Sistema Aldeia: Invocação Remota e Assíncrona de Métodos sobre Infiniband e DECK*

Rodrigo da Rosa Righi, Philippe O. A. Navaux
Programa de Pós-Graduação em Computação
Universidade Federal do Rio Grande do Sul
{rrrighi, navaux}@inf.ufrgs.br

Marcelo Pasin
Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria
pasin@inf.ufsm.br

Resumo

A linguagem Java é cada vez mais utilizada para a construção de aplicações. O próprio sistema de invocação remota de métodos (RMI) da linguagem Java proporciona a escrita de aplicações distribuídas, permitindo comunicação através de TCP/IP entre computadores. Entretanto, tal protocolo impõem penalidades de software para a obtenção de alto desempenho na comunicação. Além disso, Java RMI realiza a comunicação de maneira síncrona, o que pode também contribuir para o decréscimo da eficiência de aplicações escritas com esse sistema. Visando o uso de Java para a programação de alto desempenho em aglomerados, está em desenvolvimento o sistema Aldeia. Ele possibilita a invocação remota e assíncrona de métodos sobre as interfaces de rede Infiniband e DECK. Esse artigo descreve a estrutura do sistema Aldeia, as tecnologias e as bibliotecas utilizadas para a sua confecção.

1. Introdução

A quantidade de problemas que demandam alto poder de processamento é crescente. Para dar suporte à solução deles, são construídas arquiteturas de máquinas específicas para alto desempenho e aplicações que consigam usufruir ao máximo o paralelismo e a distribuição por elas disponibilizados. O dueto formado por máquina e aplicação é utilizado para a solução de um dado problema, de modo a obter o seu resultado em um tempo satisfatório. Na área de arquiteturas de computadores, os aglomerados de computadores (*clusters*) surgem como a solução mais utilizada atualmente para a construção de ambientes de alto desempenho. Além dos aglomerados, também observa-se o aumento da utilização de grades computacionais (*grids*)[9] para a solução de problemas complexos que necessitem de alto poder de processamento.

* Financiamento: CNPq, FIPE/UFSM e projeto LabTec-Dell

Na área referente a construção de aplicações paralelas e distribuídas, pesquisam-se ferramentas simples e eficientes que sejam capazes de facilitar a sua criação, proporcionando portabilidade e alto desempenho. Como resultado de tais esforços, têm-se o crescimento da utilização de bibliotecas que possuam uma interface comum para trabalhar com diversas tecnologias de rede. A adoção da linguagem Java para a escrita de aplicações também tem se mostrado uma boa opção neste caso. Isso ocorre porque ela proporciona as facilidades da orientação a objetos, portabilidade entre diferentes arquiteturas de máquinas e por possuir um conjunto de classes bem definido para a programação com múltiplos fluxos de execução (*multithreading*) e com memória distribuída[11].

Nesse contexto, está em desenvolvimento o sistema Aldeia. Ele possibilita a escrita de aplicações em Java segundo o paradigma de programação distribuída de invocação remota de métodos, ou **RMI** (*Remote Method Invocation*) sobre tecnologias de rede de alto desempenho. Para tal, esse sistema aproveita a programação de invocação assíncrona e remota de métodos da biblioteca Java ProActive[4]. Com o intuito de aumentar o desempenho, Aldeia altera a comunicação padrão do sistema ProActive, que é baseada no protocolo TCP/IP, e implementa a troca de mensagens entre dois computadores com as bibliotecas de comunicação **DECK**[2] e **VAPI**[19].

A biblioteca DECK objetiva possibilitar a escrita de programas paralelos e distribuídos sobre várias tecnologias de rede, como Ethernet, Myrinet e SCI. A VAPI possibilita o desenvolvimento de programas que utilizam o *hardware* Infiniband para a sua execução. A Arquitetura Infiniband[13] busca padronizar uma série de protocolos e tecnologias de rede de sistema para alto desempenho. Ela proporciona uma infra-estrutura eficiente de comunicação em nível de usuário, podendo atingir uma taxa de vazão de até 30 gigabits por segundo.

Este artigo tem por objetivo apresentar os conceitos e o desenvolvimento do sistema Aldeia. Esta apresentação

compreende a descrição da nova arquitetura de rede Infiniband, da biblioteca Java ProActive e das bibliotecas de comunicação VAPI e DECK. Também será mostrada a estrutura do Aldeia, as questões de projeto para alcançar alto desempenho, as classes que ele implementa para a construção dos serviços de rede e o ambiente de testes e de execução a ser utilizado para a sua avaliação.

2. A Arquitetura Infiniband

A arquitetura Infiniband[13], ou IBA (*Infiniband Architecture*), é uma tecnologia lançada em 1999, baseada na arquitetura VIA[6] (*Virtual Interface Architecture*), que objetiva a interconexão de redes de computadores com alta velocidade e baixa sobrecarga (latência) de comunicação. Cada adaptador de rede Infiniband define uma das três taxas de transferência de dados possíveis pela especificação: 2.5, 10 ou 30 gigabits por segundo. A IBA é composta de nós processadores, também chamados nós hospedeiros, unidades de entrada e saída de dados (E/S), chaveadores e roteadores. Na figura 1, pode ser visualizada a organização de uma rede Infiniband.

Uma unidade de E/S pode compreender um ou mais dispositivos de E/S e seus respectivos controladores. Um nó processador representa uma plataforma de computação executando um sistema operacional e aplicações do usuário. Uma rede IBA é dividida em sub-redes inter-conectadas por roteadores e, cada sub-rede compreende um ou mais chaveadores, nós processadores e unidades de E/S. A infraestrutura de comunicação suporta troca de mensagens de nós processadores entre si e de nós processadores com unidades de E/S.

Figura 1. Organização de uma rede Infiniband

Em computadores convencionais, tipo PC, um dos principais fatores limitantes no desempenho é o sistema de entrada e saída. A principal proposta da arquitetura Infiniband é substituir o barramento local compartilhado por uma malha de comunicação chaveada. Essa organização permite que muitos componentes da rede realizem comunicação concorrente com alta largura de banda. IBA fornece para os nós processadores uma transferência de mensagens sem cópias intermediárias dentro do núcleo do sistema operacional dos nós envolvidos na comunicação. Isso ocorre através da técnica de agendamento de acesso direto a memória (DMA - *Direct Memory Access*)[10].

A transmissão de dados utilizando o *hardware* Infiniband proporciona uma série de recursos requeridos pelos sistemas de servidores de aplicações e sistemas de computação em aglomerados. Entre estes recursos, pode-se citar a determinação da qualidade de serviço, melhor desempenho para troca de mensagens, gerenciamento dinâmico de rotas, alta disponibilidade e tolerância à falhas. Além disso, a tecnologia Infiniband é independente do sistema operacional e da plataforma do processador presente em um nó hospedeiro[10].

Entre as inovações propostas por Infiniband estão o serviço de transporte de datagrama confiável e a operação atômica de troca de mensagens para RDMA (*Remote DMA*)[13]. Com o serviço de datagrama confiável, é possível realizar operações de grupo (*multicast*) com confiabilidade. O RDMA atômico permite a uma aplicação ler e modificar dados com atomicidade em uma localização de memória remota. Com esta operação é possível implementar bloqueios, mecanismos de exclusão mútua e manipulação atômica de variáveis por múltiplas aplicações.

Existe uma grande iniciativa da indústria para a utilização e validação da tecnologia Infiniband. Esta tecnologia vêm sendo aplicada em duas áreas da computação. A primeira envolve os sistemas de centros de dados que processam grandes volumes de informações. A outra área onde se utiliza Infiniband é a de redes especializadas para processamento de aplicações numéricas e simulações que demandam por alto desempenho usando aglomerados.

3. Biblioteca Java ProActive

ProActive[4] é uma biblioteca Java com código livre para a computação paralela e distribuída. Ela é desenvolvida pelo Instituto INRIA, na França, como parte integrante do projeto ObjectWeb. Ela oferece características de segurança no transporte dos dados e de migração de objetos entre diferentes máquinas virtuais Java. Essa biblioteca é construída somente com o conjunto de classes padrão de Java, sem requerer nenhuma alteração nos códigos fonte da distribuição da máquina virtual nem da linguagem Java. Isso facilita a sua extensão e a torna aberta a adaptações e otimizações[4].

A biblioteca em questão oferece uma interface mais simples para a construção de aplicações se comparada com Java RMI. A utilização do sistema RMI padrão, presente na linguagem Java, faz com que o programador realize modificações no código fonte na construção de interfaces, de modo a tornar objetos locais acessíveis remotamente. Além disso, anteriormente à execução de um programa Java

RMI, é necessário gerar os procuradores (*proxies*) local e remoto, e lançar o programa que realiza o serviço de nomes.

O ProActive elimina a necessidade de geração de uma interface para o objeto remoto e realiza o lançamento do servidor de nomes e a geração dos procuradores automaticamente em tempo de execução. Na figura 2, pode ser visualizada a interface de programação ProActive. Nessa figura, têm-se dois programas, um cliente (classe HelloClient) e outro servidor (classe Hello), escritos com essa biblioteca.

```
1.  class Hello
2.  {
3.    String name;
4.    public void Hello (String temp) { name = temp; }
5.    public String sayHello () { return name; }
6.    public main (String[] argv) {
7.      Object[] params = new object[] {
8.                          new String("Hello World") };
9.      Hello hello = (Hello) ProActive.newActive(
10.                  Hello.class.getName(), params);
11.     ProActive.register(hello,
12.                          "//localhost/" + "Hello");
13.   }
14. }
15. class HelloClient
16. {
17.   public main (String[] argv) {
18.     Hello client;
19.     client = (Hello) ProActive.lookupActive(
20.       Hello.class.getName(), "//localhost/Hello");
21.     System.out.println( client.sayHello() );
22.   }
23. }
```

Figura 2. Exemplo de Programas Cliente e Servidor com a Biblioteca ProActive

O principal conceito inerente ao desenvolvimento do ProActive está relacionado ao tratamento de **objetos ativos**. Um objeto ativo pode ser acessível remotamente e é composto de um objeto padrão Java e de um fluxo de execução, chamado corpo. O corpo não é visível para o usuário e é encarregado de tratar das operações de recepção de invocações de métodos sobre o objeto associado, do armazenamento destas em uma fila de requisições, e do envio dos resultados para os chamadores de cada requisição.

A biblioteca ProActive é capaz de realizar a invocação remota de métodos de maneira assíncrona. Para tal, ela faz uso dos recursos de **objetos futuros** e de **espera pela necessidade**. Para explicar a comunicação entre objetos com ProActive, pode-se visualizar a figura 3. Essa figura apresenta a organização dos objetos no momento da execução dos programas mostrados na figura 2.

O objeto cliente passa a requisição de invocação de método para o procurador, que conhece a localização do corpo do objeto a ser chamado. O procurador na máquina local retorna um objeto futuro que representa o resultado esperado e envia uma requisição para a máquina remota. A

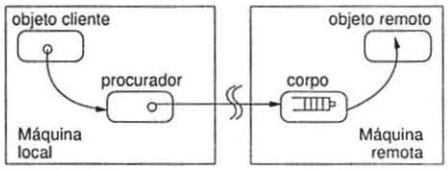

Figura 3. Objetos e Componentes ProActive

partir desse ponto, de posse do objeto futuro, o programa na máquina local pode prosseguir a sua execução automaticamente. O corpo invoca o método do objeto remoto, que processa a requisição, e envia o resultado para o procurador. O procurador, por sua vez, substitui o objeto futuro pelo objeto que representa o resultado. Caso o programa local efetuar alguma operação sobre o objeto futuro, o programa é bloqueado até a chegada efetiva do resultado da invocação. Esse processo caracteriza o recurso de espera pela necessidade, que segue o modelo de computação dirigido pelos dados (*dataflow*)[22].

Os objetos procurador e corpo, mostrados na figura 3, representam pontos finais de comunicação. Se eles estiverem localizados em máquinas diferentes, a comunicação entre eles acontece através da rede. Atualmente, ProActive realiza a troca de mensagens sobre redes configuradas com o protocolo padrão TCP/IP. Este protocolo foi inicialmente construído para sistemas distribuídos e heterogêneos como a Internet. Ele impõe penalidades de desempenho que não são necessárias para uma rede de sistema, como os aglomerados de computadores[19]. Estas penalidades compreendem cópias intermediárias de memória, verificação de somatórios, trocas de contexto entre o espaço do usuário e do núcleo do sistema operacional, entre outras[14].

O ProActive utiliza as classes padrão de soquetes TCP/IP para a conexão entre computadores: Socket e ServerSocket. O procurador do corpo na máquina local utiliza a classe Socket e atua como cliente. O corpo do objeto remoto realiza o papel de servidor de requisições e instancia um objeto da classe ServerSocket.

4. Bibliotecas de Comunicação

Esta seção apresenta as bibliotecas de comunicação VAPI e DECK que são utilizadas para o desenvolvimento do sistema Aldeia. Com essas interfaces de comunicação, Aldeia possibilitará a troca de mensagens sobre diferentes tecnologias e protocolos de rede.

4.1. Biblioteca VAPI

A biblioteca Verbs API, ou simplesmente VAPI[19], é desenvolvida pela empresa Mellanox e se propõe a implementar uma interface de programação consistente com todos as funções definidas na especificação Infiniband.

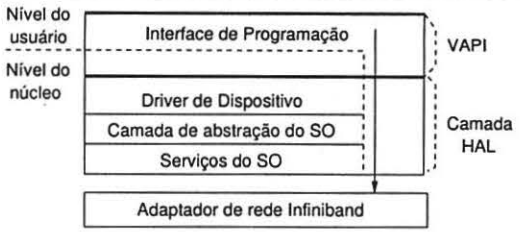

Figura 4. Arquitetura da biblioteca VAPI

A arquitetura de *software* da VAPI é ilustrada na figura 4. Esta figura apresenta a interface de programação VAPI como o nível ao topo para ambos os espaços do usuário e do núcleo do sistema operacional. Dessa forma, depois de realizadas as operações de conexão e de alocação de memória, as aplicações em nível de usuário podem trocar mensagens acessando diretamente o adaptador de rede, num processo conhecido como sobrepasso do sistema operacional.

A camada de abstração de *hardware* (HAL - *Hardware Abstraction Layer*) fornece uma interface entre as chamadas de funções VAPI e o adaptador de rede Infiniband. Esta camada faz uma abstração das características específicas de um dispositivo Infiniband. Isso torna desnecessário reescrever a camada VAPI no momento que é utilizado um adaptador de rede diferente. A camada de abstração do sistema operacional esconde os serviços fornecidos por diferentes sistemas operacionais e fornece uma interface comum para as funções de gerenciamento de memória e para as operações que tratam com interrupções.

4.2. Biblioteca DECK

A biblioteca DECK (*Distributed Execution and Communication Kernel*) foi desenvolvida no Grupo de Processamento Paralelo e Distribuído da UFRGS e proporciona suporte ao desenvolvimento de aplicações paralelas e distribuídas. Ela possibilita a manipulação de fluxos de execução, comunicação coletiva, variáveis de condição e de exclusão mútua, além da troca de mensagens tradicional. A comunicação entre dois computadores com essa biblioteca é realizada através de abstrações de caixas postais. Uma caixa postal representa um ponto final de comunicação e pode receber ou enviar mensagens.

A arquitetura da biblioteca DECK fornece uma interface de programação comum para trabalhar com diferentes tecnologias de rede. Uma de suas grandes vantagens é poder ser utilizada para a construção de ferramentas para a integração de aglomerados[1]. Atualmente, essa biblioteca possui suporte as tecnologias Ethernet[2], SCI[7] e Myrinet[18] e encontra-se em desenvolvimento a versão DECK que possui suporte para a Arquitetura de Interface Virtual (VIA)[8].

5. Sistema Aldeia

Esta seção apresenta o desenvolvimento do sistema Aldeia. Primeiramente, é discutida a motivação para o desenvolvimento desse sistema. As subseções que seguem tratam da estrutura do Aldeia, das módulos que o compõe, das questões de implementação do serviço de rede e das otimizações empregadas no sistema Aldeia. Para finalizar a seção, é apresentado o ambiente de execução no qual esse sistema será posteriormente avaliado.

5.1. Motivação

A linguagem Java vem sendo cada vez mais utilizada para a construção de aplicações paralelas e distribuídas. Para a escrita de aplicações desse gênero, ela oferece um conjunto de classes de alto nível que possibilita a invocação remota de métodos. O padrão RMI presente na distribuição Java possui a propriedade de realizar a comunicação entre objetos local e remoto em redes configuradas com o protocolo TCP/IP. As características desse protocolo fazem com que ele tenha o seu desempenho penalizado, e acabe não sendo a melhor opção para a troca de mensagens em aplicações que demandam por alto desempenho[17].

Outra questão inerente ao Java RMI é que ele realiza a invocação remota de métodos de forma síncrona. Isso faz com que o objeto chamador do método fique bloqueado até o retorno do resultado da requisição. Enquanto encontra-se bloqueado, o processo poderia realizar alguma computação útil para a solução da aplicação, aumentando a sua eficiência.

A motivação para a construção do sistema Aldeia está embasada no fato de construir um sistema, com interface em Java, para possibilitar RMI assíncrono sobre tecnologias de rede de alto desempenho. As questões de projeto desse sistema procuram ultrapassar os limites de desempenho colocados pelo padrão Java RMI. Para desenvolver o Aldeia, foram analisados sistemas RMI assíncronos existentes, como o ARMI[21] e o ProActive. O ARMI cria um fluxo de execução para cada invocação de método e usa o sistema Java RMI, que possui a sua implementação dentro da distribuição Java, para a comunicação entre objetos. Além disso, o ARMI exige que o programador tenha consciência da assincronia entre a chamada remota e a sua efetiva execução, sendo necessário verificar o final de chamadas remotas. O ProActive é uma biblioteca de classes com código aberto, possível de alterações, que implementa todo o processo de invocação assíncrona e remota de métodos de forma transparente para o usuário. Com base nessa característica, o sistema Aldeia aproveita, assim, a interface e a implementação de RMI presentes no ProActive. Com essa adoção, têm-se, além do benefício

da comunicação assíncrona, a facilidade na escrita e na execução de uma aplicação paralela e distribuída.

Com o intuito de maximizar o desempenho, o sistema Aldeia substitui a camada de rede do ProActive, construída sobre soquetes TCP, por uma interface capaz de proporcionar comunicação entre computadores com alto desempenho. A interface de rede do Aldeia é implementada com as bibliotecas de comunicação VAPI e DECK. Dessa forma, o sistema poderá ser utilizado em aglomerados que possuam tecnologias de rede Infiniband, Myrinet, SCI ou Ethernet.

Com as tecnologias Infiniband, Myrinet e SCI, o sistema Aldeia poderá transportar grandes quantidades de dados com alta taxa de largura de banda e baixa latência de comunicação, sejam esses dados os parâmetros ou o resultado de uma invocação remota de método. O sistema em pauta poderá ser utilizado tanto em redes locais, quanto em redes especializadas para o processamento de dados, como os centros de dados, ou para alto desempenho, como os aglomerados ou grades computacionais.

5.2. Estrutura

Aldeia reimplementa as classes de soquetes padrão do Java para trabalhar com as bibliotecas DECK e VAPI. Estas duas últimas estão escritas em linguagem C (compilado para código nativo) e faz-se necessária uma interface que sirva de ligação entre programas Java e C. Para facilitar a estruturação do Aldeia, ele foi logicamente dividido em módulos. A figura 5 mostra a disposição dos módulos desse sistema e as suas interações.

O sistema Aldeia aproveita a interface para o usuário e a implementação de RMI assíncrono oferecidos pelo ProActive, que já está implementada. Nela, são desenvolvidos o registro e a busca de objetos e o empacotamento e desempacotamento dos dados.

ProActive utiliza soquetes padrão para realizar a comunicação entre objetos local e remoto. Aldeia reimplementa as classes de soquetes Java para trabalhar com a biblioteca VAPI. Esse conjunto de classes especializadas se encontra no módulo chamado de Soquetes Aldeia (ver figura 5). Neste módulo são construídas as classes que tratam da conexão de dois computadores e dos canais de comunicação para a escrita e para a leitura de dados sobre as tecnologias suportadas por esse sistema. As questões inerentes a implementação deste conjunto de classes são apresentadas na seção 5.3.

As classes Java presentes no módulo de Soquetes Aldeia possuem métodos nativos, cujas implementações já estão compiladas para código nativo de máquina. O módulo 3, Adaptador VAPI (também figura 5), representa um conjunto de funções, escritos em linguagem C, que possuem a implementação de cada um desses métodos. O sistema JNI (*Java Native Interface*), presente nas distribuições Java, é

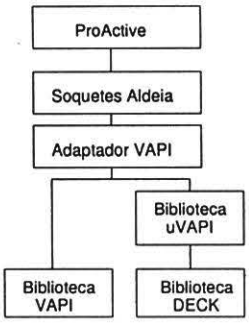

Figura 5. Módulos do sistema Aldeia

empregado para tratar da interface entre programas escritos em Java com o Adaptador VAPI, desenvolvido em linguagem C.

O Adaptador VAPI implementa funções básicas que realizam a conexão entre dois computadores e a troca de mensagens entre eles. Para isso, ele utiliza a interface de chamada de funções da biblioteca VAPI. Desta forma, para realizar comunicação através de equipamentos Infiniband, basta ligar diretamente o Adaptador VAPI com a biblioteca VAPI.

Para proporcionar comunicação transparente entre o Adaptador VAPI e DECK, está em desenvolvimento a biblioteca de adaptação chamada μVAPI. Ainda na figura 5, ela está posicionada entre o Adaptador VAPI e a biblioteca DECK. A μVAPI tem por objetivo transpor qualquer chamada da interface VAPI para a chamada de uma função equivalente DECK. Ela implementa um subconjunto de 11 funções da VAPI e algumas estruturas de dados que são utilizadas como parâmetros ou como retorno de funções. Este subconjunto de funções representa somente aquelas que são utilizadas para realizar a conexão e a transferência de dados entre dois computadores. A μVAPI, além de servir como adaptador para o sistema Aldeia, pode servir para portar aplicações escritas com a VAPI para executar sobre as tecnologias suportadas pela biblioteca DECK.

A camada mais baixa do sistema Aldeia é composta pelas bibliotecas de comunicação. Uma instância do sistema Aldeia poderá utilizar ou a biblioteca VAPI, ou a biblioteca DECK. Caso for utilizada DECK, poder-se-á trocar mensagens através de redes Myrinet, Ethernet ou SCI. Se o sistema for ligado a biblioteca VAPI, será possível a comunicação entre computadores em uma rede Infiniband.

5.3. Implementação

O Aldeia está atualmente em desenvolvimento. Apesar de não existir uma versão do sistema, várias questões de implementação foram decididas previamente. Esta subseção trata do desenvolvimento dos Soquetes Al-

deia. Neles são implementadas 4 classes Java, duas delas para realizar a conexão entre computadores e duas para a troca de mensagens. Objetos dessas classes carregam, no momento de sua instanciação, a biblioteca dinâmica do Adaptador VAPI, que possui os códigos referentes aos métodos nativos que ele implementa.

Para a conexão entre dois computadores, são implementadas duas classes Java: AldeiaSocket e AldeiaServerSocket. A primeira é utilizada por aplicações cliente e representa um ponto final de conexão. Essa classe possui o método connect(), que realiza a conexão entre dois pontos ligados com o sistema Aldeia. Um objeto da classe AldeiaServerSocket é instanciado em uma aplicação que atua como servidor. Nesta classe, é disponibilizado o método accept(), que estabelece a comunicação com um cliente e retorna um objeto que representa um ponto final de conexão.

Para realizar a troca de mensagens, a classe que representa um ponto final de conexão implementa os métodos getOutputStream() e getInputStream(). Esses métodos são responsáveis por retornar um canal de comunicação para tráfego de dados. A figura 6 apresenta as classes utilizadas para a conexão e para a troca de mensagens entre duas máquinas com o Aldeia.

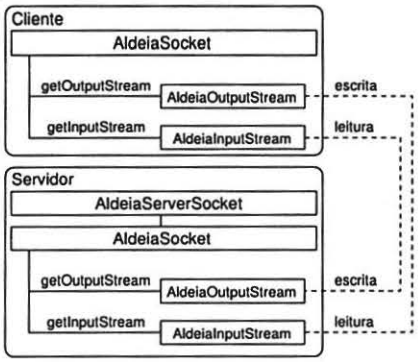

Figura 6. Organização das classes para a conexão e comunicação com Soquetes Aldeia

As classes do Aldeia, AldeiaOutputStream e AldeiaInputStream, derivam, respectivamente, das classes padrão para comunicação de dados em Java, OutputStream e InputStream. Elas representam uma abstração de um canal para a escrita e leitura de dados sobre as tecnologias de rede suportadas pelo Aldeia. Toda a troca de mensagens entre computadores com esse sistema deve necessariamente utilizá-las. As interfaces de métodos de ambas classes estão apresentadas em linguagem Java na figura 7.

Para a escrita de dados pela rede, a classe AldeiaOutputStream utiliza o método nativo writeAldeia(). Ele é invocado pelos demais métodos desta classe que realizam a escrita de dados. O método writeAldeia() possui a sua implementação escrita em linguagem C e para a sua invocação é utilizado o sistema JNI (ver subseção 5.2).

```
1.  class AldeiaOutputStream extends OutputStream
2.  {
3.        AldeiaOutputStream() {}
4.        void close() {}
5.        void flush() {}
6.        void write(byte[] b){ }
7.        void write(byte[] b, int off, int len) {}
8.        abstract void write(int b) {}
9.        private native void writeAldeia(byte[] b,
10.                   int off, int len) {}
11. }
12. class AldeiaInputStream extends InputStream
13. {
14.       AldeiaInputStream() {}
15.       int available() {}
16.       void close() {}
17.       void mark(int readlimit) {}
18.       boolean markSupported() {}
19.       abstract int read() {}
20.       int read(byte[] b) { }
21        int read(byte[] b, int off, int len) {}
22.       void reset() {}
23.       long skip(long n) {}
24.       private native void readAldeia(byte[] b,
25.                  int off, int len) {}
26. }
```

Figura 7. Interface de métodos das classes para a escrita e leitura de dados no Aldeia

Depois que dois computadores estiverem conectados utilizando o Aldeia, eles podem proceder com a troca de mensagens. A linguagem Java é orientada a objetos e, portanto, é natural que o processo de troca de mensagens consiga passar objetos através do canal de comunicação. Para tal, Java disponibiliza as classes ObjectOutputStream e ObjectInputStream. A classe ObjectOutputStream, por exemplo, realiza a escrita de objetos sobre um canal de saída de dados. Este canal é representado por um objeto do tipo OutputStream, que é passado como parâmetro para o construtor da classe em questão. Para trabalhar com o sistema Aldeia, deve ser passado um objeto da classe AldeiaOutputStream para esse construtor.

Para adaptar os soquetes Aldeia à biblioteca ProActive, é necessário somente substituir as chamadas de soquetes padrão Java, (Socket e ServerSocket), para as duas classes que descrevem o funcionamento de soquetes com Aldeia, (AldeiaSocket e AldeiaServerSocket). Esta facilidade é dada pela propriedade de polimorfismo e reusabilidade de código inerentes a linguagem Java.

Seguindo a mesma idéia da estrutura de classes apresentada na figura 6, um conjunto de soquetes Java especializados para a arquitetura VIA foi implementado e validado[5]. Essa implementação serviu de base para o desenvolvimento de RMI sobre VIA e testes realizados com soquetes VIA mostraram que ela atinge menor latência de comunicação se comparado com soquetes Java TCP.

5.4. Otimização

Para aumentar o desempenho do sistema Aldeia, são adotadas duas ações de otimização. A primeira diz respeito as ferramentas Java para melhorar a execução do Aldeia, como a utilização de um JNI otimizado e um compilador JIT (*Just In Time*)[15]. A segunda ação trata de uma otimização dentro do Aldeia, através da redução das chamadas entre programas Java e C.

As distribuições padrão do Java 2 trabalham com a invocação de métodos nativos. Nelas, o sistema JNI é utilizado para a implementação de diversas classes (incluindo java.io, java.lang e java.net), de forma a acessar as implementações em baixo nível escritas com código nativo[16]. Para otimizar as chamadas de métodos nativos, o sistema Aldeia pode utilizar a máquina virtual Java implementada pela empresa IBM[12].

Na distribuição Java da IBM, foram feitas algumas melhorias para proporcionar um sistema JNI mais eficiente. Os benefícios desse sistema incluem a reusabilidade da pilha Java para reduzir a sobrecarga na invocação de métodos, aumento da velocidade das primitivas JNI e a alteração dos métodos das bibliotecas de classes para evitar cópias desnecessárias de memória. Essa mesma distribuição Java pode fazer uso de um compilador JIT, que traduz, em tempo de execução, métodos Java diretamente para instruções de máquina. Essa técnica colabora para a rapidez da execução se comparado com a interpretação Java. JIT coloca o resultado da compilação na memória *cache*, de modo a eliminar traduções redundantes realizadas pelos interpretadores[15].

Na segunda ação para aumentar o desempenho do Aldeia, procura-se reduzir as chamadas aos métodos writeAldeia() e readAldeia(), presentes respectivamente, nas classes AldeiaOutputStream e AldeiaInputStream. Os métodos da classe AldeiaOutputStream, por exemplo, antes de chamarem writeAldeia(), trabalham com uma região de memória pré-alocada de tamanho fixo. Quando esta região estiver completada com os dados, procede-se com a troca de mensagem. Esta implementação facilita para o receptor dos dados, que sabe de antemão o tamanho deles. Mais especificamente, o tamanho de cada pacote na arquitetura Infiniband é fixo[13]. Assim, a região de memória pré-alocada possui tamanho igual ao pacote Infiniband. Isso resulta em uma melhor utilização da rede e proporciona maior eficiência para a aplicação.

5.5. Ambiente de Execução e de Testes

O sistema Aldeia, em um primeiro momento, será avaliado em aglomerados de computadores que apresentem tecnologias de rede compatíveis com as bibliotecas DECK e VAPI. Esse sistema também pode ser agregado a ferramentas destinadas a grades computacionais [9]. As grades podem compreender domínios com várias tecnologias de rede e o Aldeia, com as suas facilidades de comunicação, pode agregar mais ambientes para essa plataforma de execução.

O ambiente de testes do sistema Aldeia depende da biblioteca de comunicação utilizada, DECK ou VAPI. Caso for escolhida a primeira, tem-se a facilidade de executar o sistema sobre qualquer rede com equipamentos padrão Ethernet, Myrinet ou SCI. Para realizar testes sobre equipamentos Myrinet, está disponível no Instituto de Informática da UFRGS o aglomerado Corisco.

O aglomerado utilizado para a execução do Aldeia com a interface de comunicação VAPI é o aglomerado LabTeC (Laboratório de Tecnologia em Clusters), também localizado no Instituto de Informática da UFRGS. Nesse aglomerado, tem-se um conjunto de *hardware* da empresa Mellanox composto de 4 adaptadores de rede, cada qual com uma vazão máxima de 2.5 gigabits por segundo, cabos de interconexão e um chaveador, todos construídos com a tecnologia Infiniband. Testes com um programa simples, escrito em linguagem C, que realiza o envio e recebimento de mensagens avaliaram uma largura de banda máxima de 2.1 gigabits por segundo e uma latência de 7 microsegundos.

6. Conclusão

A evolução das linguagens de programação levam os desenvolvedores de aplicações a adotarem aquelas que possuam mais recursos e que sejam eficientes e simples. Nesse sentido, a linguagem Java tem emergido como uma das mais utilizadas. Ela possibilita reusabilidade de código, polimorfismo, herança e também proporciona portabilidade entre diferentes sistemas operacionais e arquiteturas de computadores. A queda de desempenho advinda da simulação de uma máquina virtual tem sido eficientemente melhorada com compiladores para código nativo durante a execução.

Java é empregada para a construção de aplicações paralelas e distribuídas, visto que oferece um conjunto de classes bem definido para tal finalidade. Para essas áreas, essa linguagem também fornece um sistema de alto nível que realiza a invocação remota de métodos, chamado Java RMI. Entretanto, Java RMI não se propõe a construção de aplicações que demandam por alto desempenho e que realizam bastante comunicação pela rede. Tal observação está argumentada na maneira como Java RMI é implementado. Ele realiza a comunicação sobre o protocolo TCP/IP, que impõe sobrecargas de *software*, e de forma síncrona, que contribui para reduzir a eficiência de aplicações.

O sistema Aldeia objetiva possibilitar a invocação assíncrona e remota de métodos sobre interfaces de rede de alto desempenho, como SCI, Myrinet e Infiniband. Aldeia reimplementa as classes de soquetes para a conexão de computadores. Nele também são reimplementadas as classes que representam os canais de comunicação para o trans-

porte de dados. Essas classes possuem métodos nativos, implementados previamente em linguagem C. O conjunto de funções em C que implementa os métodos nativos formam o Adaptador VAPI. Ele possui uma ligação direta com a biblioteca VAPI, permitindo comunicação em redes Infiniband. Para estabelecer compatibilidade com DECK, este adaptador utiliza as funções da μVAPI, que reimplementa um subconjunto de funções da VAPI para utilizar a biblioteca DECK.

A principal contribuição do sistema Aldeia é a possibilidade de realizar comunicação, com melhor desempenho que TCP/IP, sobre a tecnologia Infiniband. Esse trabalho representa uma facilidade para usuários que utilizam Java e possuam equipamentos Infiniband. Programas escritos com ProActive tiram proveito do sistema Aldeia sem alteração nenhuma. Além deles, outros programas quaisquer que utilizem soquetes Java podem ser facilmente modificados para usar o Aldeia. Por fim, mas não menos importante, esta é a primeira iniciativa de utilização da biblioteca DECK para a programação orientada a objetos.

As implementações dos módulos Adaptador VAPI e Soquetes Aldeia já estão finalizadas. A cargo de trabalhos futuros, têm-se a confecção da μVAPI e a análise do desempenho do Aldeia se comparado com os sistemas Java RMI e ProActive com TCP/IP. Com o intuito de aumentar o desempenho do Aldeia, almeja-se ainda alterar as classes Java que realizam a serialização de objetos. Isto pode ser feito para se tirar proveito das estruturas de dados mais complexas da VAPI, que permitem definir cartografias de memória[20] para o envio de dados. Da mesma forma, a VAPI apresenta funções de interrupção a distância que poderiam servir para aumentar a reatividade do sistema[3].

Referências

[1] M. Barreto, R. Ávila, and P. Navaux. The MultiCluster model to the integrated use of multiple workstation clusters. In *Proc. of the 3rd Workshop on Personal Computer based Networks of Workstations*, volume 1800 of *Lecture Notes in Computer Science*, pages 71–80, Cancun, 2000.

[2] M. E. Barreto, P. O. A. Navaux, and M. P. Rivière. 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*, volume 2, pages 623–637, Argentina, 1998.

[3] A. Carissimi. *Le noyau exécutif Athapascan-0 et l'exploitation de la multiprogrammation légère sur les grappes de stations multiprocesseurs.* Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble, France, Nov. 1999.

[4] D. Caromel, W. Klauser, and J. Vayssiere. Towards seamless computing and metacomputing in Java. In G. C. Fox, editor, *Concurrency Practice and Experience*, volume 10, pages 1043–1061. Wiley & Sons, Ltd., Set. - Nov. 1998.

[5] Y.-T. Chen, W.-J. Wu, C.-K. Chen, and J.-K. Lee. Building java rmi for meta-cluster servers with network processor. *10th Workshop on Compiler Techniques for High-Performance Computing (CTHPC 2004)*, 2004.

[6] Compaq, Intel, and Microsoft. Virtual Interface Architecture Specification version 1.0, Dec. 1997. Disponível em http://www.cs.cornell.edu/barr/repository/cs614/san_10.pdf.

[7] F. A. D. de Oliveira. Uma biblioteca para programação paralela por troca de mensagens de *clusters* baseados na tecnologia SCI. Master's thesis, Programa de Pós-Graduação em Computação - UFRGS, fev. 2001.

[8] L. A. de Paula e Silva and P. O. A. Navaux. Implementação da biblioteca de comunicação deck sobre o padrão de protocolo de comunicação em nível de usuário via. In *IV Escola Regional de Alto Desempenho*, volume 4, pages 155–156, Pelotas/RS, 2004.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 2003.

[10] W. T. Futral. *Infiniband Architecture Development and Deployment.* Intel Press, primeira edition, 2001.

[11] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multiparadigm communications in Java for grid computing. *Communications of the ACM*, 44(10):118–125, Out. 2001.

[12] IBM Corporation. IBM Developer kit for Linux, 2004. Disponível em http://www-106.ibm.com/developerworks/-views/java/downloads.jsp.

[13] InfiniBand Trade Association IBTA. Inifiniband architecture specification, vol. 1, release 1.1, 2002, Nov. 2002. Disponível em http://www.infinibandta.org/specs.

[14] J. Kay and J. Pasquale. Profiling and reducing processing overheads in TCP/IP. *IEEE/ACM Trans. Netw.*, 4(6):817–828, 1996.

[15] I. H. Kazi, H. H. Chen, B. Stanley, and D. J. Lilja. Techniques for obtaining high performance in java programs. *ACM Comput. Surv.*, 32(3):213–240, 2000.

[16] S. Liang. *Java Native Interface: Programmer's Guide and Specification.* Addison-Wesley, 1999.

[17] J. Maassen, R. V. Nieuwpoort, R. Veldema, H. Bal, T. Kielmann, C. Jacobs, and R. Hofman. Efficient Java RMI for parallel programming. *ACM Transactions on Programming Languages and Systems*, 23(6):747–775, Nov. 2001.

[18] C. C. Marchezan. DECK/GM: Implementação do ambiente DECK através do sistema GM para tecnologia Myrinet, 2003. Porto Alegre: UFRGS. 61p. Trabalho de Graduação.

[19] Mellanox. Introduction to infiniband. Technical report, Mellanox Technologies Inc., Santa Clara, California (EUA), 2000.

[20] M. Pasin. *Mouvement efficace de données pour la programmation parallèle irrégulière.* Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble, France, Nov. 1999.

[21] R. R. Raje, J. I. Williams, and M. Boyles. Asynchronous Remote Method Invocation (ARMI) mechanism for Java. *Concurrency: Practice and Experience*, 9(11):1207–1211, Nov. 1997.

[22] J. Veen. Dataflow machine archirecture. *ACM Computing Surveys*, 18, 1986.