanto a utilização de hardware de propósito geral nessa área.

Apesar da maioria dos grupos que realizam computação sobre agregados ainda insistirem em utilizar software de propósito geral, desenvolvido sem levar em consideração os requisitos específicos das aplicações e do ambiente de hardware paralelos, existem diversos projetos de pesquisa que tem por objetivo prover ambientes de software específicos para agregados de PCs [6, 11]. O SNOW, um desses projetos de pesquisa, propõe um ambiente de software para suportar computação de alto desempenho sobre agregados de PCs dedicados onde um sistema de suporte em tempo de execução minimalista, contendo apenas as abstrações e serviços necessários para atender aos requisitos da aplicação paralela em questão, é gerado para cada aplicação. O ambiente de software gerado é acessado através de interfaces padrão como a MPI e o POSIX, permitindo que qualquer aplicação projetada de acordo com esses padrões possa ser suportada.

3 O sistema de comunicação SNOW

A utilização de ambientes de programação paralela geralmente implica em um aumento considerável no tráfego de mensagens do sistema computacional. Além dos dados a serem processados e dos resultados referentes ao processamento desses dados, o ambiente de programação paralela necessita que mensagens de controle sejam trocadas com os nodos de processamento com o objetivo de avaliar fatores como a disponibilidade de recursos de hardware e o andamento das tarefas de processamento. Como conseqüência, a performance de um ambiente de programação paralela está intimamente relacionada à eficiência dos mecanismos de comunicação empregados pelo sistema computacional, especialmente no que se refere aos ambientes de programação para agregados de PCs.

Protocolos de comunicação, devido a sua complexidade, geralmente são organizados em uma hierarquia em camadas. Nesse tipo de arquitetura cada camada oferece um conjunto de serviços à camada superior e utiliza os serviços disponibilizados pelos protocolos das camadas inferiores. Entre as desvantagens associadas aos protocolos de comunicação em camadas podemos destacar as diversas cópias em memória dos dados sendo transmitidos como forma de comunicação entre as camadas do protocolo, tanto no envio quanto no recebimento de mensagens. Cópias desnecessárias de dados são consideradas uma das principais causas relacionadas a degradação da performance de sistemas de comunicação tradicionais e são evitadas a todo custo por sistemas de comunicação otimizados como os sistemas ULC.

Outro fator que tem um impacto considerável na performance de sistemas de comunicação tradicionais é a sua natureza monolítica. Com o objetivo de esconder a complexidade das diferentes redes físicas, sistemas de comunicação tradicionais geralmente provêm aos seus usuários um conjunto fixo de serviços e uma interface fixa de acesso a esses serviços. Embora uma interface fixa de acesso aos serviços de um sistema de comunicação seja necessária e até mesmo benéfica, diferentes aplicações necessitam de diferentes combinações de serviços de comunicação. Para algumas aplicações, o conjunto de serviços providos por um determinado sistema de comunicação pode ser insuficiente, problema comum que geralmente é endereçado através da utilização de camadas *middleware* que disponibilizam os serviços necessários mas degradam a performance de comunicação.

Além de implementações otimizadas, dois dos principais requisitos referentes ao projeto de sistemas de comunicação modernos são a extensibilidade e a configurabilidade. Sistemas de comunicação extensíveis permitem que novos serviços requisitados por uma determinada aplicação sejam implementados de maneira eficiente diretamente dentro do sistema de comunicação, dispensando a utilização de camadas *middleware*. A configurabilidade pode melhorar a performance da comunicação permitindo que serviços não utilizados sejam desligados e outros aspectos relacionados a comunicação, tais como unidade máxima de transmissão (MTU), tamanho dos buffers utilizados durante o envio e recebimento de mensagens e até mesmo algumas características do algoritmo de comunicação, sejam customizáveis.

Ao invés de prover um único protocolo monolítico o sistema de comunicação SNOW disponibiliza um conjunto de protocolos leves de comunicação que podem ser utilizados, individualmente ou em conjunto, para atender aos

requisitos específicos das aplicações. Além de manter a modularidade dos protocolos leves e suportar técnicas de implementação de alto desempenho utilizando um mecanismo de composição explícito no lugar do encapsulamento em camadas, essa abordagem oferece diversas outras vantagens, incluindo a habilidade de criar novos serviços de comunicação sob demanda e permitir que aplicações experimentem com diferentes configurações de protocolo, coletando métricas para identificar a melhor configuração para as suas necessidades. O sistema de comunicação SNOW é constituído por um framework meta-programado, responsável por prover mecanismos para selecionar, configurar e combinar diferentes *estratégias de comunicação*, e o núcleo básico de comunicação, uma abstração que implementa um algoritmo de comunicação minimalista com *pontos de ação* pré-definidos, onde as estratégias de comunicação podem agir para modificar características da comunicação.

A interação entre as estratégias de comunicação ativas e o núcleo básico de comunicação é intermediada pelo framework meta-programado e diferentes estratégias de comunicação podem ser facilmente criadas estendendo esse framework. O núcleo básico de comunicação consiste em uma implementação minimalista de um algoritmo de comunicação para uma determinada tecnologia de rede física. Só o que é estreitamente necessário para efetuar troca de mensagens entre dois nodos da rede é implementado no núcleo básico e até mesmo serviços como fragmentação devem ser projetados como estratégias de comunicação para que aplicações que não necessitam desses serviços possam desligá-los, aumentando a performance do sistema de comunicação. Neste contexto, estratégias de comunicação são componentes auto-contidos que implementam desde pequenas modificações no algoritmo básico de comunicação até protocolos leves completos. Estratégias de comunicação implementam o padrão de projeto *flyweight* [5] e podem ser projetadas independentemente da tecnologia de rede física sendo utilizada.

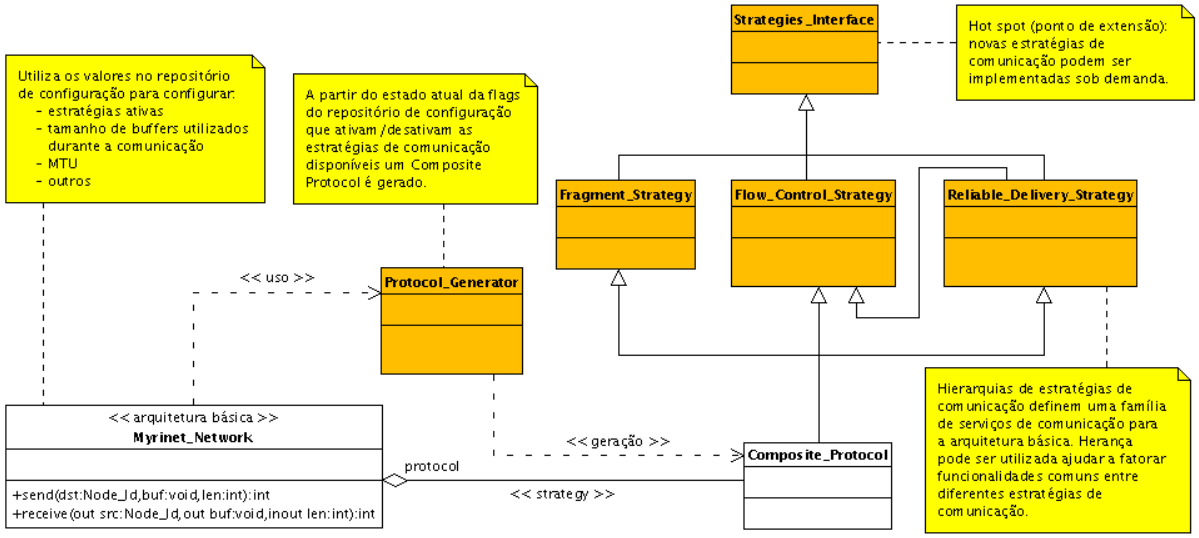

Figura 1: Diagrama de classes referente ao sistema de comunicação SNOW.

3.1 Arquitetura do sistema de comunicação

Basicamente, o sistema de comunicação SNOW pode ser dividido em duas partes: estática e dinâmica. A parte estática, meta-programada e resolvida em tempo de compilação, inclui o framework e um repositório de configuração. A parte dinâmica inclui o núcleo básico de comunicação e as estratégias de comunicação ativas, agrupadas em um componente que implementa uma variação do padrão de projeto *composite* e consiste no protocolo a ser utilizado dinamicamente. A figura 1 apresenta o diagrama de classes referente ao sistema de comunicação SNOW. As classes em cor mais escura constituem o framework meta-programado e são resolvidas em tempo de compilação. As classes em cor mais clara compõem a parte dinâmica do sistema.

O núcleo básico de comunicação e o protocolo composto implementam uma variação do padrão de projeto *strategy*. Em tempo de execução, o núcleo básico faz chamadas para métodos específicos do protocolo composto em cada um dos pontos de ação definidos por ela. O protocolo composto é gerado em tempo de compilação pelo framework meta-programado a partir das informações contidas no repositório de configuração e as diferentes composições de protocolo são utilizadas para customizar a comunicação de acordo com os requisitos das aplicações. Pelo fato de o sistema de

comunicação ter sido projetado para suportar aplicações em ambientes dedicados e para evitar o overhead associado à implementação tradicional do padrão *composite*, o protocolo composto não pode ser reconfigurado dinamicamente.

3.2 Framework meta-programado

Metaprogramação é a tecnologia chave para o desenvolvimento de sistemas adaptáveis e para a automação da criação de componentes. Metaprogramação com templates C++ é uma forma de metaprogramação limitada ao processo de compilação: o mecanismo de templates combinado com algumas outras outras funcionalidades da linguagem C++ permite a criação de componentes que são resolvidos pelo compilador da linguagem. Componentes metaprogramados podem ser utilizados para manipular as características dos componentes dinâmicos do sistema e esse é o princípio básico de funcionamento do sistema de comunicação SNOW. Além disso, o repositório de configuração do sistema de comunicação foi implementado com uma classe *trait*. Classes *trait* podem ser vistas como pequenos objetos estáticos cujo propósito é agrupar informações utilizadas por outros componentes [15].

Como mencionado anteriormente, o núcleo básico de comunicação e o protocolo composto implementam uma variação do padrão de projeto *strategy*. A principal diferença entre o mecanismo utilizado no sistema de comunicação SNOW e a implementação tradicional do *strategy* é que, no SNOW, o protocolo composto é gerado em tempo de compilação pelo suporte meta-programdo do sistema de comunicação de acordo com os requisitos da aplicação e com as informações contidas no repositório de configuração. Em tempo de compilação o componente *gerador de protocolos* percorre o repositório de configuração determinando quais estratégias de comunicação foram ativadas. As classes referentes às estratégias de comunicação ativas são utilizadas para inicializar uma nova configuração de protocolo composto que vai ser utilizada em tempo de execução. Todo o processo de seleção, configuração e combinação estática de estratégias de comunicação está sumarizada na figura 2.

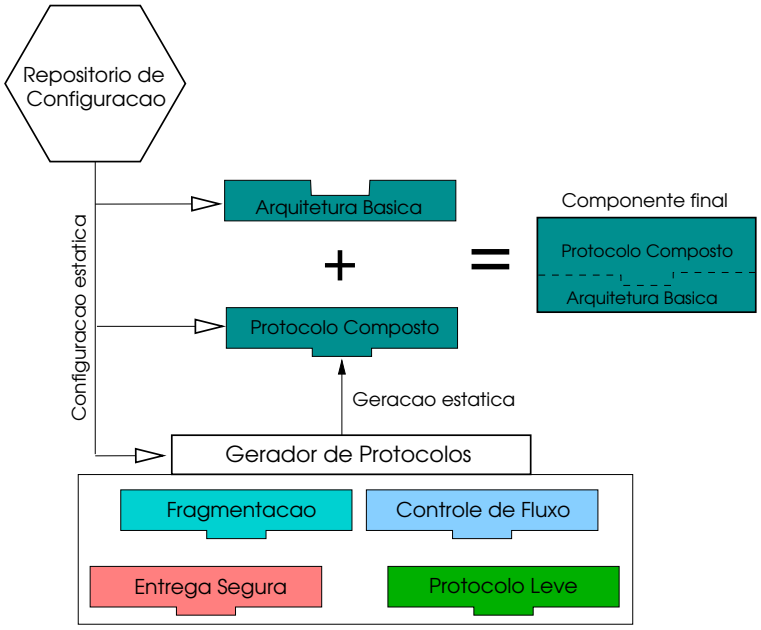

Figura 2: Seleção, configuração e combinação de protocolos leves em tempo de compilação.

O protocolo composto aceita como parâmetro de classe uma lista contendo as estratégias de comunicação ativas. Para cada método definido pelo protocolo composto, um mecanismo estático de geração de código é utilizado para percorrer a lista de estratégias de comunicação ativas gerando código para a chamada de método correspondente de cada estratégia de comunicação. Como todos os métodos definidos pelas estratégias de comunicação são declarados como *inline*, o compilador C++ substitui a chamada pelo corpo do método, evitando o overhead associado à chamada de funções. Note que o protocolo composto poderia facilmente ser implementado utilizando-se uma lista de estratégias de comunicação caso essas estendessem uma super-classe que definisse métodos virtuais. Essa abordagem teria a vantagem de possibilitar que novas estratégias fossem adicionadas ou removidas da lista em tempo de execução, permitindo que o comportamento do protocolo composto mudasse dinamicamente. Entretanto, isso implicaria em chamadas à funções virtuais, que podem ser até duas vezes mais caras do que chamadas à funções não virtuais ou até

três vezes mais caras do que uma atribuição simples, e também impossibilitaria que o *inlinig* fosse utilizado.

4 Protocolos leves de comunicação

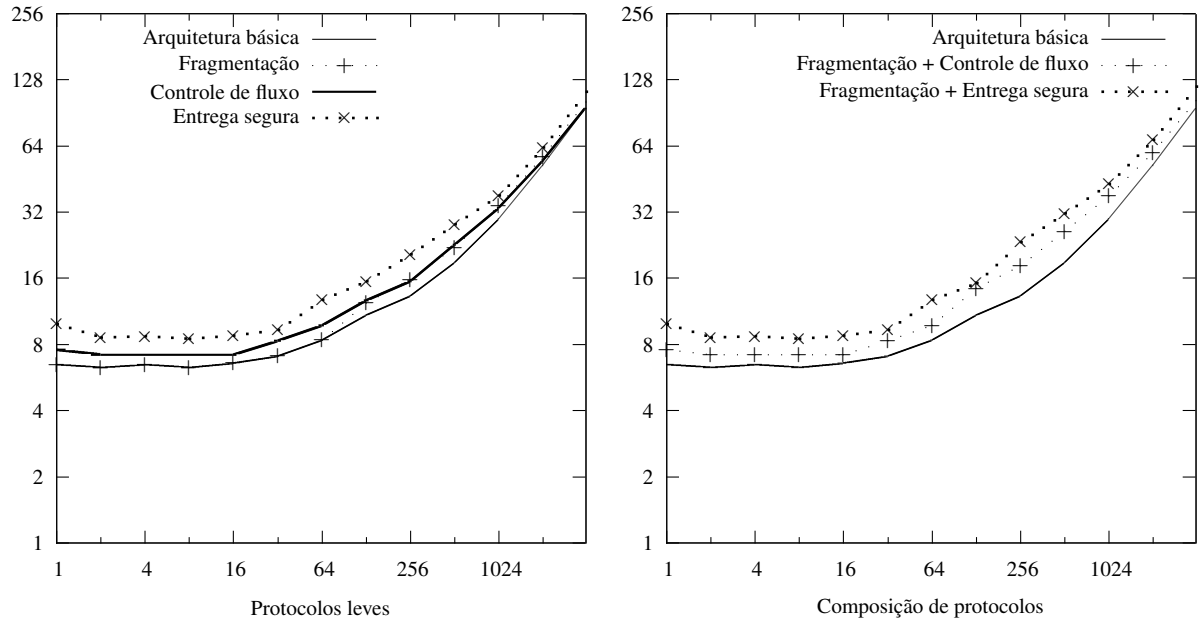

Figura 3: Latência relacionada ao núcleo básico de comunicação e aos protocolos leves implementados para o SNOW.

De maneira a validar o sistema de comunicação e como uma primeira etapa na implementação de protocolos para as aplicações paralelas suportadas pelo SNOW, algumas estratégias de comunicação que implementam protocolos leves foram desenvolvidas. Diversas aplicações necessitam de mecanismos de fragmentação, controle de fluxo e entrega segura de mensagens e por isso estratégias de comunicação que disponibilizam esses serviços foram implementadas sobre o núcleo básico de comunicação Myrinet descrita em [13].

Fragmentação é o processo de dividir mensagens grandes em pedaços menores durante o envio de acordo com a MTU do sistema de comunicação. Esses fragmentos devem ser reorganizados após o recebimento do último fragmento referente à mensagem pelo nodo receptor. No sistema de comunicação SNOW a MTU é dinamicamente configurável e portanto o comportamento do protocolo de fragmentação pode ser alterado em tempo de execução.

Controle de fluxo é o processo de ajustar o fluxo de dados durante a comunicação, com o objetivo de assegurar que o nodo receptor possa tratar todos os dados enviados a ele. Existem diversos mecanismos de controle de fluxo, dentre os quais podemos destacar a comunicação *rendezvous*, onde o nodo receptor fornece ao nodo transmissor um buffer disponível para o recebimento antes que o transmissor possa enviar a mensagem, e o mecanismo de *tokens*, onde o transmissor deve possuir créditos referentes ao receptor para que possa enviar uma mensagem. O protocolo de controle de fluxo implementado para o SNOW utiliza um dos mecanismos mais comuns para comunicação assíncrona: o *xon-xoff*. Nesse mecanismo, o receptor envia uma mensagem `xoff` para o transmissor quando os seus buffers para o recebimento de mensagens estão cheios fazendo com que o transmissor aguarde até que um `xon` seja enviado pelo nodo receptor, sinalizando que esse nodo está novamente apto a receber mensagens.

Entrega segura é o mecanismo que garante que todas as mensagens enviadas a um nodo são recebidas, mesmo que hajam problemas na rede de comunicação. Suporte eficiente a entrega segura de mensagens é um componente fundamental de qualquer sistema de comunicação, entretanto, devido a baixa taxa de erros de transmissão presente em tecnologias de rede modernas, diversos protocolos não implementam essa funcionalidade. O protocolo leve de entrega segura do SNOW foi implementado utilizando mecanismos de re-transmissão e timeout, além de estender o protocolo de controle de fluxo assegurando que as requisições de re-transmissão não sejam perdidas.

A figura 3 exibe a latência relacionada ao núcleo básico de comunicação do sistema SNOW e aos protocolos descritos acima, individualmente e combinados. A latência foi medida utilizando-se o benchmark *round-trip time* para diversos tamanhos de frame. É importante notar que o protocolo de entrega segura estende as funcionalidades do

protocolo de controle de fluxo e por isso não faria sentido apresentar a latência da combinação desses dois protocolos. Além disso, o protocolo de fragmentação baseia-se na MTU para efetuar a divisão das mensagens. Para os testes cuja performance é apresentada a MTU foi configurada em 64 bytes.

5 Trabalhos correlatos

Diversos projetos de pesquisa se propõem a desenvolver sistemas de comunicação que utilizam protocolos modulares projetados sobre frameworks. O principal diferencial do SNOW é o fato de toda a configuração, seleção e combinação de protocolos ser efetuada estaticamente através de um framework meta-programado, característica que tende a melhorar a performance de comunicação para aplicações dedicadas. Assim como o SNOW, o projeto *Tau* (Transport and up) implementa composição de protocolos sem encapsulamento em camadas através de um framework dinâmico projetado para suportar protocolos fim-a-fim.

O projeto *X-Kernel Protocol Framework* [9] consiste em um conjunto de protocolos modulares que podem ser dinamicamente ligados a um framework de comunicação que provê mecanismos para que um protocolo possa invocar operações de outros protocolos, isto é receber/enviar mensagens de/para um protocolo adjacente, além de uma coleção de bibliotecas para a manipulação de mensagens, eventos e *threads*. O X-Kernel é um projeto de pesquisa experimental onde novos protocolos de comunicação são projetados, implementados e sua performance é avaliada.

A principal vantagem relacionada às implementações dinâmicas de frameworks para a comunicação é a possibilidade de permitir que o sistema de comunicação adapte-se aos requisitos dinâmicos do sistema computacional. O projeto *CANEs* (Composable Active Netowrk Elements) consiste em um framework que inclui uma terminologia consistente, especificações de interface e um conjunto de requisitos funcionais para construção de *redes ativas* [2]. CANEs define um algoritmo de processamento de pacotes genérico que pode ser customizado através de instruções simples que podem ser inseridas em determinados pontos desse algoritmo.

Sunder e Musser [16] utilizam técnicas de meta-programação similares as utilizadas no desenvolvimento do sistema de comunicação SNOW para implementar suporte a programação orientada a aspectos. Além dos mecanismos fornecidos pela linguagem C++, o projeto também utilizou a biblioteca *Boost*, um framework meta-programado extensível que implementa funcionalidades equivalentes a STL.

6 Conclusão

A nova geração de sistemas computacionais e aplicações distribuídas vêm motivando a academia e a indústria a propor novas arquiteturas para os sistemas de comunicação modernos. Sistemas de comunicação tradicionais provêm aos seus usuários um conjunto fixo de serviços e uma pilha de protocolos monolítica, arquitetura que prejudica a performance de certas classes de aplicações e implica no desenvolvimento e utilização de camadas *middleware* que implementam os serviços que foram deixados de fora do sistema de comunicação. Além de prover mecanismos que permitam que as inovações tecnológicas disponibilizadas pelos fabricantes de redes de comunicação sejam utilizadas eficientemente, extensibilidade e configurabilidade são requisitos imprescindíveis para que os sistemas de comunicação modernos possam adaptar-se aos requisitos das aplicações.

O sistema de comunicação SNOW, cuja arquitetura foi discutida nesse artigo, foi projetado com o objetivo de atender os requisitos das aplicações dedicadas suportadas pelo ambiente de programação paralela SNOW. O principal componente desse sistema de comunicação consiste em um framework meta-programado que provê mecanismos para configurar e combinar os serviços de comunicação disponíveis. Novos serviços de comunicação podem ser implementados diretamente dentro do sistema de comunicação estendendo esse framework. Além da arquitetura do sistema de comunicação SNOW, as decisões de projeto referentes à implementação de alguns serviços de comunicação também foram discutidas. Como uma próxima etapa no desenvolvimento do ambiente de programação paralela SNOW, outros serviços de comunicação vão ser implementados e a performance do sistema de comunicação vai ser avaliada utilizando-se aplicações reais.

De maneira a validar a hipótese de que configuração estática é suficiente para o domínio de aplicações dedicadas o sistema de comunicação SNOW vai ser estendido de maneira a suportar configuração dinâmica do protocolo de comunicação. As estratégias de comunicação já desenvolvidas vão ser reutilizadas nesse novo cenário e a performance do sistema de comunicação com configuração dinâmica habilitada vai ser avaliada e comparada com a performance atual do sistema.

Referências

[1] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy. User-level interprocess communication for shared memory multiprocessors. In *ACM Transactions on Computer Systems*, pages 175–198, May 1991.

[2] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Reasoning about active network protocols. In *ICNP '98: Proceedings of the Sixth International Conference on Network Protocols*, page 31. IEEE Computer Society, 1998.

[3] Greg Burns, Raja Daoud, and James Vaigl. Lam: An open cluster environment for mpi. In *Proceedings of the Supercomputing Symposium*, pages 379–386, 1994.

[4] Antonio Augusto Frohlich, Philippe Olivier Alexander Navaux, Sergio Takeo Kofuji, and Wolfgang Schroder-Preikschat. Snow: a parallel programming environment for clusters of workstations. In *Proceedings of the 7th German-Brazilian Workshop on Information Technology*, Maria Farinha, Brazil, September 2000.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[6] Douglas P. Ghormley, David Petrou, Steven H. Rodrigues, Amin M. Vahdat, and Thomas E. Anderson. Glunix: A global layer unix for a network of workstations. *Software - Practice and Experience*, 28(9):929–961, 1998.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. In *Parallel Computing*, volume 22, pages 789–828, September 1996.

[8] H. Hellwagner, W. Karl, M. Leberecht, and H. Richter. Sci-based local-area shared-memory multiprocessor. In *Proceedings of the International Workshop on Advanced Parallel Processing Technologies - APPT'95*, Beijing, China, September 1995.

[9] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.

[10] Steven S. Lumetta, Alan M. Mainwaring, and David E. Culler. Multiprotocol active messages on a cluster of smp's. In *Proceedings of Supercomputing'97*, San Jose, CA, November 1997.

[11] Shashidhar Merugu, Samrat Bhattacharjee, Ellen W. Zegura, and Kenneth L. Calvert. Bowman: A node os for active networks. In *INFOCOM '00: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1127–1136, March 2000.

[12] Loic Prylli and Bernard Tourancheau. Bip: A new protocol designed for high performance networking on myrinet. In *IPPS/SPDP Workshops*, volume 1388 of *Lecture Notes in Computer Science*, pages 475–485, Orlando, FL, March 1998.

[13] Thiago Robert Claudino Santos and Antonio Augusto Frohlich. An application-oriented communication system for clusters of workstations. In *Proceedings of the First International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-1)*, pages 15–24, Saint-Malo, France, June 2004.

[14] Erich Strohmaier, Jack Dongarra Meuer Horst, and D. Simon. Top500 report for november 2004. Technical report, November 2004.

[15] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, third edition, 1997.

[16] Shyam Sunder and David R. Musser. A metaprogramming approach to aspect oriented programming in c++. March 2001.

[17] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. Pm: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking*, volume 1225 of *Lecture Notes in Computer Science*, pages 708–717, April 1997.