

Paralelização de uma Aplicação para Análise de Dados Micrometeorológicos Utilizando uma Abordagem *Peer-to-Peer**

Marcelo Veiga Neves¹, Tiago Scheid², Andrea Schwertner Charão²,
Guilherme Sausen Welter³, Osvaldo Luiz Leal de Moraes³

¹Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15064 – 91501-970 – Porto Alegre – RS

²Laboratório de Sistemas de Computação (LSC)
Curso de Ciência da Computação – Universidade Federal de Santa Maria (UFSM)
Informática/CT – Campus UFSM – 97105-900 – Santa Maria – RS

³Laboratório de Micrometeorologia (L μ Met)
Centro de Ciências Naturais e Exatas – Universidade Federal de Santa Maria (UFSM)
Campus UFSM – 97105-900 – Santa Maria – RS

mvneves@inf.ufrgs.br, {scheid, andrea}@inf.ufsm.br

gswelter@yahoo.com.br, ollmoraes@smail.ufsm.br

Resumo. *Este artigo apresenta uma experiência de paralelização de uma aplicação que analisa grandes conjuntos de dados micrometeorológicos, utilizando uma abordagem peer-to-peer. Para isso, desenvolveu-se um sistema baseado na plataforma JXTA, permitindo a distribuição e coordenação de tarefas em uma rede peer-to-peer. Este sistema também leva em conta a ociosidade dos recursos disponíveis para promover um melhor aproveitamento dos mesmos. Com as adaptações realizadas na aplicação e sua execução sobre o sistema desenvolvido, obteve-se ganho de desempenho com o processamento de maiores conjuntos de dados em um tempo mais curto.*

1. Introdução

O processamento paralelo e distribuído é frequentemente utilizado para resolver problemas que demandam grande poder computacional. Através da paralelização de aplicações, pode-se obter aumentos significativos de desempenho, realizando-se operações mais rapidamente ou processando-se maiores volumes de dados.

Dentre as diferentes abordagens para a construção de aplicações paralelas e distribuídas, destacam-se os sistemas *peer-to-peer* (P2P) [Milojicic et al. 2002], que permitem aproveitar o poder de processamento de múltiplos computadores interligados em rede. Esta abordagem é vantajosa para se lidar com ambientes de execução dinâmicos, com configurações heterogêneas e que não são constantemente dedicados à computação paralela e distribuída. Outra característica frequentemente associada a sistemas P2P é o aproveitamento do tempo ocioso de estações de trabalho para a execução de aplicações de alto desempenho.

*Trabalho parcialmente financiado por FAPERGS e CNPq.

Neste contexto, este artigo explora uma abordagem *peer-to-peer* para paralelização de uma aplicação que analisa grandes conjuntos de dados coletados por sensores micrometeorológicos. Os objetivos deste trabalho são, de um lado, obter ganho de desempenho com a adaptação da aplicação existente e, de outro, flexibilizar o uso dos recursos computacionais disponíveis, permitindo aproveitar a ociosidade dos mesmos. Para isso, utilizou-se a plataforma JXTA [JXTA 2001], que fornece uma infra-estrutura básica para construção de aplicações P2P.

A seção 2 a seguir fornece uma visão geral sobre a plataforma JXTA e seu uso para desenvolvimento de aplicações paralelas e distribuídas. Na seção 3 apresenta-se a aplicação alvo deste trabalho, seguida, na seção 4, da descrição do sistema desenvolvido para suportar sua execução paralela e distribuída sobre a plataforma JXTA. A seção 5 relata o trabalho de paralelização da aplicação existente, que é avaliada na seção 6.

2. Plataforma JXTA

JXTA é uma plataforma Java para desenvolvimento de aplicações distribuídas, projetada pela Sun Microsystems e desenvolvida com o auxílio de especialistas de instituições acadêmicas e industriais [JXTA 2001]. Esta plataforma tem como objetivo abstrair a complexidade de sistemas de computação P2P, garantindo ao mesmo tempo portabilidade e interoperabilidade.

Em linhas gerais, a plataforma JXTA consiste em um conjunto de protocolos baseados em XML. Através desses protocolos, JXTA define abstrações que permitem a construção de uma rede virtual P2P, provendo assim transparência quanto à localização dos recursos através da atribuição de endereços lógicos únicos. Essa rede virtual possibilita que os *peers* possam trocar mensagens independentemente da sua localização real, sendo estas mensagens roteadas de forma transparente entre nós que são cercados por *firewalls* ou que utilizam protocolos de comunicação diferentes. A comunicação entre *peers* é realizada através de *pipes*, que são canais para envio e recebimento de mensagens entre serviços e aplicações.

O uso de JXTA como infra-estrutura de base para computação distribuída é explorado em projetos como OurGrid [Cirne et al. 2003], P3 [Shudo et al. 2005], e JuxMem [Antoniou et al. 2005]. Ao iniciar-se o presente trabalho, avaliou-se a possibilidade de se utilizar OurGrid ou P3 para se atingir os objetivos propostos, uma vez que ambos sistemas permitem agregar recursos de processamento. Optou-se, no entanto, pelo desenvolvimento de um sistema próprio capaz de suportar diferentes estratégias de paralelização e distribuição da aplicação alvo, além permitir a integração de um detector de ociosidade desenvolvido em outro trabalho [Mathias 2005].

3. Análise de Dados Micrometeorológicos

A aplicação alvo considerada neste trabalho foi originalmente desenvolvida no Laboratório de Micrometeorologia ($L\mu$ Met) da Universidade Federal de Santa Maria. A área de micrometeorologia é o ramo das ciências atmosféricas que estuda os fenômenos físicos de pequena escala espaço-temporal, que ocorrem na camada limite planetária¹. Estudos na área de micrometeorologia contribuem, por exemplo, para a solução de problemas ambientais e para a previsão do tempo e do clima.

¹Camada atmosférica em contato com a superfície da Terra, com espessura da ordem de 1km.

Observações de campo são freqüentemente empregadas em estudos micrometeorológicos. Os dados coletados em estações meteorológicas, geralmente bastante numerosos, precisam ser analisados e interpretados, o que demanda um grande poder computacional. No caso do L μ Met, a aplicação em questão é normalmente utilizada para processar os dados coletados durante um dia, em uma taxa de 16 coletas por segundo. Em um dia, 1.382.400 amostras são coletadas, gerando um arquivo de aproximadamente 100 MBytes. O processamento desse arquivo é realizado em partes, ou janelas, de 2^{15} linhas. Após o processamento de cada janela, os resultados da análise são salvos em arquivos, sendo que ao final do processamento tem-se aproximadamente 70 MBytes de resultados.

Testes realizados com a aplicação em um computador mono-processado Intel Pentium IV a 2.40 GHz, com 512 MBytes de memória RAM e 512 Kbytes de memória *cache*, demonstram que, para processar um arquivo de um dia, a aplicação executa, em média, durante 24,04 minutos.

Apesar dessa aplicação ser importante para os experimentos do L μ Met, nem sempre é possível dedicar recursos à sua execução. Desta forma, pode-se utilizar estações de trabalho, ocasionalmente ociosas neste laboratório, para o processamento paralelo e distribuído da aplicação. O sistema desenvolvido para dar suporte a este processamento é descrito na seção seguinte.

4. Arquitetura do Sistema Desenvolvido

Com o intuito de aproveitar os recursos do laboratório L μ Met e aumentar o desempenho da aplicação, foi implementado um sistema capaz de prover uma plataforma dinâmica de execução, permitindo a distribuição e coordenação de tarefas em um ambiente não-dedicado (rede de estações de trabalho do laboratório). O restante desta seção descreve a implementação e funcionamento deste sistema.

4.1. Visão Geral

A arquitetura do sistema baseia-se em um modelo P2P puro [Schollmeier 2001], onde os *peers* são os computadores que integram a plataforma de execução. Neste modelo, todos os *peers* são considerados iguais e podem interagir entre si sem a interferência de um servidor centralizado. Além disso, a interação dos *peers* independe da estrutura física de rede, pois todos fazem parte de uma mesma rede virtual.

A plataforma de execução é dinâmica, ou seja, seus computadores podem estar disponíveis apenas por determinados períodos. Esse comportamento dinâmico é decorrente da natureza de disponibilização de recursos. Logo, a configuração da rede virtual P2P pode variar constantemente, pois os computadores podem ter diferentes períodos de disponibilidade.

Como os computadores não são dedicados ao sistema, há a necessidade de utilizar somente os períodos ociosos de modo a diminuir a intrusividade do sistema nas aplicações dos usuários. O sistema implementado monitora a utilização dos recursos de cada computador para detectar períodos de ociosidade computacional que podem ser aproveitados para o processamento da aplicação.

Cada computador que integra a rede virtual é considerado um *peer*. Os *peers* podem interagir entre si, através da troca de mensagens, ou receber requisições de clientes.

Clientes permitem que os usuários submetam trabalhos para a plataforma de execução. A seguir são descritos os elementos de um *peer* e o funcionamento de um cliente.

4.2. Elementos de um Peer

Cada *peer* do sistema é estruturado conforme a figura 1. Todos os elementos desta estrutura são encapsulados em um programa, que é executado em cada *peer* como um processo *daemon*. Um *peer* pode criar um ou mais trabalhadores. Trabalhadores recebem submissões de trabalhos. Os trabalhadores interagem com o monitor de ociosidade do *peer* para processar somente em períodos ociosos. O controlador do *peer* realiza descobertas de recursos e mantém a rede virtual.

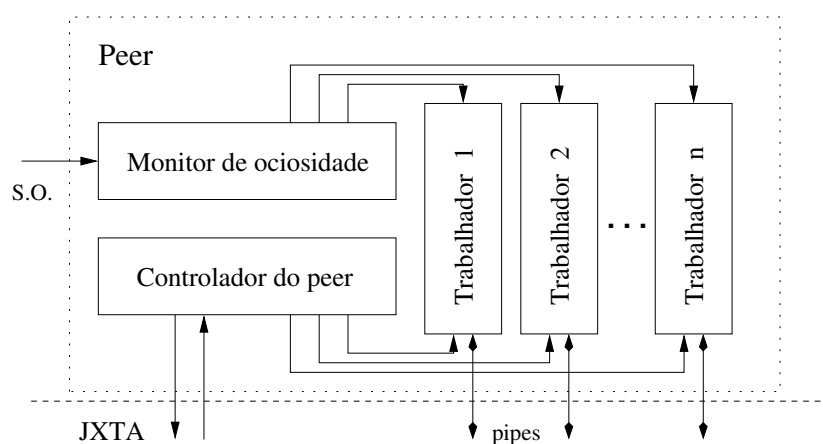


Figura 1. Estrutura de um *peer*.

A seguir descreve-se em maiores detalhes a composição e o funcionamento de cada um destes elementos.

4.2.1. Monitor de Ociosidade

O monitor de ociosidade é responsável por fornecer informações acerca da utilização dos recursos do *peer*. Isto é importante pois permite que trabalhos sejam aceitos somente quando os recursos do *peer* estiverem ociosos.

A monitoração para aproveitar períodos ociosos é um assunto complexo, que envolve muitos problemas, principalmente quando há a necessidade de não perturbar as atividades do usuário [Mathias 2005]. Não é objetivo deste trabalho resolver estes problemas. Desta forma, optou-se por monitorar ociosidade somente para guiar a aceitação de trabalhos e utilizar arquivos de configuração para informar períodos que os *peers* podem funcionar sem interferir nas atividades dos usuário.

4.2.2. Trabalhador

O trabalhador é responsável por executar o trabalho que lhe é submetido. Cada trabalhador possui um identificador único e visível a todos os *peers* da rede. Vários trabalhadores em um mesmo *peer* podem trabalhar de forma concorrente. Isto pode ser útil,

principalmente em máquinas que possuem mais de uma unidade de processamento. Trabalhadores locais e remotos podem se comunicar através da abertura de canais (*pipes*) de comunicação.

Um trabalhador pode estar em diferentes estados: livre, quando não está executando nenhum trabalho no momento; ocupado, quando já está processando algum trabalho submetido anteriormente; e indisponível, quando o trabalhador ainda faz parte da rede virtual mas por algum motivo não pode ser mais acessado, possivelmente porque seu *peer* está desligado. O fato de um trabalhador estar em estado livre não significa que ele irá aceitar um trabalho. A aceitação do trabalho também depende do estado do *peer* local, isto é, se está ocioso ou não.

4.2.3. Controlador do Peer

O controlador do *peer* é responsável por manter a plataforma de execução, isto é, realizar a manutenção da lista de *peers* ativos e monitorar a disponibilidade dos trabalhadores. Desta forma, todos os *peers* possuem listas de referências para os trabalhadores livres, ocupados e indisponíveis, podendo abrir conexões para troca de mensagens ou enviar tarefas para os trabalhadores livres.

Também é tarefa do controlador receber conexões de clientes e tratar as submissões de trabalhos. Quando uma submissão é recebida, o controlador atua como escalonador e realiza a distribuição e coordenação das tarefas.

4.3. Programa Cliente e Funcionamento do Sistema

O usuário interage com o sistema através de um programa cliente. Para submeter um trabalho, o programa cliente se conecta diretamente a um *peer* qualquer da rede virtual, por exemplo o *peer* local. O controlador do *peer* que recebeu a submissão do cliente realiza o escalonamento das tarefas na lista de trabalhadores livres da rede.

Quando um *peer* é iniciado, o controlador tenta descobrir se já existe um grupo de *peers* que formam o sistema. Se o grupo não existir, o *peer* cria o grupo tornando-se seu proprietário. Após se obter o grupo, o *peer* tenta ingressar no mesmo para participar da rede virtual.

Assim que o *peer* começa a fazer parte da rede virtual, inicia-se o processo de descoberta de recursos de JXTA. Após algum tempo, todos os *peers* já possuem as listas de referências para os trabalhadores disponíveis na rede e seus respectivos canais de comunicação. Nesse ponto os *peers* já estão inicializados e prontos para receber requisições de clientes.

Ao submeter um trabalho para a plataforma de execução P2P, o cliente envia uma requisição para um *peer* qualquer da rede (possivelmente o *peer* local). Nessa requisição estão informações sobre o trabalho que deve ser realizado, isto é, o programa (código nativo) e conjunto de dados (arquivo de entrada) que devem ser processados. De posse dessas informações, o controlador do *peer* inicia o processo de escalonamento.

5. Adaptação da Aplicação

De posse do sistema apresentado na seção anterior, foi possível paralelizar a aplicação alvo e adaptá-la para fazer uso deste sistema. Como o código da aplicação original foi

escrito em linguagem Fortran 77, optou-se por utilizar JNI (*Java Native Interface*) para lançar a aplicação no sistema implementado. Essa abordagem permite a conservação do código original, podendo ser facilmente alterado por seus desenvolvedores, sempre que necessário. Além disso, é comum que linguagens compiladas, como Fortran, obtenham um desempenho superior ao de linguagens interpretadas, como Java.

A paralelização da aplicação consistiu na sua divisão em tarefas que são distribuídas entre os trabalhadores. Como foi mencionado na seção 3, o processamento do arquivo é realizado em janelas. Com base nisso, optou-se decompor os dados de entrada em várias janelas. Logo, o primeiro passo da paralelização foi modificar o código original de modo a permitir o processamento de uma única janela de dados, especificada por um parâmetro.

Para realizar o mapeamento das tarefas sobre os computadores disponíveis foi necessário fazer algumas adaptações na plataforma de execução, de modo a incluir um escalonador. Foi implementado um escalonador simples que realiza a distribuição das tarefas aos trabalhadores livres. Este escalonador mantém uma lista de tarefas, que são enviadas para cada trabalhador da lista trabalhadores de livres. Quando um trabalhador termina de executar uma tarefa, este retorna para a fila de trabalhadores livres. Então, o escalonador lhe atribui uma nova tarefa.

O lançamento da aplicação é feito através do programa cliente. Após uma tarefa ser aceita por um trabalhador, a biblioteca contendo o código nativo responsável por processar uma tarefa é carregada e a rotina principal é executada sobre a janela de dados recebida. Assim que todas as tarefas tenham sido processadas, o cliente recebe um arquivo contendo os resultados do processamento.

6. Avaliação

Esta seção apresenta uma avaliação do sistema desenvolvido sobre a plataforma JXTA, bem como da paralelização da aplicação alvo. Como ambiente de execução, utilizou-se 5 máquinas mono-processadas Intel Pentium IV a 2.40 GHz, com 512 Mbytes de memória RAM, 512 Kbytes de memória *cache* para cada processador e adaptador de rede Gigabit Ethernet.

6.1. Avaliação do Sistema

A fim de avaliar a sobrecarga imposta pela plataforma de execução, mediu-se os tempos de processamento das seguintes operações: inicialização de um *peer*, envio e recepção de arquivos de dados e execução seqüencial da aplicação sobre o sistema desenvolvido. Há que se ressaltar que não é objetivo deste trabalho fazer uma análise completa de desempenho da plataforma JXTA, mas sim de sua influência sobre o sistema implementado.

O tempo médio de inicialização de um *peer*, na rede em questão, foi de 23,32 segundos em 300 execuções, com um desvio padrão de 4,77. O tempo de inicialização é altamente dependente da plataforma JXTA e, desta forma, do comportamento da rede. Isso é evidenciado pelo desvio padrão obtido, confirmando que o tempo de inicialização pode variar bastante a cada execução.

No que concerne o envio e recepção de arquivos, obteve-se um tempo médio de 14,58 segundos para transmissão do arquivo de dados descrito na seção 3. Como se pode

notar, o custo de transmissão é pouco significativo se comparado com o tempo total de execução da aplicação.

Os testes de execução da aplicação alvo, através da chamada de métodos nativos, mostraram um aumento significativo no tempo de execução. De fato, a aplicação original executava durante 24,04 minutos (conforme testes descritos na seção 3). Executando-se o mesmo código sobre o sistema desenvolvido, obteve-se um tempo médio de execução de 30,42 minutos. Acredita-se que esse aumento seja decorrente da utilização de JNI, já que toda a alocação de memória é realizada pela JVM (*Java Virtual Machine*).

6.2. Avaliação da Aplicação

Para avaliar a aplicação, foram realizados experimentos em uma rede P2P formada pelos computadores descritos na seção 6.1. A aplicação foi dividida em tarefas (ver seção 5), que foram escalonadas sobre os trabalhadores da rede. A tabela 1 contém os tempos totais de execução (em minutos), variando-se o número de trabalhadores da rede.

Tabela 1. Execução paralela variando-se os trabalhadores (processadores).

Número de trabalhadores	Tempo médio
1	30,07
2	15,46
3	10,45
4	8,08
5	6,06

Comparando-se a execução da aplicação, utilizando-se apenas um trabalhador, com a execução apresentada na seção 6.1, nota-se uma diminuição no tempo de execução. Esta diminuição deve-se à eliminação das leituras desnecessárias no arquivo, realizadas durante o processo de paralelização descrito na seção 5.

No gráfico presente na figura 2 é possível verificar a aceleração (*speedup*) da aplicação paralela no ambiente de execução descrito. Conforme se pode observar neste gráfico, à medida que o número de trabalhadores (processadores) aumenta, o *speedup* também aumenta.

7. Conclusão

Este trabalho apresentou o uso de uma abordagem P2P para paralelização de uma aplicação que analisa grandes conjuntos de dados coletados por sensores micrometeorológicos. A paralelização aumentou a eficiência da aplicação, diminuindo o tempo necessário para processar conjuntos de dados micrometeorológicos.

Outra contribuição deste trabalho foi o desenvolvimento de um sistema que faz uso de uma rede P2P como uma plataforma dinâmica para execução da aplicação em questão. Esse sistema permite a formação de um ambiente de execução que agrega recursos não-dedicados, podendo, no futuro, ser aproveitado para executar outras aplicações paralelas e distribuídas. Além disso, a utilização de um detector de ociosidade pode permitir uma melhor utilização dos recursos disponíveis.

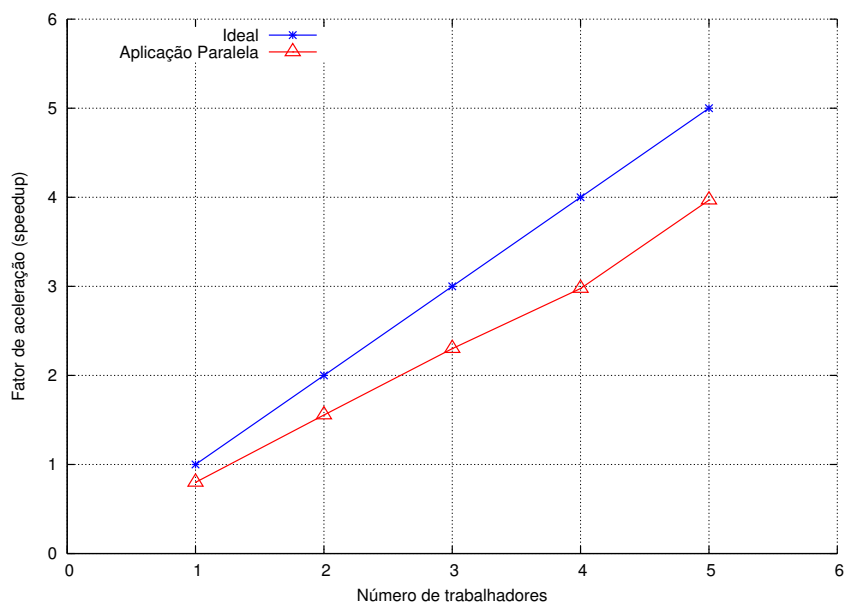


Figura 2. Fator de aceleração (*speedup*) da aplicação paralela.

Este trabalho pode ser aprimorado e servir como base para outros trabalhos, sendo que inicialmente pretende-se desenvolver uma interface mais amigável e introduzir tolerância a falhas, para melhor atender a seus usuários. Além disso, pretende-se avaliar o impacto do detector de ociosidade e realizar testes em redes com um número maior de *peers* para avaliar a escalabilidade do sistema.

Referências

- Antoniou, G., Bougé, L., and Jan, M. (2005). JuxMem: An Adaptive Supportive Platform for Data Sharing on the Grid. *Scalable Computing: Practice and Experience*, 6(3):45–55.
- Cirne, W. et al. (2003). The OurGrid Project: Running Bag-of-Tasks Applications on Computational Grids. In *SC'2003 Conference CD*. IEEE/ACM SIGARCH.
- JXTA (2001). Project JXTA Web site. <http://www.jxta.org/>. Acesso em abril de 2006.
- Mathias, E. N. (2005). Um Detector de Ociosidade para Sistemas Distribuídos Baseado em Modelos Matemáticos de Predição e de Descoberta de Padrões. Technical report, Laboratório de Sistemas de Computação, Universidade Federal de Santa Maria.
- Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2002). Peer-to-Peer Computing. Technical Report HPL-2002-57, Hewlett-Packard Labs. <http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html>.
- Schollmeier, R. (2001). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Peer-to-Peer Computing*, pages 101–102.
- Shudo, K., Tanaka, Y., and Sekiguchi, S. (2005). P3: P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources. In *Proc. Cluster Computing and the Grid 2005 (CCGrid 2005, Fifth Int'l Workshop on Global and Peer-to-Peer Computing)*, Cardiff, United Kingdom.