UFSM

Trabalho de Graduação

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

Marcelo Veiga Neves

Curso de Ciência da Computação

Santa Maria, RS, Brasil

2006

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

por

Marcelo Veiga Neves

Trabalho de Graduação apresentado ao Curso de Ciência da Computação – Bacharelado, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Trabalho de Graduação nº 208

Santa Maria, RS, Brasil

2006

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Graduação

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

elaborado por

Marcelo Veiga Neves

como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora:

Andrea Schwertner Charão
(Orientador)

Benhur Stein

Iara Augustin

Santa Maria, 6 de Janeiro de 2006.

I love deadlines. I like the whooshing sound they make as they fly by.

(Douglas Adams)

"Brindo à casa

Brindo à vida

Meus amores

Minha família".

Agradecimentos

A quantidade de agradecimentos tem uma tendência a ser proporcional ao tempo do percurso. Confesso que o meu percurso não foi dos maiores, mesmo assim tenho muito a agradecer. Acredito que junto aos agradecimentos devem vir algumas desculpas. Em especial, começo me desculpando com aqueles a quem eu talvez esqueça de agradecer.

À minha familia: minha mãe, minhas irmãs e minha sobrinha. Sem elas nada teria acontecido, nenhum esforço científico teria fruto caso não houvesse o auxílio e o aconchego profundo do amor que essas pessoas me dão. Em especial a minha mãe Icleia, pelo esforço em me proporcionar uma boa educação e formação.

À Andrea Charão, minha orientadora, por oferecer os meios e o apoio necessário durante a realização do trabalho e por sempre definir *deadlines* imaginários (única forma de me fazer escrever). Também a agradeço por ter sido, e continuar sendo, aquela que mostra o caminho, que incentiva, que se preocupa, que critica os erros e elogia os acertos. Obrigado pela confiança e, principalmente pela paciência ;-).

Aos professores do curso de Ciência da Computação pela contribuição no processo de formação e aos professores Benhur Stein e Iara Augustin por aceitarem fazer parte da banca examinadora e contribuírem para esse trabalho.

À Ju, minha namorada, pelo apoio e compreensão, mesmo nos momentos mais conturbados deste trabalho em que eu, praticamente, morava no laboratório.

Ao meu computador gandalf, por ter ficado sempre ao meu lado (ou melhor, na minha frente) e me ajudado na realização deste trabalho.

Ao Dr. Albert Hofmann por ter descoberto, em 1937, o ácido lisérgico dietilamida, substância que ficou famosa na década de 60 e influenciou o processo criativo de muitas bandas. Essas bandas criaram músicas mágicas que serviram de trilha sonora para esse trabalho e como anestésico em algumas madrugadas no laboratório.

A todos os colegas da graduação, que tornaram a minha graduação um pouco

menos chata. Em especial ao Rafael e ao Rubens, parceiros em várias cervejadas e festas que fizemos juntos.

E por último, mas não menos importante, aos integrantes do grupo LSC (Elton, Edmar, Geovani e Scheid) pela parceira durante toda a graduação. Ao colega Elton, pelas conversas viajadas de onde saíram várias "idéias geniais". Ao Edmar, pelas jogatinas e cervejadas. E também por ensinar a manter a calma quando um *deadline* se aproxima. Ao Geovani, por ensinar que as vezes devemos ser um pouco responsáveis também. Ao Scheid, por estar sempre pronto para uma festa (sempre mesmo) e pela sua grande ajuda neste e em vários outros trabalhos que fizemos junto no LSC.

:wq

Sumário

Lista de Tabelas

Lista de Figuras

RESUMO

Trabalho de Graduação
Ciência da Computação
Universidade Federal de Santa Maria

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

Autor: Marcelo Veiga Neves
Orientador: Andrea Schwertner Charão
Data e Local da Defesa: Santa Maria, 6 de Janeiro de 2006.

O processamento paralelo e distribuído é freqüentemente utilizado para resolver problemas que demandam uma grande poder computacional. Atualmente, é crescente a utilização de plataformas não-dedicadas, como redes de estações de trabalho, para o processamento paralelo e distribuído. Neste contexto, a computação *peer-to-peer* surge como uma alternativa, permitindo trabalhar com ambientes dinâmicos e aproveitar períodos ociosos. Baseado nisso, este trabalho se propõe a paralelizar uma aplicação de análise de dados meteorológicos utilizando um abordagem *peer-to-peer*. Esta aplicação é utilizada pelo Laboratório de Micrometeorologia da UFSM para processar dados coletados por sensores micrometeorológicos. Para isso, foi desenvolvido um sistema que disponibiliza um plataforma dinâmica de execução e permite a distribuição e coordenação de tarefas em ambientes não-dedicados. Além disso, o sistema permite aproveitar recursos ociosos, promovendo assim um uso mais eficiente dos recursos computacionais disponíveis. A partir da distribuição e paralelização da aplicação, torna-se possível processar um conjunto maior de dados em um tempo mais curto.

Capítulo 1

Introdução

O processamento paralelo e distribuído é freqüentemente 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.

Existem diversas arquiteturas que permitem executar programas paralelos e distribuídos. Um exemplo são os aglomerados de computadores (*clusters*), que oferecem uma boa relação custo-desempenho [BAK 99] e se tornaram bastante populares. Em *clusters* os nós são tipicamente idênticos e dedicados ao processamento de aplicações paralelas. Outra possibilidade é a utilização de plataformas não-dedicadas, como múltiplas estações de trabalho interligadas em rede.

Ao longo do tempo, surgiram várias abordagens e ferramentas para a programação paralela e distribuída, sendo geralmente voltadas a um tipo de plataforma de execução. Por exemplo, o uso do padrão MPI (*Message Passing Interface*) [GRO 96] para troca de mensagens é uma opção eficiente e portável para programação em *clusters*. No entanto, em ambientes não-dedicados e com configurações possivelmente heterogêneas, esta alternativa apresenta limitações, tais como a degradação da eficiência e a falta de suporte à dinamicidade da plataforma de execução.

Dentre outras alternativas que se apresentam para construção de aplicações paralelas e distribuídas, destacam-se os sistemas *peer-to-peer* (P2P). Esta abordagem permite trabalhar com ambientes dinâmicos, configurações heterogêneas e possivelmente aproveitar o tempo ocioso de estações de trabalho para a execução de aplicações paralelas e distribuídas. Embora já existam diversas ferramentas que auxiliam no desenvolvimento de aplicações P2P, esta área ainda é tema de muitas pesquisas.

Neste contexto, este trabalho se propõe a paralelizar uma aplicação de análise de dados meteorológicos, utilizando uma abordagem P2P. A aplicação é utilizada para o processamento de grandes conjuntos de dados que são coletados por sensores do Laboratório de Micrometeorologia da UFSM (LμMet) [LUM 2005]. O laboratório LμMet possui, como ambiente de execução, um *cluster* de computadores e uma rede de estações de trabalho formada por máquinas heterogêneas e não necessariamente dedicadas. Desta forma, este trabalho visa, ao mesmo tempo, obter ganho de desempenho, permitindo o aproveitamento dos recursos computacionais ociosos e, também, contribuir para a consolidação das pesquisas em torno da aplicabilidade de sistemas P2P.

O texto deste trabalho está dividido em capítulos. Primeiramente, é apresentada uma revisão de literatura (capítulo 2), que mostra as abordagens para computação paralela e distribuída, focalizando a computação P2P. Esse capítulo também apresenta uma comparação entre infra-estruturas para desenvolvimento de aplicações P2P. Logo após, o capítulo 3 descreve a plataforma JXTA, que foi utilizada na implementação deste trabalho. O capítulo 4 apresenta o sistema desenvolvido, bem como a aplicação que foi paralelizada. No capítulo 5, encontra-se a descrição dos testes e a análise de resultados sobre a implementação. Para finalizar, o capítulo 6 reúne as principais idéias e resultados obtidos, e também expõe os possíveis trabalhos futuros.

Capítulo 2

Abordagens para Computação Paralela e Distribuída

2.1 Plataformas de Execução Paralela e Distribuída

Por se tratar de uma solução economicamente atrativa, as arquiteturas paralelas com memória distribuída têm evoluído constantemente. Essa evolução pode ser vista a partir de plataformas como:

- **Aglomerados de computadores** (*Clusters*): são o tipo de arquitetura voltada à computação paralela mais difundida atualmente. Um *cluster* consiste de um conjunto de computadores independentes (nós) interligados por uma rede de interconexão de alta velocidade e que oferecem uma visão única do sistema [BAK 99]. Em geral, os computadores que integram um aglomerado são dedicados ao processamento paralelo e possuem a mesma arquitetura. Porém, a utilização de aglomerados compostos por recursos heterogêneos vem aumentando e sendo alvo de muitos estudos [KRE 2004].

- **Redes de estações de trabalho** (*Networks of Workstations*) [POL 96]: são uma opção de arquitetura paralela com memória distribuída. As redes de estações de trabalho são compostas por computadores tradicionais interligados por uma tecnologia de rede tradicional [ROS 2003]. A idéia em sua concepção propõe criar máquinas paralelas simplesmente através do uso da programação simultânea e ordenada de computadores pessoais (estações de trabalho), usando a rede que já os interligava. Estas máquinas não necessariamente são dedicadas ao processamento paralelo.

- **Grades de computadores** (*Grids*) [FOS 99] são a mais recente alternativa de arquitetura para execução de programas paralelos. Os *grids* podem ser caracterizadas como um conjunto de sistemas geograficamente distribuídos, interligados por uma rede de larga escala. Essa rede pode ser a própria Internet ou uma rede especial de alta velocidade para obter a largura de banda requerida por suas aplicações. Assim como as redes de estações de trabalho, os computadores participantes de um *grid* podem não ser dedicados às aplicações paralelas.

O modelo de programação paralela mais utilizado em arquiteturas com memória distribuída é o modelo com trocas de mensagens. Nesse modelo, as aplicações paralelas são compostas por um conjunto de processos distribuídos, entre os computadores da arquitetura paralela, com o auxílio de uma biblioteca de troca de mensagens. Em *clusters* de computadores é comum a utilização de bibliotecas de comunicação mais tradicionais, geralmente MPI. No entanto, em ambientes que possuem comportamento dinâmico o uso dessas bibliotecas pode não ser viável. Nesses casos, uma alternativa pode ser a computação P2P.

2.2 Computação *Peer-to-Peer*

O termo *peer-to-peer* (P2P) [MIL 2002] refere-se à classe de sistemas e aplicações que emprega recursos distribuídos — como poder computacional, capacidade de armazenamento ou largura de banda — para realizar uma operação crítica de um modo descentralizado. No contexto deste trabalho, essa operação crítica é o processamento distribuído.

Conceitualmente, a computação P2P é uma alternativa aos modelos centralizados e cliente-servidor. Na sua forma mais pura, o modelo P2P não utiliza servidores e todos os participantes são considerados *peers*.

Comparado com os modelos tradicionais (cliente-servidor), P2P oferece algumas vantagens provenientes da eliminação de servidores. A descentralização elimina pontos únicos de falhas e pode diminuir a ocorrência de gargalos. Além disso, pode-se alcançar uma maior escalabilidade, pois os *peers* possuem autonomia, o que diminui os custos de gerenciamento.

2.2.1 Infra-estruturas para Aplicações *Peer-to-Peer*

Aplicações P2P podem ser divididas em três grandes categorias de acordo com seu objetivo: para computação paralela, para compartilhamento de arquivos e para colaboração [MIL 2002]. Este trabalho analisou algumas plataformas que fornecem uma infra-estrutura para o desenvolvimento e execução de aplicações distribuídas P2P. Optou-se por analisar as plataformas XtremWeb [FED 2000], BOINC [AND 2002], JXTA [KRI 2001] e ProActive P2P [CAR 98]. Estas plataformas foram escolhidas, principalmente, por serem *open source* e fazerem parte de projetos ativos na área de computação P2P.

Aplicações desenvolvidas utilizando XtremWeb e BOINC não conseguem trabalhar sem a presença do um servidor central. Por causa do grande controle exercido pelo servidor central, pode não ser apropriado classificar estas plataformas como plataformas para desenvolvimento de aplicações P2P [SIN 2004]. No entanto, estas plataformas foram incluídas nessa comparação por serem utilizado para a criação de aplicações como a do projeto SETI@home [KOR 2001], que é considerado P2P por muitos autores [MIL 2002, LOD 2003, LO 2004].

2.2.1.1 XtremWeb

XtremWeb é uma plataforma para computação P2P desenvolvida na Universidade de Paris-Sud, França [FED 2000]. Este sistema foi originalmente projetado para o estudo de modelos de execução para a Computação Global [CAR 97], mas tornou-se uma plataforma completa para execução de aplicações distribuídas P2P.

XtremWeb possui uma arquitetura que segue um modelo coordenador-trabalhador. Um nó coordenador gerencia um conjunto de tarefas (*bag of tasks*) e coordena o escalonamento destas sobre um conjunto de máquinas voluntárias (trabalhadores). Normalmente, uma ou mais aplicações são instaladas nos trabalhadores e clientes podem submeter tarefas ao coordenador. Nesse caso, as tarefas submetidas são registradas no coordenador para serem então escalonadas para os trabalhadores. Para ambientes de uso compartilhado, esse escalonamento pode utilizar apenas períodos de ociosidade dos trabalhadores.

Todos os trabalhadores são nós voluntários. Dessa forma, o coordenador não tem controle sobre os nós disponíveis, pois todas as ações e conexões são iniciadas pelos trabalhadores. Por isso, é possível trabalhar com ambientes bastante dinâmicos. Esse modelo é conhecido com *pull* e implica na independência de todos os

componentes, não sendo permitida a comunicação entre *peers*.

2.2.1.2 BOINC

BOINC (*Berkeley Open Infrastructure for Network Computing*) [AND 2002] é uma plataforma para computação distribuída que permite utilizar recursos públicos. BOINC foi desenvolvido pelo Laboratório de Ciências Espaciais de Berkeley, mesmo grupo desenvolveu e continua a operar o projeto SETI@home. BOINC tem como principal objetivo avançar o paradigma de computação que utiliza recursos públicos, e encorajar uma boa parte dos usuários de computador do mundo a disponibilizar o tempo ocioso de seus computadores para um ou mais projetos de pesquisa.

BOINC possui uma arquitetura cliente-servidor. O servidor é centrado em um banco de dados relacional, o qual armazena a descrição das aplicações e todas as informações necessárias para o seu funcionamento. As funções do servidor são executadas por um conjunto de *web services* e processos *daemons*, que juntos realizam o escalonamento de tarefas, atendimento aos clientes via RPC, carga de arquivos e recebimento dos resultado final da computação. Já os clientes são responsáveis por executar a aplicação no computador voluntário, quando o mesmo estiver ocioso.

A geração de unidades de trabalho fica sob responsabilidade do desenvolvedor de aplicações. Para isso, BOINC oferece uma API para criação de unidades de trabalho no banco de dados. É importante mencionar que BOINC não possui suporte para comunicação entre clientes.

2.2.1.3 JXTA

JXTA é uma plataforma para desenvolvimento de aplicações distribuídas projetada pela Sun Microsystems e desenvolvida com o auxílio de alguns especialistas de instituições acadêmicas e industriais [JXT 2005]. Esta plataforma tem como objetivo resolver problemas de computação P2P e, ao mesmo tempo, alcançar características como interoperabilidade e portabilidade.

JXTA consiste em um conjunto de protocolos, onde cada um deles é definido pela troca de algumas mensagens entre os participantes, sendo que cada mensagem tem um formato pré-definido. Estes protocolos definem, por exemplo, a busca por novos nós, busca por recursos, busca por informações sobre os *peers* e seus estados, criação de grupos de *peers*, etc.

JXTA não implementa nenhum tipo de sistema para computação distribuída.

No entanto, os protocolos definidos por JXTA permitem a criação um rede virtual de *peers* que abstrai a rede física. Eles possibilitam que os *peers* possam trocar mensagens independentemente da sua localização real, roteando as mensagens de forma transparente entre nós cercados por *firewalls* ou que utilizam protocolos de comunicação diferentes. JXTA também permite que os *peers* se comuniquem sem conhecer ou precisar gerenciar mudanças físicas na topologia da rede, garantindo que redes com comportamentos dinâmicos possam ser utilizadas de forma transparente.

2.2.1.4 ProActive P2P

ProActive é um *middleware* que busca oferecer um modelo de programação concorrente e distribuída com transparência [CAR 98]. Esse *middleware* está sendo desenvolvido em um esforço conjunto entre o grupo de pesquisa Oasis, da Université de Nice Sophia Antipolis e o instituto francês INRIA (*Institute National de Recherche en Informatique et en Automatique*), sendo parte do consórcio europeu ObjectWeb. ProActive oferece uma biblioteca para computação distribuída e paralela em Java, um ambiente gráfico de monitoração e lançamento de tarefas e uma infra-estrutura P2P.

O objetivo principal da infra-estrutura P2P de ProActive é a utilização transparente de ciclos de processadores disponíveis em estações de trabalho presentes em instituições, combinados com grades computacionais e aglomerados de computadores. O funcionamento do ambiente P2P é guiado por arquivos de configuração onde se determinam os horários de funcionamento dos *peers*, os protocolos de comunicação utilizados e a lista de *peers* conhecidos. A localização de *peers*, para a formação da rede P2P, é feita através de um algoritmo de busca BFS (*Breadth-First Search*), sendo que *peers* não acessíveis diretamente comunicam-se através de um mecanismo transparente de encaminhamento de mensagens.

Os mecanismos de busca de *peers* e redirecionamento de mensagens permitem a implementação de aplicativos sem preocupações relativas à localização ou estrutura física na qual a rede P2P organiza-se. Entretanto, é obrigatória a criação de arquivos de configuração para cada *peer* que compõe a rede, o que pode ser uma tarefa complexa, especialmente em ambientes amplos.

2.2.2 Comparação

Comparando essas plataformas, pôde-se observar algumas particularidades relevantes. Em XtremWeb e BOINC, foi possível observar a existência de um servidor central, que gerencia a distribuição de tarefas, e a não possibilidade de comunicação entre os *peers*, o que inviabiliza a construção de aplicações que possuam dependências de dados. Em ProActive P2P, notou-se a existência de uma interface gráfica para o lançamento de tarefas e possibilidade de integração com outros sistemas bastante difundidos como Globus [FOS 97] e Ibis [NIE 2002]; no entanto, observou-se a necessidade de programar com o modelo de objetos ativos, utilizado pelo *middleware* ProActive. Em JXTA, percebeu-se que o mesmo não implementa algoritmos de escalonamento e distribuição de tarefas; no entanto, oferece uma liberdade maior ao desenvolvedor da aplicação através de sua API.

Com base nos resultados dessa comparação, escolheu-se a plataforma JXTA para a implementação deste trabalho. O próximo capítulo descreve essa plataforma em maiores detalhes.

Capítulo 3

A Plataforma JXTA

A idéia básica de JXTA é formar uma camada de rede virtual, independente da rede física utilizada pelos dispositivos, provendo transparência de acesso aos participantes. Funcionalidades e serviços comuns são encapsulados para esconder a complexidade das implementações e facilitar o desenvolvimento de aplicativos P2P [JXT 2005].

O projeto JXTA define algumas abstrações de rede que permitem a construção de uma rede virtual P2P, representada pela figura 3.1. Uma rede virtual provê transparência quanto à localização dos recursos através da atribuição de endereços lógicos. Os *peers* da rede virtual se auto-organizam em grupos, chamados *peer groups*. Todos os recursos em uma rede virtual JXTA são publicados através de uma representação chamada *advertisements*. Já a comunicação é feita por troca de mensagens através de canais (*pipes*) que podem ser abertos entre dois *peers*.

Este capítulo descreve as principais abstrações e componentes de uma rede virtual JXTA.

3.1 Organização da Rede

Uma rede virtual JXTA é composta por *peers*. Cada *peer* opera de forma independente e assincrona em relação a outros *peers*, e é unicamente identificado por seu *peerID*.

Peer é qualquer entidade que possa participar e interagir com a rede, seja ela um computador, um processo ou até mesmo um usuário. Para ser considerado um *peer* essa entidade deve ao menos entender os protocolos de comunicação básicos definidos pela tecnologia, como o *Peer Resolver Protocol* e *EndPoint Router Protocol*,

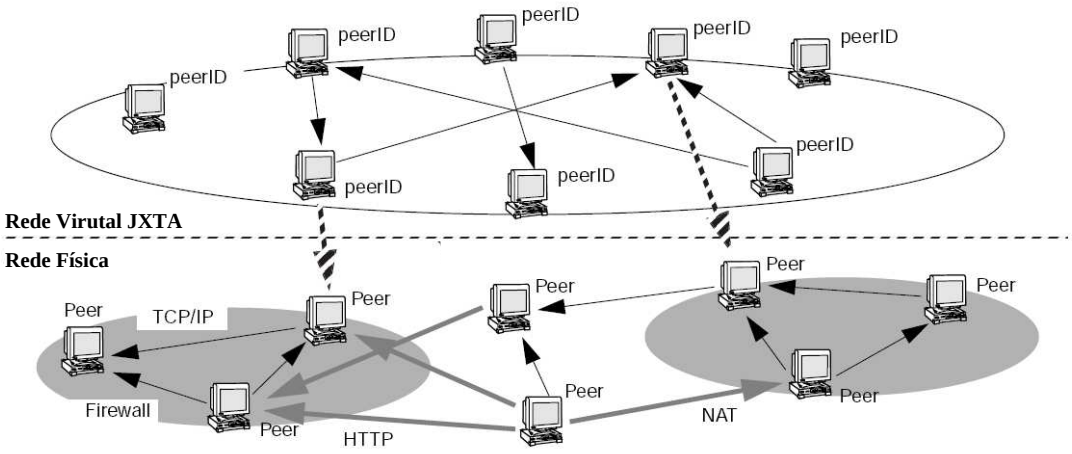

Figura 3.1: Rede Virtual JXTA. Fonte: [TRA 2003], figura adaptada.

explicados mais adiante neste capítulo.

A organização da rede entre os *peers* é feita por grupos, chamados *peer groups*. Cada *peer group* é identificado unicamente por um *peerGroupID*. Quaisquer *peers* podem fazer parte de um mesmo grupo, independente de sua localização física. A figura 3.2 ilustra essa organização. Quando um *peer* é inicializado, ele ingressa (*join*) no *NetPeerGroup*, que é o grupo raíz do projeto JXTA, podendo, posteriormente, participar de outros grupos, desde que tenha permissão para isso.

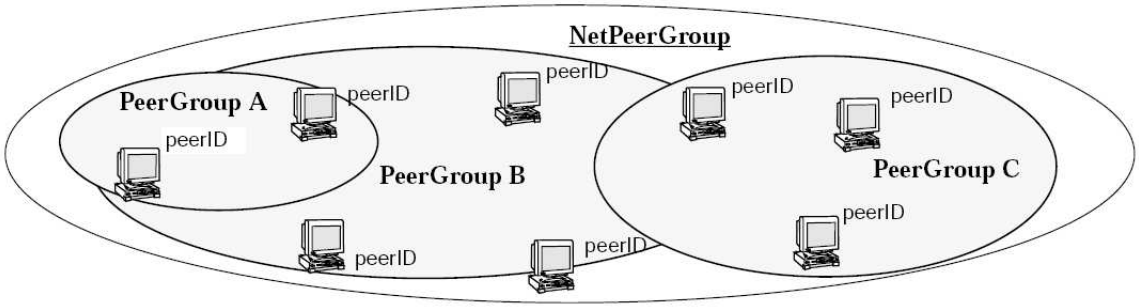

Figura 3.2: Organização da rede em *peer groups*. Fonte: [TRA 2003].

3.2 Descoberta de Recursos

JXTA utiliza um modelo de endereçamento lógico uniforme e independente de localização. Todos os recursos de rede (*peer*, *pipe*, *peer group*, serviços, dados, etc.)

são associados a um JXTA ID único. JXTA IDs são objetos abstratos que permitem múltiplas representações, como IPv6 e MAC, para coexistir na mesma rede JXTA. A implementação atual utiliza identificadores aleatórios de 128 bits, permitindo que cada *peer* possa gerar seus próprios IDs únicos.

Os recursos em uma rede JXTA são representados por *advertisements*. *Advertisements* são descritores de recursos em uma linguagem neutra representados como documentos XML. A figura 3.3 apresenta um *advertisement* que descreve um *peer group*. O campo GID contém o ID do grupo, MSID contém a referência para um outro *advertisement* que descreve todos os serviços do grupo, Name contém o nome e Desc a sua descrição.

```
<jxta:PGA xmlns:jxta=http://jxta.org>
    <GID> urn:jxta:jxta−NetGroup </Id>
    <MSID> urn:jxta:uuid−DEADBEEFDEAFBABAFEEDBABE000000010206 </MSID>
    <Name> NetPeerGroup </Name>
    <Desc> NetPeerGroup by default </Desc>
</jxta:PGA>
```

Figura 3.3: *Advertisement* de um *peer group*.

Todas as resoluções em uma rede JXTA são realizadas por *rendezvous peers*, que são responsáveis pela descoberta dos *advertisements*. Os protocolos do projeto JXTA não especificam como a busca por *advertisements* deve proceder, mas provê um *framework* de um protocolo de resolução genérico que deve ser implementado. *Rendezvous peers* mantém em suas *caches* locais os índices dos *advertisement*, que são referências para o *peer* que tem o *advertisement* em sua *cache*.

3.3 Comunicação entre *Peers*

A comunicação entre *peers* é realizada através de *pipes*, que são canais para envio e recebimento de mensagens entre serviços e aplicações. *Pipes* disponibilizam uma abstração que provê caixas de entrada e saída, não necessitando, assim, conhecer a localização física do outro *peer*. *Pipes* podem ser anunciados e descobertos através de *advertisements*, que são identificados unicamente pelo seu ID.

As mensagens enviadas através de *pipes* são chamadas JXTA *messages*. Uma JXTA *message* é um conjunto de elementos com nome e conteúdo, sendo que o conteúdo pode ser de qualquer tipo. O formato dessas mensagens pode ser em binário ou XML. O primeiro possui um maior desempenho, enquanto o segundo

garante portabilidade, pois possibilita uma representação de dados única.

3.4 Roteamento de Mensagens

Um *peer* sempre tenta conectar-se diretamente ao seu destino, caso não consiga, ele usa um *relay peer*. *Relay peers* são responsáveis por fazer a ponte entre *peers* que não têm acesso direto entre si (*firewall*, NAT). Desta forma, *Relay peers* possibilitam um *peer* enviar mensagens para outro *peer* inacessível a ele ou temporariamente indisponível.

O *advertisement* de um *peer* contém uma lista de *relay peers* preferenciais para ajudar na resolução de rotas. Quando uma conexão direta não é possível, um *relay peer* é usado, e este por sua vez repete o processo. Deste modo, uma mensagem pode passar por mais de um *relay peer*, dependendo da configuração da rede.

3.5 Protocolos JXTA

JXTA implementa vários protocolos que permitem a organização da rede virtual. Eles são descritos brevemente abaixo.

- *Peer Discovery Protocol* (PDP): é o protocolo no qual um *peer* publica seu próprio *advertisement* ou descobre os *advertisements* de outros *peers*.

- *Peer Resolver Protocol* (PRP): é o protocolo usado para enviar uma consulta de resolução genérica para um ou mais *peers* e receber uma, ou várias, respostas para essa consulta.

- *Peer Information Protocol* (PIP): é o protocolo pelo qual um *peer* pode obter informação de estado de um outro *peer*, como *uptime*, carga de tráfego, capacidades, etc.

- *Rendezvous Protocol* (RVP): é o protocolo pelo qual os *peers* podem se inscrever para um serviço de propagação. Em um grupo, um *peer* pode tornar-se *rendezvous peer* ou ouvinte de um *rendezvous peer*. RVP permite mandar mensagem para todos os ouvintes do serviço.

- *Endpoint Routing Protocol* (ERP): é o protocolo pelo qual um *peer* pode descobrir a rota usada para enviar mensagens a um outro *peer*. Caso uma rota

não seja mais possível por ausência de algum *peer*, pode-se usar o ERP para determinar uma nova rota.

Capítulo 4

Sistema Desenvolvido

Este capítulo apresenta o desenvolvimento de um sistema que disponibiliza uma plataforma dinâmica de execução e permite a distribuição e coordenação de tarefas em ambientes não-dedicados. Este sistema foi desenvolvido utilizando uma abordagem P2P e projetado para permitir a distribuição e paralelização de uma aplicação de análise de dados meteorológicos.

As seções seguintes apresentam a aplicação alvo, o sistema desenvolvido e a distribuição e paralelização desta aplicação.

4.1 Aplicação Alvo

A aplicação alvo consiste no processamento de grandes conjuntos de dados, que foram coletados por sensores micrometeorológicos do Laboratório LμMet. Estudos na área de micrometeorologia geralmente envolvem simulações para estudar processos físicos e experimentações através de coleta de informações. Estas experimentações demandam a análise e interpretação, com grande precisão, das observações feitas. O processamento dos dados coletados, geralmente, demanda um grande poder computacional. Desta forma, a paralelização da aplicação trará benefícios a seus usuários, já que tornará possível analisar os dados rapidamente e em uma vazão maior.

Micrometeorologia é a parte da meteorologia que estuda os fenômenos físicos que ocorrem em regiões limitadas da superfície da Terra, menor que 1 km^2, e em um curto período de tempo, menos que 1 dia [DOB 2003]. A física das baixas camadas da atmosfera é importante, pois grandes variações nas condições de tempo e clima são encontradas nessas camadas. Através do estudo das camadas próximas ao solo,

pode-se dizer que a micrometeorologia estuda os seguintes aspectos: turbulência, já que próximo ao solo o ar encontra alguns obstáculos; transferências de calor, campos de temperatura nas primeiras camadas, evaporação, radiação, poluição, etc.

A aplicação que foi paralelizada é normalmente utilizada para processar os dados coletados durante um dia. Os sensores coletam informações a 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. A posição das janelas no arquivo é calculada levando em conta um atraso (*lag*) de 30 minutos, como mostra a figura 4.1.

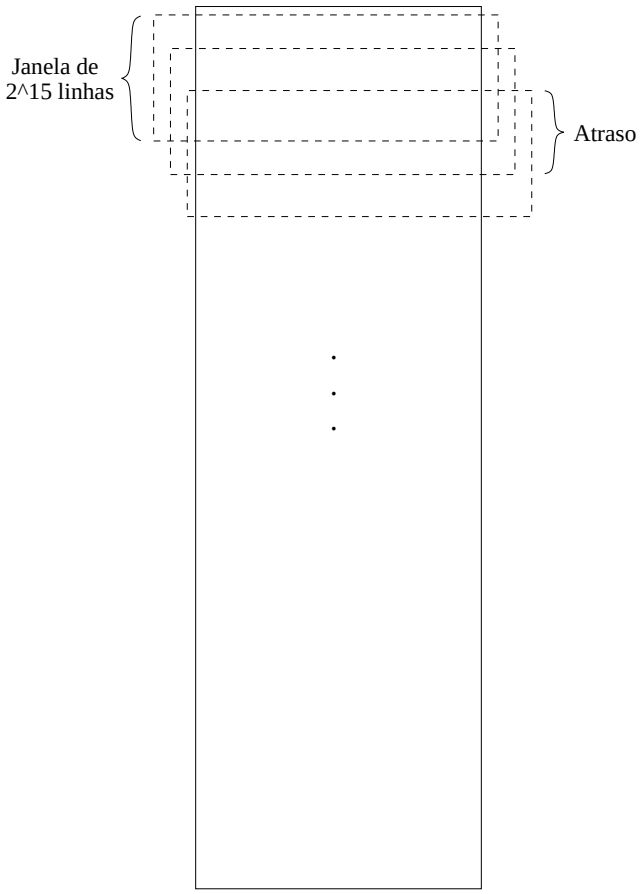

Figura 4.1: Arquivo de entrada divido em janelas.

No inicio da execução, as estruturas de dados são inicializadas, incluindo a matriz que contém os dados da janela, e o cálculo de alguns parâmetros micrometeorológicos é realizado, tais como direção do vento, componentes turbulentas e condições de estabilidade. Depois, algumas subrotinas realizam transformações

sobre a matriz.

A figura 4.2 mostra o fluxo de dados entre as subrotinas da aplicação. Primeiramente, é realizada uma rotação 3D [KAI 94], depois as tendências dos dados turbulentos são calculadas, seguido da janela Bell Taper [STU 88]. O próximo passo é o cálculo de algumas transformadas rápidas de Fourier (*Fast Fourier Transform*, ou FFT) [PRE 92] e o cálculo dos espectros (*Discrete Energy Spectrum*) [STU 88] para cada uma das colunas. Depois é chamada a rotina que realiza o cálculo das bandas para suavização dos espectros. No final, testa-se a qualidade dos espectros através do método dos mínimos quadrados e da rotina de autocorrelação.

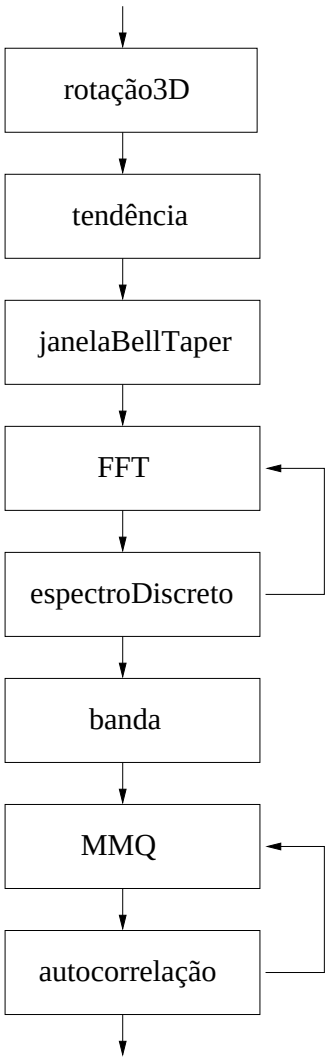

Figura 4.2: Fluxo de dados das subrotinas da aplicação.

Após o processamento de uma janela, os resultados da análise são salvos em

arquivos. Para um arquivo de entrada contendo as informações coletadas durante um dia, a aplicação gera aproximadamente 70 MBytes de resultados.

Testes realizados em uma máquina mono-processada 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. O sistema operacional utilizado nesses testes foi GNU/Linux (distribuição Gentoo 1.4), com *kernel* versão 2.6.12. Todo o código da aplicação é escrito em FORTRAN 77 e o compilador utilizada nos testes foi GNU Fortran (GCC) versão 3.3.6.

Embora seja útil, a execução dessa aplicação não é prioritária; então, não existem recursos dedicados ao seu processamento no laboratório LμMet. Desta forma, a primeira parte deste trabalho consiste na implementação de um sistema capaz de distribuir a aplicação entre os computadores que estejam disponíveis. A estrutura deste sistema será aproveitada posteriormente na paralelização da aplicação. A próxima seção apresenta uma solução P2P para esse problema.

4.2 Arquitetura do Sistema

Com o intuito de aproveitar os recursos do laboratório LμMet, foi implementado um sistema capaz de distribuir a aplicação entre os computadores que estiverem momentaneamente disponíveis. O restante desta seção descreve a implementação e funcionamento deste sistema.

4.2.1 Visão Geral

A arquitetura do sistema baseia-se em um modelo P2P puro [SCH 2001], onde os *peers* são os computadores que integram a plataforma de execução. Em um modelo P2P puro (figura 4.3), 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 da mesma rede virtual.

O sistema implementado oferece transparência quanto à localização dos *peers* que integram a plataforma de execução (rede virtual P2P). Os recursos necessários para a execução das tarefas, como códigos e dados (arquivos de entrada), podem estar distribuídos na rede e também são acessados de forma transparente. Essa

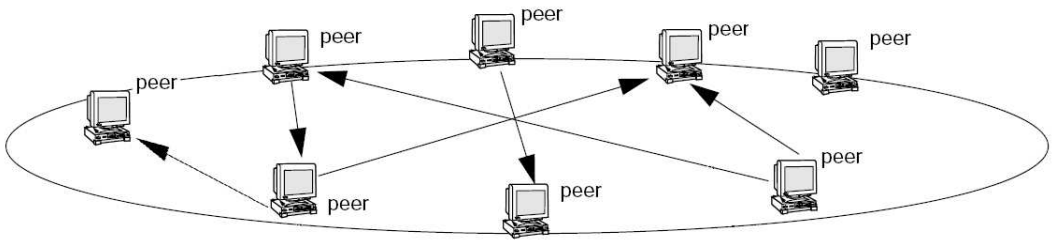

Figura 4.3: Rede baseada no modelo P2P puro. Fonte: [TRA 2003].

transparência é alcançada graças a descoberta de recursos de JXTA e será descrito na seção 4.2.5.

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 interferência 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.

4.2.2 Elementos de um Peer

Cada *peer* do sistema é estruturado conforme a figura 4.4. 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 e, se forem capazes, aceitam os mesmos. Um trabalhador aceita um trabalho somente se ele possuir todos os recursos necessários à sua execução e se seu *peer* estiver em estado ocioso. Os trabalhadores interagem com o monitor de ociosidade para descobrir se o *peer* local está em es-

tado ocioso. O controlador do *peer* realiza descobertas de recursos e mantém a rede virtual.

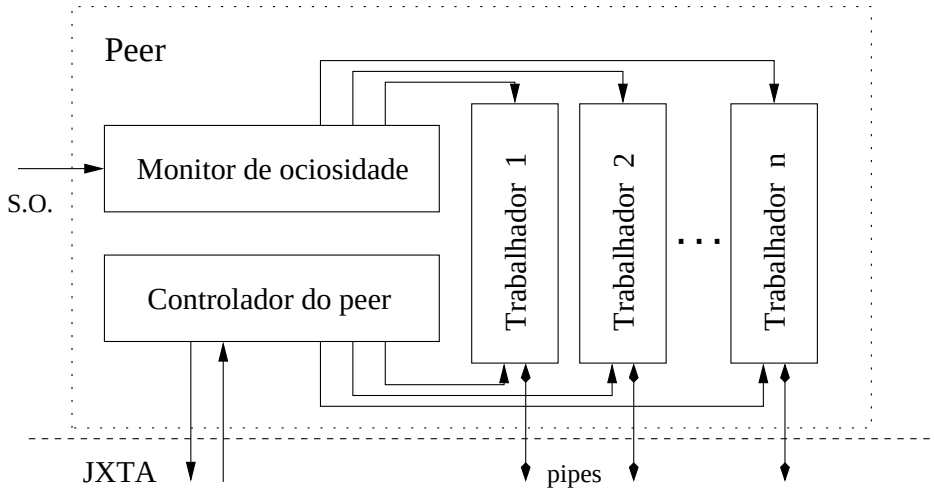

Figura 4.4: Estrutura de um *peer*.

As subseções seguintes descrevem em detalhes a composição e o funcionamento de cada um destes elementos.

4.2.2.1 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.

O problema da utilização de períodos disponíveis pode ser tratado de diferentes maneiras. Em ProActive P2P, os períodos de funcionamento do *peer* são definidos através de arquivos de configuração que determinam os horários de funcionamento dos *peers* (por exemplo, das 22:00 às 6:00 horas). Durante esses horários o sistema tem a permissão para utilizar os recursos da máquina como recursos dedicados. Já o sistema BOINC implementa uma proteção de tela (*screen saver*), que monitora, junto ao sistema operacional, as interrupções dos dispositivos de entrada, e entra em execução quando nenhuma atividade do usuário é detectada. XtremWeb utiliza outra abordagem, a monitoração de índices de cargas para detectar períodos de baixa utilização dos recursos.

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 [MAT 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.

A monitoração de ociosidade é realizada através da coleta e análise de índices de carga para verificar se o *peer* local está em estado ocioso. Esses índices são métricas que quantificam a carga submetida a um elemento do sistema [FER 88]. A inexistência de carga ou um valor muito pequeno identifica um sistema ocioso (por exemplo, utilização do tempo de CPU abaixo de 5%).

4.2.2.2 Trabalhador

O trabalhador (*worker*) 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 local e remotos podem se comunicar através da abertura de canais (*pipes*) de comunicação.

Um trabalhador pode estar em diferentes estados:

- livre: é um trabalhador que não está executando nenhum trabalho no momento, logo está pronto para receber submissões.
- ocupado: é um trabalhador que já está processando algum trabalho submetido anteriormente, então não pode aceitar novas submissões.
- indisponível: é um trabalhador que ainda faz parte da rede virtual mas por algum motivo não pode mais ser 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. O trabalhador utiliza as informações coletadas pelo monitor de ociosidade para decidir se seu *peer* é capaz de aceitar um trabalho sem comprometer as atividades do proprietário do computador.

4.2.2.3 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 e podem 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.

4.2.3 Cliente

O usuário interage com o sistema através de um programa cliente. Para submeter uma trabalho, o programa cliente se conecta diretamente a um *peer* qualquer da rede virtual, por exemplo o *peer* local. O *peer* que recebeu a submissão do cliente propaga a mesma para a lista de trabalhadores livres que possuem os recursos necessários para sua execução.

4.2.4 Funcionamento

Quando um *peer* é iniciado, o controlador tenta descobrir se já existe um *peer group* para 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 (*join*) 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, qual o programa (código nativo) e conjunto de dados (arquivo de entrada) que devem ser executados. De posse dessas informações, o controlador do *peer* inicia o processo de escalonamento, este processo funciona de formas diferentes para a versão distribuída e paralela da aplicação.

4.2.5 Implementação

Durante o planejamento do sistema, primeiramente foram estudadas diferentes plataformas para computação P2P. Após ser escolhido JXTA pelos fatores apresentados no capítulo 3, projetou-se uma estrutura que pudesse atender as necessidades do sistema e, então, partiu-se para a implementação do mesmo. Esta seção apresenta uma descrição de alguns aspectos importantes relacionados à implementação.

Embora JXTA possua implementações nas linguagens C, Java e Perl, optou-se por utilizar Java para o desenvolvimento desse trabalho. A escolha de Java deve-se ao fato dessa ser uma linguagem flexível e portável. Além disso, Java é uma linguagem orientada a objetos que possui vantagens tais como polimorfismo, herança e criação de tipos abstratos de dados, entre outras. Para simplificar as operações de descoberta de recursos e comunicação, utilizou-se uma camada de abstração para JXTA, chamada YaJal [BUC 2005] (*Yet Another JXTA Abstraction Layer*).

Como mencionado no capítulo 3, os *peers* de uma rede virtual JXTA se auto-organizam em grupos, chamados *peer groups*. Neste trabalho foi definido um grupo que possui um ID único. Na criação do grupo é possível especificar *login* e senha para aumentar a segurança da rede virtual (neste caso a autenticação é realizada pelo protocolo de *membership* do JXTA, mencionado na seção 3.5).

A monitoração de ociosidade foi implementada utilizando um detector de ociosidade desenvolvido em um outro trabalho [MAT 2005]. Esse detector dispara eventos informando quando o *peer* local está entrando ou saindo de um estado de ociosidade computacional. O estado do *peer* é conhecido através da coleta de métricas diretamente do sistema operacional. Também é possível a obtenção de informações coletadas por ferramentas de monitoração de *cluster* (Ganglia [MAS 2004], SCMS [UTH 99], PCP [GOO 2005] e Parmon [BUY 2000]), que eventualmente podem estar em funcionamento em um *cluster*, bem como via SNMP (*Simple Network Management Protocol*), em redes gerenciáveis. O detector de ociosidade pode utilizar algoritmos de predição, que são aplicados sobre os dados coletados, de modo a prever se os recursos continuarão ociosos por um período grande de tempo.

O detector de ociosidade disponibiliza um método que informa o estado atual do *peer*. Quando um trabalhador recebe uma submissão, via coordenador, ele interroga o monitor de ociosidade para verificar se seu nó está em estado ocioso e só então aceita o trabalho. Se o *peer* estiver ocupado, uma mensagem de recusa é retornada.

A definição de períodos de funcionamento é feita através de um arquivo de configuração. Este arquivo contém os intervalos que o *peer* deve funcionar. por exemplo, 12:00-14:00 indica que o *peer* deve funcionar das 12 às 14 horas. Quando o período de funcionamento termina, o *peer* é desligado e uma nova execução é agendada, junto ao sistema operacional, para o início do próximo período.

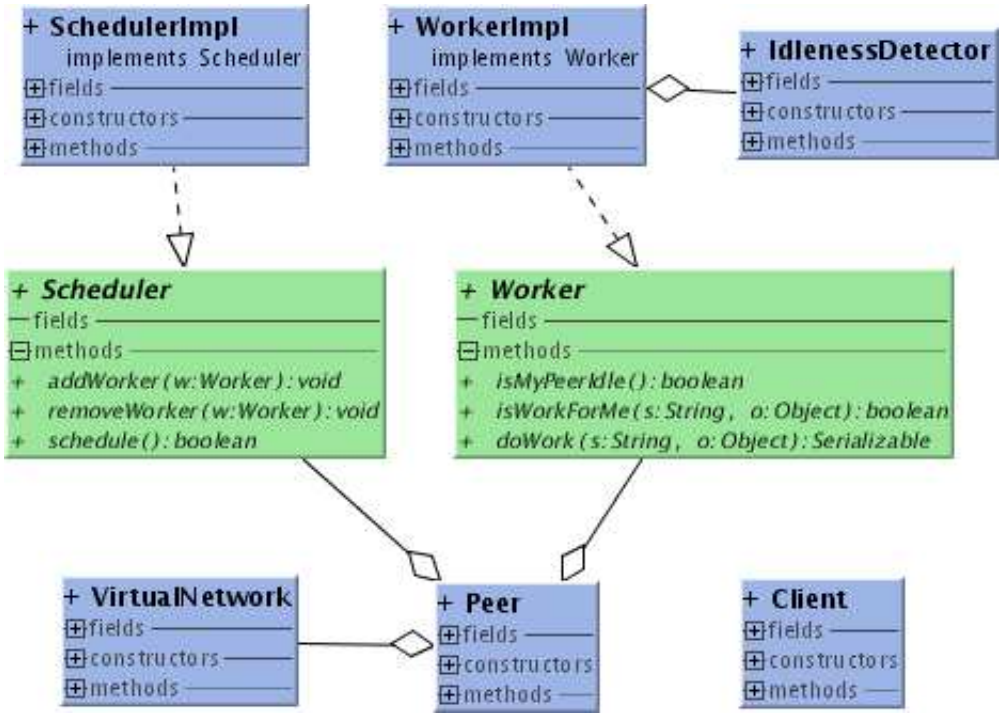

Figura 4.5: Diagrama de classes simplificado.

A figura 4.5 apresenta o diagrama de classes da implementação do sistema. Como pode ser observado no diagrama, a implementação do Peer foi contemplada através de uma classe Java chamada Peer. Um objeto dessa classe existirá em cada um dos *peers* da rede. Esse objeto instancia uma classe chamada VirtualNetwork, que mantém a rede JXTA, e uma implementação da interface Scheduler, que realiza o escalonamento dinâmico das tarefas.

O módulo trabalhador foi estruturado através de uma classe que implementa os métodos da interface Worker. Isto permite que o sistema possa ser adaptado para outras aplicações, através da reimplementação do trabalhador. Uma ou mais instâncias dessa classe existirão em cada Peer e utilizam a classe IdlenessDetector para descobrir o estado do *peer* local e tomar decisões relativas a ociosidade computacional.

A implementação do programa cliente consiste de uma classe Client capaz de se comunicar, através de *sockets* TCP, com uma instância de Peer.

4.3 Distribuição da Aplicação

O passo seguinte da implementação foi a distribuição da aplicação utilizando a plataforma de execução descrita na seção anterior. Utilizando essa plataforma, foi possível executar a aplicação em vários *peers* a fim de processar diferentes conjuntos de dados simultaneamente.

O código da aplicação alvo foi escrito em linguagem FORTRAN 77. Desta forma, 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 sabido que linguagens compiladas, como FORTRAN, geralmente obtêm um desempenho superior ao de limguagens interpretadas, como Java.

Como mencionado na seção anterior, todos os recursos, incluindo os códigos nativos (programas FORTRAN compilados para várias arquiteturas) e os arquivos de entrada contendo os dados micrometeorológicos, são publicados na rede P2P através de arquivos descritores em XML, os *advertisements*. A figura 4.6 mostra a descrição da publicação (*advertisement*) de uma biblioteca de código nativo. Quando um trabalho é submetido a um *peer*, descobre-se quais trabalhadores possuem os recursos necessários para sua execução. Neste processo, os *advertisements* locais e remotos são consultados para verificar a existência dos recursos.

```
<ResourceAdvertisement>
    <Id> urn:jxta:uuid-A58E4211C06C42F49B218B60B497B4488C64791B846E4D94B8
     B8F0FD37339386600 </Id>
    <Type> NativeCode </Type>
    <Name> libTest.so </Name>
    <Version> 20051110 </Version>
    <Desc> Biblioteca para análise de dados micrometeorológicos </Desc>
</ResourceAdvertisement>
```

Figura 4.6: *Advertisement* de uma biblioteca de código nativo.

O lançamento da aplicação é feito através do programa cliente, descrito na seção 4.2.3. Após o trabalho ser aceito pelo trabalhador, a biblioteca contendo o código nativo é carregada e a rotina principal da aplicação é executada com o arquivo referente ao conjunto de dados requisitado. No final da execução, o cliente é noti-

ficado e os resultados do processamento são disponibilizados na forma de arquivos em um diretório do *peer* local.

4.4 Paralelização da Aplicação

A paralelização da aplicação consistiu na divisão em tarefas que são distribuídas entre os processadores. Para isso utilizou-se a plataforma de execução descrita nesse capítulo.

Em alguns casos, a paralelização de um algoritmo pode ser bastante intuitiva e simples. No entanto, a utilização de metodologias de paralelização, que permitam maximizar as possibilidades de paralelização e analisar alternativas, pode reduzir os custos da paralelização e evitar escolhas ruins. Neste trabalho, optou-se por utilizar a metologia conhecida pelo acrônimo PCAM [FOS 95], que divide o processo de paralelização em quatro estágios distintos: particionamento, comunicação, aglomeração e mapeamento.

O particionamento consiste em decompor o processamento e os dados, que serão operados por esse processamento, em pequenas tarefas. Nesta fase, os problemas práticos, como número de processadores, são ignorados e a atenção é focada na descoberta de oportunidades de execução em paralelo. A divisão em tarefas pode ser feita seguindo duas abordagens principais: a decomposição funcional, onde divide-se a computação e depois define-se o conjunto de dados; e decomposição por domínio (ou dados), onde divide-se os dados em pequenos conjuntos e depois define-se o processamento que será executado sobre esses dados.

Na fase de particionamento desta aplicação, foram investigadas alternativas de decomposição, tanto funcional quanto de dados. Para isso foram realizados testes para identificar o perfil de execução da aplicação (*profiling*). A tabela 4.1 contém a porcentagem de tempo gasto por cada subrotina.

Primeiramente, verificou-se a possibilidade de uma decomposição funcional, tomando subrotinas como nível de abstração mais baixo. No entanto, analisando a tabela 4.1, percebe-se que uma única subrotina da aplicação (autocorrelacao) consome mais de 98 % do tempo total de processamento. Como o nível de abstração mais baixo, escolhido para esse trabalho, é o nível de subrotinas, partiu-se para uma outra abordagem: a decomposição por domínios.

Conforme verificado na figura 4.1, o processamento do arquivo é realizado

Subrotina	Tempo
rotacao3D	0,130 %
tendencia	0,040 %
janelaBellTaper	0,040 %
FFT	0,460 %
espectroDiscreto	0.120 %
banda	0,030 %
MMQ	0,001 %
autocorrelacao	98,670 %

Tabela 4.1: Porcentagem de tempo consumido por cada subrotina.

em janelas. Com base nisso, preferiu-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.

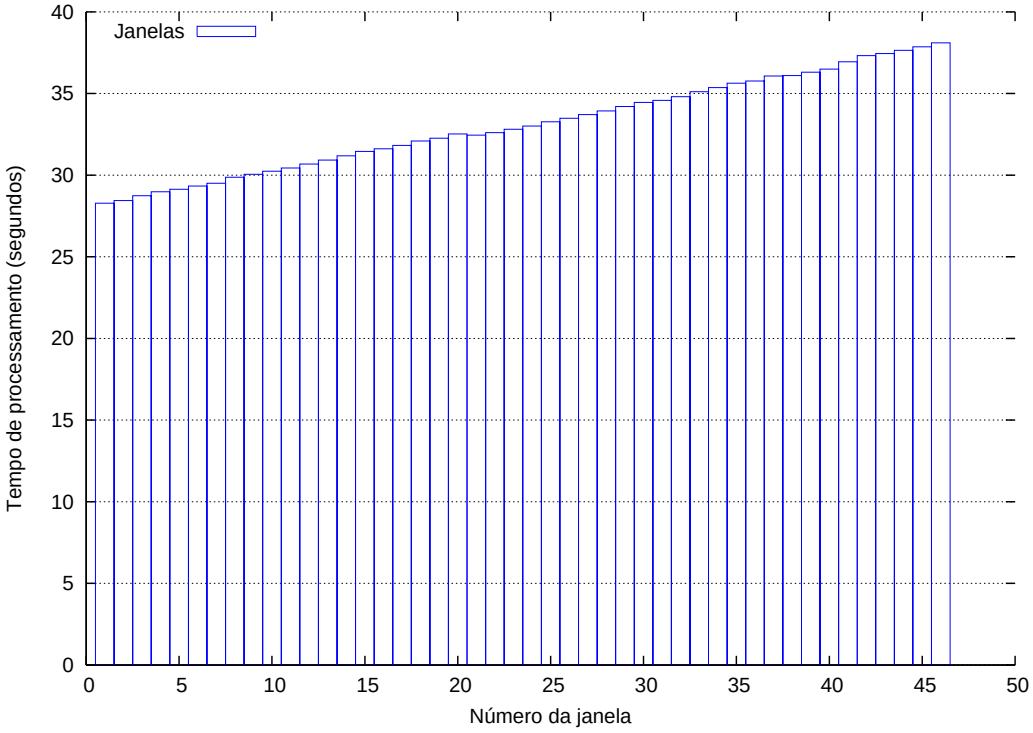

Figura 4.7: Tempos de execução de cada janela de dados.

O gráfico da figura 4.7 mostra os tempos de execução de cada janela para o processamento de um arquivo contendo os dados referentes a um dia de coleta. O ambiente de execução é o mesmo descrito na seção 4.1. Observando esse gráfico,

percebe-se que, embora as janelas tenham o mesmo tamanho, o tempo de execução aumenta para as janelas que possuem os dados mais distantes do início do arquivo. Uma nova análise do código fonte da aplicação mostrou que isso é decorrente do posicionamento da janela no arquivo de entrada, que é realizada através da leitura de todas as janelas anteriores. Desta forma, a primeira fase da paralelização também tratou da eliminação das leituras desnecessárias.

A abordagem utilizada dispensou as etapas de aglomeração e comunicação, definidas na metodologia PCAM, pois não existe dependência entre as janelas de dados. Desta forma, passou-se direto a fase de mapeamento, responsável por atribuir as tarefas aos processadores disponíveis. Para realizar o mapeametno 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 ele retorna para a fila de trabalhadores livres. Então, o escalonador lhe atribui uma nova tarefa. Da mesma forma, quando um novo *peer* ingressa na rede P2P, este publica seus trabalhadores, que também passam a fazer parte da lista de trabalhadores livres.

De forma semelhante à versão distribuída, 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 recebida. Assim que todas as tarefas tenham sido processadas, o cliente é informado sobre o final de execução e os resultados são disponibilizados na forma de arquivos em um diretório do *peer* local.

Capítulo 5

Avaliação

Este capítulo apresenta uma avaliação do sistema desenvolvido sobre a plataforma JXTA, bem como das versões distribuída e paralela da aplicação alvo.

5.1 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. Além disso, outros autores já se dedicaram à avaliação do desempenho desta plataforma [SEI 2002, HAL 2003, ANT 2005].

Os resultados desta avaliação encontram-se na tabela 5.1. Para obtê-los, utilizou-se um ambiente de execução formado por 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. As máquinas estão ligadas em rede através de um *switch* 3COM modelo 4900 funcionando a 100 Mbps. O sistema operacional utilizado em todas as máquinas é GNU/Linux (distribuição Gentoo 1.4), com *kernel* versão 2.6.12. A versão de JXTA utilizada foi a 2.3.3 com JDK versão 1.5.0.

O tempo médio de inicialização de um *peer*, na rede em questão, foi de 23,32 segundos em 300 execuções. Vale ressaltar que a inicialização do *peer* envolve a descoberta/criação do grupo de *peers*, descoberta dos *peers* que participam deste grupo, criação e publicação dos trabalhadores e inicialização das estruturas de dados

Tabela 5.1: Avaliação dos custos do sistema.

	Tempo médio	Desvio padrão
Inicialização do *peer*	23,32	4,77
Envio de arquivo de dados	14,59	1,10
Processamento	1825,20	9,05
Retorno dos resultados	9,79	1,94

utilizadas pelo controlador do *peer*. O tempo de inicialização é altamente dependente da plataforma JXTA e, desta forma, do comportamento da rede. Isso é evidenciado pelo desvio padrão de 4,77, confirmando que o tempo pode variar bastante a cada execução.

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 4.1). Executando-se o mesmo código sobre o sistema desenvolvido, obteve-se um tempo médio de execução de 30,42 minutos (1825,20 segundos). 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*).

No que concerne o envio e recepção de arquivos, os tempos médios obtidos foram, respectivamente, 14,59 e 9,79 segundos. Nos testes realizados, foram enviados os arquivos de dados e de resultados utilizados pela aplicação e descritos na seção 4.1. O arquivo de dados de entrada possui 106.7 MBytes e os arquivos de resultados possui, no total, 69.2 MBytes. Como se pode notar, o custo de transmissão não é muito significativo se comparado com o tempo total de execução da aplicação alvo.

5.2 Avaliação da Aplicação Distribuída

O objetivo da avaliação da versão distribuída da aplicação é verificar quantos conjuntos de dados (arquivos contendo os dados de 1 dia de coleta) é possível processar em um período de tempo. Para realizar os testes foi construída uma rede P2P formada por 3 dos computadores descritos na seção anterior, sendo lançado um *peer* em cada computador. Após a inicialização dos *peers*, foram realizadas submissões de trabalhos para um dos *peers*, utilizando o programa cliente. O controlador desse *peer* distribui os trabalhos entre os trabalhadores da lista de trabalhadores livres.

A tabela 5.2 contém os tempos totais (em minutos), desde a recepção da submissão até o final da execução, coletados no *peer* que recebeu as submissões. Cada trabalho submetido resulta no processamento de um arquivo de 1 dia de coleta dos sensores.

Tabela 5.2: Execução distribuída em uma rede formada por 3 *peers*.

Número de trabalhos (1 dia)	Tempo médio (min)
1	30,76
2	31,18
3	32,66

Os resultados da tabela 5.2, permitem algumas observações:

- o tempo necessário para processar um arquivo é 30,76 minutos. Duplicando-se a quantidade de dados a ser processados o tempo passa a 31,18, o que representa um aumento de 1,36 % no tempo de processamento.
- comparando com a versão seqüencial, que processava um arquivo em 24,04 minutos, a versão distribuída obteve um ganho de 35 % ao processar 2 arquivos simultaneamente. Ao processar 3 arquivos simultaneamente, esse ganho passa a 55 %.

5.3 Avaliação da Aplicação Paralela

Para avaliar a aplicação paralela, foram realizados experimentos em uma rede P2P formada pelos computadores descritos na seção 5.1. A aplicação foi divida em tarefas (ver seção 4.4), que foram escalonadas sobre os trabalhadores da rede. A tabela 5.3 contém os tempos totais de execução (em minutos) variando-se o número de trabalhadores da rede.

Comparando-se a execução da aplicação paralela com a execução da aplicação distribuída (tabela 5.2) utilizando-se apenas um trabalhador, nota-se uma diminuição no tempo de execução. Esta diminuição deve-se à eliminação das leituras desnecessárias no arquivo, durante o processo de paralelização descrito na seção 4.4.

Uma maneira de avaliar o desempenho de uma aplicação paralela é compará-la com sua versão seqüencial. Portanto, um indicador da qualidade de uma paralelização é a relação entre o tempo de execução do algoritmo seqüencial e o tempo de

Tabela 5.3: Execução paralela variando o número de trabalhadores (processadores).

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

execução do algoritmo paralelo. Essa medida é chamada de *speedup* ou fator de aceleração, e é definida como

$$S_p = \frac{T_s}{T_p(N)},$$

onde T_s é o tempo de execução do algoritmo seqüencial e $T_p(N)$ é o tempo de execução do algoritmo paralelo em N processadores. Desta forma, pode-se dizer que quanto maior o *speedup* melhor é o algoritmo paralelo, sendo $S_p = N$ considerado o *speedup* ideal.

No gráfico presente na figura 5.1 é possível verificar o *speedup* da aplicação paralela no ambiente de execução descrito. Conforme pode-se observar neste gráfico, à medida que o número de trabalhadores (processadores) aumenta, o *speedup* também aumenta. O *speedup* da aplicação paralela ficou sempre abaixo do ideal devido à sobrecarga imposta pela plataforma.

A eficiência (ϕ) de um algoritmo paralelo expressa-se através da razão entre o *speedup* obtido e o número de processadores utilizados (N) [KRO 96], conforme a fórmula:

$$\phi = \frac{S_p}{N}.$$

A tabela 5.4 apresenta uma análise da eficiência da aplicação paralela para diferentes números de trabalhadores. Observando-se essa tabela, percebe-se a viabilidade da abordagem de paralelização utilizada. Mesmo no caso em que foram usados somente 2 trabalhadores, o *speedup* foi de 1.555, o que corresponde a 77.8 % de eficiência. O *speedup* ideal, neste caso, seria 2 e corresponderia a 100 % de eficiência. Porém, os fatores externos, como tempo de comunicação e sobrecarga da plataforma, tornam esses índices impossíveis de alcançar.

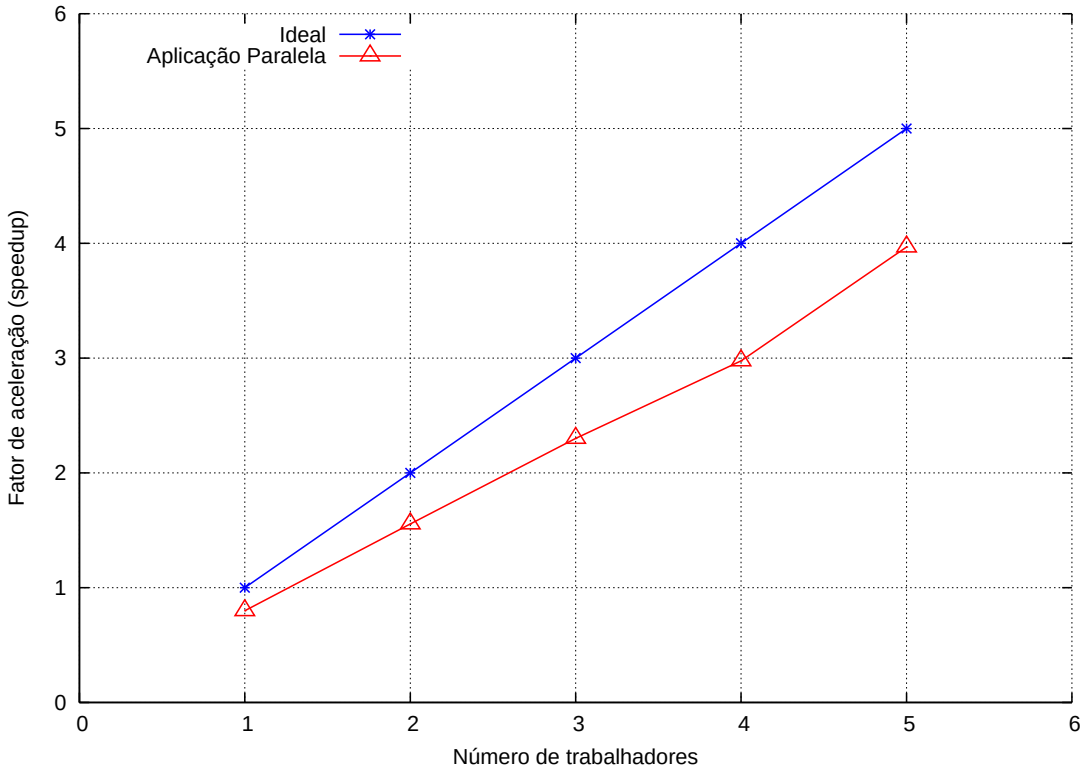

Figura 5.1: Fator de aceleração (*speedup*) da aplicação paralela.

Tabela 5.4: Eficiência da Aplicação Paralela

Número de trabalhadores (N)	*Speedup* (S_p)	Eficiência (ϕ)
2	1.555	77.8 %
3	2.300	76.7 %
4	2.976	74.4 %
5	3.967	79.3 %

Capítulo 6

Conclusão

Este trabalho apresentou a distribuição e paralelização de uma aplicação que realiza o processamento de grandes conjuntos de dados. Esta aplicação é utilizada pelo Laboratório LμMet para analisar os dados coletados por sensores micrometeorológicos.

Primeiramente, foi realizado o estudo e análise de infra-estruturas para aplicações *peer-to-peer*. A plataforma escolhida foi JXTA, por fornecer as funcionalidades e flexibilidade necessárias para implementar a distribuição e paralelização desta aplicação. O próximo passo foi o estudo da aplicação alvo, sendo gerados alguns perfis de execução (*profiling*), para analisar o seu comportamento, e análise do código fonte FORTRAN, a fim de se obter o fluxo de dados e conhecer as dependências entre as rotinas. Depois, seguiu-se para a implementação da distribuição e paralelização da aplicação.

A distribuição da aplicação tornou possível o processamento de vários conjuntos de dados simultaneamente. Além disso, a utilização de um detector de ociosidade pode permitir uma melhor utilização dos recursos disponíveis no Laboratório LμMet. Através da avaliação da distribuição da aplicação, comprovou-se um ganho de desempenho comparado a versão seqüencial. Já a paralelização aumentou a eficiência da aplicação diminuindo o tempo necessário para processar um conjunto de dados.

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. Esse sistema pode, no futuro, ser adaptado para executar outras aplicações paralelas e distribuídas.

Este trabalho pode ser melhorado e servir como base para outros trabalhos, sendo que pretende-se melhorar a interface com os usuários e introduzir tolerância

a falhas, para que possa ser mais facilmente utilizado. Além disso, pretende-se realizar testes em redes com um número maior de *peers* para avaliar a escalabilidade do sistema.

Como trabalhos futuros, pretende-se também adaptar o sistema para uma aplicação que possua dependência entre os dados, a fim de avaliar seu comportamento em um ambiente dinâmico. Também pretende-se estudar métodos de *sandboxing* para evitar a execução de códigos maliciosos e permitir a utilização do sistema em ambientes não confiáveis.

Referências Bibliográficas

[AND 2002] ANDERSON, D. **BOINC, Berkeley Open Infrastructure for Network Computing**. http://boinc.ssl.berkeley.edu: [s.n.], 2002.

[ANT 2005] ANTONIU, G. et al. Performance Evaluation of JXTA Communication Layers. In: FIFTH INTERNATIONAL WORKSHOP ON GLOBAL AND PEER-TO-PEER COMPUTING (GP2PC 2005), 2005, Cardiff, UK. **Anais...** [S.l.: s.n.], 2005.

[BAK 99] BAKER, M.; BUYYA, R. Cluster Computing at a Glance. In: BUYYA, R. (Ed.). **High Performance Cluster Computing**. Upper Saddle River, NJ: Prentice Hall PTR, 1999. v.1, Architectures and Systems, p.3–47. Chap. 1.

[BUC 2005] BUCCIARELLI, D. **Yet Another JXTA Abstraction Layer**. http://yajal.sourceforge.net/. Acessado em dezembro de 2005.

[BUY 2000] BUYYA, R. PARMON: A Portable and Scalable Monitoring System for Clusters. **Software Practice and Experience**, v.30, n.7, p.723–739, jun 2000.

[CAR 97] CARDELLI, L. Global Computation. **SIGPLAN Not.**, New York, NY, USA, v.32, n.1, p.66–68, 1997.

[CAR 98] CAROMEL, D.; KLAUSER, W.; VAYSSIÈRE, J. Towards Seamless Computing and Metacomputing in Java. **Concurrency: Practice and Experience**, v.10, n.11–13, p.1043–1061, 1998.

[DOB 2003] DOBSON, J. **Optimisation of a Large-Scale Scientific Code**. 2003. Dissertação (Mestrado em Ciência da Computação) — The University of Edinburgh.

[FED 2000] FEDAK, G. et al. XtremWeb : A Generic Global Computing Platform. In: CCGRID'2001 - SPECIAL SESSION GLOBAL COMPUTING ON PERSONAL DEVICES, 2000. **Anais...** IEEE press, 2000.

[FER 88] FERRARI, D.; ZHOU, S. An Empirical Investigation of Load Indexes for Load Balancing Applications. In: IFIP WG 7.3 ISCPM, MEASUREMENT AND EVALUATION, 12., 1988. **Proceedings...** North-Holland, 1988. p.515–528.

[FOS 99] FOSTER, I.; KESSELMAN, C. (Eds.). **The Grid**: Blueprint for a New Computing Infrastructure. [S.l.]: MORGAN-KAUFMANN, 1999. 259–278p.

[FOS 97] FOSTER, I.; KESSELMAN, C. Globus: A Metacomputing Infrastructure Toolkit. , v.11, n.2, p.115–128, 1997.

[FOS 95] FOSTER, I. **Designing and Building Parallel Program**. Reading, MA: Addison-Wesley Publishing Compagny, 1995.

[GOO 2005] GOODWIN, M. **Performance Co-Pilot Web page**. http://oss.sgi.com/projects/pcp/. Acessado em novembro de 2005.

[GRO 96] GROPP, W. et al. High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. **Parallel Computing**, v.22, n.6, p.789–828, Sept. 1996.

[HAL 2003] HALEPOVIC, E.; DETERS, R. **JXTA Performance Study**. Department of Computer Science, University of Saskatchewan, Saskatoon, Canada.: [s.n.], 2003.

[JXT 2005] JXTA. **Project JXTA Web site**. http://www.jxta.org/. Acessado em novembro de 2005.

[KAI 94] KAIMAL, J. C.; FINNIGAN, J. J. **Atmospheric Boundary Layer Flows - Their Structure and Measurement**. New York: Oxford University Press, 1994.

[KOR 2001] KORPELA, E. et al. **SETI@home**: Massively Distributed Computing for SETI. v.3, n.1.

[KRE 2004] KREUTZ, D. L.; CERA, M. C.; STEIN, B. O. Alguns Aspectos de Desempenho e Utilização de Aglomerados de Computadores Heterogêneos. In: IV CONGRESSO BRASILEIRO DE COMPUTAçãO (CBCOMP), 2004, Itajaí - SC. **Anais...** [S.l.: s.n.], 2004.

[KRI 2001] KRISHNAN, N. **The Jxta Solution to P2P**. 2001.

[KRO 96] KRONSJÖ, L. PRAM models. In: **Parallel and Distributed Computing Handbook**. New York: McGraw-Hill, 1996.

[LO 2004] LO, V. et al. Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet. In: PROCEESINGS OF 3RD INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS (IPTPS'04), 2004, San Diego, CA, USA. **Anais...** [S.l.: s.n.], 2004.

[LOD 2003] LODYGENSKY, O. et al. Augernome and Xtremweb: Monte Carlos Computation on a Global Computing Platform. **ECONF**, v.C0303241, p.THAT001, 2003.

[LUM 2005] LUMET. Laboratório de Micrometeorologia - UFSM. http://www.ufsm.br/lumet. Acessado em novembro de 2005.

[MAS 2004] MASSIE, M.; CHUN, B.; CULLER, D. **The Ganglia Distributed Monitoring System**: Design, Implementation, and Experience. California: University of California, Berkeley Technical Report, 2004.

[MAT 2005] MATHIAS, E. N.; CHARÃO, A. S. **Um Detector de Ociosidade para Sistemas Distribuídos, Baseado em Modelos Matemáticos de predições e de Descoberta de Padrões**. Universaide Federal de Santa Maria: [s.n.], 2005. Trabalho de Graduação (Ciência da Computação).

[MIL 2002] MILOJICIC, D. et al. **Peer-to-Peer Computing**. Palo Alto: Hewlett-Packard Company: HP Labs, 2002. (HPL-2002-57).

[NIE 2002] NIEUWPOORT, R. V. van et al. Ibis: An Efficient Java-based Grid Programming Environment. In: ACM JAVA GRANDE - ISCOPE CONF., 2002. **Proceedings...** [S.l.: s.n.], 2002. p.18 – 27.

[POL 96] POLLATOS, S.; CANDLIN, R. Parallelism on a Network of Workstations. In: TDP96: TELECOMMUNICATION - DISTRIBUTION - PARALLELISM, 1996, Agelonde, La Londe Les Maures, France. **Proceedings...** [S.l.: s.n.], 1996. p.439–454? Parallelisme sur un Reseau de Stations de Travail.

[PRE 92] PRESS, W. H. et al. **Numerical Recipes in FORTRAN**: The Art of Scientific Computing. 2.ed. Cambridge, UK: Cambridge University Press, 1992. xxvi + 963p.

[ROS 2003] ROSE, C. A. F. D.; NAVAUX, P. O. A. **Arquiteturas Paralelas**. Porto Alegre - RS: Sagra Luzzatto, 2003.

[SCH 2001] SCHOLLMEIER, R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: PEER-TO-PEER COMPUTING, 2001. **Anais...** [S.l.: s.n.], 2001. p.101–102.

[SEI 2002] SEIGNEUR, J.-M. **JXTA Pipes Performance**. Dublin 2, Ireland: Distributed Systems Group. Department of Computer Science. Trinity College, 2002.

[SIN 2004] SINGH, A.; HAAHR, M. **A Survey of P2P Middlewares**. Ireland: Distributed Systems Group, Department of Computer Science. Trinity College, 2004.

[STU 88] STULL, R. B. **An Introduction to Boundary Layer Meteorology**. [S.l.]: Kluwer Academic Publishers, 1988. 307-310p.

[TRA 2003] TRAVERSAT, B. et al. **Project JXTA 2.0 Super-Peer Virtual Network**.

[UTH 99] UTHAYOPAS, P.; RUNGSAWANG, A. SCMS: An Extensible Cluster Management Tool for Beowulf Cluster. In: SUPERCOMPUTING'99 (CD-ROM), 1999, Portland, OR. **Proceedings...** ACM SIGARCH and IEEE, 1999. Department of Computer Engineering, Kasetsart University.