SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: ____ / ____ / ____

Assinatura : _____

MidHPC: Um suporte para a execução transparente de aplicações em *grids* computacionais

José Augusto Andrade Filho

Orientador:

Prof. Dr. Rodrigo Fernandes de Mello

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC, USP, como parte dos requisitos para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Junho de 2008

A meus pais

Agradecimentos

Agradeço, primeiramente, a Deus, pelo dom da vida e pelas grandes oportunidades.

A meus pais, Vilma e Guga, pelo apoio incondicional em todas as horas, por serem um modelo de conduta e pelas “broncas” nas horas em que elas devem ser dadas :).

A minhas irmãs, Mônica e Shirley, pelos momentos divertidos e pela cooperação e apoio em todas as horas.

A minhas avós, tios (em especial a tia Valma e a meu padrinho Chico) e primos (em especial a Jéssica e a Andrezza) que tanto me apoiaram nessa jornada.

A meu orientador e amigo, Rodrigo Fernandes de Mello, pela dedicação ao me orientar, além do imenso apoio (em diversos aspectos) e da paciência homérica com minhas falhas e dúvidas.

A meus colegas e amigos da USP, em especial ao Marcelo, Lucas e Cláudia pela ajuda e apoio durante a realização deste trabalho. Ao Evgueni, Bert, Tott, Geraldo, Matheus, Michelle, Japa, Lourenço e Gard pela ajuda e por proporcionarem um ambiente de trabalho alegre.

À CAPES e ao CNPq, pelo apoio financeiro.

*“Computers are incredibly fast,
accurate and stupid. Human
beings are incredibly slow,
inaccurate and brilliant.
Together they are powerful
beyond imagination.”*

Albert Einstein

Sumário

1	Introdução	1
1.1	Contextualização	1
1.2	Objetivo	3
1.3	Estrutura	4
2	Escalonamento de Processos	7
2.1	Considerações Iniciais	7
2.2	Taxonomia de Algoritmos de Escalonamento de Processos	7
2.2.1	Classificação Hierárquica	7
2.2.2	Classificação Plana	10
2.3	Balanceamento de Carga	11
2.4	Considerações Finais	11
3	<i>Grids</i> Computacionais	13
3.1	Considerações Iniciais	13
3.2	Definição e Características	13
3.3	Infra-estrutura	15
3.4	Classificação dos <i>Grids</i>	15
3.5	Diferenças entre outros domínios	16
3.6	Implementações de <i>Grids</i>	16
3.6.1	SETI@Home	16
3.6.2	Globus Toolkit	18
3.6.3	Progrid	19
3.6.4	NAREGI	19
3.6.5	InteGrade	20
3.6.6	OurGrid	22
3.6.7	XGrid	23
3.7	Considerações finais	24

4	Técnicas Envolvidas no Desenvolvimento do MidHPC	25
4.1	Considerações Iniciais	25
4.2	Aprendizado Baseado em Instâncias	26
4.3	Algoritmos Genéticos	28
4.4	Ferramenta de <i>Checkpoint</i>	30
4.5	Memória Compartilhada Distribuída	31
4.6	Modelos de Programação	34
4.7	Considerações Finais	36
5	MidHPC: <i>Middleware for High Performance Computing</i>	37
5.1	Considerações Iniciais	37
5.2	Brokers Global e Local	38
5.3	Shell	39
5.4	Scheduler	40
5.4.1	Route	41
5.4.2	RouteGA	42
5.5	Extração, Classificação e Predição	43
5.6	Suporte de DSM do MidHPC	44
5.7	Integração dos módulos	45
5.8	Considerações Finais	49
6	Utilizando o MidHPC	51
6.1	Considerações Iniciais	51
6.2	Instalação	51
6.3	Configuração do MidHPC	52
6.3.1	Configuração do Broker Global	52
6.3.2	Configuração do Broker Local	52
6.3.3	Configuração do Scheduler	53
6.3.4	Configuração do Shell	53
6.4	Arquitetura Implementada	54
6.4.1	Montagem do Ambiente	55
6.4.2	Remoção de Elementos	59
6.5	Execução de uma Aplicação Genérica	61
6.5.1	Passo-a-Passo de Execução	62
6.6	Considerações Finais	67
7	Conclusões	69
7.1	Conclusões e Contribuições	69
7.2	Colaborações	71
7.3	Publicações geradas	71

7.4	Trabalhos futuros	72
Apêndices		73
A Broker Global		73
A.1	Visão Geral	73
A.2	Interações	74
A.3	Troca de Mensagens	74
A.4	Mensagens de Saída	75
A.5	Mensagens de Entrada	80
A.6	Interações SQL	82
A.7	Arquivo de configuração	83
B Broker Local		85
B.1	Visão Geral	85
B.2	Interações	85
B.3	Troca de Mensagens	86
B.4	Mensagens de Saída	86
B.5	Mensagens de Entrada	89
B.6	Arquivo de configuração	92
C Shell		93
C.1	Visão Geral	93
C.2	Interações	93
C.3	Requisições	94
C.4	Arquivo de configuração	94
D Scheduler		97
D.1	Visão Geral	97
D.2	Interações	98
D.3	Troca de Mensagens	98
D.4	Mensagens de Saída	99
D.5	Mensagens de Entrada	101
D.6	Interações SQL	102
D.7	Aplicações e comandos	104
D.8	Arquivo de interação: <i>pc.xml</i>	105
D.9	Arquivo de interação: <i>config_ibl.xml</i>	105
D.10	Arquivo de interação: <i>scheduling.xml</i>	105
D.11	Arquivo de interação: <i>environment.xml</i>	105
D.12	Arquivo de configuração	111

E	Monitores	113
E.1	Visão Geral	113
E.2	Interações	113
E.3	Monitor de Sistema	113
E.4	Monitor de Processos	115
E.5	Interações SQL	115
F	Benchmark	117
F.1	Visão Geral	117
F.2	Ferramentas	117
F.2.1	Informação sobre Memória	117
F.2.2	Capacidade em Mips	118
F.2.3	Desempenho de Disco	118
F.2.4	Desempenho de Memória	118
F.3	Arquivo de Informação <i><pc.xml></i>	118
G	Interceptação de <i>Threads</i>	121
H	IBL	123
H.1	Visão Geral	123
H.2	Interações	123
H.3	Arquivo de configuração <i>config_ibl.xml</i>	124
H.4	Saída do IBL (resultado)	126
I	Tabelas da Base de Dados	127
I.1	Visão Geral	127
I.2	Descrição das tabelas	127
I.2.1	Tabela: <i>Computer</i>	127
I.2.2	Tabela: <i>RTT</i>	128
I.2.3	Tabela: <i>Application</i>	128
I.2.4	Tabela: <i>Process</i>	130
I.2.5	Tabela: <i>History</i>	130
I.2.6	Tabela: <i>System</i>	131
	Referências	133

Lista de Figuras

2.1	Classificação Hierárquica de Escalonamento de Processos	8
3.1	Estrutura dos <i>clusters</i> do InteGrade (Goldchleger <i>et al.</i> , 2002)	21
3.2	Arquitetura do XGrid	23
4.1	Comportamento da função de ponderação	28
4.2	Operador de Cruzamento: um ponto	29
4.3	Operador de Cruzamento: dois pontos	30
5.1	Estrutura do MidHPC	38
5.2	Comunicação entre os módulos de Scheduler	40
5.3	Ordem de execução dos módulos do MidHPC	45
5.4	Diagrama de Seqüência: Início do Broker (Caso 1)	46
5.5	Diagrama de Seqüência: Início do Broker (Caso 2)	46
5.6	Diagrama de Seqüência: Início do Scheduler	47
5.7	Diagrama de Seqüência: Iniciando uma aplicação	48
6.1	Exemplo de arquivo de configuração de um Broker Global	52
6.2	Exemplo de arquivo de configuração de um Broker Local	53
6.3	Exemplo de arquivo de configuração de um Scheduler	54
6.4	Exemplo de arquivo de configuração de um Shell	54
6.5	Arquitetura implementada para testes	55
6.6	Executando um <i>mps</i> sem um Broker Global	55
6.7	Iniciando um Broker Global	55
6.8	Executando um <i>mps</i> com um Broker Global ativo	56
6.9	Iniciando um Broker Local	56
6.10	Resultado da inclusão de um Broker Local	56
6.11	Resultado do <i>mps</i> após a inclusão do Broker Local	56
6.12	Iniciando outro Broker Global	57
6.13	Resultado da requisição de registro do novo Broker Global	57

6.14	Execução do <i>mps</i> após a inclusão do segundo Broker Global	57
6.15	Resultado de um <i>mps</i> com dois Brokers Globais e quatro Brokers Locais	58
6.16	Iniciando um Scheduler	58
6.17	Resultado da inclusão de um Scheduler	58
6.18	Execução de um <i>mps</i> após a inclusão de um Scheduler	59
6.19	Resultado de um <i>mps</i> com o ambiente completo	60
6.20	Interrupção de um Scheduler	60
6.21	Broker Local - removendo um Scheduler	61
6.22	Resultado de um <i>mps</i> após a remoção de um Scheduler	61
6.23	Interrompendo um Broker Local	61
6.24	Mensagem recebida pelo Broker Global	62
6.25	Resultado de um <i>mps</i> após a remoção do Broker Local	62
6.26	Finalizando um Broker Global	62
6.27	Mensagem de remoção recebida pelo Broker Global	63
6.28	Execução de um <i>mps</i> após a finalização do Broker Global	63
6.29	Localização da aplicação exemplo	63
6.30	Executando a aplicação no MidHPC	63
6.31	Broker Global Primário recebe o pedido de início de aplicação	64
6.32	Broker Global repassa a aplicação para um Broker Local	64
6.33	Broker Local envia a aplicação para um Scheduler	64
6.34	Início da aplicação e envio da mensagem de reinício	65
6.35	Broker Local repassa a mensagem para o Broker Global	65
6.36	Broker Global encaminha a requisição para o Broker Global correto	65
6.37	Broker Global encaminha a requisição para o Broker Local definido	66
6.38	Broker Local encaminha a mensagem para o Scheduler	66
6.39	No Scheduler, a aplicação é reiniciada	66
6.40	Resultado de um <i>mps</i> que mostra a aplicação em execução	66
6.41	Execução de um <i>mps</i> após o término da aplicação	67
6.42	Execução de um <i>mps</i> com uma nova aplicação quem contém duas <i>threads</i>	67
6.43	Trecho de código do <i>testmutex: Producer</i>	68
6.44	Trecho de código do <i>testmutex: Consumer</i>	68
6.45	Trecho de código do <i>testmutex: Main</i>	68
A.1	Interações do Broker Global com outros módulos	74
A.2	Interações do Broker Global com outros módulos - mensagens	75
B.1	Interações do Broker Global com outros módulos	86
B.2	Interações do Broker Local com outros módulos - mensagens	86
C.1	Interações do Shell com outros módulos	93

D.1	Interações do Scheduler com outros módulos (parte 1)	98
D.2	Interações do Scheduler com outros módulos (parte 2)	99
D.3	Interações do Scheduler com outros módulos - mensagens (parte 1)	100
D.4	Interações do Scheduler com outros módulos - mensagens (parte 2)	100
E.1	Interações dos Monitores com outros módulos	114
G.1	Interceptação da criação de <i>threads</i>	121
H.1	Interações do IBL com outros módulos	123

Lista de Tabelas

3.1	Características de <i>Clusters</i> , <i>Grids</i> e sistemas <i>P2P</i>	17
3.2	Características presentes nas implementações de <i>Grids</i>	24
A.1	Descrição das interações do Broker Global com outros módulos	74
B.1	Descrição das interações do Broker Local com outros módulos	86
C.1	Descrição das interações do Shell com outros módulos	94
D.1	Descrição das interações do Scheduler com outros Módulos (parte 1)	98
D.2	Descrição das interações do Scheduler com outros módulos (parte 2)	99
E.1	Descrição das interações dos Monitores com outros módulos	114
H.1	Descrição das interações do IBL com outros Módulos	123
I.1	Descrição dos campos da tabela <i>Computer</i>	128
I.2	Descrição dos campos da tabela <i>RTT</i>	128
I.3	Descrição dos campos da tabela <i>Application</i>	128
I.4	Descrição dos campos da tabela <i>Process</i>	130
I.5	Descrição dos campos da tabela <i>History</i>	131
I.6	Descrição dos campos da tabela <i>System</i>	131

Lista de Símbolos

AFS	<i>Andrew File System</i>
AGs	Algoritmos Genéticos
API	<i>Application Programming Interface</i>
Bash	<i>Bourne Again Shell</i>
BG	Broker Global
BIL	<i>Best Imaginary Level Scheduling</i>
BL	Broker Local
DFS	<i>Distributed File System</i>
DNS	<i>Domain Name Server</i>
DPWP	<i>Dynamic Policy Without Preemption</i>
DSM	<i>Distributed Shared Memory</i>
EP	Elemento de Processamento
FIFO	<i>First In First Out</i>
Gid	<i>Global Identifier</i>
GNU	<i>GNU is Not Unix</i>
HTTP	<i>HyperText Transfer Protocol</i>
IBL	<i>Instance-Based Learning</i>
IP	<i>Internet Protocol</i>
LAN	<i>Local Area Network</i>
LLB	<i>List-Based Load Balancing</i>
MidHPC	<i>Middleware for High Performance Computing</i>
MIMD	<i>Multiple Instruction Mutiple Data</i>
MIPS	Milhões de Instruções Por Segundo
MPI	<i>Message Passing Interface</i>
NAREGI	<i>NAtional REsearch Grid Initiative</i>
OGSA	<i>Open Grid Service Architecture</i>
OpenMP	<i>Open Multi-Processing</i>
Pid	<i>Process Identifier</i>

Pthreads	Implementação do padrão POSIX de <i>threads</i>
PVM	<i>Parallel Virtual Machine</i>
P2P	<i>Peer-to-Peer</i>
QoS	<i>Quality of Service</i>
RTT	<i>Round-Trip Time</i>
SETI	<i>Search for ExtraTerrestrial Intelligence</i>
SSHFH	<i>Secure SHell File System</i>
TDS	<i>Task Duplication Scheduling</i>
XML	<i>eXtended Markup Language</i>

Resumo

Pesquisas em sistemas paralelos e distribuídos de alto desempenho apresentam limitações no que se refere a análise, projeto, implementação e execução automática e transparente de aplicações. Essas limitações motivaram o projeto do MidHPC (do inglês *Middleware for High Performance Computing*, ou seja, *Middleware* para Computação de Alto Desempenho), que balanceia transparente e automaticamente cargas de trabalho considerando a capacidade dos recursos computacionais e o comportamento das aplicações envolvendo: processamento, acesso a disco, memória e rede. Para utilizar todo o potencial do MidHPC, aplicações devem ser escritas utilizando o modelo de programação concorrente, tal como o padrão POSIX de *threads* (*pthread*). Aplicações desenvolvidas seguindo esse modelo de programação podem ser executadas em ambientes de *Grid* sem alteração de código fonte ou recompilação. Durante a execução, tarefas de uma mesma aplicação paralela comunicam-se, transparentemente, por meio de um sistema de memória compartilhada distribuída. O objetivo deste trabalho foi desenvolver alguns dos módulos do projeto MidHPC e integrar demais ferramentas que haviam sido previamente desenvolvidas pelo grupo. Este trabalho permite aplicar, em ambientes reais, todos os conceitos de escalonamento de processos estudados e desenvolvidos durante o projeto MidHPC.

Abstract

Current researches on parallel and distributed systems present limitations regarding the analysis, design, implementation and automatic execution of high performance applications. Those limitations motivated the design of **MidHPC** (Middleware for High Performance Computing), which transparently and automatically balances workloads considering computing resources capacities and application behavior such as: processing, network, memory and disc accesses. In order to use all of the **MidHPC** potential, applications must be developed following the concurrent programming model, using the POSIX thread standard (**pthread**s). Applications developed according to this programming model can be executed in the Grid environment with no source code modification nor recompilation. During the execution, tasks of the same application communicate, transparently, using a distributed shared memory system. The objective of this master thesis was to develop modules of the **MidHPC** project and integrate tools that were previously developed by the research group. This work allows applying, in real-world environments, all process scheduling concepts studied and developed during the **MidHPC** project.

Introdução

1.1 Contextualização

A disponibilidade de microprocessadores de baixo custo e a evolução das redes de computadores tornaram economicamente viável o desenvolvimento de sistemas distribuídos. Em tais sistemas, processos comunicam-se a fim de realizar a mesma tarefa computacional. Além de diminuir o custo, esses sistemas permitem a construção de ambientes escaláveis e mais flexíveis que máquinas paralelas (Buyya, 1999).

Com o advento dos sistemas distribuídos, pesquisas foram desenvolvidas com o objetivo de: maximizar o uso dos recursos computacionais; obter alto desempenho; e desenvolver ambientes e bibliotecas de comunicação para simplificar o projeto e implementação de sistemas distribuídos (Sterling, 2000).

Posteriormente, os conceitos de sistemas distribuídos evoluíram partindo de ambientes construídos em redes locais para ambientes internet¹. Nesses ambientes, redes interagem entre si através de roteadores. Assim, surgem novas necessidades de pesquisas para o desenvolvimento de técnicas de alocação de recursos e obtenção de alto desempenho em ambientes de larga escala. Esses ambientes, localizados em regiões geograficamente distintas e compostos por recursos computacionais de capacidade heterogênea, são denominados *Grids* computacionais (Stockinger, 2007).

Dada a complexidade desses ambientes, oriunda da heterogeneidade e da distância entre os recursos, políticas de escalonamento de tarefas tiveram que ser adaptadas. Escalonadores, anteriormente utilizados em ambientes homogêneos, apresentam um desempenho inferior quando utilizados em *Grids* computacionais. Para melhorar o desempenho desses escalonadores e, conseqüentemente, de aplicações paralelas, diversas pesquisas foram

¹Ambientes internet referem-se a redes interconectadas e não à rede mundial de computadores.

conduzidas nas áreas de balanceamento de carga, imagem única de sistema, protocolos de baixa latência, bibliotecas de comunicação e arquiteturas de *hardware* de alto desempenho (Apon & Baker, 2000).

Entretanto, os resultados dessas pesquisas apresentavam limitações em relação ao desenvolvimento e execução de aplicações, pois, em geral, exigiam que o usuário tivesse conhecimento sobre características específicas de ambientes, bibliotecas e técnicas de distribuição e balanceamento de carga.

O orientador deste trabalho, juntamente com demais colaboradores, procurando solucionar esse impasse, passou a desenvolver pesquisas com o objetivo de criar um ambiente transparente e distribuído para a execução de aplicações paralelas. Inicialmente, o foco dessas pesquisas estava em *clusters* de computadores e, posteriormente, foi modificado para ambientes de *Grid* computacional.

Os primeiros trabalhos realizados abordavam a avaliação de desempenho de protocolos de comunicação, utilizando modelos que exploravam as características de diferentes ambientes e bibliotecas (Ferreira & Mello, 2004; Dodonov *et al.*, 2005). Os estudos realizados permitiram a extração de dados sobre a latência e *overhead* de pacotes utilizando diferentes protocolos e redes, os quais serviram de base para compreender o ambiente no qual as aplicações paralelas são executadas.

Os resultados de medições de rede e a possibilidade de compreender a distribuição de cargas de trabalho (processamento) motivaram pesquisas com o objetivo de projetar um modelo de migração de processos. Esse modelo, baseado nos trabalhos de Harchol-Balter & Downey (1997), possibilitou avaliar o tempo de execução de processos, como projetado pelos outros autores, como as cargas de trabalho sobressalentes (Mello & Senger, 2004).

Com a viabilização da migração de processos, novas pesquisas foram realizadas sobre escalonamento, resultando no desenvolvimento de algoritmos que adotam técnicas como: localidade de computadores para a distribuição de processos em *Grids* computacionais, algoritmos genéticos (Mello *et al.*, 2007), otimização baseada em colônias de formigas (Nery *et al.*, 2006), conhecimento sobre a ocupação de recursos para a distribuição de processos, dentre outras pesquisas (Mello *et al.*, 2004; Ishii *et al.*, 2005; Reis *et al.*, 2005; Senger *et al.*, 2005; Mello *et al.*, 2006b).

As pesquisas sobre escalonamento de processos motivaram a realização de estudos sobre o comportamento de aplicações paralelas e distribuídas, uma vez que o uso eficiente de um ambiente distribuído depende do conhecimento de características específicas de cada aplicação, tais como o uso de recursos de processamento, memória, acesso a disco e rede (Dodonov *et al.*, 2006; Dodonov & Mello, 2007).

A integração dessas pesquisas motivou o desenvolvimento do projeto MidHPC (*Middleware for High Performance Computing*) que provê um *middleware* para a execução de aplicações paralelas e distribuídas desenvolvidas segundo o paradigma concorrente (Sethi, 1989).

A principal contribuição do MidHPC é oriunda do fato de que projetistas de *software* possam utilizar apenas o paradigma concorrente, usando *threads*, para o desenvolvimento de aplicações que são, automática e transparentemente, paralelizadas e distribuídas. O MidHPC oferece suporte a essas aplicações (*multithreads*) ao interceptar a criação de *threads* para transformá-las em processos, de modo a permitir o balanceamento de carga transparente. A comunicação entre esses processos é realizada por meio de um suporte de DSM, que é transparente ao usuário.

A utilização de módulos de inteligência artificial para conduzir o balanceamento de carga é um diferencial do MidHPC em relação a outras implementações de *Grids*. Esses módulos consideram otimizações em comunicação e em tempo de resposta de aplicações.

O modelo de programação do MidHPC torna a complexidade de escalonamento, de transferência de mensagens e de gerenciamento de ambiente transparentes ao desenvolvedor. Assim, o projetista de *software* precisa apenas levar em consideração os problemas relativos à aplicação em desenvolvimento, como gerenciamento de condições de corrida.

1.2 Objetivo

O presente trabalho tem como objetivo o desenvolvimento de alguns módulos e a integração de outras ferramentas que compõem o projeto do MidHPC (*Middleware for High Performance Computing*)². O MidHPC permite a execução de aplicações paralelas e distribuídas que seguem o paradigma concorrente (Sethi, 1989) em ambientes distribuídos de larga escala, tais como *Grids* computacionais. Visando esse fim, foram realizadas as seguintes tarefas, ressaltando que o termo *Desenvolvimento* refere-se aos módulos projetados e implementados neste mestrado e o termo *Utilização* refere-se à adoção de módulos previamente projetados e implementados pelo grupo:

- Desenvolvimento³o módulo **Broker** para gerenciar, de maneira escalável, os computadores que participam do ambiente de *Grid*. O Broker é dividido em dois submódulos, um Local para controle dos Schedulers e um Global, para gerenciar a interação entre redes distintas;
- Desenvolvimento do módulo **Scheduler** que permite acoplar um algoritmo de balanceamento de carga responsável pela tomada de decisões;
- Desenvolvimento do módulo **Shell**, baseado no GNU/Bash, que implementa a interface com usuários. Esse Shell provê ferramentas para início e término de aplicações, além de funções para verificar a execução de processos e a carga do sistema;

²Website do projeto MidHPC: <http://antrax.icmc.usp.br/~augusto/midhpc>

³d

-
- Utilização de uma técnica para permitir o uso do paradigma concorrente por meio da conversão, transparente, de *threads* para processos, o que permite a transferência de carga;
 - Utilização de um **Monitor** que extrai características da aplicação e as armazena em uma base de dados (*knowledge base*). Tais informações são posteriormente utilizadas pelo Scheduler, através da técnica de aprendizado baseado em instâncias (**IBL**);
 - Utilização de um suporte de memória compartilhada distribuída (**DSM** – *Distributed Shared Memory*) que visa permitir a comunicação entre processos.

As questões relativas a segurança, alta disponibilidade e replicação não são abordadas no contexto deste trabalho. Este trabalho permite aplicar, em ambientes reais, todos os conceitos de escalonamento de processos estudados e desenvolvidos durante o projeto MidHPC.

1.3 Estrutura

Esta dissertação de mestrado apresenta a seguinte organização:

- **Capítulo 2**, Escalonamento de Processos, apresenta a Taxonomia de Algoritmos de Escalonamento (seção 2.2), descrevendo seus conceitos e detalhando o Balanceamento de Carga, tópicos essenciais no projeto MidHPC;
- **Capítulo 3**, *Grids* Computacionais, apresenta definições, características e exemplos de implementações de *Grids* computacionais;
- **Capítulo 4**, Técnicas Envolvidas no Desenvolvimento do MidHPC, detalha as técnicas utilizadas no desenvolvimento do MidHPC: Aprendizado Baseado em Instâncias (seção 4.2), Algoritmos Genéticos (4.3), Ferramentas de *Checkpoint* (seção 4.4) e Memória Compartilhada Distribuída (seção 4.5). Por fim, apresenta Modelos de Programação (seção 4.6) que são utilizados no desenvolvimento de aplicações paralelas e distribuídas;
- **Capítulo 5**, MidHPC: *Middleware for High Performance Computing*, apresenta o desenvolvimento do projeto, detalhando a estrutura e os módulos principais (Brokers Local e Global, Shell e Scheduler), além de abordar a integração desses (seção 5.7);
- **Capítulo 6**, Utilizando o MidHPC, detalha os requisitos de instalação do MidHPC, apresenta a montagem de um ambiente e a execução de aplicações;
- **Capítulo 7**, Conclusões, que apresenta Contribuições e atividades futuras a serem realizadas;

- Por fim são apresentados os **Apêndices**, que contêm informações detalhadas sobre a implementação de cada um dos módulos, e as **Referências**.

Escalonamento de Processos

2.1 Considerações Iniciais

O projeto MidHPC tem como principal função o escalonamento de processos, com o objetivo de balanceamento de carga, em ambientes distribuídos heterogêneos de larga escala. Assim, fez-se necessário um estudo sobre a Taxonomia de Algoritmos de Escalonamentos de Processos proposta por Casavant & Kuhl (1988) de modo a fornecer uma visão geral sobre os algoritmos de escalonamento de processos e permitindo compreender maiores detalhes sobre o funcionamento do MidHPC.

2.2 Taxonomia de Algoritmos de Escalonamento de Processos

Casavant & Kuhl (1988) propuseram uma taxonomia para enumerar e definir termos utilizados para qualificar algoritmos de escalonamento de processos de acordo com a organização e utilização das informações para o escalonamento. Essa taxonomia foi dividida em dois aspectos apresentados nas seções seguintes: hierárquico e plano.

2.2.1 Classificação Hierárquica

No aspecto hierárquico (figura 2.1), os autores propuseram uma estrutura de árvore para classificação das técnicas de escalonamento de processos. Essa classificação é apresentada a seguir.

Figura 2.1: Classificação Hierárquica de Escalonamento de Processos

Local *versus* Global

A primeira divisão na hierarquia proposta contém dois ramos: Local e Global. Esse nível subdivide algoritmos de escalonamento segundo a localização dos processos. O Local refere-se ao escalonamento em máquinas com apenas um processador e o Global refere-se ao escalonamento em sistemas multiprocessados.

Para máquinas com apenas um processador, o escalonamento é feito pelo sistema operacional, assim, não faz sentido discorrer sobre esse ramo, pois, este trabalho é voltado para o escalonamento de processos em sistemas distribuídos.

Estático *versus* Dinâmico

Os algoritmos de escalonamento classificados como Globais são subdivididos em Estáticos e Dinâmicos. Essa subdivisão indica o momento em que as atribuições ou escalonamentos são realizados. No caso do escalonamento Estático as decisões são tomadas na fase de projeto. No Dinâmico, as decisões de escalonamento são tomadas durante a execução dos processos.

No ramo Estático dessa classificação, a análise comparativa enfocou os sub-ramos descritos a seguir:

Ótimo *versus* Subótimo

Os algoritmos Estáticos são classificados em Ótimos e Subótimos. O Ótimo requer que todas as informações necessárias para escalonamento estejam disponíveis. Sobre tais informações são aplicadas funções e seus resultados definem a alocação dos processos. Quando um problema não é computacionalmente factível, soluções Subótimas devem ser

utilizadas. Dentre as soluções Subótimas, pode-se adotar duas abordagens: Aproximação e Heurística.

Aproximação *versus* Heurística

A diferença entre as técnicas estáticas subótimas de Aproximação e Heurística é que a primeira, em vez de procurar pela melhor resposta em todo espaço de soluções do problema, busca uma boa solução. Essa, é obtida adotando uma métrica ou função de avaliação que analisa parâmetros e restrições do problema de escalonamento. A segunda, técnica Heurística, é baseada em análises sobre o conhecimento *a priori* de informações do sistema e do comportamento da aplicação.

Técnicas Ótimas e Subótimas Aproximadas

Casavant & Kuhl (1988) unificam os conceitos das técnicas Globais, Estáticas Ótimas e Subótimas Aproximadas para classificar quatro técnicas:

1. Busca e enumeração do espaço de solução (Shen & Tsai (1985) *apud* Casavant & Kuhl (1988));
2. Teoria dos grafos (Bokhari (1979); Stone (1978); Stone & Bokhari (1978) *apud* Casavant & Kuhl (1988));
3. Programação matemática (Bokhari (1981) *apud* Casavant & Kuhl (1988));
4. Teoria de filas (Kleinrock (1976) *apud* Casavant & Kuhl (1988)).

Soluções Dinâmicas

Nesse ramo da classificação Global, encontram-se algoritmos de escalonamento que tomam decisões durante a execução dos processos. Esses algoritmos são descritos e comparados de acordo com o nível que assumem na árvore (figura 2.1).

Fisicamente Distribuídos *versus* Não-distribuídos

Os algoritmos Dinâmicos são subdivididos em Fisicamente Distribuídos e Não-distribuídos. Os Não-distribuídos são representados por algoritmos onde a responsabilidade de escalonamento é de um único processador. Nos Distribuídos, a responsabilidade é partilhada entre processadores.

Cooperativo *versus* Não-cooperativo

Os algoritmos Fisicamente Distribuídos são subdivididos em Cooperativos e Não-cooperativos. Nesses algoritmos, cada processador executa um módulo de *software* que

captura suas informações locais de carga e capacidade de processamento, as quais são utilizadas para tomada de decisões. Nos Cooperativos, o módulo de *software* de um processador utiliza informações dos demais módulos que executam em outros processadores e, com base nessas informações, são tomadas decisões de alocação de processos. Nos Não-cooperativos, os processadores consideram somente as informações locais para a tomada de decisões.

Nesse ponto da taxonomia, soluções ótimas, subótimas aproximadas e subótimas por heurísticas podem ser utilizadas. A discussão apresentada no ramo estático aplica-se, também, nesse caso.

2.2.2 Classificação Plana

Segundo Casavant & Kuhl (1988), alguns tópicos não podem ser acoplados à estrutura hierárquica, pois dificultam a compreensão e, por esse motivo, foram enumerados em uma classificação auxiliar, denominada Plana. Os termos contidos nessa classificação são apresentados e definidos a seguir:

1. *Adaptativo versus Não-adaptativo*: Algoritmos de escalonamento de processos Adaptativos são aqueles que utilizam informações do sistema para modificar suas heurísticas na tomada de decisões. Os Não-adaptativos desconsideram mudanças no sistema;
2. *Atribuição Inicial versus Reatribuição Dinâmica*: Na Atribuição Inicial, uma vez que um processo foi alocado a um processador ele não é mais transferido. Na Reatribuição Dinâmica (ou migração), um processo pode ser transferido para outro processador durante sua execução;
3. *Balanceamento de Carga*: Algoritmos de escalonamento de processos que visam Balanceamento de Carga distribuem, eqüitativamente, a carga de processos no sistema. Através dessa técnica, minimiza-se o tempo de resposta dos processos e a variância na carga dos processadores. O balanceamento de carga utiliza-se da técnica de Reatribuição Dinâmica;
4. *Compartilhamento de Carga*: Algoritmos de escalonamento de processos que visam Compartilhamento de Carga buscam atribuir processos a todos elementos de processamento do sistema;
5. *Probabilístico*: Esse termo refere-se às políticas de escalonamento que fazem escolhas probabilísticas para alocação de processos, de acordo com funções de distribuição de probabilidades.

2.3 Balanceamento de Carga

Segundo o aspecto plano da taxonomia apresentada por Casavant & Kuhl (1988) os algoritmos com suporte a balanceamento de carga distribuem, eqüitativamente, a carga de trabalho entre elementos de processamento (EPs) do sistema, diminuindo os tempos de resposta dos processos (Shivaratri *et al.*, 1992).

Esta seção apresenta maiores detalhes sobre os principais componentes de um algoritmo de balanceamento de carga definidos por Shivaratri *et al.* (1992). Tais detalhes devem auxiliar na melhor compreensão das operações realizadas pelos algoritmos de balanceamento de carga e, conseqüentemente, do projeto MidHPC que adota tais conceitos. São eles:

1. *Política de transferência*: determina se um EP está em um estado de carga que permita participar de uma operação de transferência de processos, seja como emissor ou receptor. Os receptores são aqueles cuja carga de trabalho é baixa, isto é, estão ociosos, já os emissores ou servidores, são aqueles que se encontram sobrecarregados;
2. *Política de seleção*: é responsável por decidir qual processo será transferido para outro EP do ambiente. Se não existirem processos a serem transferidos, o elemento não é mais considerado um emissor, mesmo que esteja sobrecarregado;
3. *Política de localização*: tem como função encontrar o melhor emissor de processos para um determinado receptor, ou vice-versa. O melhor emissor é aquele que está mais sobrecarregado e o melhor receptor, o que está mais ocioso;
4. *Política de informação*: sua função é decidir sobre a periodicidade e captura de informações sobre o estado de carga dos EPs. Tais informações são utilizadas para a tomada de decisões de balanceamento. Esses dados são utilizados para a obtenção de um índice de carga que identifica a ocupação de cada EP. Esse índice é utilizado pela política de localização para definir emissores e receptores.

Bons algoritmos de balanceamento de carga utilizam poucos recursos computacionais do ambiente e conseguem alocá-los de forma otimizada, aumentando o desempenho das aplicações.

2.4 Considerações Finais

Este capítulo apresentou conceitos sobre escalonamento de processos e sua classificação hierárquica e plana segundo Casavant & Kuhl (1988). Após identificar o termo balanceamento de carga nessa taxonomia, foram apresentados conceitos adotados pelos algoritmos de balanceamento de carga (Shivaratri *et al.*, 1992). Esses conceitos são importantes,

uma vez que a principal atividade do MidHPC é o escalonamento de processos em ambientes distribuídos. Segundo a taxonomia apresentada, os dois algoritmos de escalonamento atualmente suportados pelo MidHPC são classificados como: Global, Dinâmico, Fisicamente Distribuído, Cooperativo, Sub-ótimo por Heurísticas, contemplando, também, o Balanceamento de Carga.

Grids Computacionais

3.1 Considerações Iniciais

Neste capítulo são abordados conceitos e sistemas voltados para a implementação de *Grids* computacionais. Inicialmente são apresentadas motivações para o desenvolvimento de *Grids* computacionais, em seguida, definições de *Grids*, infra-estrutura, classificação e a diferença entre *Grids* e outros domínios. Por fim, são apresentadas algumas implementações de *Grids*.

3.2 Definição e Características

Computadores são utilizados para modelar e simular problemas complexos de ciência e engenharia, diagnosticar condições médicas, controlar equipamentos industriais, prever o tempo, gerenciar portfólios de ações e vários outros propósitos (Foster & Kesselman, 2000). Esses problemas necessitam de processamento de alto desempenho. Uma das primeiras abordagens para solucionar tal questão foi por meio da utilização de supercomputadores paralelos.

Com o avanço da tecnologia e a conseqüente melhoria dos computadores pessoais, surgiu a idéia de utilizá-los para realizar as tarefas antes feitas pelos supercomputadores. Inicialmente, os computadores compartilhavam seus recursos de processamento em uma rede local e, em geral, apresentavam a mesma configuração (ambiente homogêneo). Esses agrupamentos são também denominados *clusters*. Com a melhoria das conexões, os agrupamentos começaram a ser interligados com o intuito de reduzir custos, dando origem à infra-estrutura denominada *Grid* Computacional.

Stevens *et al.* (1997) definiram o termo *Grid* em analogia às redes de energia elétrica

(*Electric Power Grids*). Assim como a rede de energia elétrica, os *Grids* computacionais permitem um acesso de baixo custo a recursos compartilhados (Foster & Kesselman, 2000).

Bote-Lorenzo *et al.* (2004) definem um *Grid* computacional como uma infra-estrutura de *hardware* e *software* distribuída geograficamente, em larga escala, composta por recursos heterogêneos interligados e compartilhados. Esses recursos pertencem a várias organizações administrativas e são coordenados para prover um suporte computacional transparente, confiável e persistente para uma grande variedade de aplicações. As aplicações podem realizar computação distribuída, computação de alta vazão (*high-throughput*), computação sob demanda, computação colaborativa ou computação multimídia. À partir dessa definição foram enumeradas diversas características, as quais foram estudadas por Stockinger (2007):

1. *Larga escala*: é necessário que *Grids* suportem grandes quantidades de recursos, levando em consideração a manutenção de desempenho, à medida que essa quantidade aumenta;
2. *Distribuição geográfica*: recursos podem estar distribuídos em locais geograficamente distantes;
3. *Heterogeneidade*: recursos são os mais variados possíveis, tanto em termos de *software* quanto de *hardware*;
4. *Controle descentralizado*: recursos do *Grid* são gerenciados por diversas organizações;
5. *Compartilhamento de recursos*: apesar de pertencerem a organizações distintas, os recursos são compartilhados entre todos os participantes;
6. *Transparência*: virtualmente, *Grids* são vistos como um único computador, que representa a soma da capacidade de todos os recursos;
7. *Consistência*: *Grids* devem ser construídos sobre padrões existentes, sejam de *hardware* ou *software*, de modo a facilitar a interconexão dos recursos;
8. *Persistência*: *Grids* devem se adaptar ao ambiente dinâmico e manter o acesso aos recursos;
9. *Segurança*: acesso seguro aos recursos é uma característica importante, visto que o controle dos recursos é de responsabilidade de organizações distintas;
10. *Suporte a aplicação*: *Grids* têm suporte a diversos tipos de aplicações, porém, alguns são voltados para tarefas específicas;

11. *Modelo computacional*: em geral, *Grids* têm suporte a diversos modelos de execução (*batch*, interativo, paralelo e distribuído etc);
12. *Modelo de licença*: devido à sua origem acadêmica, existe uma tendência de usar *softwares* de código aberto.

3.3 Infra-estrutura

Para prover as características enumeradas na seção 3.2, uma infra-estrutura de *Grid* computacional deve conter um conjunto de técnicas, relacionadas a seguir (Stockinger, 2007):

1. *Modelagem de recursos*: descreve os recursos disponíveis, suas capacidades e as suas relações, de modo a facilitar a descoberta e o gerenciamento;
2. *Monitoração e notificação*: permite identificar os estados de um recurso e notificar a aplicação (ou a infra-estrutura) sobre a alteração desses estados;
3. *Alocação*: identifica recursos que possam ser utilizados para executar a aplicação;
4. *Provisionamento, gerenciamento de ciclo de vida e liberação*: permite que um recurso seja configurado automaticamente para ser usado por uma aplicação, gerenciando sua utilização durante execução e liberando o recurso ao término da aplicação;
5. *Relatório e auditoria*: gerencia o uso dos recursos compartilhados pelos usuários.

Além disso, um *Grid* disponibiliza aos usuários um ponto único de entrada para executar aplicações.

3.4 Classificação dos *Grids*

Grids computacionais podem ser classificados de acordo com a relação entre as instituições que os mantêm. Assim, são divididos em colaborativos e *enterprise*.

Como essa classificação é baseada nas relações entre as instituições que mantêm o *Grid*, uma mesma implementação pode ser classificada em quaisquer das classes. Por exemplo, uma determinada implementação ao ser utilizada em conjunto por vários departamentos de uma mesma instituição é classificada como *enterprise*. Porém, essa mesma implementação pode ser classificada como colaborativa se um conjunto de instituições diferentes (empresas, universidades etc) utilizá-la.

Vale ressaltar que alguns autores classificam um *cluster*, que tem como objetivo alto desempenho/vazão computacional, como um *Grid*. Porém, *clusters* tendem a ser estáticos, homogêneos e gerenciados por um único domínio. Assim, *clusters* não deveriam ser denominados de *Grids*, a não ser que estejam conectados a outros *clusters*.

3.5 Diferenças entre outros domínios

Segundo Stockinger (2007), ao final da década de 1990, *Grids* computacionais surgiram como um novo domínio na ciência da computação, embora diversas técnicas e protocolos tenham sido herdados de outros domínios tais como a computação distribuída. Assim, *Grids* computacionais não podem ser discutidos isoladamente, pois essa área da computação apresenta diversas intersecções com outras.

Grid versus Computação Distribuída

Devido à sobreposição de diversas características é difícil diferenciar o domínio de *Grids* computacionais da computação distribuída. Para Stockinger (2007), alguns pesquisadores consideram o *Grid* como um caso especial de computação distribuída. A escalabilidade (possibilidade de aumentar o número de recursos computacionais) e a transparência (independência de plataforma e capacidade de usar recursos heterogêneos de *hardware* e *software*) são características particulares que evidenciam sua complexidade.

Entretanto, alguns autores afirmam que não existe uma diferença real entre *Grids* computacionais e a computação distribuída, pois eles são complementares e um parte do outro (Stockinger, 2007).

Grid versus Clusters e sistemas P2P

Um conjunto de características de *Clusters*, *Grids* e sistemas *P2P* é apresentado na tabela 3.1. Os principais fatores para diferenciar um *cluster* de *Grids* computacionais e sistemas *P2P* vêm do fato dele apresentar gerenciamento centralizado, posse única dos recursos computacionais e arquiteturas homogêneas. Diferenciar sistemas *P2P* de *Grids* torna-se cada vez mais complexo, pois o primeiro pode ser utilizado para a construção de diferentes aplicações, incluindo o processamento distribuído, característico de *Grids*.

3.6 Implementações de *Grids*

Nesta seção são apresentadas algumas implementações de *Grids* computacionais.

3.6.1 SETI@Home

O projeto SETI@Home, desenvolvido pela Universidade da Califórnia em Berkeley, tem o propósito de analisar sinais de rádio captados no observatório em Arecibo (Porto Rico), em busca de vida inteligente fora da Terra (*Search for ExtraTerrestrial Intelligence*) (Korpela *et al.*, 2001). Historicamente, é considerado como o primeiro *Grid* computacional, porém, é voltado para uma aplicação específica, a análise de sinais.

Tabela 3.1: Características de *Clusters*, *Grids* e sistemas *P2P*

Característica	<i>Cluster</i>	<i>Grid</i>	<i>P2P</i>
População	PCs comuns	Computadores de ponta	PCs <i>desktop</i>
Posse	Única	Múltiplas	Múltiplas
Descoberta	Serviços membros	Índice centralizado & informação descentralizada	Descentralizada
Gerenciamento de usuário	Centralizado	Descentralizado	Descentralizado
Gerenciamento de recurso	Centralizado	Distribuído	Distribuído
Alocação/Escalonamento	Centralizado	Descentralizado	Descentralizado
Imagem única de sistema	Sim	Não	Não
Escalabilidade	Centenas	Milhares	Milhões
Capacidade	Garantida	Variável, mas alta	Variável
Vazão	Média	Alta	Muito Alta
Velocidade (latência de banda)	Baixa, alta	Alta, baixa	Alta, baixa

Fonte: Stockinger (2007)

A arquitetura desse sistema é simples: dados capturados por um rádio-telescópio em Arecibo são gravados em fita e enviados para Berkeley (EUA), onde são divididos em pequenas unidades de trabalho (*work units*), que correspondem a 10KHz de largura de banda de 107 segundos de duração (aproximadamente 350Kb). Essas unidades são transferidas para um meio de armazenamento temporário e, posteriormente, distribuídas entre os usuários. Os clientes SETI@home são executados em computadores pessoais espalhados pelo mundo. Basicamente, eles requisitam uma unidade de trabalho, processam a mesma, enviam os resultados para o servidor central e, nesse momento, recebem outra unidade.

O protocolo de comunicação entre os clientes e o servidor é baseado em HTTP, de maneira a permitir que máquinas atrás de *firewalls* consigam contatar o servidor¹. Não há dependência entre as unidades de trabalho, o que implica em não existir comunicação entre clientes. Além disso, a conexão à Internet é necessária apenas no momento de envio e recepção de dados, permitindo o processamento *off-line*. O cliente periodicamente salva o estado da computação de maneira a permitir o progresso, mesmo se o computador pessoal é freqüentemente ligado e desligado.

O projeto SETI@home tem um sistema de escalonamento de processos simplificado, enviando uma unidade de trabalho para os computadores requisitantes. Isso é possível pois operações em uma unidade de trabalho são independentes. Uma mesma unidade de trabalho é replicada, para diversos computadores, de modo a garantir que seja executada. Um computador solicita uma outra unidade de trabalho somente após finalizar uma análise.

¹Um *firewall* pode bloquear requisições para a porta 80, porém, geralmente essa porta não é bloqueada.

3.6.2 Globus Toolkit

O *Globus Toolkit*, atualmente na versão 4, é um conjunto de ferramentas de *software* e módulos que implementam a arquitetura OGSA (*Open Grid Service Architecture*), definida pelo comitê *The Globus Alliance* ². Essa arquitetura apresenta, basicamente, as seguintes funcionalidades:

1. *Localização e alocação de recursos*: o mecanismo de localização de recursos é importante, pois a maioria das aplicações não conhece exatamente o local dos recursos requisitados, principalmente quando a carga e a disponibilidade dos recursos variam. A alocação de recursos diz respeito ao escalonamento e envolve tarefas básicas, tais como: criação de processos, acesso a dados, etc. O *Globus Resource Allocation Manager* (GRAM) fornece alocação de recursos, criação de processos, monitoração e gerenciamento de serviços. O GRAM mapeia requisições escritas em uma linguagem de especificação de recursos (RSL - *Resource Specification Language*) em comandos para escalonadores e computadores locais;
2. *Interface de autenticação*: fornece mecanismos básicos de autenticação, utilizados para identificar tanto usuários quanto recursos. O *Grid Security Infrastructure* (GSI) oferece uma interface simples de autenticação, com suporte a controle local sobre direitos de acesso e mapeamento de identidades globais de usuários em identidades locais;
3. *Comunicação*: oferece mecanismos básicos para comunicação. Deve permitir a implementação abrangente de vários métodos de comunicação, tais como: passagem de mensagens, chamada de procedimento remoto, memória compartilhada distribuída, *multicast*, etc. No *Globus*, o módulo de comunicação é baseado na biblioteca de comunicação *Nexus* (Foster & Kesselman, 1999). O *Nexus* define cinco abstrações básicas: nós, contexto, *threads*, canais de comunicação e requisições de serviço remoto. As funções do *Nexus* que manipulam tais abstrações formam a interface de comunicação do *Globus*. Essa interface é utilizada amplamente por outros módulos do *Globus* e também para construção de serviços de alto nível, incluindo ferramentas para programação paralela;
4. *Serviço de informação de recursos unificado*: permite, em tempo de execução, a coleta de informações a respeito da infra-estrutura do *Grid* e do seu estado. Esse mecanismo permite a atualização, a coleta de informações e o controle de acesso;
5. *Acesso a dados*: oferece acesso remoto para armazenamento persistente. O módulo de acesso a dados do *Globus* trata do problema de armazenamento de alto desempenho, assim que ocorre acesso a sistemas de arquivos paralelos e dispositivos de

²<http://www.globus.org>.

entrada e saída embutidos na rede, tal como o HPSS (*High Performance Storage System*). O GridFTP (*Grid File Transfer Protocol*) implementa uma variedade de operações automáticas e programadas para movimentação e estratégias para acesso a dados, habilitando programas que executam em locais remotos para leitura e escrita de dados.

O *Globus Toolkit* também utiliza tecnologias de serviços *Web*, tais como: SOAP (*Simple Object Access Protocol*), WSDL (*Web Service Definition Language*) e WSIL (*Web Services Inspection Language*), que suportam gerenciamento de estado distribuído, inspeção, descoberta, invocação e notificações distribuídas. Especificamente, a interface dos serviços do *Grid* é definida em formato WSDL.

Outros sistemas são baseados na arquitetura do *Globus*, tais como o ESG (*Earth System Grid*) que tem como propósito modelar e simular o clima terrestre (Foster *et al.*, 2003).

3.6.3 Progrid

O *Progrid* (Costa *et al.*, 2003), desenvolvido pela Universidade Federal de São Carlos, é uma arquitetura para gerenciamento e operação de *Grids* baseada em servidores *proxy*.

No *Progrid*, o servidor *proxy* atua como um ponto de interconexão entre as redes que compõem o *Grid*. Para incorporar a estrutura do *Progrid* a uma rede, não é necessário realizar modificações na infra-estrutura, pois os serviços são oferecidos em um ponto único localizado na borda da rede. Nesse ambiente, a comunicação entre os computadores é realizada pelo *proxy* através de uma conexão segura (utilizando SSL).

Para a execução de uma aplicação é utilizado uma versão MPI modificada com suporte a comunicação através de *proxies*. Essa modificação é transparente ao usuário e à aplicação, que não necessita de alterações para executar ambiente.

É necessário que cada rede participante do ambiente tenha, pelo menos, um *proxy*, de modo que esse permita a comunicação entre as tarefas gerenciadas pelo MPI. Esse *proxy* também é responsável por controlar e gerenciar as informações da rede em que está alocado. O *status* geral do ambiente é obtido através da compilação dos dados dos *proxies* de todas as redes, o que diminui o *overhead* no controle de comunicação.

3.6.4 NAREGI

O propósito geral no NAREGI (Matsuoka *et al.*, 2005) é desenvolver os fundamentos tecnológicos de *Grids* voltados para computação científica de larga escala. Esse projeto é dividido em seis grupos de pesquisa e desenvolvimento (*Work Package – WP*). Cada um dos WPs enfoca seus trabalhos em uma área e serão descritos, em linhas gerais, a seguir:

1. O WP-1 é responsável pelos *middlewares*, de camada mais baixa e intermediária, para o gerenciamento de recursos como *superscheduler* (voltado para a alocação de

tarefas em recursos que possam atender seus requisitos), **GridVM** (para controle de recursos locais) e serviços de informação no *Grid*;

2. O WP-2 coordena o desenvolvimento de ferramentas básicas para programação no *Grid*, que consistem, principalmente, de: **GridRPC**, que possibilita a distribuição de aplicações em ambientes de larga escala, e **GridMPI**, que provê suporte a aplicações desenvolvidas em MPI;
3. O WP-3 trabalha com ferramentas de *Grids* para o usuário final, incluindo *Grid workflows* (ferramenta visual para preparar, submeter e consultar aplicações executando em recursos remotos), ambientes para resolução de problemas (**PSE**, *Problem-Solving Environment*, utilizado para incorporar as aplicações ao *Grid*) e ferramentas de visualização (específica para nanosimulações);
4. O WP-4 lida com a junção (*packing*) e o gerenciamento da configuração dos *softwares* do projeto, que provê um serviço de gerenciamento de informações sobre recursos, escalável e seguro, utilizado na execução de um ambiente de *Grid* computacional de larga escala e multidisciplinar;
5. O WP-5 investiga questões de rede, segurança e gerenciamento para infra-estrutura de *Grids* de alto desempenho, como medidas de tráfego em tempo real, provisionamento de *QoS*, roteamento otimizado entre organizações virtuais e protocolos de transferência de dados;
6. Por último, o WP-6 desenvolve componentes específicos de *middlewares* para a utilização do *Grid* em aplicações de nanociência de larga escala.

A maneira com que esses módulos são utilizados depende de como a aplicação foi projetada: para aplicações legadas, ou seja, que não foram desenvolvidas para ambientes de *Grid*, o usuário especifica onde e como, de uma maneira declarativa, suas tarefas serão preferivelmente executadas; para aplicações compostas por vários processos, é preciso definir um *workflow* entre as tarefas. Em seguida, o *superscheduler* decide, automaticamente, onde os processos serão executados.

Novas aplicações, que assumem a execução em paralelo em *Grids*, são desenvolvidas com o auxílio das ferramentas **GridRPC** e **GridMPI**, que possibilitam paralelização de aplicações e portabilidade. Outras facilidades como transferência de arquivos e instalação são realizadas em conjunto com o **GridPSE**.

3.6.5 InteGrade

O *InteGrade* (Goldchleger *et al.*, 2002; Goldchleger, 2005) permite que aplicações paralelas possam ser executadas em um ambiente distribuído, beneficiando-se do poder

computacional ocioso que existe nas organizações. Isso é conseguido ao integrar recursos e máquinas de usuário em laboratórios compartilhados em uma intranet ou em um *Grid*. Ele tem arquitetura orientada a objetos, onde cada módulo do sistema comunica-se com os demais a partir de chamadas de método remotas. O *InteGrade* utiliza CORBA (Object Management Group, 2002) como sua infra-estrutura de objetos distribuídos que facilita a implementação, uma vez que a comunicação entre os módulos do sistema é abstraída pelas chamadas de método remotas.

Esse ambiente é estruturado em *clusters* que podem conter estações de trabalhos ou máquinas dedicadas ao *Grid*. Esses *clusters* são organizados de maneira hierárquica para permitir a inclusão de diversos recursos. A figura 3.1 mostra as principais funções que uma máquina do *cluster* pode realizar: Os gerenciadores de *cluster* (*Cluster Manager*) representam computadores responsáveis por gerenciar aquele *cluster* e a comunicação com os demais; uma máquina do usuário (*User Node*) é aquela que submete aplicações ao *Grid*; uma provedora de recursos (*Resource Provider Node*) exporta parte de sua capacidade para o *Grid* e, em geral, são estações de trabalho; e um recurso dedicado (*Dedicated Node*) é reservado exclusivamente para utilização pelo *Grid*. Vale ressaltar que esses papéis podem ser sobrepostos, por exemplo, uma máquina de usuário pode ser uma provedora de recurso.

Figura 3.1: Estrutura dos *clusters* do *InteGrade* (Goldchleger *et al.*, 2002)

O gerenciamento interno do *cluster* é realizado, conjuntamente, pelo gerenciador de recursos locais (LRM, *Local Resource Manager*) e pelo gerenciador de recursos globais (GRM, *Global Resource Manager*). O LRM é executado em todas as máquinas do *cluster* que compartilham recursos e tem por objetivo coletar informações sobre a utilização de memória, CPU, disco e rede, que são enviadas para o GRM periodicamente. O GRM, então, utiliza essas informações para o escalonamento dentro do *cluster*.

Quando um usuário submete uma aplicação para execução, o GRM seleciona máquinas com recursos disponíveis para serem candidatas a executar a aplicação, com base nas informações locais. Em seguida, devido à natureza dinâmica do *Grid*, o GRM verifica se o melhor candidato tem recursos disponíveis naquele momento e o reserva para iniciar

a execução. Caso o recurso não esteja disponível, outro candidato é selecionado e o processo se repete. É importante ressaltar que recursos em quaisquer um dos *clusters* do *Grid* podem ser acessados, caso seja necessário.

O LUPA (*Local Usage Pattern Analyzer*) e o GUPA (*Global Usage Pattern Analyzer*) gerenciam a coleta e a análise dos padrões de uso dentro de um *cluster*. O LUPA executa em máquinas do *cluster* e coleta dados sobre padrões de uso. Baseando-se nesses dados, é derivado um padrão de uso semanal para a máquina o qual é enviado periodicamente para o GUPA. Essa informação sobre os padrões de uso é disponibilizada ao GRM para a tomada de decisões de balanceamento de carga.

O centro de controle (NCC, *Node Control Center*) permite que o dono do provedor de recursos defina condições para o compartilhamento. Parâmetros como período de uso, porção de recursos compartilhado no *Grid* ou condições para considerar o recurso como disponível podem ser configurados com essa ferramenta.

O ASCT (*Application Submission and Control Tool*) permite que um usuário submeta aplicações para execução no *Grid*. O usuário pode especificar pré-requisitos, como *hardwares* ou *softwares* a serem utilizados, e requisitos dos recursos, como quantidade mínima de memória, dentre outras. Essa ferramenta pode ser utilizada para monitorar o progresso da aplicação.

O *InteGrade* utiliza o modelo de programação paralela BSP desenvolvido por Goldchleger (2005).

3.6.6 OurGrid

O projeto *OurGrid* (Andrade *et al.*, 2003) identifica quatro pontos-chave na execução de aplicações do tipo *bag-of-tasks* em *Grids*. São eles: 1) ambiente de *Grid*, 2) permissão de acesso automática, 3) escalonamento de aplicação eficiente, e 4) mecanismo de tratamento de erro.

O OurGrid utiliza o *broker* chamado *MyGrid*, que escalona a aplicação nos diversos recursos que o usuário tenha permissão de acesso, seja por uma infraestrutura de *Grid* (como o Globus) ou via acesso remoto. O *MyGrid* foi projetado para ser transparente e completo, englobando todo o ciclo de produção, e apresentado em duas visualizações: a *GridMachine*, que provê os serviços básicos necessários para usar essa máquina e a *GridMachineProvider*, que abstrai os gerenciadores de recursos. Esse modelo simplifica a interoperação com novos recursos.

Sob esse projeto está em desenvolvimento um sistema P2P de compartilhamento de recursos visando aplicações *bag-of-tasks*, chamado *OurGrid Community*³. O intuito dessa comunidade é servir como uma rede de favores: um usuário disponibiliza seus recursos ociosos com o objetivo de poder utilizar os de outros quando necessitar. Além disso,

³<http://www.ourgrid.org>.

com uma conta autônoma, a rede de favores garante a igualdade no compartilhamento de recursos, incentivando doações de recursos.

O sistema de escalonamento utilizado pelo OurGrid é denominado de WQR – *Workqueue with Replication* – (Paranhos *et al.*, 2003), que apresenta um bom desempenho sem utilizar informações de recursos ou das tarefas ao replicar aplicações em execução em máquinas disponíveis no ambiente.

3.6.7 XGrid

A tecnologia *XGrid* da *Apple* é parte integrante dos sistemas Mac OS X e Mac OS X Server. Por esse motivo sua configuração e utilização é simplificada, necessitando apenas habilitar esse serviço e preparar a infra-estrutura.

A arquitetura do XGrid é dividida em três camadas, como mostra a figura 3.2. Essa arquitetura é composta por clientes (1), uma unidade controladora (2) e os agentes dedicados, parciais ou internet (3, 4 e 5 respectivamente).

Figura 3.2: Arquitetura do XGrid

O elemento principal do *XGrid* é o *Controller* (2). Ele é responsável por distribuir as aplicações do Cliente (1) entre os agentes do *Grid* (3, 4 e 5). Ao término da execução, o *Controller* coleta os resultados e os apresenta ao Cliente.

O escalonamento de processos no XGrid é feito seguindo a ordem de chegada das tarefas no controlador. A partir desse elemento, o processo é encaminhado para o computador mais rápido naquele momento. É possível definir dependências nas tarefas de modo que essas sejam escalonadas em uma ordem determinada.

O XGrid não é heterogêneo quanto a sistema operacional, pois, necessita de computadores com sistema Mac OS X, dificultando sua utilização. Outro problema é a presença de um elemento centralizador único, pois, caso esse pare de funcionar, todo o Grid perde sua funcionalidade.

3.7 Considerações finais

Este capítulo apresentou definições de *Grids* computacionais propostas por Bote-Lorenzo *et al.* (2004) e Stockinger (2007). Tomando como referência a definição desses autores, as principais características foram apresentadas e comentadas.

A tabela 3.2 apresenta um comparativo entre as características do MidHPC e das implementações descritas (SETI@Home, seção 3.6.1; Globus Toolkit, seção 3.6.2; ProGrid, seção 3.6.3; NAREGI, seção 3.6.4; InteGrade, seção 3.6.5; OurGrid, seção 3.6.6; e XGrid, seção 3.6.7).

Tabela 3.2: Características presentes nas implementações de *Grids*

Característica	SETI@Home	Globus	ProGrid	NAREGI	InteGrade	OurGrid	XGrid	MidHPC
Suporte a aplicações <i>bag-of-tasks</i>	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Suporte ao escalonamento de <i>threads</i>	Não	Não	Não	Não	Não	Não	Não	Sim
Suporte a comunicação entre processos	Não	Não	Sim	Não	Não	Não	Não	Sim
Sistema Operacional	Windows	Unix	Linux	Unix	Linux	Linux e Windows	Mac OS	Linux
Uso de info. de comportamento	Não	Não	Não	Não	Sim (apenas computador)	Não	Não	Sim (aplicação e computador)
Arquitetura de <i>Software</i>	BOINC ⁴	OGSA ⁵	-	OGSA	OGSA CORBA	-	-	-

Como pode ser observado nessa tabela apenas o MidHPC tem suporte ao escalonamento de *threads*, o que facilita o desenvolvimento de aplicações paralelas e permite o uso de aplicações legadas (*multithread*).

Um outro ponto importante é a utilização de informações de comportamento com o objetivo de otimizar a execução de aplicações. Dentre as implementações apresentadas, apenas o InteGrade utiliza esse tipo de informação ao coletar o histórico de uso de computadores, enquanto o MidHPC utiliza tanto informações de uso de computadores como relativas ao comportamento de execução de aplicações.

Ressalta-se, também, que somente o MidHPC e o Progrid apresentam suporte a comunicação entre processos. Entretanto, enquanto o Progrid utiliza um sistema semelhante ao MPI que exige conhecimento do usuário para utilizar a biblioteca de comunicação, o MidHPC permite a comunicação entre processos de maneira transparente por meio de suporte de DSM.

Informações adicionais sobre as características do MidHPC são descritas no capítulo 5.

Técnicas Envolvidas no Desenvolvimento do MidHPC

4.1 Considerações Iniciais

Diversas técnicas foram desenvolvidas no escopo do projeto MidHPC. Este trabalho de mestrado foi responsável pela unificação dessas técnicas com o objetivo de construir um ambiente que realiza o balanceamento de carga de processos, adotando técnicas de inteligência artificial. Esse ambiente suporta aplicações legadas (*multithread*) e permite comunicação transparente entre processos.

As técnicas de inteligência artificial utilizadas foram: Aprendizado Baseado em Instâncias (seção 4.2), que é usado para caracterizar o comportamento médio de execução de aplicações; e Algoritmos Genéticos (seção 4.3), utilizados para otimizar o balanceamento de carga com base no comportamento das aplicações. Como o balanceamento pressupõe o uso de reatribuição dinâmica (migração), *checkpoints* (seção 4.4) são usados para permitir essa tarefa.

Esse ambiente suporta o escalonamento das diferentes linhas de execução de aplicações *multithread* por meio da conversão de *threads* em processos (discutido no capítulo 5 e no apêndice G). Esses processos são escalonados no *Grid* e comunicam-se, transparentemente, por meio de um suporte de DSM (seção 4.5). Essa conversão permite que desenvolvedores utilizem o modelo de programação concorrente, altamente difundido.

4.2 Aprendizado Baseado em Instâncias

Aprendizado Baseado em Instâncias (*Instance-Based Learning* – IBL) é um paradigma de aprendizado que orienta a construção de algoritmos para encontrar instâncias similares em uma base de experiências, o qual visa aproximar funções de domínio real ou discreto (Aha *et al.*, 1991; Mitchell, 1997). O aprendizado consiste simplesmente em armazenar dados de treinamento e utilizá-los para realizar buscas e aproximações.

Cada instância é composta de um conjunto de atributos, os quais podem ser classificados como entradas ou saídas: atributos de entrada descrevem as condições em que uma experiência foi observada; enquanto atributos de saída representam os resultados de acordo com as condições descritas pelos atributos de entrada.

Os algoritmos IBL calculam a similaridade entre uma instância de consulta, x_q , e as instâncias da base de experiências, retornando um conjunto de instâncias relacionadas ao ponto de consulta. Nesses algoritmos, apenas as instâncias mais relevantes são usadas para classificar e realizar predições para o ponto de consulta. Eles podem construir uma aproximação distinta por meio de uma função objetivo para cada ponto de consulta apresentado.

O algoritmo IBL, proposto por Senger *et al.* (2006), é baseado no aprendizado dos k -vizinhos mais próximos (*k-nearest neighbor learning*). O método assume que todas as instâncias em uma base de experiências correspondem a pontos em um espaço $\mathbb{R}^n$ e que uma função de distância, Euclidiana, é utilizada para definir pontos próximos à consulta.

Seja uma instância x descrita por um vetor de atributos:

$$(a_1(x), a_2(x), \dots, a_n(x)) \quad (4.1)$$

onde $a_r(x)$ descreve o valor do r -ésimo atributo de um instância x .

A distância entre duas instâncias x_i e x_j corresponde a $d(x_i, x_j)$:

$$E(x_q, x) = \sqrt{\sum_{i=1}^n d(a_i(x_q), a_i(x))^2} \quad (4.2)$$

A função de distância $d(a_i, x_q)$ é definida pelas Equações 4.3 e 4.4.

$$d(x, y) = \begin{cases} 1 & \text{se } x \text{ ou } y \text{ estão faltando atributos} \\ d_n(x, y) & \text{se os atributos são nominais} \\ \|x - y\| & \text{caso contrário} \end{cases} \quad (4.3)$$

$$d_n(x, y) = \begin{cases} 0 & \text{se } x = y \\ 1 & \text{caso contrário} \end{cases} \quad (4.4)$$

Um problema recorrente da distância Euclidiana é que quando um dos atributos apresenta um intervalo de valores muito mais amplo, ele pode dominar os valores obtidos com a equação de distância básica, levando a um pobre desempenho classificatório do algoritmo (Wilson & Martinez, 1997). Por exemplo, se o atributo responsável por armazenar o número de EPs (elementos de processamento) apresentar valores entre 1 e 1024 e o atributo que representa o identificador da fila estiver entre 1 e 5, a influência do identificador da fila será dominada pelo número de EPs alocados. Pode-se solucionar essa questão por meio da normalização dos atributos numéricos em uma mesma escala.

Senger *et al.* (2006) adotam uma função de normalização linear $h(x_i)$, que é usada para reduzir os atributos numéricos a uma mesma escala, entre 0 e 1. Seja $\min_{x_i}$ o menor valor na base de experiências para o atributo x_i e $\max_{x_i}$ o maior valor para o mesmo atributo, a função $h(x_i)$ corresponde a:

$$h(x_i) = \frac{x_i - \min_{x_i}}{\max_{x_i} - \min_{x_i}} \quad (4.5)$$

Assim, para aproximar uma função de domínio de valores reais $f : \mathbb{R}^n \rightarrow \mathbb{R}$, utilizando pesos para cada um dos k vizinhos mais relevantes, de acordo com a distância ao ponto de consulta x_q , é utilizada a equação 4.6.

$$f'(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i} \quad (4.6)$$

A função de peso w_i deve gerar resultados próximos de um valor constante quando a distância tende a 0 e aproximar-se de 1 à medida que a distância tende ao infinito. Embora diferentes funções de ponderação possam ser empregadas, Senger *et al.* (2006) adotam uma função Gaussiana centrada no ponto x_q com uma variância t , conforme equação 4.7.

$$w_i(x) = e^{-\frac{E(x_q, x)}{t^2}} \quad (4.7)$$

A Figura 4.1 apresenta exemplos de valores de w para um intervalo de distâncias Euclidianas entre 0 e 10, empregando 0.125, 0.250, 0.500, 1.000 e 2.000 como valores de t^2 .

Através das técnicas de ponderação descritas, o algoritmo proposto por Senger *et al.* (2006) apresenta a vantagem de encontrar aplicações paralelas similares trabalhando com atributos em valores reais (como número de EPs utilizados, memória utilizada e tempo de execução) e atributos de valores discretos, tais como: usuário que submeteu a aplicação e grupo de trabalho ao qual pertence, assim como o identificador da fila do *software* de escalonamento.

No MidHPC a técnica IBL é utilizada para obter o comportamento médio de uso de recursos de uma aplicação. Essa técnica é acionada antes de iniciar aplicações e tem sido utilizada para parametrizar o algoritmo de escalonamento de processos.

Figura 4.1: Comportamento da função de ponderação

O IBL é usado do seguinte modo: suponha uma aplicação α composta por três processos (α_1, α_2 e α_3). Durante a execução da aplicação, os processos α_1, α_2 e α_3 são monitorados e seus dados de ocupação de recursos são salvos em uma base de dados. Em seguida, a média de uso dos recursos por esses processos é calculada e armazenada. É com base nessas médias históricas de uso que o IBL define o comportamento médio de ocupação de recursos ao escalonar uma nova aplicação.

4.3 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são aplicados como técnicas de busca e otimização em diversas áreas. São baseados no mecanismo de seleção natural, focando na sobrevivência do indivíduo mais apto. AGs nem sempre encontram a melhor solução possível, porém, fornecem boas soluções locais para problemas NP-completos (Papadimitriou, 1994).

A solução de problemas utilizando algoritmos genéticos envolve dois aspectos: codificação da solução na forma de cromossomos, onde cada cromossomo representa uma possível solução, e uma função de aptidão (*fitness*) que é aplicada para encontrar a melhor solução.

Várias técnicas de codificação podem ser utilizadas para diferentes tipos de problemas, como *strings* binárias, *bitmaps*, números reais dentre outras. A função de aptidão é responsável por avaliar as possíveis soluções. Essa função recebe um cromossomo como parâmetro e retorna um número real que representa a qualidade da solução obtida, por exemplo, o quão adequada é a solução para o problema em estudo.

Cromossomos mais aptos são identificados e armazenados durante o processo de evolução. Os menos aptos, por outro lado, são eliminados. Diferentes técnicas podem ser aplicadas para a identificação dos melhores cromossomos, como a seleção proporcional,

seleção por *ranking* e a seleção baseada em torneio (Back *et al.*, 1999).

Na seleção proporcional, indivíduos são transferidos para a próxima geração de acordo com o valor proporcional ao resultado de sua função de aptidão. Uma das possíveis implementações dessa técnica consiste no uso de uma roleta, dividida em N partes, com N sendo o número de indivíduos (cromossomos) da população atual. O tamanho de cada parte é proporcional ao valor da função de aptidão de cada indivíduo. A roleta é girada N vezes e, a cada turno, o indivíduo apontado é selecionado e inserido na próxima geração.

A seleção baseada em *ranking* pode ser dividida em duas etapas. Durante a primeira, as soluções são ordenadas de acordo com seu valor da função de aptidão. Com a lista ordenada, cada indivíduo recebe um novo valor de aptidão de acordo com sua posição no *ranking*. Após isso, um procedimento que seleciona os indivíduos conforme sua posição no *ranking* é aplicado. Assim, os indivíduos de melhores posições têm maiores chances de serem selecionados.

Na seleção baseada em torneio não se atribui automaticamente probabilidades a indivíduos. Um torneio de tamanho k é definido, com $k \geq 2$ indivíduos. Então, k indivíduos são escolhidos aleatoriamente na população atual, e seus valores de aptidão são comparados e o indivíduo com melhor valor de aptidão é selecionado para reprodução. O valor de k é definido pelo usuário, representando a pressão de seleção, isto é, a velocidade com a qual os indivíduos mais aptos vão dominar a população, gerando o extermínio dos mais fracos.

Uma vez selecionados os indivíduos para reprodução, é necessário modificar suas características genéticas usando técnicas de reprodução conhecidas como operadores genéticos. Os operadores mais comuns são o cruzamento e a mutação.

O operador de cruzamento permite a troca de material genético entre dois indivíduos, conhecidos como pais, combinando suas informações de modo a aumentar a possibilidade de gerar um novo indivíduo com melhores características que os originais (Hinterding, 2000).

O cruzamento de um ponto é o mais usado. Para aplicá-lo, dois indivíduos (pais) são selecionados e dois novos indivíduos são criados a partir deles (filhos). Um único ponto de quebra aleatório é selecionado nos cromossomos pais, e novos cromossomos são criados à partir da combinação dos primeiros, como mostrado na figura 4.2. A figura 4.2 (a) mostra os indivíduos pais e o ponto de separação marcado pelo símbolo $|$. Novos indivíduos criados a partir da combinação dos cromossomos pais são mostrados na figura 4.2 (b), ilustrando o operador de cruzamento.

Figura 4.2: Operador de Cruzamento: um ponto

O cruzamento de dois pontos também é bastante utilizado. Semelhante ao de um ponto, onde dois indivíduos são selecionados e dois novos, criados. A diferença é que esse operador utiliza dois pontos de quebra aleatórios, ao invés de apenas um. A figura 4.3 exemplifica um cruzamento com os locais de quebra representados pelo símbolo |. A figura 4.3 (a) mostra os cromossomos pais e (b) os gerados após o cruzamento.

Figura 4.3: Operador de Cruzamento: dois pontos

O operador de mutação é utilizado para alterar um único gene por um valor aleatório. Quando um indivíduo é representado por um *bitmap*, esse operador escolhe de forma aleatória um gene do cromossomo e troca seu valor de 1 para 0 e vice-versa. O objetivo do operador de mutação é manter a diversidade da população, sempre permitindo que um cromossomo cubra um amplo espaço de busca (Hinterding, 2000). Esse operador é geralmente aplicado com baixa probabilidade, pois, com alta, o resultado tende a ser aleatório.

No contexto do MidHPC, os conceitos abordados nessa seção foram utilizados para o desenvolvimento do algoritmo de balanceamento de carga RouteGA (Mello *et al.*, 2007), que é detalhado na seção 5.4.2.

4.4 Ferramenta de *Checkpoint*

Uma ferramenta de *checkpoint* captura o contexto de um processo, como registradores, memória e arquivos abertos, de modo a permitir o reinício da aplicação, no mesmo ou em outro processador.

É possível classificar as ferramentas de *checkpoint* conforme a forma de implementação (Zhang *et al.*, 2005; Kim, 2005): modificações do *kernel*, implementação de módulos e no espaço de usuário. Os mecanismos que modificam o *kernel* são implementados por meio de alterações núcleo do sistema operacional. Esses mecanismos têm a vantagem de oferecer acesso direto às estruturas do *kernel* e a desvantagem de depender das estruturas da versão utilizada, aumentando a complexidade de implementação e dificultando a portabilidade.

Os mecanismos que implementam módulos de *kernel* interceptam chamadas de sistema (Kyiv National Taras Shevchenko University, 2007; Janakiraman *et al.*, 2005; Osman *et al.*, 2002; Shankar, 2005). Alguns desses mecanismos necessitam da adaptação de bibliotecas de funções (Plank *et al.*, 1995; Iskra *et al.*, 2000; Sankaran *et al.*, 2003). Eles têm a vantagem de diminuir a dependência do *kernel*.

Finalmente, alguns sistemas de *checkpoint* são implementados como programas de

usuário que podem necessitar de privilégios de *root*. Como outros sistemas de *checkpoint* ele também captura o estado de um processo em execução e o grava em um arquivo executável. Esse arquivo pode ser utilizado para reiniciar o processo posteriormente.

*CryoPID*¹, sistema utilizado pelo MidHPC, é uma ferramenta de *checkpoint* implementada como programa de usuário. Ela apresenta o diferencial de não necessitar de permissões de superusuário (*root*) para realizar o *checkpoint* (executa em sistemas Linux-*kernel* 2.4 e 2.6).

Essa ferramenta permite que uma imagem completa do executável seja gerada, inclusive com as bibliotecas dinâmicas utilizadas para a execução em máquinas com versões diferentes. Sua execução é simplificada, pois é necessário apenas o comando *freeze* *<output-file>* *<pid>*, onde *output-file* é o nome do arquivo que conterá o *checkpoint* e *pid* representa o identificador do processo sobre o qual será realizado o *checkpoint*. Executado esse comando, um arquivo binário executável, de nome *<output-file>*, é gerado com as informações do processo e o seu reinício será efetuado com a execução desse arquivo.

No MidHPC, o *checkpoint* é utilizado para possibilitar a migração de processos. Atualmente, ele é acionado no início da aplicação, após a conversão de *threads* em processos, para que esses sejam migrados de acordo com o resultado do algoritmo de escalonamento. Vale ressaltar que esse algoritmo pode indicar a migração de processos em execução, o que significa realizar um *checkpoint* sobre eles.

4.5 Memória Compartilhada Distribuída

Em um DSM (*Distributed Shared Memory*), cada processador participante de um sistema distribuído oferece parte de sua memória local para os demais, visando ampliar o espaço de endereçamento global de memória (Cox *et al.*, 1997; Protic *et al.*, 1996). O suporte de DSM deve oferecer consistência de acesso aos dados para tarefas, localizadas em processadores distintos, que compartilham regiões de memória.

Um mecanismo de DSM pode ser definido como uma memória que oferece acesso distribuído a dados de maneira uniforme, transparente e consistente. Assim, dados localizados na memória compartilhada são tratados como se estivessem na memória local.

Um suporte a DSM pode ser classificado de acordo com três itens: algoritmos, implementação e modelo de consistência, detalhados a seguir.

Algoritmos

Os suportes de DSM consideram algoritmos com diferentes abordagens para distribuição e consistência de acesso aos dados.

¹Disponível em: <http://cryopid.berlios.de>.

Com relação à distribuição de dados, duas estratégias podem ser utilizadas: replicação e migração. Na replicação, várias cópias de um mesmo arquivo podem existir na memória local de diferentes processadores. Essas cópias permitem que cada processador acesse a informação localmente, evitando, assim, atrasos de comunicação. A replicação beneficia aplicações que efetuam, principalmente, a leitura de dados.

Na migração, há somente uma cópia do dado, que trafega pelo meio de comunicação. Sempre que um processador necessita de um dado, esse é copiado para a memória local, através da rede. A migração garante a exclusividade de acesso ao dado, porém, prejudica o acesso simultâneo.

Quanto à consistência de acesso aos dados, os algoritmos utilizam três abordagens:

1. **SRSW** (*Single Reader / Single Writer*): nessa abordagem há apenas um processo realizando escritas e outro realizando leituras de dados da memória compartilhada. Para esse tipo de algoritmo geralmente é utilizado um servidor central que controla o acesso à memória compartilhada, o que pode levar a um baixo desempenho;
2. **MRSW** (*Multiple Reader / Single Writer*): nessa abordagem vários processos realizam leituras enquanto somente um faz operações de escrita na memória compartilhada. Essa abordagem utiliza a replicação dos dados para servir a várias requisições de leitura. Esse tipo de algoritmo apresenta um bom desempenho para operações de leitura, visto que os processadores acessam cópias locais da memória. Porém, o desempenho para a escrita é baixo, pois é necessário que todas as cópias sejam atualizadas sempre que um conteúdo for modificado;
3. **MRMW** (*Multiple Reader / Multiple Writer*): esse tipo de algoritmo utiliza a replicação de dados tanto para a leitura quanto para a escrita de dados na memória compartilhada, permitindo, assim, a existência de múltiplos leitores e escritores. Apesar de oferecer um bom desempenho para as operações de escrita, essa abordagem utiliza excessivamente o meio de comunicação, devido às mensagens para atualização de dados.

Implementação

Um suporte de DSM pode ser classificado, de acordo com seu nível de implementação, em: *Software*, *Hardware* e Híbrido.

O primeiro tipo utiliza recursos de *software* para gerenciar distribuição, acesso e consistência dos dados localizados na memória distribuída. Esse tipo de sistema é mais difundido, pois não necessita de dispositivos específicos e, geralmente, não apresenta restrições quanto ao sistema operacional. Os sistemas de DSM em nível de *software* podem ser implementados tanto em nível de usuário quanto em nível de sistema.

Os sistemas implementados em nível de usuário não necessitam de modificações no sistema operacional, o que o torna mais flexível e independente. Nesses sistemas, todo o processamento de dados é feito pelas aplicações que acessam a memória compartilhada, ou por uma biblioteca de funções. Contudo, a utilização do espaço de usuário para esse processamento prejudica o desempenho dessas implementações.

Por outro lado, os sistemas implementados em nível de sistema são menos flexíveis e dependem de modificações no sistema operacional. Mas, apresentam um melhor desempenho, pois, todo o processamento de dados, ou uma parte dele, é feito pelo sistema operacional. Alguns exemplos de sistema de memória compartilhada distribuída no nível de software são: *Mermaid* (Zhou *et al.*, 1990), *Blizzard* (Schoinas *et al.*, 1994), *Mirage* (Fleisch & Popek, 1989), *Clouds* (Ramachandran & Khalidi, 1989), *Linda* (Ahuja *et al.*, 1986), dentre outros.

O segundo tipo de suporte a DSM necessita de componentes e dispositivos de *hardware*. O desempenho dessa abordagem é superior, uma vez que o processamento é feito por um *hardware* específico. Porém, ela é restrita, uma vez que os sistemas devem suportar e ter o mesmo dispositivo *hardware*. Entre os exemplos desse tipo de sistemas de memória compartilhada distribuída pode-se mencionar: *Dash* (Lenoski *et al.*, 1992), *SCI* (Karl *et al.*, 1999), *DDM* (Hagersten *et al.*, 1992), *Merlin* (Li *et al.*, 1995), entre outros.

No terceiro tipo de suporte a DSM, o híbrido, operações de memória compartilhada são realizadas por *softwares* e em dispositivos de *hardware* específicos. Exemplos dessa classe de sistemas de memória distribuída incluem *Plus* (Bisiani & Ravishankar, 1990), *Galactica Net* (Carter *et al.*, 1995), *Alewife* (Agarwal *et al.*, 1999), *Flash* (Heinrich *et al.*, 1994), *Typhoon* (Reinhardt *et al.*, 1994) e *Shrimp* (Liao *et al.*, 1998), entre outros.

Modelo de Consistência

Um sistema de DSM é também classificado, de acordo com o modelo de consistência, como forte ou fraco.

Um modelo de consistência forte apresenta uma menor complexidade de implementação ao custo de uma maior latência de acesso à memória. Modelos fracos, ou relaxados, utilizam técnicas de reorganização da memória, *pipelining* e sobreposição de dados, que melhoram o desempenho das operações de entrada e saída ao custo de maior complexidade computacional.

Modelos de consistência fortes são classificados em: seqüencial e controlado pelo processador. O primeiro, utiliza uma mesma seqüência de acesso à memória para todos os processadores do sistema, por exemplo, uma fila de acesso baseada no protocolo FIFO (*First In First Out*). O segundo é semelhante ao anterior, porém, a ordem dos acessos é controlada por um único dispositivo do sistema. Esse modelo, oferece um melhor desempenho, mas é dependente do *hardware* utilizado.

Modelos de consistência fracos utilizam-se de acesso comum à memória e acesso sincro-

nizado, sendo que a consistência da memória é verificada apenas quando ocorrem acessos sincronizados. Os acessos sincronizados podem ser de dois tipos: requisição (*acquire*) e liberação de páginas (*release*). Alguns exemplos de modelos de consistência fracos são apresentadas a seguir.

No modelo denominado *release consistency*, os acessos comuns de leitura e escrita podem ser realizados apenas nas páginas que foram requisitadas previamente (*acquire*) e as modificações são efetuadas após a liberação (*release*) dessas páginas. Nas técnicas *lazy release consistency* (Amza *et al.*, 1996) e *entry consistency*, as modificações das páginas não são armazenadas na memória compartilhada após a liberação das mesmas. As alterações somente são feitas na memória compartilhada na próxima requisição às páginas modificadas, sendo efetuadas de maneira tardia, visando obter maior desempenho. No caso do *lazy release consistency* a consistência é mantida por uma camada de *software*, enquanto no *entry consistency* essa é realizada por meio de recursos do compilador.

O sistema de memória compartilhada distribuída utilizado pelo MidHPC (detalhado na seção 5.6) foi desenvolvido e implementado por Dodonov (2006). Esse sistema permite o acesso distribuído aos dados por múltiplos leitores e escritores simultaneamente e o modelo de consistência utilizado foi o seqüencial.

Atualmente, um arquivo (localizado no sistema de arquivos distribuídos) é utilizado como memória e é mapeado pelos processos. Para realizar operações de leitura e escrita, comandos como *dsm_write* e *dsm_read* são utilizados. Como atividade futura, espera-se modificar esse suporte de DSM para que essas operações sejam transparentes ao usuário.

4.6 Modelos de Programação

Um modelo de programação é um conjunto de tecnologias de *softwares* para escrever algoritmos de acordo com o sistema (*hardware* ou *software*) utilizado. Segundo Lee & Talia (2003) os modelos de programação são classificados em:

1. *Estados compartilhados*: tipicamente associados com modelos de execução e linguagens síncronas direcionadas para máquinas de memória compartilhada ou de memória distribuída com alta largura de banda e de baixa latência. O *JavaSpaces* é um exemplo desse tipo de modelo;
2. *Passagem de mensagens*: processos executam em espaços de endereços distintos, onde informações são trocadas através de mensagens. Essas mensagens são geradas por funções implementadas em bibliotecas. Esse modelo força o desenvolvedor a verificar exatamente onde uma comunicação potencialmente custosa pode ocorrer. O PVM, o MPI e suas variantes são exemplos do modelo de passagem de mensagens;
3. *RPC (Remote Procedure Call)* e *RMI (Remote Method Invocation)*: esses modelos, implementados como construções de linguagem, provêem um mecanismo simples

para gerenciar computação remota. Além de gerenciar o fluxo de controle e de dados, RPC e RMI possibilitam a checagem de aridade e de tipos de argumentos. Eles também podem ser utilizados para construir outros modelos para programação em *Grids*;

4. *Ponto a ponto (P2P)*: computação P2P permite o compartilhamento de recursos computacionais e serviços sem intermediários. Esse modelo aproveita-se da capacidade computacional dos *desktops* e da conectividade de rede. Em um sistema P2P, computadores que antes eram apenas clientes também atuam com servidores, assumindo o papel que for mais eficiente para o sistema. Isso reduz a carga em servidores dedicados e permite a realização de serviços especializados eficientemente.

Dentre as ferramentas que implementam diferentes modelos de programação vale destacar o PVM (*Parallel Virtual Machine*), o MPI (*Message Passing Interface*) e o OpenMP (*Open Multi-Processing*), pois esses são os mais utilizados e os que mais se assemelham ao modelo de programação utilizado no MidHPC.

PVM é um conjunto integrado de ferramentas de *software* e bibliotecas, que foi projetado para permitir que uma rede de computadores heterogêneos possa ser utilizada como um único processador paralelo distribuído. O modelo de computação PVM é baseado na idéia de que uma aplicação é composta por várias tarefas. Cada tarefa é responsável por uma parte da carga de trabalho da aplicação. Essas tarefas são executadas na máquina virtual e a sincronização é feita através de trocas de mensagens definidas pelos usuários (Beguelin *et al.*, 1994).

O MPI é um padrão para a comunicação de dados em computação paralela definido pelo Message Passing Interface Forum (1993), diferentemente do PVM que é um conjunto de *software* e bibliotecas. Nesse padrão, os processos de uma aplicação comunicam-se por meio de funções que controlam o envio e o recebimento de mensagens. Existem diversas implementações do padrão MPI, dentre elas o LAM-MPI² e o MPICH³.

As funções do OpenMP suportam a utilização de multiprocessamento de memória compartilhada em ambiente multi-plataforma (para C/C++ e Fortran) e consiste em um conjunto de diretivas de compilação, bibliotecas de procedimentos e variáveis de ambiente que influenciam o comportamento de execução. As linhas de comando originais (processamento em série) não precisam, em geral, ser modificadas quando paralelizada com o OpenMP, diminuindo, assim, o risco de introduzir erros. Entretanto, o OpenMP executa de modo eficiente apenas em sistemas de memória distribuída. Além disso, apresenta baixo paralelismo, pois ele depende da paralelização de *loops*, mantendo seqüencial um percentual relativamente alto de códigos fora de *loops* (Quinn, 2004).

²Disponível em: <http://www.lam-mpi.org/>.

³Disponível em: <http://www-unix.mcs.anl.gov/mpi/mpich1/>.

O modelo de programação do MidHPC é baseado no padrão de *threads* POSIX, permitindo que aplicações legadas possam ser executadas com pouca ou nenhuma adaptação. Novas aplicações podem ser desenvolvidas utilizando-se conceitos amplamente difundidos (programação concorrente), sem a necessidade de adquirir conhecimentos específicos de modelos ou bibliotecas. Assim, o desenvolvedor precisará se preocupar apenas com sua aplicação e com problemas clássicos da implementação, que envolvem sincronização de variáveis e condições de corrida.

4.7 Considerações Finais

As técnicas apresentadas neste capítulo são utilizadas pelo MidHPC para otimizar o balanceamento de carga e para tornar transparente a interação do usuário com o ambiente.

O IBL é utilizado para obter o comportamento médio de uso de recursos de uma aplicação. Essa técnica é acionada antes de se iniciar uma aplicação e tem sido utilizada para parametrizar o algoritmo de escalonamento de processos. Atualmente, esse algoritmo de escalonamento utiliza Algoritmos Genéticos para otimizar o balanceamento de carga, esse algoritmo foi denominado RouteGA (Mello *et al.*, 2007) e é detalhado na seção 5.4.2.

A migração de processos, com base no resultado do algoritmo RouteGA, é conduzida por meio da transferência de arquivos de *checkpoint* entre computadores (gerados pela ferramenta *CryoPID*). A comunicação desses processos ocorre por meio de um suporte de DSM.

Para executar no MidHPC, as aplicações devem seguir o modelo de programação baseado no padrão de *threads* POSIX. Devido ao uso do modelo de programação concorrente, as aplicações podem ser desenvolvidas utilizando-se conceitos amplamente difundidos (programação concorrente), sem a necessidade de adquirir conhecimentos específicos de modelos ou bibliotecas. Assim, o desenvolvedor precisará se preocupar apenas com sua aplicação e com problemas clássicos da implementação, que envolvem sincronização de variáveis e condições de corrida.

MidHPC: *Middleware for High Performance Computing*

5.1 Considerações Iniciais

O objetivo do MidHPC (*Middleware for High Performance Computing*) é simplificar o processo de análise, projeto, implementação e execução de aplicações paralelas em ambientes heterogêneos de larga escala.

Outras implementações de *Grid* apresentadas no capítulo 3 abordaram essas questões. Entretanto, essas implementações não suprem necessidades de transparência de uso, suporte a aplicações legadas e otimização de comunicação e escalonamento.

Além de suprir tais necessidades, o MidHPC oferece suporte a aplicações legadas (*multithreads*) ao interceptar a criação de *threads* para convertê-las em processos e permitir o balanceamento de carga transparente. A comunicação entre esses processos é realizada por meio de um suporte de DSM transparente. O escalonamento de processos é conduzido com base em módulos de inteligência artificial que consideram o comportamento de aplicações.

Com o objetivo de fornecer transparência no uso de um *Grid*, o MidHPC organiza e gerencia os recursos disponíveis no ambiente por meio dos módulos **Broker Global** e **Broker Local** (ambos descritos na seção 5.2). O suporte a aplicações legadas e o balanceamento de carga são providos pelo módulo **Scheduler** (seção 5.4), que adota técnicas de inteligência artificial. Usuários acessam recursos do *Grid* e requisitam a execução de aplicações por meio de um módulo baseado no GNU/bash, denominado **Shell** (seção 5.3). A figura 5.1 apresenta a disposição desses módulos no MidHPC.

O Scheduler utiliza informações sobre capacidade dos recursos e comportamento das

Figura 5.1: Estrutura do MidHPC

aplicações para tomar decisões de escalonamento, adotando o algoritmo de balanceamento de carga RouteGA parametrizado pela técnica de inteligência artificial IBL. A parametrização é baseada em informações extraídas por um módulo denominado Monitor. Esse módulo coleta informações das aplicações durante a execução, as quais podem ser utilizadas para predição de comportamento futuro, como proposto por Dodonov (2006). Essas informações são armazenadas na base de dados e posteriormente utilizadas pelo módulo IBL para permitir ao RouteGA planejar a migração de processos com o objetivo de otimizar o tempo de resposta das aplicações.

Processos de aplicações paralelas comunicam-se de maneira transparente através de um sistema de memória compartilhada distribuída (DSM - *Distributed Shared Memory*). A implementação atual do DSM executa sobre um sistema de arquivos distribuídos (DFS - *Distributed File System*) que é acessível pelos computadores do ambiente.

O presente trabalho tem como objetivo o desenvolvimento dos módulos Broker Global, Broker Local Scheduler e Shell. Além da integração das demais ferramentas que compõem o projeto MidHPC.

5.2 Brokers Global e Local

O Broker é o módulo utilizado para conectar, de modo transparente, as redes de computadores que compõem o *Grid*. Esse módulo é composto por dois gerenciadores distintos: B_L (Broker Local) e o B_G (Broker Global). Cada B_G é responsável por gerenciar um conjunto de B_L .

O gerenciador B_L é responsável por oferecer e controlar o acesso de computadores de uma rede local ao *Grid*. Esse gerenciador armazena uma lista que contém informações sobre os computadores que executam o módulo Scheduler (o que é um requisito para oferecer recursos ao *Grid*). Essas informações incluem capacidade de processamento, latência de acesso à memória principal e virtual, vazão de leitura e escrita em disco e a carga¹ atual de cada computador em sua rede. Tais informações são resumidas em médias e desvios-padrão. Essas medidas-resumo e o número de computadores na rede do B_L são submetidos ao e armazenados pelo B_G .

A interligação entre redes é feita através dos Brokers Globais. O primeiro Broker Global iniciado exercerá o papel de Primário. As requisições de usuários, oriundas do Shell, que iniciam aplicações ou obtêm informações sobre suas execuções são atendidas pelo gerenciador B_G Primário.

O Broker Global resume as informações de cada um de seus B_L s em uma única entrada replicada em todos os Brokers Globais para fins de tolerância a falhas. Além disso, a localização (endereço de rede) dos outros B_G s também é armazenada.

A integração dos Brokers com os outros módulos do MidHPC está detalhada na seção 5.7 e os apêndices A e B apresentam informações detalhadas sobre a implementação do Broker Global e Local, respectivamente.

5.3 Shell

Para simplificar a interação do usuário foram desenvolvidas ferramentas para um Shell, utilizadas para submissão e controle da execução de aplicações. O Shell executa sobre um sistema de arquivos distribuídos que contém uma imagem do sistema operacional Linux². Atualmente, o sistema de arquivos utilizado é o NFS (*Network File System*) que centraliza a imagem do sistema operacional e oferece uma visão única dessa partição para os demais computadores do sistema, inclusive Brokers e Schedulers.

Um usuário, quando necessita executar aplicações paralelas e distribuídas, deve apenas copiar sua aplicação para a imagem única do sistema de arquivos do *Grid*. Caso a imagem do sistema operacional *Linux* não contenha alguma das bibliotecas, o computador onde a aplicação foi lançada instala as dependências³.

Para controlar a execução de aplicações e gerenciar o ambiente, o usuário pode utilizar as seguintes ferramentas implementadas no Shell:

1. **Mkill**: a aplicação *mkill* requisita que o Broker Global interrompa a execução de uma aplicação. Essa requisição será redirecionada para qualquer Scheduler que

¹Obtida pelo *load average* que é acessível por diversos comandos do *Linux*. Atualmente é utilizado o comando *w*.

²Atualmente baseada no *Kurumin Linux*.

³Utilizando a ferramenta *apt-get* para obter as bibliotecas que faltam. Essa operação é feita automaticamente.

execute um processo da aplicação, para que seja possível interrompê-lo (usando o comando *kill* do Unix);

2. **Mps**: a aplicação *mps* requisita informações sobre o ambiente. Isso inclui os Brokers Globais, Broker Locais, Schedulers, tabela de RTT (entre Brokers Globais) e aplicações (com seus processos);
3. **Mstart**: a aplicação *mstart* requisita ao Broker Global o início de uma aplicação.

O apêndice C apresenta mais informações sobre o módulo Shell. As interações com outros módulos são apresentadas na seção 5.7.

5.4 Scheduler

O módulo Scheduler tem funções de gerenciamento de recursos e de troca de informações com o Broker Local e com o algoritmo de balanceamento de carga.

As informações trocadas com o Broker Local são detalhadas na seção 5.7, visto que é por meio do Broker Local que o Scheduler se integra ao ambiente.

A troca de informações entre o módulo Scheduler e o algoritmo de balanceamento de carga é feita através de dois arquivos no padrão XML, conforme a figura 5.2. O primeiro contém dados sobre capacidades e carga dos recursos computacionais envolvidos, aplicações em execução e localização dos processos da aplicação. O segundo, contém a resposta do algoritmo de escalonamento (como os processos serão alocados nos recursos disponíveis).

Figura 5.2: Comunicação entre os módulos de Scheduler

O módulo Scheduler permite a substituição de seu algoritmo de escalonamento, uma vez que há uma interface bem definida por meio de arquivos no padrão XML.

A seguir são apresentados os algoritmos de balanceamento de carga Route, resultado de estudos realizados pelo grupo de pesquisa (Mello *et al.*, 2006b), e RouteGA, também desenvolvido pelo grupo (Mello *et al.*, 2007). O algoritmo RouteGA é atualmente utilizado como escalonador de processos do MidHPC.

5.4.1 Route

O Route é um algoritmo de balanceamento de carga totalmente distribuído, voltado para ambientes de *Grid*, onde existe uma grande quantidade de recursos, heterogeneidade, alta latência de comunicação (quando comparado com máquinas paralelas virtuais executadas em LANs) e um grande número de usuários. Os algoritmos de balanceamento de carga utilizam-se de redistribuição dinâmica de processos para prover uma distribuição equitativa, baseando-se na capacidade computacional dos recursos (Shivaratri *et al.*, 1992). O algoritmo Route foi desenvolvido à partir do conceito de roteamento de mensagens, o qual é utilizado para definir uma vizinhança de computadores. Cada computador pode distribuir processos de aplicações paralelas entre seus vizinhos.

Para entender esse algoritmo, suponha três redes de computadores distintas: α , β e γ . Cada computador, composto por diferentes capacidades de processamento (memória, disco e acesso a rede), deve executar um módulo capaz de tomar decisões de escalonamento. Depois de iniciar esse módulo em um computador $C_{n,\alpha}$ (computador n da rede α), calcula-se o atraso de comunicação dessa rede, definido como RTT_α (RTT – Round-Trip Time). O atraso (ou latência de comunicação) é o custo para transmitir e receber uma mensagem de tamanho mínimo entre dois computadores da rede α . Após essa etapa define-se o RTT máximo, $RTT_{vizinhanca}$, usado para definir a vizinhança de $C_{n,\alpha}$, dada na equação 5.1, onde k é um parâmetro de atraso para comunicação entre computadores. O valor de k pode ser definido de maneira fixa ou adaptativa, baseando-se nas capacidades das redes envolvidas no *Grid*.

$$RTT_{vizinhanca} = RTT_\alpha * k, k \geq 1 \quad (5.1)$$

Todos os computadores do ambiente que apresentarem um atraso de comunicação, em relação ao computador $C_{n,\alpha}$, menor que $RTT_{vizinhanca}$ são definidos como vizinhos. A distribuição dos processos da aplicação paralela é feita sobre tal vizinhança. Quando um computador $C_{n,\alpha}$ inicia uma aplicação paralela, ele requisita informações sobre a carga de seus vizinhos com o objetivo identificar os mais ociosos.

Como a distribuição de carga é feita entre vizinhos, e a vizinhança é definida pelo atraso de comunicação entre os computadores, o uso de uma rede de alto desempenho garante menor atraso de sincronização entre os processos da mesma aplicação paralela. Assim, se um recurso computacional está sobrecarregado, ele avaliará a carga de seus vizinhos. Migrações de processos (ou tarefas) são conduzidas utilizando o modelo de custo de migração definido por Mello & Senger (2004). Esse modelo considera a carga dos processos, o quanto foi e será executado, além do custo de transferi-los para outros computadores.

5.4.2 RouteGA

Melhorias no algoritmo Route foram realizadas gerando um novo balanceador de carga denominado RouteGA (Mello *et al.*, 2007)⁴ que otimiza a primeira alocação de processos.

Para ilustrar o problema da primeira alocação no Route, suponha que um computador c_i inicie uma aplicação paralela a_j composta por n tarefas, ou processos $(p_0, \dots, p_n)$, em um tempo t e que a vizinhança v desse computador, definida pelo atraso de comunicação (equação 5.1), é formada pelos recursos $vc_0, \dots, vc_k$, que apresentam, respectivamente, capacidades de processamento de $cap_0, \dots, cap_k$. Suponha, também, que cada recurso da vizinhança tem uma longa fila de processos gerando alta carga por longos períodos.

No Route, os processos são designados para os v vizinhos do computador c_i , que os redistribuem, de acordo com o modelo de migração proposto por Mello & Senger (2004), até convergir para áreas do *Grid* capazes de atender esses processos. Essa convergência pode ser lenta, o que aumenta o tempo de resposta das aplicações paralelas. Altos tempos de resposta implicam em demora na execução de aplicações e diminuem, conseqüentemente, o desempenho do sistema.

Essa limitação motivou o desenvolvimento do RouteGA, que utiliza algoritmos genéticos e otimiza a etapa de alocação considerando as necessidades da aplicação, a capacidade e a carga do ambiente. As necessidades (o quê, quando e quanto) são armazenadas em uma base de experiências. Um algoritmo IBL é executado utilizando essa base para encontrar informações de modo a parametrizar o algoritmo genético do RouteGA. Para isso, o IBL requer detalhes das aplicações paralelas tais como: nome do executável, parâmetros e usuário. O IBL retorna a média e o desvio padrão de uso dos recursos, incluindo a largura de banda necessária para comunicação entre processos, que são usados para definir a vizinhança para o RouteGA. Quanto maior e mais freqüente a troca de informações entre processos de uma mesma aplicação paralela, menor deve ser a latência entre os processadores vizinhos considerado na alocação do RouteGA.

O algoritmo genético gera soluções, ou cromossomos, que são avaliados de acordo com uma função de aptidão. Um exemplo de cromossomo é $i = \{0, 3, 8, 5, 7\}$, onde cada índice j do vetor representa um processo que compõe a aplicação paralela (o tamanho do cromossomo $|i| = 5$, representa o número de processos da aplicação paralela) e cada elemento do vetor representa onde o processo será alocado. Por exemplo, o processo de índice 0 será alocado no computador 0, o de índice 1 no computador 3 e assim por diante.

O algoritmo genético do RouteGA considera a função de aptidão multi-objetivo (F_i) apresentada na equação 5.2, onde: MRT_i é o tempo de resposta do processo, com maior custo computacional, de uma aplicação paralela; η é um parâmetro para evitar divisões por zero e LV_i é a variação de carga que a solução proposta causa ao ambiente (equação

⁴*Outstanding paper* no IEEE 21st International Conference on Advanced Information Networking and Applications – 444 submissões.

5.3).

$$F_i = \frac{1.0}{MRT_i + \eta} + \frac{1.0}{LV_i + \eta} \quad (5.2)$$

$$LV_i = \sum_{j=1}^{|i|} (C_j - \frac{\sum_{j=1}^{|i|} C_j}{j})^2 j \quad (5.3)$$

O tempo de resposta MRT_i é obtido pela equação 5.4 onde C_j é o custo de execução total para o processo j . A equação 5.5 define C_j , onde PM_j é o custo de processamento total do processo j , CAP_d é a capacidade do computador d onde o processo j está alocado, e CC_j é o custo de comunicação do processo j com qualquer outro processo da mesma aplicação paralela executando em outros processadores.

$$MRT_i = \max(C_j), j = 0, \dots, |i| - 1 \quad (5.4)$$

$$C_j = \frac{PM_j}{CAP_d} + CC_j \quad (5.5)$$

A equação 5.6 define como o custo de comunicação é obtido, onde $CPS(i_j, i_k)$ é o custo de comunicação entre tarefas nos índices j e k do cromossomo i que executam, respectivamente, nos processadores i_j e i_k .

$$CC_j = \sum_{j=1}^{|i|} \sum_{k=1, j \neq k}^{|i|} CPS(i_j, i_k) \quad (5.6)$$

A função multi-objetivo visa minimizar o tempo de resposta do processo com maior custo (MRT_i) e causar a menor variação de carga no ambiente, o que significa um melhor balanceamento. Quanto maior o valor F_i , melhor é a representação da solução.

Após executar o algoritmo genético por várias gerações, onde, em cada uma, n indivíduos são criados, cruzados e mutados, ele deve convergir para uma solução válida que apresente o maior valor de aptidão que, nesse caso, significa uma melhor alocação de processos no *Grid*.

Esse algoritmo é usado na primeira distribuição de processos de uma aplicação paralela no *Grid*. Depois disso, processos podem ser migrados, de acordo com o modelo de migração apresentado por Mello & Senger (2004), entre computadores vizinhos, tal como no Route.

5.5 Extração, Classificação e Predição

No MidHPC, existem duas frentes de trabalho para a extração, classificação e predição. A primeira segue o modelo proposto por Dodonov *et al.* (2006), que utiliza redes neurais e abordagens estatísticas. A segunda, mais simples, utiliza a técnica IBL para obter médias

de uso de recursos.

No modelo proposto por Dodonov *et al.* (2006), a primeira fase é responsável por monitorar a execução de aplicações utilizando uma ferramenta de análise, a qual é baseada no projeto *GridBox* (Dodonov *et al.*, 2004). Essa abordagem intercepta as seguintes chamadas pelo Monitor: leitura e escrita em disco, envio e recebimento em rede. Entre pares de eventos consecutivos, captura-se a carga de processamento e a ocupação de memória gerada pelo processo. As informações obtidas são utilizadas para fazer previsões sobre o comportamento da aplicação.

Em seguida, o comportamento da aplicação é classificado com o objetivo de redução de dimensionalidade. Essa redução elimina pequenos ruídos dos dados, que aumentam o desempenho da etapa de previsão. A classificação é feita utilizando uma arquitetura de rede neural *ART-2A*, que é adaptativa, não supervisionada e *on-line* (Mello *et al.*, 2005), classificando padrões sem conhecimento prévio.

O comportamento classificado da aplicação é utilizado por duas técnicas de previsão diferentes e independentes: a primeira cria uma cadeia de Markov para definir arcos entre os estados classificados, de acordo com a variação do comportamento da aplicação durante sua execução; a segunda constrói uma série temporal, seguindo as mudanças da aplicação, que é submetida para uma rede neural de previsão, capaz de aproximar o comportamento futuro. Essa previsão é utilizada para tomar decisões sobre a migração de processos.

A abordagem utilizando a técnica IBL faz uso de parte do modelo descrito anteriormente, mais especificamente da extração de comportamento de aplicações, para preencher a base de dados que é adotada para obter médias de ocupação de recursos, conforme detalhado na seção 4.2.

Atualmente, o monitor, a base de dados e o IBL estão integrados ao MidHPC, os quais são utilizados para caracterizar comportamento histórico de aplicações visando otimizar o balanceamento de carga (Mello *et al.*, 2007). Paralelamente, outras pesquisas (Dodonov *et al.*, 2005; Dodonov, 2006; Dodonov & Mello, 2007) têm sido desenvolvidas com o intuito de prever o comportamento de aplicações com base nas informações capturadas da execução corrente.

5.6 Suporte de DSM do MidHPC

O MidHPC utiliza um suporte de DSM para a comunicação entre processos. Para compreender o seu funcionamento considere uma aplicação α , composta por um grupo de processos α_{T_n} . Os processos α_{T_1} e α_{T_2} comunicam-se através da mudança do valor da variável x , que é compartilhada por ambos. Portanto, quando α_{T_1} modifica o valor de x , essa alteração também é percebida em α_{T_2} . Como α_{T_1} e α_{T_2} podem estar em computadores distintos, o DSM (*Distributed Shared Memory*) é utilizado para manter a consistência do valor em memória.

Quando a aplicação α inicia, o módulo Scheduler cria, no sistema de arquivos distribuídos, um arquivo⁵ f que será utilizado como memória principal pelo DSM. A função *mmap* faz a associação desse arquivo com a aplicação α . Assim, quando essa aplicação alocar um espaço de memória, ela utilizará uma área no arquivo f . O tamanho do arquivo f é determinado pela informação contida na base de conhecimento IBL (uso média da memória principal), sendo o padrão de 32Mb.

O módulo DSM intercepta as chamadas POSIX para controle de condição de corrida, como *mutex* e *semáforos*, visando criar e modificar variáveis compartilhadas armazenadas no arquivo do DSM. Esse módulo utiliza o modelo de consistência seqüencial (Adve & Ghacharloo, 1996), também adotado no padrão POSIX (IEEE, 1993). Qualquer operação de leitura e escrita é executada no arquivo do DSM e o desenvolvedor é responsável por utilizar *locks* e *unlocks* para manter a memória consistente em caso de condição de corrida (como no problema produtor/consumidor).

Para utilizar o DSM, faz-se necessária a adoção de um sistema de arquivos distribuídos – DFS (*Distributed File System*), pois o arquivo f precisa ser acessível pelos computadores que executam a aplicação α . Qualquer DFS pode ser utilizado desde que ele permita a leitura distribuída, mesmo que os dados sejam centralizados. Se os dados forem distribuídos, ou replicados, o desempenho do DSM aumenta.

Atualmente, o sistema de arquivos distribuídos utilizado em experimentos é o NFS (*Network File System*), devido à sua simplicidade de instalação e configuração. Sistemas de arquivos como o SSHFS (que provê a segurança), o AFS e o CODA (que aumentam o desempenho), dentre outros, podem ser utilizados como DFS base, o qual pode ser alterado sem haver modificações na aplicação nem em qualquer módulo do MidHPC.

5.7 Integração dos módulos

Esta seção detalha a integração e uso dos módulos do MidHPC, que devem ser iniciados em uma ordem pré-definida (figura 5.3) para colocar em funcionamento o *Grid*.

Figura 5.3: Ordem de execução dos módulos do MidHPC

⁵Utilizando a linha de comando Unix `dd if=/dev/zero of=MEMORY_FILE bs=1M count=32` – para um arquivo com 32Mb de tamanho, por exemplo.

De acordo com a figura 5.3, observa-se que, após iniciar o Broker Local, pode-se escolher entre iniciar o módulo Scheduler ou o Shell. Porém, todas as funcionalidades do MidHPC somente estarão disponíveis caso hajam módulos Schedulers iniciados, pois esses são os responsáveis pela execução de aplicações.

O primeiro módulo a ser iniciado é o Broker Global (seção 5.2) que procura por um Broker Global Primário. Se esse não responder, o Broker iniciado assume e desempenha o papel de Primário (figura 5.4).

Figura 5.4: Diagrama de Seqüência: Início do Broker (Caso 1)

Caso o Broker Primário responda, o gerenciador B_G iniciado registra-se no Primário (figura 5.5). Em seguida, dois tipos de mensagens são enviadas pelo Broker Global Primário: o primeiro, para o novo B_G com as localizações dos demais B_G s no *Grid* (endereços IP) e o segundo, para outros B_G s informando sobre a localização do novo Broker Global.

Figura 5.5: Diagrama de Seqüência: Início do Broker (Caso 2)

A busca pelo Broker Global Primário utiliza uma entrada em DNS dinâmico, tal como *broker.midhpc.usp.br*. Para verificar se esse Broker está ativo, tenta-se uma conexão com esse nome. Caso não seja bem sucedida, o Broker Global que tentou a conexão modifica a entrada no DNS e, assim, novas requisições serão direcionadas a ele. Embora o nome seja utilizado para referenciar um Broker Global como Primário, não existe diferença entre os Brokers Globais, pois eles replicam todas as informações. Caso o Primário fique inoperante, o primeiro Broker Global que requisitar alguma informação assumirá o papel

de Primário. Além disso, periodicamente, ocorre uma verificação para identificar quais Brokers estão ativos.

A distância entre os B_{GS} é definida na forma de latência do sistema de comunicação (RTT – *Round-Trip Time*) que é calculada, periodicamente, de modo a manter um valor atualizado. Essa medida é realizada através do comando *ping* e é armazenada em uma base de dados.

Após o início do Broker Global, deve-se iniciar o Broker Local que se registra no Broker Global indicado em seu arquivo de configuração. À partir desse momento, a infraestrutura básica do ambiente está criada.

Com os Brokers Global e Local em execução, o próximo módulo a ser iniciado é o Scheduler (seção 5.4). Ao iniciar, o Scheduler verifica as capacidades do computador⁶ e envia uma mensagem para o Broker Local definido no arquivo de configuração. É através dessa mensagem que o Scheduler é adicionado na lista do Broker Local que, por sua vez, retorna ao Scheduler o endereço IP do Broker Global. Em seguida, o Scheduler armazena as informações de capacidade (quantidade e desempenho de memória principal e *swap*, capacidade em Mips e desempenho de disco), juntamente com os endereços IPs dos Brokers Local e Global (utilizados para identificar a rede à qual o Scheduler faz parte), na base de dados. Posteriormente, o gerenciador do B_L atualiza suas informações, devido à adição de um novo Scheduler no ambiente, e envia uma mensagem para atualizar as informações do Broker Global (Figura 5.6).

Figura 5.6: Diagrama de Sequência: Início do Scheduler

O módulo Shell necessita que apenas o Broker Global esteja em execução, pois ele não se conecta diretamente a um Scheduler. Entretanto, nenhuma aplicação pode ser iniciada sem Schedulers em execução, pois é por meio deles que recursos computacionais

⁶Capacidade em Mips, *throughput* de leitura e escrita em disco, quantidade e desempenho de memórias principal e *swap*. Essas informações são coletadas através de ferramentas de *benchmark* proposta por Mello & Senger (2006).

são disponibilizados ao *Grid*.

Requisições de usuário são enviadas ao Broker Global, que é responsável por respondê-la. Requisições provenientes do comando *mps* solicitam ao Broker Global que colete informações sobre aplicações e ambiente, para que essas sejam exibidas para o usuário. Aquelas oriundas de um comando *mkill <gid>* fazem com que o Broker Global envie mensagens para interromper a execução dos processos da aplicação com o identificador *<gid>*. Esse *<gid>* caracteriza todos os processos da mesma aplicação.

Quando uma requisição *mstart myapp* (onde *myapp* é o nome da aplicação seguida de seus parâmetros) é recebida pelo Broker Global Primário, um B_G com menor índice de carga I_B é selecionado para iniciar a execução. O índice I_B é definido na equação 5.7, onde B_{Cap} é o somatório da capacidade de todos os computadores do Broker Global, e B_{Carga} é o somatório da carga a ser executada.

$$I_B = \frac{B_{Carga}}{B_{Cap}} \quad (5.7)$$

O Broker Global envia as informações da aplicação para o Broker Local de menor carga em sua rede, que seleciona um Scheduler, também com menor carga, para enviar a aplicação. Em seguida, essa aplicação é iniciada e suas *threads* são convertidas em processos. *Checkpoints* são gerados sobre esses processos que têm suas execuções interrompidas. O Scheduler executa o algoritmo IBL para obter as médias e desvios-padrão de consumo de CPU, memória, disco e rede. Com as informações do IBL e de todo o ambiente, o algoritmo de balanceamento de carga é executado pelo Scheduler. O algoritmo de balanceamento retorna uma distribuição de processos que será utilizada para realizar a migração. Após migrados, os processos são reiniciados e seus comportamentos, monitorados. As etapas que fazem parte da execução de uma aplicação estão demonstradas na figura 5.7.

Figura 5.7: Diagrama de Seqüência: Iniciando uma aplicação

Atualmente, há dois algoritmos de escalonamento implementados: Route e RouteGA. Ambos consideram a vizinhança para a distribuição de processos contudo, somente o

RouteGA analisa os resultados gerados pelo algoritmo IBL. Os valores médios de ocupação obtidos com o IBL são utilizados para parametrizar o algoritmo genético e obter uma boa distribuição de processos, conforme apresentado na seção 5.4.2.

Para suportar a migração, utiliza-se uma ferramenta de *checkpoint* (atualmente o *CryoPID*) que armazena o contexto dos processos no DFS na forma de arquivos binários, os quais são utilizados para reiniciar os processos em outros computadores. Após migrar os processos, o módulo Scheduler, responsável pela inicialização, atualiza as informações sobre a localização dos processos na base de dados.

5.8 Considerações Finais

Neste capítulo foram apresentados os componentes do MidHPC e suas relações, destacando as trocas de mensagens para a execução e gerenciamento de aplicações.

O modelo de programação do MidHPC torna a complexidade de escalonamento, de transferência de mensagens e de gerenciamento de ambiente transparentes ao desenvolvedor. Assim, o desenvolvedor precisa levar em consideração apenas os problemas relativos à aplicação que está desenvolvendo, como gerenciamento de condições de corrida.

Utilizando o MidHPC

6.1 Considerações Iniciais

O capítulo 5 apresentou os módulos do MidHPC e suas interações. Este capítulo detalha a instalação e os requisitos necessários para realizá-la. Apresenta, ainda, a montagem de um ambiente, mostrando os arquivos de configuração utilizados e a arquitetura construída. Finalmente, apresenta a execução de uma aplicação genérica, detalhada passo-a-passo.

6.2 Instalação

Para instalar o MidHPC e executar seus módulos, os seguinte itens são necessários:

1. Linux *kernel* 2.6.x;
2. PostgreSQL (ou outro banco de dados);
3. Compilador e máquina virtual *Java*;
4. Compilador *C/C++*.

Com esses requisitos atendidos, é necessário apenas descompactar o arquivo de instalação e executar um comando *make*. Esse comando inicia a compilação de todos os módulos do MidHPC e executa o *script* de criação da base de dados, atualmente em PostgreSQL (as tabelas da base de dados são detalhadas no apêndice I).

Em seguida, o usuário precisa modificar os arquivos de configuração dos módulos para que se adequem à sua necessidade. Após configurá-los, é possível iniciar os módulos Broker Global, Broker Local e Scheduler.

6.3 Configuração do MidHPC

Para configurar o ambiente do MidHPC é necessário definir quais computadores serão responsáveis por executar cada módulo. Além disso, é preciso definir a entrada DNS (*Domain Name Server*) que servirá de referência para o Broker Global Primário.

A seguir são apresentadas as configurações dos módulos Broker Global, Broker Local, Scheduler e Shell.

6.3.1 Configuração do Broker Global

O Broker Global necessita que o usuário configure as seguintes informações:

1. Endereço IP local utilizado;
2. Porta local;
3. Nome do Broker Global Primário;
4. Porta do Broker Global Primário.

A figura 6.1 apresenta um exemplo de arquivo de configuração que será utilizado no ambiente exemplo, detalhado na seção 6.4. Informações de implementação do Broker Global podem ser encontradas no apêndice A, inclusive a definição de cada um dos itens do arquivo de configuração.

```
<global-broker>
  <ip>192.168.0.181</ip>
  <port>2225</port>
  <bg_root>bg-usp.no-ip.info</bg_root>
  <bg_root_port>2225</bg_root_port>

  <net_latency>/midhpc/extras/net-latency.sh 5</net_latency>
  <username>postgres</username>
  <passwd></passwd>
  <driver>org.postgresql.Driver</driver>
  <url>jdbc:postgresql://192.168.0.1:5432/MIDHPCDB</url>
</global-broker>
```

Figura 6.1: Exemplo de arquivo de configuração de um Broker Global

6.3.2 Configuração do Broker Local

O Broker Local necessita que o usuário modifique as seguintes informações:

1. Endereço IP local;

2. Porta local;
3. Endereço IP do Broker Global;
4. Porta do Broker Global.

Desse modo, o Broker Local apenas se conecta ao Broker Global definido. Detalhes de implementação sobre o Broker Local encontram-se no apêndice B.

Um exemplo de arquivo de configuração do Broker Local é apresentado na figura 6.2. Esse arquivo é utilizado no ambiente exemplo construído.

```
<local-broker>  
  <ip>192.168.0.2</ip>  
  <port>2220</port>  
  <bg_ip>192.168.0.181</bg_ip>  
  <bg_port>2225</bg_port>  
</local-broker>
```

Figura 6.2: Exemplo de arquivo de configuração de um Broker Local

6.3.3 Configuração do Scheduler

Para o módulo Scheduler, o usuário precisa modificar os seguintes itens:

1. Endereço IP local;
2. Porta local;
3. Endereço IP do Broker Local;
4. Porta do Broker Local.

Assim, o Scheduler irá se conectar apenas ao Broker Local definido no arquivo de configuração. O apêndice D apresenta mais detalhes sobre o Scheduler.

O arquivo de configuração apresentado na figura 6.3 é utilizado no ambiente exemplo que é construído na seção 6.4.

6.3.4 Configuração do Shell

Todas as ferramentas do Shell compartilham o mesmo arquivo de configuração, pois, necessitam comunicar-se apenas com o Broker Global Primário. Assim, apenas a entrada DNS e a porta para o Broker Global Primário são necessárias no arquivo de configuração, que é exemplificado na figura 6.4.

O apêndice C detalha o módulo Shell e as ferramentas que dele fazem parte.

```

<scheduler>
  <ip>192.168.0.181</ip>
  <port>2215</port>
  <bl_ip>192.168.0.2</bl_ip>
  <bl_port>2220</bl_port>
  <cmd_alg>/midhpc/startalg.sh</cmd_alg>
  <cmd_info>/midhpc/schedinfo.sh</cmd_info>
  <use_ibl>1</use_ibl>
  <ibl_mask_file>/midhpc/config/config_mask.xml</ibl_mask_file>
  <cmd_ibl>/midhpc/ibl.sh</cmd_ibl>
  <cmd_meminfo>/midhpc/extras/memInfo/memInfo.sh</cmd_meminfo>
  <cmd_media>/midhpc/extras/media/medias</cmd_media>
  <pc_capacity>/midhpc/config/pc.xml</pc_capacity>
  <bench_mips>/midhpc/extras/mips/build/mips</bench_mips>
  <bench_mem>/midhpc/extras/memo/bench_memo.sh</bench_mem>
  <bench_disc>/midhpc/extras/disc/build/discio /tmp/bench_disc</bench_disc>
  <arg_usage>0.9</arg_usage>
  <arg_increment>0.05</arg_increment>
  <process_monitor>/midhpc/extras/monitor/gridbox.sh</process_monitor>
  <system_monitor>/midhpc/extras/monitor/monitor.sh</system_monitor>
  <run_threads>/midhpc/extras/monitor/run.sh</run_threads>
  <kill_app>/midhpc/extras/monitor/kill_app.sh</kill_app>
  <checkpoint>freeze</checkpoint>
  <username>postgres</username>
  <passwd></passwd>
  <driver>org.postgresql.Driver</driver>
  <url>jdbc:postgresql://192.168.0.1:5432/MIDHPCDB</url>
</scheduler>

```

Figura 6.3: Exemplo de arquivo de configuração de um Scheduler

```

<shell-tools>
  <bg_root>bg-usp.no-ip.info</bg_root>
  <bg_root_port>2225</bg_root_port>
</shell-tools>

```

Figura 6.4: Exemplo de arquivo de configuração de um Shell

6.4 Arquitetura Implementada

A arquitetura representada na figura 6.5 foi construída para que testes fossem realizados no ambiente. Nessa figura, encontram-se informações como endereço IP, porta e nome de cada computador do ambiente.

Figura 6.5: Arquitetura implementada para testes

6.4.1 Montagem do Ambiente

Inicialmente, a aplicação *mps* é executada (figura 6.6). Como não existe um Broker Global Primário definido, um erro é apresentado.

```
guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====
ERROR: Primary Broker is not active.
Impossible to show environment status.
guto@gandalf:~/midhpc$
```

Figura 6.6: Executando um *mps* sem um Broker Global

Assim, é preciso iniciar um Broker Global para que esse seja definido como Primário. Esse passo é demonstrado na figura 6.7.

```
guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181             Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
```

Figura 6.7: Iniciando um Broker Global

O detalhe da figura 6.7 mostra que o Broker Global Primário estava inativo e que essa função passou a ser exercida pelo Broker Global iniciado (IP 192.168.0.181). Agora, é possível executar um *mps* (figura 6.8) e obter uma resposta.

Com um Broker Global Primário ativo, o *mps* retorna um resultado. Porém, não existem Broker Locais ou Scheduler ativos no ambiente.

Continuando com a construção do ambiente, é iniciado um Broker Local (figura 6.9).

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info      IP: 192.168.0.181      Port: 2225) ===
** Broker Global: gandalf      IP: 192.168.0.181      Port: 2225
    Total Mips: 0.0 Load Avg: 0.0  BLs: 0  Schedulers: 0

=== Computers ===

=== RTT Table (From BG to BG) ===
guto@gandalf:~/midhpc$ █

```

Figura 6.8: Executando um *mps* com um Broker Global ativo

```

augusto@moone:~/midhpc$ make lb1
MidHPC - Local Broker
    GB IP: 192.168.0.181    GB Port: 2225
    IP: 192.168.0.2 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
█

```

Figura 6.9: Iniciando um Broker Local

Esse Broker Local (IP 192.168.0.2) conectou-se ao Broker Global, definido no arquivo de configuração, para requisitar a sua inclusão no ambiente. O resultado dessa requisição pode ser visto no detalhe da figura 6.10.

```

guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
    Primary: bg-usp.no-ip.info
    GB Primary IP: 192.168.0.181    GB Primary Port: 2225
    IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2 2220)
█

```

Figura 6.10: Resultado da inclusão de um Broker Local

Executando novamente um *mps*, é possível identificar o Broker Local que foi iniciado (figura 6.11, no detalhe).

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info      IP: 192.168.0.181      Port: 2225) ===
** Broker Global: gandalf      IP: 192.168.0.181      Port: 2225
    Total Mips: 0.0 Load Avg: 0.0  BLs: 1  Schedulers: 0
*** Broker Local: moone      IP: 192.168.0.2      Port: 2220
    Mips: 0.0      Schedulers: 0      Load Avg: 0.0

=== Computers ===

=== RTT Table (From BG to BG) ===
guto@gandalf:~/midhpc$ █

```

Figura 6.11: Resultado do *mps* após a inclusão do Broker Local

Em seguida, um outro Broker Global (IP 192.168.0.6) é iniciado (figura 6.12). Nesse

ponto, já existe um Broker Global Primário ativo e, portanto, o registro é requisitado (detalhe 1). Após o registro (figura 6.13, detalhe), o Broker Global Primário envia a lista de Brokers Globais em execução no ambiente (figura 6.12, detalhe 2).

```
augusto@brianna:~/midhpc$ make gb2
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.6 Port 2225

Primary Broker Global alive. Registering. 1
Broker Global started. Port: 2225.
New GB added (IP: 192.168.0.181). 2
```

Figura 6.12: Iniciando outro Broker Global

```
guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2 2220)
Adding GB brianna(192.168.0.6 2225)
```

Figura 6.13: Resultado da requisição de registro do novo Broker Global

Executando novamente o *mps* é possível identificar os dois Brokers Globais, sendo que um deles contém um Broker Local ativo. Como existem dois Brokers Globais ativos no ambiente, a tabela de RTT é preenchida com a distância (latência em milissegundos) entre eles (figura 6.14).

```
guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
    Total Mips: 0.0 Load Avg: 0.0 BLs: 1 Schedulers: 0
    *** Broker Local: moone IP: 192.168.0.2 Port: 2220
        Mips: 0.0 Schedulers: 0
    Load Avg: 0.0
** Broker Global: brianna IP: 192.168.0.6 Port: 2225
    Total Mips: 0.0 Load Avg: 0.0 BLs: 0 Schedulers: 0

=== Computers ===

=== RTT Table (From BG to BG) ===
    From: 192.168.0.6 To: 192.168.0.181 RTT: 0.134ms
    From: 192.168.0.181 To: 192.168.0.6 RTT: 0.117ms

guto@gandalf:~/midhpc$
```

Figura 6.14: Execução do *mps* após a inclusão do segundo Broker Global

Em seguida, mais três Brokers Locais são iniciados. A disposição desses Brokers Locais segue o arquitetura definida na figura 6.5 e o resultado de um *mps* após a inclusão é mostrado na figura 6.15.

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info) IP: 192.168.0.181 Port: 2225 ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
    Total Mips: 0.0 Load Avg: 0.0 BLs: 2 Schedulers: 0
    *** Broker Local: tera IP: 192.168.0.3 Port: 2220
        Mips: 0.0 Load Avg: 0.0 Schedulers: 0
    *** Broker Local: moone IP: 192.168.0.2 Port: 2220
        Mips: 0.0 Load Avg: 0.0 Schedulers: 0

** Broker Global: brianna IP: 192.168.0.6 Port: 2225
    Total Mips: 0.0 Load Avg: 0.0 BLs: 2 Schedulers: 0
    *** Broker Local: bebel IP: 192.168.0.7 Port: 2220
        Mips: 0.0 Load Avg: 0.0 Schedulers: 0
    *** Broker Local: brooke IP: 192.168.0.5 Port: 2220
        Mips: 0.0 Load Avg: 0.0 Schedulers: 0

=== Computers ===

=== RTT Table (From BG to BG) ===
    From: 192.168.0.6 To: 192.168.0.181 RTT: 0.126ms
    From: 192.168.0.181 To: 192.168.0.6 RTT: 0.117ms

guto@gandalf:~/midhpc$ █

```

Figura 6.15: Resultado de um *mps* com dois Brokers Globais e quatro Brokers Locais

Finalmente, é iniciado um Scheduler (figura 6.16). Esse, conecta-se ao Broker Local definido no arquivo de configuração, além disso, inclui suas informações na base de dados e recebe um identificador (sua chave primária).

```

guto@gandalf:~/midhpc$ make sched1
MidHPC - Scheduler
    GB IP: 192.168.0.181
    LB IP: 192.168.0.2    LB Port: 2220
    IP: 192.168.0.181    Port 2215

ID do Novo PC: 1
Broker Local is alive. Registering.
█

```

Figura 6.16: Iniciando um Scheduler

O Broker Local, ao receber o pedido de inclusão, insere o Scheduler em sua lista (figura 6.17).

```

augusto@moone:~/midhpc$ make lb1
MidHPC - Local Broker
    GB IP: 192.168.0.181    GB Port: 2225
    IP: 192.168.0.2 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.181 Porta: 2215 Name: gandalf Mips: 2514.437
█

```

Figura 6.17: Resultado da inclusão de um Scheduler

Ao executar um *mps* (figura 6.18), o Broker Global mostra que existe um Scheduler em sua rede (detalhe 1) e, mais abaixo (detalhe 2), é mostrada a informação do Scheduler que foi iniciado.

O ambiente completo é composto por dois Brokers Globais, quatro Brokers Locais e oito Schedulers.

Um *mps* com o ambiente completo é mostrado na figura 6.19.

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====
=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info) IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
1 Total Mips: 2514.437 Load Avg: 0.306875 BLs: 2 Schedulers: 1
*** Broker Local: tera IP: 192.168.0.3 Port: 2220
Mips: 0.0 Schedulers: 0 Load Avg: 0.0
*** Broker Local: moone IP: 192.168.0.2 Port: 2220
Mips: 2514.437 Schedulers: 1 Load Avg: 0.61375

** Broker Global: brianna IP: 192.168.0.6 Port: 2225
Total Mips: 0.0 Load Avg: 0.0 BLs: 2 Schedulers: 0
*** Broker Local: bebel IP: 192.168.0.7 Port: 2220
Mips: 0.0 Schedulers: 0 Load Avg: 0.0
*** Broker Local: brooke IP: 192.168.0.5 Port: 2220
Mips: 0.0 Schedulers: 0 Load Avg: 0.0

=== Computers ===
2
Name: gandalf IP: 192.168.0.181 BL IP: 192.168.0.2 BG IP: 192.168.0.181
Capacity: 2514.437 Mips
Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
Main Memory: 465.734375Mb Performance: (0.0022654600)x + (0.0002766800)
Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

=== RTT Table (From BG to BG) ===
From: 192.168.0.181 To: 192.168.0.6 RTT: 0.134ms
From: 192.168.0.6 To: 192.168.0.181 RTT: 0.139ms

guto@gandalf:~/midhpc$

```

Figura 6.18: Execução de um *mps* após a inclusão de um Scheduler

6.4.2 Remoção de Elementos

A remoção de um módulo do ambiente é feita através de sua interrupção por meio de um *Ctrl+C* ou *kill <pid>*. Ao ser interrompido, o módulo envia uma mensagem informando sobre sua interrupção.

Inicialmente, um Scheduler é interrompido por um *Ctrl+C* (figura 6.20). O Scheduler escolhido (IP 192.168.0.3) envia a mensagem de interrupção (no detalhe) para seu Broker Local (IP 192.168.0.3).

Ao receber a mensagem, o Broker Local remove o Scheduler da sua lista (figura 6.21, no detalhe).

Ao executar um *mps* (figura 6.22), é possível notar que o Broker Local de IP 192.168.0.3, agora, mostra apenas um Scheduler em sua rede (detalhe 1). Além disso, a informação referente ao Scheduler foi removida (detalhe 2).

Para remover um Broker Local, é necessário que seus Schedulers sejam primeiramente removidos. Ao interromper um Broker Local, uma mensagem é enviada para o Broker Global definido no arquivo de configuração (figura 6.23, no detalhe).

Ao receber a mensagem, o Broker Global remove o Broker Local de sua lista (figura 6.24, no detalhe).

Ao executar um *mps*, o Broker Local removido não mais aparece na lista e o Broker Global mostra que existe apenas um Broker Local em sua lista.

Ao finalizar um Broker Global, uma mensagem é enviada a todos os Brokers Globais (figura 6.26, no detalhe). Semelhante ao Broker Local, para finalizar um Broker Global todos os Brokers Locais e Schedulers devem ser previamente finalizados.

Assim que outro Broker Global recebe a mensagem, ele atualiza sua lista removendo o Broker Global indicado pela mensagem (figura 6.23).

Após a remoção do Broker Global, ao executar um *mps*, é possível ver que apenas 1

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====
=== Applications ===
=== Brokers (Primary: bg-usp.no-ip.info IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
    Total Mips: 10057.748 Load Avg: 14.060810546875 BLs: 2 Schedulers: 4
    *** Broker Local: tera IP: 192.168.0.3 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 27.75
    *** Broker Local: moone IP: 192.168.0.2 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.37162109374999996

** Broker Global: brianna IP: 192.168.0.6 Port: 2225
    Total Mips: 10057.748 Load Avg: 0.010312499999999999 BLs: 2 Schedulers: 4
    *** Broker Local: bebel IP: 192.168.0.7 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.0
    *** Broker Local: brooke IP: 192.168.0.5 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.041249999999999995

=== Computers ===
Name: gandalf IP: 192.168.0.181 BL IP: 192.168.0.2 BG IP: 192.168.0.181
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: moone IP: 192.168.0.2 BL IP: 192.168.0.2 BG IP: 192.168.0.181
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: tera IP: 192.168.0.3 BL IP: 192.168.0.3 BG IP: 192.168.0.181
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: aria IP: 192.168.0.4 BL IP: 192.168.0.3 BG IP: 192.168.0.181
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: brooke IP: 192.168.0.5 BL IP: 192.168.0.5 BG IP: 192.168.0.6
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: brianna IP: 192.168.0.6 BL IP: 192.168.0.5 BG IP: 192.168.0.6
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: bebel IP: 192.168.0.7 BL IP: 192.168.0.7 BG IP: 192.168.0.6
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

Name: anita IP: 192.168.0.8 BL IP: 192.168.0.7 BG IP: 192.168.0.6
    Capacity: 2514.437 Mips
    Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
    Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
    Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

=== RTT Table (From BG to BG) ===
From: 192.168.0.181 To: 192.168.0.6 RTT: 0.127ms
From: 192.168.0.6 To: 192.168.0.181 RTT: 0.131ms

guto@gandalf:~/midhpc$ █

```

Figura 6.19: Resultado de um *mps* com o ambiente completo

```

augusto@tera:~/midhpc$ make sched3
MidHPC - Scheduler
GB IP: 192.168.0.181
LB IP: 192.168.0.3 LB Port: 2220
IP: 192.168.0.3 Port 2215

ID do Novo PC: 3
Broker Local is alive. Registering.
Sending shutdown message to 192.168.0.3.
make: *** [sched3] Error 130

augusto@tera:~/midhpc$ █

```

Figura 6.20: Interrupção de um Scheduler

Broker Local está ativo (figura 6.28).

```

augusto@tera:~/midhpc$ make lb2
MidHPC - Local Broker
GB IP: 192.168.0.181 GB Port: 2225
IP: 192.168.0.3 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.3 Porta: 2215 Name: tera Mips: 2514.437
Adding Scheduler: IP: 192.168.0.4 Porta: 2215 Name: aria Mips: 2514.437
Removing Scheduler 192.168.0.3 from the environment.

```

Figura 6.21: Broker Local - removendo um Scheduler

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====
=== Applications ===
GID: 1 Name: /nfs/tabuada/tabuada Parameters: 1

=== Brokers (Primary: bg-usp.no-ip.info IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
Total Mips: 7543.311 Load Avg: 20.759319704388087 BLs: 2 Schedulers: 3
*** Broker Local: tera IP: 192.168.0.3 Port: 2220 Mips: 2514.437 Schedulers: 1 Load Avg: 41.298125250381034 1
*** Broker Local: moone IP: 192.168.0.2 Port: 2220 Mips: 5028.874 Schedulers: 2 Load Avg: 0.2205141583951415

** Broker Global: brianna IP: 192.168.0.6 Port: 2225
Total Mips: 10057.748 Load Avg: 0.046388178424207005 BLs: 2 Schedulers: 4
*** Broker Local: bebel IP: 192.168.0.7 Port: 2220 Mips: 5028.874 Schedulers: 2 Load Avg: 0.0
*** Broker Local: brooke IP: 192.168.0.5 Port: 2220 Mips: 5028.874 Schedulers: 2 Load Avg: 0.10636368407158428

=== Computers ===
Name: gandalf IP: 192.168.0.181 BL IP: 192.168.0.2 BG IP: 192.168.0.181
Capacity: 2514.437 Mips
Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)
Name: moone IP: 192.168.0.2 BL IP: 192.168.0.2 BG IP: 192.168.0.181
Capacity: 2514.437 Mips
Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)
Name: aria IP: 192.168.0.4 BL IP: 192.168.0.3 BG IP: 192.168.0.181
Capacity: 2514.437 Mips
Disc Throughput Read: 57.913Mb/s Disc Throughput Write: 39.529Mb/s
Main Memory: 465.734375Mb Performance: (0.0022054600)x + (0.0002766800)
Swap Memory: 486.30078125Mb Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)
Name: brooke IP: 192.168.0.5 BL IP: 192.168.0.5 BG IP: 192.168.0.6

```

Figura 6.22: Resultado de um *mps* após a remoção de um Scheduler

```

augusto@tera:~/midhpc$ make lb2
MidHPC - Local Broker
GB IP: 192.168.0.181 GB Port: 2225
IP: 192.168.0.3 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.3 Porta: 2215 Name: tera Mips: 2514.437
Adding Scheduler: IP: 192.168.0.4 Porta: 2215 Name: aria Mips: 2514.437
Removing Scheduler 192.168.0.3 from the environment.
Removing Scheduler 192.168.0.4 from the environment.
Sending shutdown message to 192.168.0.181.
make: *** [lb2] Error 130

augusto@tera:~/midhpc$

```

Figura 6.23: Interrompendo um Broker Local

6.5 Execução de uma Aplicação Genérica

Para executar uma aplicação será adotado o ambiente apresentado na figura 6.5.

A aplicação a ser executada é composta por uma única *thread* que executa cálculos e armazena os resultados em um arquivo. Para que possa ser executada, essa aplicação

```

guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2 2220)
Adding GB brianna(192.168.0.6 2225)
Adding Local Broker tera(192.168.0.3 2220)
app#/nfs/tabuada/tabuada 1#guto#users
Application (/nfs/tabuada/tabuada 1) sent to BG: 192.168.0.6
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
Removing Local Broker 192.168.0.3 from the environment.

```

Figura 6.24: Mensagem recebida pelo Broker Global

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
GID: 1      Name: /nfs/tabuada/tabuada      Parameters: 1

=== Brokers (Primary: bg-usp.no-ip.info      IP: 192.168.0.181      Port: 2225) ===
** Broker Global: gandalf      IP: 192.168.0.181      Port: 2225
    Total Mips: 5028.874      Load Avg: 0.24531450843123104      BLs: 1      Schedulers: 2
    *** Broker Local: moone      IP: 192.168.0.2      Port: 2220
        Mips: 5028.874      Schedulers: 2      Load Avg: 0.24531450843123104
    *** Broker Local: brianna      IP: 192.168.0.6      Port: 2225
        Total Mips: 10057.748      Load Avg: 0.07508777099163655      BLs: 2      Schedulers: 4
        *** Broker Local: bebel      IP: 192.168.0.7      Port: 2220
            Mips: 5028.874      Schedulers: 2      Load Avg: 0.0
        *** Broker Local: brooke      IP: 192.168.0.5      Port: 2220
            Mips: 5028.874      Schedulers: 2      Load Avg: 0.15916548314090462

=== Computers ===

```

Figura 6.25: Resultado de um *mps* após a remoção do Broker Local

```

augusto@brianna:~/midhpc$ make gb2
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.6 Port 2225

Primary Broker Global alive. Registering.
Broker Global started. Port: 2225.
New GB added (IP: 192.168.0.181).
Application (/nfs/tabuada/tabuada 1 - Gid: 1) sent to BL: 192.168.0.7
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
Removing Local Broker 192.168.0.7 from the environment.
Removing Local Broker 192.168.0.5 from the environment.
Sending shutdown message to 192.168.0.181.
make: *** [gb2] Error 130

augusto@brianna:~/midhpc$

```

Figura 6.26: Finalizando um Broker Global

deve estar em um diretório do ambiente, de modo que todos os computadores possam ter acesso à mesma.

A seguir é detalhado o passo-a-passo da execução dessa aplicação genérica.

6.5.1 Passo-a-Passo de Execução

A aplicação a ser executada está localizada no diretório */nfs/tabuada*. Esse diretório está numa área que todos os computadores tem acesso (figura 6.29) por meio de um

```

guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2 2220)
Adding GB brianna(192.168.0.6 2225)
Adding Local Broker tera(192.168.0.3 2220)
app#/nfs/tabuada/tabuada 1#guto#users
Application (/nfs/tabuada/tabuada 1) sent to BG: 192.168.0.6
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
Removing Local Broker 192.168.0.3 from the environment.
Removing Global Broker 192.168.0.6 from the environment.

```

Figura 6.27: Mensagem de remoção recebida pelo Broker Global

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====
=== Applications ===
      GID: 1          Name: /nfs/tabuada/tabuada          Parameters: 1

=== Brokers (Primary: bg-usp.no-ip.info      IP: 192.168.0.181      Port: 2225) ===
** Broker Global: gandalf      IP: 192.168.0.181      Port: 2225
   Total Mips: 5028.874      Load Avg: 0.22852600489927977      BLs: 1      Schedulers: 2
   *** Broker Local: moone      IP: 192.168.0.2      Port: 2220
   Mips: 5028.874      Schedulers: 2      Load Avg: 0.22852600489927977

=== Computers ===
      Name: gandalf      IP: 192.168.0.181      BL IP: 192.168.0.2      BG IP: 192.168.0.181
      Capacity: 2514.437 Mips
      Disc Throughput Read: 57.913Mb/s      Disc Throughput Write: 39.529Mb/s
      Main Memory: 465.734375Mb      Performance: (0.0022654600)x + (0.0002766800)
      Swap Memory: 486.36078125Mb      Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)
      Name: moone      IP: 192.168.0.2      BL IP: 192.168.0.2      BG IP: 192.168.0.181
      Capacity: 2514.437 Mips
      Disc Throughput Read: 57.913Mb/s      Disc Throughput Write: 39.529Mb/s
      Main Memory: 465.734375Mb      Performance: (0.0022654600)x + (0.0002766800)
      Swap Memory: 486.36078125Mb      Performance: (0.0000106100)x^2 + (-0.0027663400)x + (0.5140557300)

=== RTT Table (From BG to BG) ===
guto@gandalf:~/midhpc$ █

```

Figura 6.28: Execução de um *mps* após a finalização do Broker Global

compartilhamento via *NFS*.

```

guto@gandalf:/nfs/tabuada$ ls
Makefile tabuada* tabuada.c
guto@gandalf:/nfs/tabuada$ █

```

Figura 6.29: Localização da aplicação exemplo

Para iniciar a aplicação, a ferramenta do Shell *mstart* é utilizada. A aplicação é denominada */nfs/tabuada/tabuada* e possui parâmetro 1 (figura 6.30).

```

guto@gandalf:~/midhpc$ ./mstart /nfs/tabuada/tabuada 1
guto@gandalf:~/midhpc$ █

```

Figura 6.30: Executando a aplicação no MidHPC

Ao executar o comando *mstart*, o Shell conecta-se ao Broker Global Primário e requisita o início da aplicação. O Broker Global Primário cria um *Gid* (identificador global) para a aplicação ao adicioná-la na base de dados. Em seguida, verifica qual dos Brokers Globais

da lista possui menor carga, o escolhido, neste caso, foi o Broker Global 192.168.0.6 (figura 6.31, no detalhe).

```
guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2    2220)
Adding GB brianna(192.168.0.6    2225)
Adding Local Broker tera(192.168.0.3    2220)
app#/nfs/tabuada/tabuada 1#guto#users
Application (/nfs/tabuada/tabuada 1) sent to BG: 192.168.0.6
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
```

Figura 6.31: Broker Global Primário recebe o pedido de início de aplicação

O Broker Global selecionado, ao receber o pedido da aplicação, envia a requisição para o Broker Local com menor carga, no caso o 192.168.0.7 (figura 6.32, no detalhe).

```
augusto@brianna:~/midhpc$ make gb2
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.6 Port 2225

Primary Broker Global alive. Registering.
Broker Global started. Port: 2225.
New GB added (IP: 192.168.0.181).
Application (/nfs/tabuada/tabuada 1 - Gid: 1) sent to BL: 192.168.0.7
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
```

Figura 6.32: Broker Global repassa a aplicação para um Broker Local

Por sua vez, o Broker Local seleciona o Scheduler com menor carga para iniciar a aplicação, no caso o 192.168.0.7 (figura 6.33).

```
augusto@bebel:~/midhpc$ make lb4
MidHPC - Local Broker
GB IP: 192.168.0.6    GB Port: 2225
IP: 192.168.0.7 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.7    Porta: 2215    Name: bebel    Mips: 2514.437
Adding Scheduler: IP: 192.168.0.8    Porta: 2215    Name: anita    Mips: 2514.437
Recieved: /nfs/tabuada/tabuada 1 (Gid: 1)
Scheduler selected: 192.168.0.7 Port: 2215
Application sent to Scheduler: 192.168.0.7
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
```

Figura 6.33: Broker Local envia a aplicação para um Scheduler

Ao receber a aplicação, o Scheduler executa o IBL para obter uma predição do comportamento médio da mesma (detalhes sobre o IBL no apêndice H e na seção 4.2). Com os dados da predição, o Scheduler inicia o escalonamento de processos para definir onde a aplicação executará. Nesse caso, foi definido que a aplicação deverá executar no computador 1 (figura 6.34, detalhe 1). Assim, uma mensagem de reinício da aplicação é enviada para o computador escolhido, no caso o 192.168.0.181.


```

augusto@bebel:~/midhpc$ make sched7
MidHPC - Scheduler
  GB IP: 192.168.0.6
  LB IP: 192.168.0.7      LB Port: 2220
  IP: 192.168.0.7 Port 2215

ID do Novo PC: 7
Broker Local is alive. Registering.
Recieved: /nfs/tabuada/tabuada 1 (Gid: 1)
Starting IBL.
Load Balancing
Original: 7
New: 1
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1

```

Figura 6.34: Início da aplicação e envio da mensagem de reinício

Ao receber a mensagem, o Broker Local a encaminha para o Broker Global (figura 6.35).

```

augusto@bebel:~/midhpc$ make lb4
MidHPC - Local Broker
  GB IP: 192.168.0.6      GB Port: 2225
  IP: 192.168.0.7 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.7  Porta: 2215  Name: bebel  Mips: 2514.437
Adding Scheduler: IP: 192.168.0.8  Porta: 2215  Name: anita  Mips: 2514.437
Recieved: /nfs/tabuada/tabuada 1 (Gid: 1)
Scheduler selected: 192.168.0.7  Port: 2215
Application sent to Scheduler: 192.168.0.7
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1

```

Figura 6.35: Broker Local repassa a mensagem para o Broker Global

Em seguida, o Broker Global modifica a requisição (mudando de *restart* para *start*) e a envia para o Broker Global definido na mensagem (figura 6.36).

```

augusto@brianna:~/midhpc$ make gb2
MidHPC - Global Broker
  Primary: bg-usp.no-ip.info
  GB Primary IP: 192.168.0.181  GB Primary Port: 2225
  IP: 192.168.0.6 Port 2225

Primary Broker Global alive. Registering.
Broker Global started. Port: 2225.
New GB added (IP: 192.168.0.181).
Application (/nfs/tabuada/tabuada 1 - Gid: 1) sent to BL: 192.168.0.7
restart#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1

```

Figura 6.36: Broker Global encaminha a requisição para o Broker Global correto

No Broker Global correto, a mensagem é encaminhada para o Broker Local definido (figura 6.37).

Do mesmo modo, o Broker Local encaminha a mensagem para o Scheduler (figura 6.38).

No Scheduler, a aplicação é reiniciada por meio do *checkpoint* definido na mensagem. Nesse momento, o Monitor de Processos inicia a coleta de informações sobre o comportamento (detalhes do Monitor no apêndice E).

```

guto@gandalf:~/midhpc$ make gb1
MidHPC - Global Broker
Primary: bg-usp.no-ip.info
GB Primary IP: 192.168.0.181    GB Primary Port: 2225
IP: 192.168.0.181    Port 2225

Primary Broker Global not active.
Info: Assuming Primary Global Broker bg-usp.no-ip.info
Broker Global started. Port: 2225.
Adding Local Broker moone(192.168.0.2    2220)
Adding GB brianna(192.168.0.6    2225)
Adding Local Broker tera(192.168.0.3    2220)
app#/nfs/tabuada/tabuada 1#guto#users
Application (/nfs/tabuada/tabuada 1) sent to BG: 192.168.0.6
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1

```

Figura 6.37: Broker Global encaminha a requisição para o Broker Local definido

```

augusto@moone:~/midhpc$ make lb1
MidHPC - Local Broker
GB IP: 192.168.0.181    GB Port: 2225
IP: 192.168.0.2 Port 2220

Global Broker alive. Registering.
Local Broker started. Port: 2220.
Adding Scheduler: IP: 192.168.0.181    Porta: 2215    Name: gandalf    Mips: 2514.437
Adding Scheduler: IP: 192.168.0.2    Porta: 2215    Name: moone    Mips: 2514.437
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1

```

Figura 6.38: Broker Local encaminha a mensagem para o Scheduler

```

guto@gandalf:~/midhpc$ make sched1
MidHPC - Scheduler
GB IP: 192.168.0.181
LB IP: 192.168.0.2    LB Port: 2220
IP: 192.168.0.181    Port 2215

ID do Novo PC: 1
Broker Local is alive. Registering.
start#/nfs/tabuada/bebel-7870.checkpoint#1#192.168.0.181#192.168.0.2#192.168.0.181#1
PID: 6239
Application restarted. ←

```

Figura 6.39: No Scheduler, a aplicação é reiniciada

Assim, no *mps* é possível ver que a aplicação é composta por uma *thread*, que está execução no computador 192.168.0.181 (figura 6.40).

```

guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===
GID: 1      Name: /nfs/tabuada/tabuada      Parameters: 1
PID: 6239   Computer: gandalf      IP: 192.168.0.181

=== Brokers (Primary: bg-usp.no-ip.info    IP: 192.168.0.181    Port: 2225) ===
** Broker Global: gandalf    IP: 192.168.0.181    Port: 2225
   Total Mips: 10057.748    Load Avg: 7.970693734064699    BLs: 2    Schedulers: 4
   *** Broker Local: tera    IP: 192.168.0.3    Port: 2220
       Mips: 5028.874    Schedulers: 2    Load Avg: 15.765573459416629
   *** Broker Local: moone    IP: 192.168.0.2    Port: 2220
       Mips: 5028.874    Schedulers: 2    Load Avg: 0.17581400871276853

```

Figura 6.40: Resultado de um *mps* que mostra a aplicação em execução

Ao final da execução da aplicação, dados sobre a *thread* são removidos do ambiente. O *mps* mostra que não existem mais *threads* em execução, indicando o fim da aplicação

(figura 6.41).

```
guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===


|        |                            |               |
|--------|----------------------------|---------------|
| GID: 1 | Name: /nfs/tabuada/tabuada | Parameters: 1 |
|--------|----------------------------|---------------|


=== Brokers (Primary: bg-usp.no-ip.info IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
    Total Mips: 10057.748 Load Avg: 9.148961080837601 BLs: 2 Schedulers: 4
    *** Broker Local: tera IP: 192.168.0.3 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 18.080016024386147
    *** Broker Local: moone IP: 192.168.0.2 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.21790613728905556
```

Figura 6.41: Execução de um *mps* após o término da aplicação

Posteriormente, uma outra aplicação foi executada. Ela é composta por duas *threads* que compartilham informações e está localizada em */nfs/p-c-mutex/testmutex*. A figura 6.42 apresenta um *mps* com a aplicação *testmutex* em execução, mostrando que as *threads* estão executando nos computadores 192.168.0.7 e 192.168.0.8.

```
guto@gandalf:~/midhpc$ ./mps
===== MidHPC Grid System -- Environment Information =====

=== Applications ===


|        |                                |                 |
|--------|--------------------------------|-----------------|
| GID: 1 | Name: /nfs/tabuada/tabuada     | Parameters: 1   |
| GID: 2 | Name: /nfs/p-c-mutex/testmutex | Parameters:     |
|        | PID: 4415 Computer: bebel      | IP: 192.168.0.7 |
|        | PID: 4417 Computer: anita      | IP: 192.168.0.8 |


=== Brokers (Primary: bg-usp.no-ip.info IP: 192.168.0.181 Port: 2225) ===
** Broker Global: gandalf IP: 192.168.0.181 Port: 2225
    Total Mips: 10057.748 Load Avg: 17.49335919937947 BLs: 2 Schedulers: 4
    *** Broker Local: tera IP: 192.168.0.3 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 26.376923179103418
    *** Broker Local: moone IP: 192.168.0.2 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 8.60979521965552

** Broker Global: brianna IP: 192.168.0.6 Port: 2225
    Total Mips: 10057.748 Load Avg: 0.07813709876387004 BLs: 2 Schedulers: 4
    *** Broker Local: bebel IP: 192.168.0.7 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.0011463165283203125
    *** Broker Local: brooke IP: 192.168.0.5 Port: 2220
        Mips: 5028.874 Schedulers: 2 Load Avg: 0.11423245435877549
```

Figura 6.42: Execução de um *mps* com uma nova aplicação quem contém duas *threads*

O código da aplicação */nfs/p-c-mutex/testmutex* é mostrado nas figuras 6.43, 6.44 e 6.45 que apresentam, respectivamente, os módulos *Producer*, *Consumer* e *Main*. É importante salientar o uso das funções de acesso ao DSM para escrita (*dsm_write*), leitura (*dsm_read*) e controle de concorrência (*dsm_lock* e *dsm_unlock*). Essas funções serão modificadas em trabalhos futuros de modo a tornar completamente transparente o uso do DSM pelas *threads*.

6.6 Considerações Finais

Este capítulo mostrou a configuração e montagem de um ambiente exemplo, além da execução de uma aplicação genérica. Mostrando, passo-a-passo, o resultado de cada

```
void producer(int *i) {
    for (;;) {
        dsm_lock((void *)i);
        dsm_read((void *)i);
        (*i)++;
        printf("Producer: %d\n", *i);
        dsm_write((void *)i);
        dsm_unlock((void *)i);
        sleep(1);
    }
}
```

Figura 6.43: Trecho de código do *testmutex: Producer*

```
void consumer(int *i) {
    for (;;) {
        dsm_read((void *)i);
        printf("Costumer: %d\n", *i);
        sleep(1);
    }
}
```

Figura 6.44: Trecho de código do *testmutex: Consumer*

```
int main(int argc, char *argv[]) {
    pthread_t tid_prod, tid_cons;
    int *i;
    i = (int *)nmalloc(sizeof(i));
    *i = 0;
    pthread_create(&tid_prod, NULL, (void* (*)(void *)) &producer, i);
    pthread_create(&tid_cons, NULL, (void* (*)(void *)) &consumer, i);
    return 0;
}
```

Figura 6.45: Trecho de código do *testmutex: Main*

comando e as mensagens que são trafegadas.

Conclusões

7.1 Conclusões e Contribuições

O desenvolvimento de uma aplicação distribuída pode ser dividido em três etapas: análise, projeto e implementação.

Na análise, o problema e as técnicas computacionais necessárias para resolvê-lo são definidos. Na etapa de projeto, a modularização é um dos aspectos a ser considerado e os módulos devem ser o mais independentes possível, de modo a permitir execução em paralelo com o mínimo de comunicação e, assim, possibilitar o aumento de desempenho.

A modularização deve considerar as características das tarefas, tais como: processamento, comunicação, memória e acesso a disco. Na etapa denominada implementação, os módulos são escritos segundo modelos de programação tais como MPI (MPI Forum, 1997), PVM (Beguelin *et al.*, 1994), OpenMP (Quinn, 2004).

Após a implementação, é possível identificar uma outra fase relevante, a execução. Durante essa fase, as tarefas da aplicação são escalonadas de acordo com a disponibilidade de recursos e, tradicionalmente, é função do desenvolvedor avaliar o desempenho desses, a fim de definir como o escalonamento será feito. A complexidade dessa definição é alta, pois é preciso conhecer o comportamento da aplicação durante a execução, assim como o dos recursos computacionais disponíveis, a fim de evitar sua ociosidade em detrimento da sobrecarga de outros (Casavant & Kuhl, 1988). Uma alternativa para reduzir essa complexidade é o uso de algoritmos de balanceamento de carga, que automatizam e homogenizam a distribuição de processos baseando-se na ocupação de recursos. Tais algoritmos visam distribuir os processos igualmente entre os computadores do sistema (Shivaratri *et al.*, 1992).

A modularização, na etapa de projeto, e a alocação de tarefas, na etapa de execução,

são os problemas mais complexos no desenvolvimento e utilização de uma aplicação paralela e distribuída. Visando abordar tais problemas, o projeto MidHPC (*Middleware for High Performance Computing*) foi proposto, o qual é composto pelos módulos: **Shell**, responsável pela submissão de aplicações, fazendo a interface com usuários e desenvolvedores; **Broker**, que conecta as redes que compõem o *Grid* e é responsável por responder as requisições do usuário; e **Scheduler**, que obtém informações sobre os recursos e sobre a aplicação para tomar decisões sobre o escalonamento. Além desses três módulos principais, existem outros secundários, tais como o **Monitor** e o **DSM**. O Monitor é responsável por coletar informações para o Scheduler e para a realização de predições sobre o comportamento futuro da aplicação (utilizando a técnica **IBL**). A predição possibilita que o MidHPC tome decisões sobre a migração de tarefas para otimizar o uso dos recursos. O **DSM** é um suporte a memória compartilhada distribuída que permite a comunicação, de maneira transparente, entre as tarefas da aplicação sendo, também, o responsável por distribuir os dados no *Grid* de modo a otimizar o acesso a escrita e a leitura.

A solução utilizada no MidHPC para reduzir as dificuldades da modularização e da implementação é utilizar um modelo de programação conhecido pelo desenvolvedor, pois, geralmente, ele precisa adquirir conhecimentos sobre linguagens e bibliotecas específicas para desenvolver aplicações paralelas e distribuídas.

O modelo de programação do MidHPC é baseado no padrão de *threads* POSIX, para que aplicações legadas possam ser executadas no MidHPC com pouca ou nenhuma adaptação. Novas aplicações podem ser desenvolvidas utilizando-se conceitos amplamente divulgados, sem a necessidade de adquirir conhecimentos específicos de um modelo ou biblioteca. Assim, o desenvolvedor precisará se preocupar apenas com sua aplicação e com problemas clássicos da implementação, que envolvem sincronização de variáveis e condições de corrida.

Para simplificar a etapa de execução o MidHPC usa, no módulo Scheduler, o algoritmo de balanceamento de carga RouteGA(Mello *et al.*, 2007), o qual se utiliza de conhecimento prévio do comportamento da aplicação para parametrizar um algoritmo genético e, assim, gerar uma distribuição otimizada de processos sobre computadores.

A quantidade de recursos que o RouteGA considera no escalonamento é limitada pela quantidade de comunicação entre os processos, quanto maior a troca de informação entre os processos, menor deve ser a distância entre os computadores em termos de atraso de rede. Aqueles cujo atraso é menor do que o definido pelo Scheduler, baseado no conhecimento prévio da aplicação, são considerados vizinhos. Com essa distribuição, o próprio *middleware* se encarrega de fazer a migração dos processos na vizinhança e, mesmo em máquinas diferentes, a comunicação não é alterada devido ao uso do DSM.

Além de simplificar o desenvolvimento e a execução de uma aplicação paralela e distribuída, o MidHPC permite que seus próprios módulos, ou componentes desses, sejam substituídos sem que haja a necessidade de modificar outra parte do *middleware*. Por

exemplo, é possível trocar o algoritmo de balanceamento de carga, utilizado no módulo Scheduler, do RouteGA para o Route (Mello *et al.*, 2006a,b) e manter os outros módulos inalterados. Essa flexibilidade ainda permite que o MidHPC utilize vários algoritmos de balanceamento de carga diferentes ao mesmo tempo, desde que em módulos Scheduler diferentes. O único requisito para que essa troca seja realizada é que o novo módulo atenda ao protocolo de mensagens do MidHPC.

7.2 Colaborações

A realização do presente trabalho contou com a colaboração de Rodrigo Fernandes de Mello, desenvolvendo os módulos referentes a Interceptação de *Threads*, Monitor e IBL (em Java). Evgueni Dodonov colaborou desenvolvendo os módulos de DSM e Monitor. Finalmente, o Prof. Dr. Luciano Senger, da Universidade Estadual de Ponta Grossa – Paraná, colaborou com o desenvolvimento do módulo IBL (em C). O Algoritmo de escalonamento RouteGA foi desenvolvido em colaboração com Rodrigo Fernandes de Mello e Luciano Senger (Mello *et al.*, 2007).

7.3 Publicações geradas

Este projeto de mestrado resultou na publicação dos seguintes artigos:

- MELLO, R. F. de; **ANDRADE FILHO, J. A.**; SENER, L. J.; YANG, L. T.. *RouteGA: A Grid Load Balancing Algorithm with Genetic Support*. In: The IEEE 21st International Conference on Advanced Information Networking and Applications, 2007, Ontário. The IEEE 21st International Conference on Advanced Information Networking and Applications. Ontário : IEEE Computer Society, 2007. v. 1. p. 1-8;
- MELLO, R. F. de; **ANDRADE FILHO, J. A.**; DODONOV, E.; LI, K.-C.; YANG, L. T.. *Automatic and Transparent High Performance Computing Support for Grids*. In: IEEE TENCON 2007, 2007, Taipei. Proceedings of the IEEE TENCON 2007, 2007. v. 1. p. 1-4;
- MELLO, R. F. de; **ANDRADE FILHO, J. A.**; DODONOV, E.; ISHII, R. P.; YANG, L. T.. *Optimizing distributed data access in Grid environments by using artificial intelligence techniques*. In: The Fifth International Symposium on Parallel and Distributed Processing and Applications, 2007, Niagara Falls. Proceedings on The Fifth International Symposium on Parallel and Distributed Processing and Applications, 2007. v. 1. p. 1-12.

Além da submissão do seguinte artigo, que está em processo de análise:

-
- **ANDRADE FILHO, J. A.;** MELLO, R. F. de; SENER, L. J.; YANG, L. T.; LI, K.-C.. *Toward an Efficient Middleware for Multithreaded Applications in Computational Grid*. Submetido ao: 2008 IEEE 11th International Conference on Computational Science and Engineering. Resultado a partir de: 21 de Março de 2008.

O artigo *RouteGA: A Grid Load Balancing Algorithm with Genetic Support* recebeu o prêmio de *Outstanding paper* no IEEE *21st International Conference on Advanced Information Networking and Applications* (AINA 2007) – dentre 444 submissões.

7.4 Trabalhos futuros

O desenvolvimento do MidHPC abordado neste trabalho de mestrado pode ser ampliado, aprofundando-se nos temas a seguir enumerados:

1. Pesquisas relativas à segurança do ambiente, especialmente no que se refere ao acesso ao ambiente e compartilhamento de dados e informações;
2. Desenvolvimento de outros algoritmos de escalonamento de processos ou o aprimoramento dos já existentes;
3. Melhoria da interface com o usuário, pois a atual é composta apenas por aplicações de linha de comando;
4. Análise de desempenho do *middleware* em ambiente real, utilizando diversos algoritmos de escalonamento de processos;
5. Comparação dos resultados experimentais do RouteGA (Mello *et al.*, 2007) e do Route (Mello *et al.*, 2006a,b) com os resultados obtidos em ambiente real;
6. Sistemas de extração, classificação e predição de comportamento das aplicações (em estudo pelo aluno Evgueni Dodonov);
7. Distribuição e replicação de dados, tanto das informações do Broker quanto dos dados do usuário;
8. Utilização de uma base de dados distribuída para permitir a replicação de informações e melhorar o desempenho no acesso a dados em locais distintos do *Grid*;
9. Estudos para verificar a possibilidade de rebalanceamento periódico de cargas.

Broker Global

A.1 Visão Geral

O módulo Broker é utilizado para interligar transparentemente redes de computadores e construir uma plataforma de *Grid*. Esse módulo é composto por dois gerenciadores distintos: BG (Broker Global) e BL (Broker Local). Vários BLs podem estar ligados a um único BG. Este apêndice trata especificamente do *Broker Global*.

A conexão entre Brokers através da Internet é controlada pelo gerenciador BG. Informações sobre aplicações em execução no *Grid* e sobre os BLs são armazenadas nesses gerenciadores que também armazenam réplicas de informações de outros Brokers Globais e a localização dos mesmos. Essa replicação mantém o ambiente em execução quando qualquer Broker Global falha. Somente aplicações nos BLs do Broker que falhou são interrompidas.

Requisições de usuário, originadas do Shell, são atendidas por um gerenciador BG. Essas requisições iniciam aplicações ou obtêm informações presentes em quaisquer dos Brokers Globais no ambiente.

Um BG gerencia vários BLs e sumariza algumas de suas informações, que são médias e desvios-padrões das seguintes características: capacidade em MIPS, uso e performance de memória (principal e *swap*). Essas informações são replicadas entre BGs para tolerância a falhas.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Troca de mensagens;
- Mensagens de saída;

- Mensagens de entrada;
- Interações SQL; e
- Arquivo de configuração.

A.2 Interações

A figura A.1 apresenta as interações do Broker Global com os demais módulos do MidHPC. Essas interações são descritas na tabela A.1.

Figura A.1: Interações do Broker Global com outros módulos

Tabela A.1: Descrição das interações do Broker Global com outros módulos

Origem	Destino	Como	Descrição
Shell	Broker Global	Mensagem	Requisição (<i>mstart</i> , <i>mkill</i> , <i>mps</i>)
Broker Global	Shell	Mensagem	Resposta a requisições
Broker Global	Broker Local	Mensagem	Encaminhar requisições do Broker Global
Broker Local	Broker Global	Mensagem	Informações do ambiente, encaminhar requisições
Broker Global	Broker Global Primário	Mensagem	Registro; Encaminhar requisições do Scheduler.
Broker Global Primário	Broker Global	Mensagem	Informações do ambiente; Encaminhar requisições.
Broker Global Primário	Base de dados	SQL	Inserir aplicações; Atualizar valores de RTT.
Base de dados	Broker Global Primário	SQL	Novo GID; Informações do ambiente.

A.3 Troca de Mensagens

A figura A.2 apresenta as mensagens que são trafegadas pelo Broker Global. Essas mensagens são detalhadas nas seções seguintes.

Como o Broker Global Primário é apenas um Broker Global que é designado pelo nome DNS (*bg_root*, no arquivo de configuração), as mensagens trocadas com o Primário serão mostradas apenas na seção de mensagens de saída (seção A.4).

Figura A.2: Interações do Broker Global com outros módulos - mensagens

A.4 Mensagens de Saída

A seguir, são detalhadas as mensagens que são originadas no Broker Global com destino a outros módulos.

- stop # procKey # procPid # gbIP # lbIP # schedIP
 - Origem: Broker Global
 - Destino: Broker Global Primário
 - Função: Requisição para interromper a execução do processo *procPid* e criar um *checkpoint* sobre o mesmo
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- stop_p # procKey # procPid # gbIP # lbIP # schedIP

-
- Origem: Broker Global Primário
 - Destino: Broker Global
 - Função: Requisição ao Broker Global *gbIP* para parar o processo *procPid*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **stop_p # procKey # procPid # gbIP # lbIP # schedIP**
 - Destino: Broker Local
 - Função: Requisição ao Broker Local *lbIP* para parar o processo *procPid*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **restart # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Origem: Broker Global
 - Destino: Broker Global Primário
 - Função: Requisita o reinício de um processo (*checkpointt*) da aplicação *appGid*
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **start # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Origem: Broker Global Primário

- Destino: Broker Global
- Função: Requisita ao Broker Global *gbIP* o reinício da execução do processo
- Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **start # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Destino: Broker Local
 - Função: Requisita ao Broker Local *lbIP* o reinício da execução do processo
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **kill # procPid # procKey # compKey # gbIP # lbIP # schedIP**
 - Origem: Broker Global Primário
 - Destino: Broker Global (listado em *gbIP*)
 - Função: Requisita ao Scheduler *schedIP* a interrupção do processo *procKey*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **compKey**: a chave primária (no banco de dados) do computador que executa o processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **kill # procPid # procKey # compKey # gbIP # lbIP # schedIP**
 - Origem: Broker Global

-
- Destino: Broker Local (listado em *lbIP*)
 - Função: Requisita ao Scheduler *schedIP* a interrupção do processo *procKey*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **compKey**: a chave primária (no banco de dados) do computador que executa o processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **app # appGid # appName [Params] # userName**
 - Origem: Broker Global Primário
 - Destino: Broker Global (aquele com menor carga)
 - Função: requisita o início da aplicação *appName [Params]*
 - Parâmetros:
 - * **appGid**: Gid da aplicação criado pelo Broker Global Primário;
 - * **appName [Params]**: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName**: o usuário que requisitou o início da aplicação.
 - **app # appGid # appName [Params] # userName**
 - Origem: Broker Global
 - Destino: Broker Local
 - Função: requisita o início da aplicação *appName [Params]*
 - Parâmetros:
 - * **appGid**: Gid da aplicação criado pelo Broker Global Primário;
 - * **appName [Params]**: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName**: o usuário que requisitou o início da aplicação.
 - **add_bg # gbIP # gbPort # gbName # loadAvg # lbAmmount # totalMips**
 - Origem: Broker Global
 - Destino: Broker Global Primário
 - Função: Requisita a inclusão de um Broker Global no ambiente

- Parâmetros:
 - * **gbIP**: endereço IP do Broker Global
 - * **gbPort**: porta do Broker Global
 - * **gbName**: nome do computador que executa o Broker Global
 - * **loadAVG**: média da carga dos Brokers Locais
 - * **lbAmmount**: quantidade de Brokers Locais na rede do Broker Global
 - * **totalMips**: Somatório de mips dos Brokers Locais
- **new_bg # gbIP # gbPort # gbName # loadAvg # lbAmmount # totalMips**
 - Origem: Broker Global Primário
 - Destino: Broker Global (outros BGs no ambiente)
 - Função: Informa a inclusão de um novo Broker Global no ambiente
 - Parâmetros:
 - * **gbIP**: endereço IP do Broker Global
 - * **gbPort**: porta do Broker Global
 - * **gbName**: nome do computador que executa o Broker Global
 - * **loadAVG**: média da carga dos Brokers Locais
 - * **lbAmmount**: quantidade de Brokers Locais na rede do Broker Global
 - * **totalMips**: Somatório de mips dos Brokers Locais
- **load_bg # gbIP # loadAvg # lbAmmount # totalMips**
 - Origem: Broker Global
 - Destino: Outros Brokers Globais no ambiente
 - Função: Atualizar as informações de carga do Broker Global
 - Parâmetros:
 - * **gbIP**: endereço IP do Broker Global a ser atualizado
 - * **loadAVG**: média da carga dos Brokers Locais
 - * **lbAmmount**: quantidade de Brokers Locais na rede do Broker Global
 - * **totalMips**: Somatório de mips dos Brokers Locais
- **del_bg # gbIP**
 - Origem: Broker Global
 - Destino: Outros Brokers Globais no ambiente
 - Função: Requisita a remoção do Broker Global *gbIP* da lista de Brokers Globais ativos

-
- Parâmetros:
 - * **gbIP**: endereço IP do Broker Global a ser removido.
 - **mps #**
 - Origem: Broker Global Primário
 - Destino: Broker Global
 - Função: Requisita informações adicionais sobre o ambiente
 - Parâmetros:
 - * nenhum.

A.5 Mensagens de Entrada

A seguir são detalhadas as mensagens originadas de outros módulos com destino ao Broker Global.

- **add_lb # lbIP # lbPort # lbName # loadAvg # schedulerAmmount # totalMips**
 - Origem: Broker Local
 - Destino: Broker Global
 - Função: Requisita a inclusão de um Broker Local na rede do Broker Global
 - Parâmetros:
 - * **lbIP**: endereço IP do Broker Local;
 - * **lbPort**: porta do Broker Local;
 - * **lbName**: nome do computador que executa o Broker Local;
 - * **loadAvg**: média da carga dos Schedulers
 - * **schedulerAmmount**: quantidade de Schedulers na rede do Broker Local
 - * **totalMips**: Somatório de mips dos Schedulers
- **stop # procKey # procPid # gbIP # lbIP # schedIP**
 - Origem: Broker Local
 - Destino: Broker Global Primário
 - Função: Requisição para interromper a execução do processo *procPid* e criar um *checkpoint* sobre o mesmo
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;

- * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **restart # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Origem: Broker Local
 - Destino: Broker Global
 - Função: Requisita o reinício de um processo (*checkpoint*) da aplicação *appGid*
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **kill # appGid**
 - Origem: Shell
 - Destino: Broker Global Primário
 - Função: Requisita a interrupção da aplicação *appGid*
 - Parâmetros:
 - * **appGid**: o identificador da aplicação a ser finalizada.
 - **del_lb # lbIP**
 - Origem: Broker Local
 - Destino: Broker Global
 - Função: Requisita a remoção do Broker Local *lbIP* da lista de Brokers Locais
 - Parâmetros:
 - * **lbIP**: endereço IP do Broker Local a ser removido.
 - **load_lb # lbIP # loadAvg # schedulerAmmount # totalMips**
 - Origem: Broker Local
 - Função: Atualizar as informações de carga do Broker Local
 - Parâmetros:

-
- * `lbIP`: endereço IP do Broker Local a ser atualizado;
 - * `loadAvg`: média da carga dos Schedulers;
 - * `schedulerAmmount`: quantidade de Schedulers na rede do Broker Local;
 - * `totalMips`: Somatório de mips dos Schedulers.
- `ps #`
 - Origem: Shell
 - Destino: Broker Global Primário
 - Função: Requisita informações sobre o ambiente (Brokers Globais e Locais, Schedulers e aplicações em execução)
 - Parâmetros:
 - * nenhum.
- `app # appId # appName [Params] # userName`
 - Origem: Shell
 - Destino: Broker Global Primário
 - Função: requisita o início da aplicação `appName [Params]`
 - Parâmetros:
 - * `appId`: Gid da aplicação criado pelo Broker Global Primário;
 - * `appName [Params]`: aplicação (com seus parâmetros) a ser iniciada;
 - * `userName`: o usuário que requisitou o início da aplicação.

A.6 Interações SQL

O Broker Global atualiza as informações na base de dados quando:

- Uma nova aplicação é iniciada e um *gid* é definido;
- O valor de RTT (latência de comunicação) para outro Broker Global é atualizado;
- O usuário requisita um *mips*;
- O usuário requisita um *kill*.

A.7 Arquivo de configuração

A seguir é mostrado um exemplo de um arquivo de configuração do Broker Global.

```
<global-broker>
  <ip>192.168.0.181</ip>
  <port>2225</port>
  <bg_root>bg-usp.no-ip.info</bg_root>
  <bg_root_port>2225</bg_root_port>
  <net_latency>/midhpc/extras/net-latency.sh 5</net_latency>

  <username>postgres</username>
  <passwd></passwd>
  <driver>org.postgresql.Driver</driver>
  <url>jdbc:postgresql://192.168.0.1:5432/MIDHPCDB</url>
</global-broker>
```

Onde:

- **<ip>**: Endereço IP do Broker Global;
- **<port>**: Porta do Broker Global;
- **<bg_root>**: Nome do Broker Global Primário (será convertido para endereço IP);
- **<bg_root_port>**: Porta do Broker Global Primário;
- **<net_latency>**: Programa para calcular a latência de rede (atualmente usando *ping*);
- **<username>**: Usuário do banco de dados;
- **<passwd>**: Senha do usuário do banco de dados;
- **<driver>**: Driver JDBC;
- **<url>**: Url JDBC para a base de dados.

Broker Local

B.1 Visão Geral

O módulo Broker é utilizado para interligar transparentemente redes de computadores e construir uma plataforma de *Grid*. Esse módulo é composto por dois gerenciadores distintos: BG (Broker Global) e BL (Broker Local). Vários BLs podem estar ligados a um único BG. Este apêndice trata especificamente do *Broker Local*.

O gerenciador BL é responsável por oferecer e controlar o acesso ao *Grid* por recursos em uma rede local. Esse gerenciador armazena uma lista com informações sobre todos os recursos que executam o Scheduler, o que é um requisito para participar do *Grid*. Essas informações incluem capacidade em mips, carga, *throughput* de leitura/escrita em disco e memória.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Troca de mensagens;
- Mensagens de saída;
- Mensagens de entrada; e
- Arquivo de configuração.

B.2 Interações

A figura B.1 apresenta as interações do Broker Local com os demais módulos do MidHPC. Essas interações são descritas na tabela B.1.

Figura B.1: Interações do Broker Global com outros módulos

Tabela B.1: Descrição das interações do Broker Local com outros módulos

Origem	Destino	Como	Descrição
Broker Local	Broker Global	Mensagem	Registro; Encaminhar requisições do Scheduler.
Broker Global	Broker Local	Mensagem	Encaminhar requisições (de usuários ou BGs).
Broker Local	Scheduler	Mensagem	Encaminhar requisições do Broker Global.
Scheduler	Broker Local	Mensagem	Registro; Atualização de carga; Requisições (migração e reinício de processos.)

B.3 Troca de Mensagens

A figura B.2 apresenta as mensagens que são trafegadas pelo Broker Global. Essas mensagens são detalhadas nas seções seguintes.

Figura B.2: Interações do Broker Local com outros módulos - mensagens

B.4 Mensagens de Saída

A seguir, são detalhadas as mensagens que são originadas no Broker Local com destino a outros módulos.

- add_bl # lbIP # lbPort # lbName # loadAvg # schedulerAmmount # totalMips
 - Destino: Broker Global
 - Função: Requisita a inclusão do Broker Local na rede do Broker Global
 - Parâmetros:
 - * lbIP: endereço IP do Broker Local;

- * **lbPort**: porta do Broker Local;
 - * **lbName**: nome do computador que executa o Broker Local;
 - * **loadAvg**: média da carga dos Schedulers;
 - * **schedulerAmmount**: quantidade de Schedulers na rede do Broker Local;
 - * **totalMips**: Somatório de mips dos Schedulers.
- **stop # procKey # procPid # gbIP # lbIP # schedIP**
 - Destino: Broker Global
 - Função: Requisição para interromper a execução do processo *procPid* e criar um *checkpoint* sobre o mesmo
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **stop_p # procKey # procPid # gbIP # lbIP # schedIP**
 - Destino: Scheduler
 - Função: Requisição ao Scheduler *schedIP* para parar o processo *procPid* e criar um *checkpoint* sobre o mesmo
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **restart # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Destino: Broker Global
 - Função: Requisita o reinício de um processo (*checkpoint*) da aplicação *appGid*
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;

-
- * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **start # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Destino: Scheduler
 - Função: Requisita ao Scheduler *schedIP* o reinício da execução do processo
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **del_lb # lbIP**
 - Destino: Broker Global
 - Função: Requisita a remoção do Broker Local *lbIP* da lista de Brokers Locais
 - Parâmetros:
 - * **lbIP**: endereço IP do Broker Local a ser removido.
- **load_lb # lbIP # loadAvg # schedulerAmount # totalMips**
 - Destino: Broker Global
 - Função: Atualizar as informações de carga do Broker Local
 - Parâmetros:
 - * **lbIP**: endereço IP do Broker Local;
 - * **loadAvg**: média da carga dos Schedulers;
 - * **schedulerAmount**: quantidade de Schedulers na rede do Broker Local;
 - * **totalMips**: Somatório de mips dos Schedulers.
- **app # appGid # appName [Params] # userName**
 - Destino: Scheduler
 - Função: requisita o início da aplicação *appName [Params]*
 - Parâmetros:

- * **appGid**: Gid da aplicação criado pelo Broker Global Primário;
 - * **appName [Params]**: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName**: o usuário que requisitou o início da aplicação.
- **kill # procPid # procKey # compKey # gbIP # lbIP # schedIP**
 - Destino: Scheduler (listado em *schedIP*)
 - Função: Requisita ao Scheduler *schedIP* a interrupção do processo *procPid*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **compKey**: a chave primária (no banco de dados) do computador que executa o processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.

B.5 Mensagens de Entrada

A seguir são detalhadas as mensagens originadas de outros módulos com destino ao Broker Local.

- **add_s # schedIP # schedPort # schedName # schedMips # schedLoad**
 - Origem: Scheduler
 - Função: Requisita a inclusão de um Scheduler na rede do Broker Local
 - Parâmetros:
 - * **schedIP**: endereço IP do Scheduler;
 - * **schedPort**: porta do Scheduler;
 - * **schedName**: nome do computador que executa o Scheduler;
 - * **schedMips**: Capacidade em Mips do Scheduler;
 - * **schedLoad**: Carga atual do Scheduler.
- **stop # procKey # procPid # gbIP # lbIP # schedIP**
 - Origem: Scheduler
 - Função: Requisição para interromper a execução do processo *procPid* e criar um *checkpoint* sobre o mesmo.

-
- Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **stop_p # procKey # procPid # gbIP # lbIP # schedIP**
 - Origem: Broker Global
 - Função: Requisição ao Scheduler *schedIP* para parar o processo *procPid* e criar um *checkpoint* sobre o mesmo.
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **restart # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Origem: Scheduler
 - Função: Requisita o reinício de um processo (*checkpoint*) da aplicação *appGid*
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - **start # checkpoint # appGid # gbIP # lbIP # schedIP**
 - Origem: Broker Global
 - Função: Requisita ao Scheduler *schedIP* o reinício da execução do processo
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;

- * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **kill # appGid**
 - Origem: Broker Global
 - Função: Requisita a interrupção da aplicação *appGid*
 - Parâmetros:
 - * **appGid**: o identificador da aplicação a ser finalizada.
 - **app # appGid # appName [Params] # userName**
 - Origem: Broker Global
 - Função: requisita o início da aplicação *appName [Params]*
 - Parâmetros:
 - * **appGid**: Gid da aplicação criado pelo Broker Global Primário;
 - * **appName [Params]**: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName**: o usuário que requisitou o início da aplicação.
 - **load_s # schedIP # schedLoad**
 - Origem: Scheduler
 - Função: Atualizar as informações de carga do Scheduler
 - Parâmetros:
 - * **schedIP**: endereço IP do Scheduler a ser atualizado;
 - * **schedLoad**: carga atual do Scheduler.
 - **del_s # schedIP**
 - Origem: Scheduler
 - Função: Requisita a remoção do Scheduler *schedIP* da lista dos Schedulers
 - Parâmetros:
 - * **schedIP**: endereço IP do Scheduler a ser removido.

B.6 Arquivo de configuração

A seguir é mostrado um exemplo de um arquivo de configuração do Broker Local.

```
<local-broker>
  <ip>192.168.0.181</ip>
  <port>2220</port>
  <bg_ip>192.168.0.181</bg_ip>
  <bg_port>2225</bg_port>
</local-broker>
```

Onde:

- **<ip>**: Endereço IP do Broker Local;
- **<port>**: Porta do Broker Local;
- **<bg_ip>**: Endereço IP do Broker Global;
- **<bg_port>**: Porta do Broker Global.

C.1 Visão Geral

Para iniciar uma aplicação paralela, o usuário precisa apenas copiar os binários e bibliotecas para o sistema de arquivos distribuídos, sobre o qual está a imagem do sistema. Depois disso, o usuário inicia a aplicação executando o comando *mstart* e o MidHPC trata da distribuição das tarefas. O Shell também apresenta ferramentas semelhantes ao *kill* e ao *ps*, encontradas no Unix, para auxiliar no gerenciamento de processos e do ambiente.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Requisições; e
- Arquivo de configuração.

C.2 Interações

A figura C.1 apresenta as interações do Shell com os demais módulos do MidHPC. Essas interações são descritas na tabela C.1.

Figura C.1: Interações do Shell com outros módulos

Tabela C.1: Descrição das interações do Shell com outros módulos

Origem	Destino	Como	Descrição
Shell	Broker Global Primário	Mensagem	Requisições(<i>mstart</i> , <i>mkill</i> , <i>mps</i>).
Broker Global Primário	Shell	Mensagem	Resposta a requisições.

C.3 Requisições

A seguir, são detalhadas as requisições de usuário que são feitas através do Shell. Essas requisições são geradas pelos comandos: *mkill*, *mps* e *mstart*.

- **Mkill:** `kill # appGid`
 - A aplicação *mkill* requisita que o Broker Global interrompa a execução de uma aplicação. Essa requisição será redirecionada para qualquer Scheduler que execute um processo da aplicação, para que seja possível interrompê-lo (usando o comando *kill* do Unix).
 - Parâmetros
 - * **appGid:** o identificador da aplicação a ser finalizada.
- **Mps:** `ps_info #`
 - A aplicação *mps* requisita informações sobre o ambiente. Isso inclui os Brokers Globais, Broker Locais, Schedulers, tabela de RTT (entre Brokers Globais) e aplicações (com seus processos).
 - Parâmetros
 - * nenhum.
- **Mstart:** `app # appName [Params] # userName`
 - A aplicação *mstart* requisita ao Broker Global o início da aplicação *appName* [*Params*].
 - Parâmetros
 - * **appName** [*Params*]: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName:** o usuário que requisitou o início da aplicação.

C.4 Arquivo de configuração

A seguir é mostrado um exemplo de um arquivo de configuração do Shell.

```
<shell-tools>
  <bg_root>bg-usp.no-ip.info</bg_root>
  <bg_root_port>2225</bg_root_port>
</shell-tools>
```

Onde:

- **<bg_root>**: Nome do Broker Global Primário (será convertido para endereço IP);
- **<bg_root_port>**: Porta do Broker Global Primário.

Scheduler

D.1 Visão Geral

O módulo Scheduler é responsável por tomar decisões sobre o balanceamento de carga. O primeiro algoritmo implementado foi o Route. Depois disso, o conceito de módulos de otimização foi desenvolvido e o RouteGA foi o primeiro módulo implementado, que utiliza algoritmos genéticos para otimizar a distribuição de processos.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Troca de mensagens;
- Mensagens de saída;
- Mensagens de entrada;
- Interações SQL;
- Aplicações e comandos;
- Arquivo de interação: *pc.xml*;
- Arquivo de interação: *scheduling.xml*;
- Arquivo de interação: *config_ibl.xml*;
- Arquivo de interação: *environment.xml*; e
- Arquivo de configuração.

D.2 Interações

A figura D.1 apresenta as interações do Scheduler com o os seguintes módulos do MidHPC: Broker Local, IBL, RouteGA e Monitor de Sistema. Essas interações são descritas na tabela D.1.

Figura D.1: Interações do Scheduler com outros módulos (parte 1)

Tabela D.1: Descrição das interações do Scheduler com outros Módulos (parte 1)

Origem	Destino	Como	Descrição
Scheduler	Broker Local	Mensagem	Registro; Encaminhar requisições do Scheduler.
Broker Local	Scheduler	Mensagem	Encaminhar requisições do Broker Global;
Scheduler	IBL	Arquivo XML	Classificar e prever (informação da aplicação).
IBL	Scheduler	Saída	Predições.
Scheduler	RouteGA	Arquivo XML	Informação sobre o ambiente.
RouteGA	Scheduler	Arquivo XML	Decisão de escalonamento.
Scheduler	Monitor de Sistema	Aplicação	Iniciar o monitor de sistema(comando <i>shell</i>).
Scheduler	Base de Dados	SQL	Inserir computadores e processos.

A figura D.2 apresenta as interações do Scheduler com: Monitor de Processos, Checkpoint, Camada de Interceptação e com o Benchmark. Essas interações são descritas na tabela D.2.

D.3 Troca de Mensagens

As figuras D.3 e D.4 apresentam as mensagens que são trafegadas pelo Scheduler. Essas mensagens são detalhadas nas seções seguintes.

Figura D.2: Interações do Scheduler com outros módulos (parte 2)

Tabela D.2: Descrição das interações do Scheduler com outros módulos (parte 2)

Origem	Destino	Como	Descrição
Scheduler	Monitor de Processos	Aplicação	Iniciar o monitor de processos (comando no <i>shell</i>).
Scheduler	Interceptação	Comando	Iniciar aplicações para interceptar chamadas de sistema (comando no <i>shell</i>).
Interceptação	Scheduler	Arquivo de saída	Obter os PIDs dos processos gerados.
Scheduler	Checkpoint	Comando	Fazer o checkpoint do processo.
Benchmark	Scheduler	Arquivo XML	Capacidades do computador (mips, <i>throughput</i> de disco e desempenho da memória).
Scheduler	Benchmark	Comando	Calcular capacidades.

D.4 Mensagens de Saída

A seguir, são detalhadas as mensagens que são originadas no Scheduler com destino a outros módulos.

- `add_s # schedIP # schedPort # schedName # schedMips # schedLoad`
 - Destino: Broker Local
 - Função: Requisita a inclusão do Scheduler na rede do Broker Local
 - Parâmetros:
 - * `schedIP`: endereço IP do Scheduler;
 - * `schedPort`: porta do Scheduler;
 - * `schedName`: nome do computador que executa o Scheduler;
 - * `schedMips`: Capacidade em Mips do Scheduler;
 - * `schedLoad`: Carga atual do Scheduler.
- `stop # procKey # procPid # gbIP # lbIP # schedIP`
 - Destino: Broker Local
 - Função: Requisição para interromper a execução do processo `procPid` e criar um *checkpoint* sobre o mesmo.

Figura D.3: Interações do Scheduler com outros módulos - mensagens (parte 1)

Figura D.4: Interações do Scheduler com outros módulos - mensagens (parte 2)

– Parâmetros:

- * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
- * **procPid**: o *pid* do processo;
- * **gbIP**: endereço IP do Broker Global que mantém o processo;
- * **lbIP**: endereço IP do Broker Local que mantém o processo;
- * **schedIP**: endereço IP do Scheduler que executa o processo.

- restart # checkpoint # appGid # gbIP # lbIP # schedIP

– Destino: Broker Local

– Função: Requisita o reinício de um processo (*checkpoint*) da aplicação *appGid*

– Parâmetros:

- * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
- **load_s # schedIP # schedLoad**
 - Destino: Broker Local
 - Função: Atualizar as informações de carga do Scheduler
 - Parâmetros:
 - * **schedIP**: endereço IP do Scheduler a ser atualizado;
 - * **schedLoad**: carga atual do Scheduler.
 - **del_s # schedIP**
 - Destino: Broker Local
 - Função: Requisita a remoção do Scheduler *schedIP* da lista dos Schedulers
 - Parâmetros:
 - * **schedIP**: endereço IP do Scheduler a ser removido.

D.5 Mensagens de Entrada

A seguir são detalhadas as mensagens originadas de outros módulos com destino ao Scheduler.

- **stop_p # procKey # procPid # gbIP # lbIP # schedIP**
 - Origem: Broker Local
 - Função: Requisição ao Scheduler *schedIP* para parar o processo *procPid* e criar um *checkpoint* sobre o mesmo
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.

-
- `start # checkpoint # appGid # gbIP # lbIP # schedIP`
 - Origem: Broker Local
 - Função: Requisita ao Scheduler *schedIP* o reinício da execução do processo
 - Parâmetros:
 - * **checkpoint**: o nome do *checkpoint* gerado;
 - * **appGid**: o id da aplicação (na base de dados);
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.
 - `app # appGid # appName [Params] # userName`
 - Origem: Broker Local
 - Função: requisita o início da aplicação *appName [Params]*
 - Parâmetros:
 - * **appGid**: Gid da aplicação criado pelo Broker Global Primário;
 - * **appName [Params]**: aplicação (com seus parâmetros) a ser iniciada;
 - * **userName**: o usuário que requisitou o início da aplicação.
 - `kill # procPid # procKey # compKey # gbIP # lbIP # schedIP`
 - Origem: Broker Local
 - Função: Requisita ao Scheduler *schedIP* a interrupção do processo *procPid*
 - Parâmetros:
 - * **procKey**: a chave primária (no banco de dados) do processo a ser interrompido;
 - * **procPid**: o *pid* do processo;
 - * **compKey**: a chave primária (no banco de dados) do computador que executa o processo;
 - * **gbIP**: endereço IP do Broker Global que mantém o processo;
 - * **lbIP**: endereço IP do Broker Local que mantém o processo;
 - * **schedIP**: endereço IP do Scheduler que executa o processo.

D.6 Interações SQL

O Scheduler atualiza a base de dados quando:

- Execução inicia;
 - *INSERT INTO computer (ip, name, qtd_memPr, qtd_memSw, mips, trhd, twhd, coef_la, coef_lb, coef_o2a, coef_o2b, coef_o2c, comp_alive, bl_ip, bg_ip) VALUES ('<schedIP>', '<schedName>', '<mainMemQtd>', '<swapMemQtd>', '<schedMips>', '<trHd>', '<twHd>', '<memPerfA>', '<memPerfB>', '<swapPerfA>', '<swapPerfB>', '<swapPerfC>', '1', '<lbIP>', '<gbIP>');*
 - * <schedIP>: Endereço IP;
 - * <schedName>: Nome do computador;
 - * <mainMemQtd>: Capacidade de memória principal (Mb);
 - * <swapMemQtd>: Capacidade de memória swap (Mb);
 - * <schedMips>: Capacidade em Mips;
 - * <trHd>: Throughput de leitura em disco (Mb/s);
 - * <twHd>: Throughput de escrita em disco (Mb/s);
 - * <memPerfA>: Coeficiente *a* de desempenho da memória principal;
 - * <memPerfB>: Coeficiente *b* de desempenho da memória principal;
 - * <swapPerfA>: Coeficiente *a* de desempenho da memória swap;
 - * <swapPerfB>: Coeficiente *b* de desempenho da memória swap;
 - * <swapPerfC>: Coeficiente *c* de desempenho da memória swap;
 - * '1' (comp_alive): Define o computador como ativo;
 - * <lbIP>: O Broker Local do Scheduler;
 - * <gbIP>: O Broker Global do Scheduler.
 - *SELECT currval('computer_comp_key_seq') AS schedKey;*
 - * Obtém o identificador (chave primária na base de dados) do Scheduler.
- Execução finaliza;
 - *UPDATE computer SET comp_alive = '0' WHERE comp_key = '<schedKey>';*
 - * Atualiza o Scheduler <schedKey> para inativo;
- Um novo processo é iniciado;
 - *INSERT INTO process (proc_gid, proc_pid, proc_comp_id, proc_time, proc_alive, proc_mcost) VALUES ('<gid>', '<pid>', '<schedKey>', 'now', '1', '<filesize>');*
 - * <gid>: Gid da aplicação que gerou esse processo;

-
- * `<pid>`: Pid do processo;
 - * `<schedKey>`: O ID do Scheduler que executa o processo;
 - * `'now'` (`proc_time`): Hora atual;
 - * `'1'` (`proc_alive`): Marca o processo como ativo;
 - * `<filesize>`: Tamanho do arquivo de checkpoint em *Kbytes*.
 - *SELECT currval('process_proc_key_seq') AS procKey;*
 - * Obtém o identificador (chave primária na base de dados) do Processo.
 - Um processo é interrompido.
 - *UPDATE process SET proc_alive = '0' WHERE proc_key = '<procKey>';*
 - * Atualiza o processo `<procKey>` para inativo;

D.7 Aplicações e comandos

O Scheduler requisita que:

- O Monitor de Sistema inicie sua execução para monitorar o uso. Detalhes sobre o monitor de sistemas no apêndice E;
 - Linha de comando: `monitor <comp_id>`
 - * `<comp_id>`: Identificador (chave primária no banco de dados) do Scheduler.
- O Monitor de Processos inicie a execução para monitorar o uso do sistema por um único processo. Detalhes sobre o monitor de processos no apêndice E;
 - Linha de comando: `gridbox <pid> <proc_key>`
 - * `<pid>`: O pid do processo;
 - * `<proc_key>`: Identificador (chave primária no banco de dados) do processo.
- O *benchmark* seja executado, de modo a verificar as capacidades;
 - Uma outra aplicação lida com o *benchmark*. Em seu início, ela recebe os programas (caminhos) que serão utilizados. Esses programas são listados no arquivo de configuração do Scheduler. Detalhes sobre as ferramentas de *benchmark* no apêndice F.
- O *checkpoint* salve o estado do processo em um arquivo para possibilitar a migração;
 - Linha de comando: `freeze <filename> <pid>`

- * `<filename>`: O arquivo de *checkpoint* a ser gerado;
 - * `<pid>`: O *pid* do processo.
- A camada de interceptação seja executada. Essa não é uma chamada direta, pois ao requisitar o início da aplicação essa camada é utilizada.
 - A camada de interceptação converte *threads* em processos. Mais detalhes sobre essa camada no apêndice G.

D.8 Arquivo de interação: *pc.xml*

O arquivo que contém informações sobre o computador é detalhado no apêndice F.

D.9 Arquivo de interação: *config_ibl.xml*

O arquivo de configuração do IBL é detalhado no apêndice H.

D.10 Arquivo de interação: *scheduling.xml*

Após a execução do algoritmo de balanceamento de carga, atualmente o RouteGA, um arquivo contendo informações sobre a alocação de processos é gerado. Um exemplo desse arquivo encontra-se a seguir:

```
<scheduling>
  <process>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</process>
  <original>1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4</original>
  <new>3 4 1 4 4 1 2 1 2 2 3 3 1 2 3 4</new>
</scheduling>
```

Onde:

- `<process>`: identificador do processo (chave primária na base de dados);
- `<original>`: alocação de processos original;
- `<new>`: nova alocação de processos.

D.11 Arquivo de interação: *environment.xml*

Nessa seção, é apresentado um exemplo de arquivo que contém informações sobre o ambiente (aplicações, Schedulers e tabela RTT). É a partir desse arquivo que o algoritmo de balanceamento de carga toma decisões.

```
<midhpc-info>
  <application_count>1</application_count>
  <total_process_count>8</total_process_count>
  <application gid="1">
    <process_count>8</process_count>
    <scheduled>
      <process id="1" comm="1.0" spare="5.0" mcost="1110.0">1</process>
      <process id="2" comm="1.0" spare="5.0" mcost="1210.0">1</process>
      <process id="3" comm="1.0" spare="5.0" mcost="1310.0">1</process>
      <process id="4" comm="1.0" spare="5.0" mcost="1410.0">1</process>
      <process id="5" comm="1.0" spare="5.0" mcost="1120.0">2</process>
      <process id="6" comm="1.0" spare="5.0" mcost="1220.0">2</process>
      <process id="7" comm="1.0" spare="5.0" mcost="1320.0">2</process>
      <process id="8" comm="1.0" spare="5.0" mcost="1420.0">2</process>
    </scheduled>
    <process_load>
      <spare_mips>5.0</spare_mips>
      <mips_avg>10.0</mips_avg>
      <bytes_in_avg>10.0</bytes_in_avg>
      <bytes_out_avg>10.0</bytes_out_avg>
      <blocks_in_avg>10.0</blocks_in_avg>
      <blocks_out_avg>10.0</blocks_out_avg>
      <pagefaults_avg>10.0</pagefaults_avg>
      <pagereclaims_avg>10.0</pagereclaims_avg>
      <mips_stdv>10.0</mips_stdv>
      <bytes_in_stdv>10.0</bytes_in_stdv>
      <bytes_out_stdv>10.0</bytes_out_stdv>
      <blocks_in_stdv>10.0</blocks_in_stdv>
      <blocks_out_stdv>10.0</blocks_out_stdv>
      <pagefaults_stdv>10.0</pagefaults_stdv>
      <pagereclaims_stdv>10.0</pagereclaims_stdv>
    </process_load>
    <predicted>
      <mips_avg_ibl>15.0</mips_avg_ibl>
      <mips_stdv_ibl>0.0</mips_stdv_ibl>
      <bytes_in_avg_ibl>0.0</bytes_in_avg_ibl>
      <bytes_in_stdv_ibl>0.0</bytes_in_stdv_ibl>
      <bytes_out_avg_ibl>0.0</bytes_out_avg_ibl>
      <bytes_out_stdv_ibl>0.0</bytes_out_stdv_ibl>
    </predicted>
  </application gid="1">
</midhpc-info>
```

```

    <blocks_in_avg_ibl>0.0</blocks_in_avg_ibl>
    <blocks_in_stdv_ibl>0.0</blocks_in_stdv_ibl>
    <blocks_out_avg_ibl>0.0</blocks_out_avg_ibl>
    <blocks_out_stdv_ibl>0.0</blocks_out_stdv_ibl>
    <pagefaults_avg_ibl>0.0</pagefaults_avg_ibl>
    <pagefaults_stdv_ibl>0.0</pagefaults_stdv_ibl>
    <pagereclaims_avg_ibl>0.0</pagereclaims_avg_ibl>
    <pagereclaims_stdv_ibl>0.0</pagereclaims_stdv_ibl>
  </predicted>
</application>
<computers>
  <computer_count>2</computer_count>
  <computer>
    <id>1</id>
    <name>aria</name>
    <ip>192.168.0.4</ip>
    <mips>2400.0</mips>
    <main_performance_A>0.081232312</main_performance_A>
    <main_performance_B>2.7525E-4</main_performance_B>
    <swap_performance_A>1.061E-4</swap_performance_A>
    <swap_performance_B>-0.00276634</swap_performance_B>
    <swap_performance_C>0.51405573</swap_performance_C>
    <hd_read>52.2</hd_read>
    <hd_write>38.3</hd_write>
    <main_memory>1024.0</main_memory>
    <swap_memory>512.0</swap_memory>
    <bl_ip>192.168.0.1</bl_ip>
    <bg_ip>192.168.0.1</bg_ip>
  </computer>
  <computer>
    <id>2</id>
    <name>tera</name>
    <ip>192.168.0.3</ip>
    <mips>2800.0</mips>
    <main_performance_A>0.081232312</main_performance_A>
    <main_performance_B>2.7525E-4</main_performance_B>
    <swap_performance_A>1.061E-4</swap_performance_A>
    <swap_performance_B>-0.00276634</swap_performance_B>
    <swap_performance_C>0.51405573</swap_performance_C>
  </computer>
</computers>

```

```

        <hd_read>62.2</hd_read>
        <hd_write>40.3</hd_write>
        <main_memory>1024.0</main_memory>
        <swap_memory>512.0</swap_memory>
        <bl_ip>192.168.0.6</bl_ip>
        <bg_ip>192.168.0.6</bg_ip>
    </computer>
</computers>
<rtt id_from="1" ip_from="192.168.0.1" idto="2" ip_to="192.168.0.6"
    value="20.0"/>
<rtt id_from="2" ip_from="192.168.0.6" idto="1" ip_to="192.168.0.1"
    value="15.0"/>
</midhpc-info>

```

Onde:

- <application_count>: Quantidade de aplicações no ambiente;
- <total_process_count>: Total de processos em execução no ambiente;
- <application gid="1">: Informações sobre a aplicação *gid*;
 - <process_count>: Quantidade de processos da aplicação;
 - <scheduled>: Escalonamento atual;
 - * <process id="1" comm="1.0" spare="5.0" mcost="1110.0">: Onde o processo *id* está executando. Esse processo comunicou *comm* bytes, tem *spare* mips a serem executados e apresenta um custo de migração de *mcost* Kbytes.
 - <process_load>: Carga utilizada pelo processo até o momento;
 - * <spare_mips>: Quantidade de Mips que faltam ser executados;
 - * <mips_avg>: Média de Mips executados por cada processo da aplicação;
 - * <bytes_in_avg>: Média de bytes recebidos (rede) por cada processo da aplicação;
 - * <bytes_out_avg>: Média de bytes enviados (rede) por cada processo da aplicação;
 - * <blocks_in_avg>: Média de blocos de memória lidos por cada processo da aplicação;
 - * <blocks_out_avg>: Média de blocos de memória escritos por cada processo da aplicação;
 - * <pagefaults_avg>: Média de *pagefaults* por cada processo da aplicação;

- * `<pagereclaims_avg>`: Média de *pagereclaims* por cada processo da aplicação;
 - * `<mips_stdv>`: Desvio padrão de Mips executados por cada processo da aplicação;
 - * `<bytes_in_stdv>`: Desvio padrão de bytes recebidos (rede) por cada processo da aplicação;
 - * `<bytes_out_stdv>`: Desvio padrão de bytes enviados (rede) por cada processo da aplicação;
 - * `<blocks_in_stdv>`: Desvio padrão de blocos de memória lidos por cada processo da aplicação;
 - * `<blocks_out_stdv>`: Desvio padrão de blocos de memória escritos por cada processo da aplicação;
 - * `<pagefaults_stdv>`: Desvio padrão de *pagefaults* por cada processo da aplicação;
 - * `<pagereclaims_stdv>`: Desvio padrão de *pagereclaims* por cada processo da aplicação.
- `<predicted>`: Carga da aplicação predita utilizando a técnica IBL.
- * `<mips_avg_ibl>`: Média predita de Mips executados por cada processo da aplicação;
 - * `<mips_stdv_ibl>`: Desvio padrão predito de Mips executados por cada processo da aplicação;
 - * `<bytes_in_avg_ibl>`: Média predita de bytes recebidos (rede) por cada processo da aplicação;
 - * `<bytes_in_stdv_ibl>`: Desvio padrão predito de bytes recebidos (rede) por cada processo da aplicação;
 - * `<bytes_out_avg_ibl>`: Média predita de bytes enviados (rede) por cada processo da aplicação;
 - * `<bytes_out_stdv_ibl>`: Desvio padrão predito de bytes enviados (rede) por cada processo da aplicação;
 - * `<blocks_in_avg_ibl>`: Média predita de blocos de memória lidos por cada processo da aplicação;
 - * `<blocks_in_stdv_ibl>`: Desvio padrão predito de blocos de memória lidos por cada processo da aplicação;
 - * `<blocks_out_avg_ibl>`: Média predita de blocos de memória escritos por cada processo da aplicação;
 - * `<blocks_out_stdv_ibl>`: Desvio padrão predito de blocos de memória escritos por cada processo da aplicação;

-
- * <pagefaults_avg_ibl>: Média predita de *pagefaults* por cada processo da aplicação;
 - * <pagefaults_stdv_ibl>: Desvio padrão predito de *pagefaults* por cada processo da aplicação;
 - * <pagereclaims_avg_ibl>: Média predita de *pagereclaims* por cada processo da aplicação;
 - * <pagereclaims_stdv_ibl>: Desvio padrão predito de *pagereclaims* por cada processo da aplicação.
- <computers>: Informações sobre os computadores no ambiente;
 - <computer_count>: Quantidade de computadores no ambiente que executam o Scheduler;
 - <computer>: Informações sobre um computador específico;
 - * <id>: Chave primária do computador no banco de dados;
 - * <name>: Nome do computador;
 - * <ip>: Endereço IP do computador;
 - * <mips>: Capacidade em Mips;
 - * <main_performance_A>: Coeficiente *a* de desempenho da memória principal;
 - * <main_performance_B>: Coeficiente *b* de desempenho da memória principal;
 - * <swap_performance_A>: Coeficiente *a* de desempenho da memória *swap*;
 - * <swap_performance_B>: Coeficiente *b* de desempenho da memória *swap*;
 - * <swap_performance_C>: Coeficiente *c* de desempenho da memória *swap*;
 - * <hd_read>: *Throughput* de leitura em disco (*Mb/s*);
 - * <hd_write>: *Throughput* de escrita em disco (*Mb/s*);
 - * <main_memory>: Capacidade de memória principal (*Mb*);
 - * <swap_memory>: Capacidade de memória *swap* (*Mb*);
 - * <bl_ip>: Endereço IP do Broker Local;
 - * <bg_ip>: Endereço IP do Broker Global;
 - <rtt id_from="1" ip_from="192.168.0.1" id_to="2" ip_to="192.168.0.6" value="20.0"/>: Distância RTT entre dois computadores (*id_from* e *id_to*) em milissegundos.

D.12 Arquivo de configuração

A seguir é mostrado um exemplo de um arquivo de configuração do Broker Local.

```
<scheduler>
  <ip>192.168.0.181</ip>
  <port>2215</port>
  <bl_ip>192.168.0.181</bl_ip>
  <bl_port>2220</bl_port>
  <cmd_alg>/midhpc/startalg.sh</cmd_alg>
  <cmd_info>/midhpc/schedinfo.sh</cmd_info>
  <use_ibl>1</use_ibl>
  <ibl_mask_file>/midhpc/config/config_mask.xml</ibl_mask_file>
  <cmd_ibl>/midhpc/ibl.sh</cmd_ibl>
  <cmd_meminfo>/midhpc/extras/memInfo/memInfo.sh</cmd_meminfo>
  <cmd_media>/midhpc/extras/media/medias</cmd_media>
  <pc_capacity>/midhpc/config/pc.xml</pc_capacity>
  <bench_mips>/midhpc/extras/mips/build/mips</bench_mips>
  <bench_mem>/midhpc/extras/memo/bench_memo.sh</bench_mem>
  <bench_disc>/midhpc/extras/disc/build/discio /tmp/bench_disc</bench_disc>
  <arg_usage>0.9</arg_usage>
  <arg_increment>0.05</arg_increment>
  <process_monitor>/midhpc/extras/monitor/gridbox.sh</process_monitor>
  <system_monitor>/midhpc/extras/monitor/monitor.sh</system_monitor>
  <run_threads>/midhpc/extras/monitor/run.sh</run_threads>
  <kill_app>/midhpc/extras/monitor/kill_app.sh</kill_app>
  <checkpoint>freeze</checkpoint>
  <username>postgres</username>
  <passwd></passwd>
  <driver>org.postgresql.Driver</driver>
  <url>jdbc:postgresql://192.168.0.1:5432/MIDHPCDB</url>
</scheduler>
```

Onde:

- **<ip>**: Endereço IP do Broker Local;
- **<port>**: Porta do Broker Local;
- **<bl_ip>**: Endereço IP do Broker Global;
- **<bl_port>**: Porta do Broker Global;

-
- **<cmd_alg>**: Comando para executar o algoritmo de balanceamento de carga;
 - **<cmd_info>**: Comando para extrair informações do ambiente e criar um arquivo padrão XML;
 - **<use_ibl>**: Se a técnica IBL deve ser usada ou não;
 - **<ibl_mask_file>**: Arquivo de configuração do IBL;
 - **<cmd_ibl>**: Comando para executar o IBL;
 - **<cmd_meminfo>**: Comando para obter informações sobre a memória;
 - **<cmd_media>**: Comando para calcular as médias na tabela *Application* no banco de dados;
 - **<pc_capacity>**: Arquivo padrão XML que contém informações sobre o computador;
 - **<bench_mips>**: Programa para calcular a capacidade em MIPS do computador;
 - **<bench_mem>**: Programa para calcular o desempenho da memória;
 - **<bench_disc>**: Programa para calcular o desempenho do disco;
 - **<arg_usage>**: Argumento utilizado para indicar quanto da memória total do sistema (*principal + swap*) será utilizado na avaliação do desempenho;
 - **<arg_increment>**: Argumento que indica o incremento em cada passo da avaliação do desempenho da memória;
 - **<process_monitor>**: Aplicação do monitor de processos (caminho);
 - **<system_monitor>**: Aplicação do monitor de sistema (caminho);
 - **<run_threads>**: Comando para executar a aplicação e converter as *threads* em processos;
 - **<kill_app>**: Comando para interromper uma aplicação (*kill*);
 - **<checkpoint>**: Comando utilizado para realizar os *checkpoints*;
 - **<username>**: Usuário do banco de dados;
 - **<passwd>**: Senha do usuário do banco de dados;
 - **<driver>**: Driver JDBC;
 - **<url>**: Url JDBC para a base de dados.

Monitores

E.1 Visão Geral

O MidHPC utiliza dois monitores diferentes pra capturar informações. O Monitor de Sistema captura informações sobre o sistema como um todo (considerando todas as aplicações, inclusive aquelas que não fazem parte do *Grid*) enquanto que o Monitor de Processos colhe informações sobre um único processo.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Monitor de Sistema;
- Monitor de Processos; e
- Interações SQL.

E.2 Interações

A figura E.1 apresenta as interações do Scheduler com o os seguintes módulos do MidHPC: Broker Local, IBL, RouteGA e Monitor de Sistema. Essas interações são descritas na tabela E.1.

E.3 Monitor de Sistema

Ao iniciar o Monitor de Sistema, o Scheduler envia sua identificação (chave primária da tabela *computer*). Assim, os dados podem ser armazenados com sua referência correta.

Figura E.1: Interações dos Monitores com outros módulos

Tabela E.1: Descrição das interações dos Monitores com outros módulos

Origem	Destino	Como	Descrição
Scheduler	Monitor de Sistema	Aplicação	Iniciar o monitor de sistema(comando <i>shell</i>).
Scheduler	Monitor de Processos	Aplicação	Iniciar o monitor de processos (comando no <i>shell</i>).
Monitor de Sistema	Base de dados	SQL	Inserir dados coletados.
Monitor de Processos	Base de dados	SQL	Inserir dados coletados.

Esse monitor, periodicamente, registra o estado do sistema e considera todas as aplicações que possam estar executando (pertencentes ou não ao *Grid*).

Os dados obtidos pelo Monitor de Sistema são (representam cada linha na base de dados):

- Tempo decorrido desde o início do monitoramento;
- Tempo usado pelo processador como *usertime*;
- Tempo usado pelo processador como *systemtime*;
- Percentual do tempo gasto pelo processador como *usertime* (*userload*);
- Percentual do tempo gasto pelo processador como *systemtime* (*systemload*);
- Valores máximos de *userload* e *systemload*;
- Tráfego de rede em ambas as direções (entrada e saída);
- Uso do disco, *throughputs* de leitura e escrita.

E.4 Monitor de Processos

Ao iniciar o Monitor de Processos, o Scheduler envia sua identificação (chave primária da tabela *computer*) e a identificação do processo (chave primária da tabela *process*), assim os dados podem ser armazenados com suas referências. Diferente do Monitor de Sistema, esse monitor registra o estado de um único processo.

Os dados obtidos pelo Monitor de Processos são (representam cada linha na base de dados):

- Hora atual do sistema;
- Identificador do evento;
- Tempo usado pelo processo no processador como *usertime*;
- Tempo usado pelo processo no processador como *systemtime*;
- *Pagefaults* e *Pagereclaims* (falhas de memória);
- Uso do disco (leitura e escrita);
- Os endereços de memória de onde os dados foram lidos ou escritos;
- Quantidade total de *bytes* lidos ou escritos;
- A descrição do evento, isto é, *read*, *write*, *open*, *close*.

E.5 Interações SQL

Todos os dados que são obtidos pelos monitores são armazenados na base de dados. Os seguintes comandos SQL são utilizados para esse fim:

- Monitor de Sistema:
 - `INSERT INTO system (time_at, usertime, systemtime, userload, maxuserload, systemload, maxsystemload, bytesin, maxbytesin, bytesout, maxbytesout, disk_throughput_read, disk_throughput_write, disk_bytes_read, disk_bytes_write, comp_id) VALUES (...);`
- Monitor de Processos:
 - `INSERT INTO history (hist_time, hist_id, stime, utime, pagefaults, pagereclaims, blocks_in, blocks_out, addr_in, bytes_in, addr_out, bytes_out, descr, proc_id) VALUES (...);`

Benchmark

F.1 Visão Geral

O MidHPC utiliza ferramentas de *benchmark* para identificar as características de computadores que executam o módulo Scheduler. Toda vez que o Scheduler inicia, ele verifica se o arquivo que contém as informações sobre o PC (por exemplo, *pc.xml*) existe. Caso esse arquivo não exista as ferramentas de *benchmark* são utilizadas para criar esse arquivo.

As ferramentas de *benchmark* são definidas no arquivo de configuração do Scheduler. Este apêndice está dividido nas seguintes seções:

- Ferramentas;
 - Informação sobre Memória;
 - Capacidade em Mips;
 - Desempenho de Disco;
 - Desempenho de Memória.
- Arquivo de Informação *<pc.xml>*.

F.2 Ferramentas

F.2.1 Informação sobre Memória

É utilizada para calcular as capacidades de memória do computador.

- Comando Linux utilizado: `cat /proc/meminfo | grep <info> | tr -d " " | tr -d "kB" | cut -d ":" -f 2`

-
- O parâmetro `<info>` pode ser:
 - *MemTotal*: Total de memória principal;
 - *MemFree*: Memória principal disponível;
 - *SwapTotal*: Total de memória *swap*;
 - *SwapFree*: Memória *swap* disponível.

F.2.2 Capacidade em Mips

É utilizado para calcular a capacidade em Mips do computador. É baseado no *Tom Kerrigan's Simple Chess Program* (Kerrigan, 2008).

F.2.3 Desempenho de Disco

É utilizado para calcular o *throughput* de disco. Executa várias operações de leitura e escrita para calcular as capacidades. O programa retorna dois valores:

- *Throughput* de leitura em disco em *Mb/s*
- *Throughput* de escrita em disco em *Mb/s*

F.2.4 Desempenho de Memória

É utilizado para calcular o desempenho de memória. Várias operações de alocação de memória são utilizadas para determinar as capacidades. Uma tabela com o tempo necessário para alocar diferentes quantidades de memória é gerada. Duas regressões são realizadas a partir dessa tabela, uma para a memória principal (equação linear) e uma para a memória de *swap* (equação quadrática).

A saída do *benchmark* de desempenho de memória é composto por cinco valores que, em ordem de saída, representam os coeficientes das equações:

- $f(x) = b + ax$
- $f(x) = c + bx + ax^2$

F.3 Arquivo de Informação `<pc.xml>`

```
<pc_capacity>
  <mips>2514.437</mips>
  <hd_read>57.913</hd_read>
  <hd_write>39.529</hd_write>
```

```
<main_memory>465.734375</main_memory>
<swap_memory>486.30078125</swap_memory>
<main_performance_A>0.00220546</main_performance_A>
<main_performance_B>2.7668E-4</main_performance_B>
<swap_performance_A>1.061E-5</swap_performance_A>
<swap_performance_B>-0.00276634</swap_performance_B>
<swap_performance_C>0.51405573</swap_performance_C>
</pc_capacity>
```

Onde:

- <mips>: Capacidade do processador em Mips;
- <hd_read>: *Throughput* de leitura em disco (*Mb/s*);
- <hd_write>: *Throughput* de escrita em disco (*Mb/s*);
- <main_memory>: Quantidade de memória principal;
- <swap_memory>: Quantidade de memória *swap*;
- <main_performance_A>: Coeficiente *a* de desempenho da memória principal;
- <main_performance_B>: Coeficiente *b* de desempenho da memória principal;
- <swap_performance_A>: Coeficiente *a* de desempenho da memória *swap*;
- <swap_performance_B>: Coeficiente *b* de desempenho da memória *swap*;
- <swap_performance_C>: Coeficiente *c* de desempenho da memória *swap*;

Interceptação de *Threads*

O MidHPC intercepta as chamadas de sistema para a criação de *threads* (`pthread_create`) para que seja possível criar processos ao invés de *threads* (figura G.1). Isso é feito para permitir a migração das tarefas.

Threads, agora convertidas em processos, comunicam-se através de um arquivo localizado no DSM que atua como a memória. Assim, toda vez que uma *thread* modificar um valor na memória, essa modificação é feita nesse arquivo.

Figura G.1: Interceptação da criação de *threads*

O DSM deve executar sobre um sistema de arquivos distribuídos (DFS) para permitir o acesso ao arquivo de memória pelos computadores no ambiente.

H.1 Visão Geral

O módulo IBL utiliza a técnica definida na seção 4.2.

Este apêndice está dividido nas seguintes seções:

- Interações;
- Arquivo de configuração *config_ibl.xml*;
- Saída do IBL (resultado);

H.2 Interações

A figura H.1 apresenta as interações do IBL com outros módulos do MidHPC. Essas interações são descritas na tabela H.1.

Figura H.1: Interações do IBL com outros módulos

Tabela H.1: Descrição das interações do IBL com outros Módulos

Origem	Destino	Como	Descrição
Scheduler	IBL	Arquivo XML	Classificar e prever (informação da aplicação).
IBL	Scheduler	Saída	Predições.
Base de Dados	IBL	SQL	Resposta a requisição.

H.3 Arquivo de configuração *config_ibl.xml*

A seguir é mostrado um exemplo de um arquivo de configuração do IBL que é gerado pelo Scheduler para cada um dos parâmetros a serem preditos.

```
<ibl>
  <dimension>35</dimension>
  <neighborhood>2</neighborhood>
  <sigma>0.125</sigma>

  <table-name>application</table-name>
  <username>postgres</username>
  <passwd></passwd>
  <train-select-add></train-select-add>

  <driver>org.postgresql.Driver</driver>
  <url>jdbc:postgresql://192.168.0.1:5432/MIDHPCDB</url>
  <pkname>app_gid</pkname>
  <distance-field-name>app_distance</distance-field-name>
  <distance-field>5</distance-field>

  <setz id="0" value="0.0" attrtype="C" />
  <setz id="1" value="0.0" attrtype="N" />
  <setz id="2" value="0.0" attrtype="N" />
  <setz id="3" value="0.0" attrtype="N" />
  <setz id="4" value="0.0" attrtype="N" />
  <setz id="5" value="1.0" attrtype="C" />
  <setz id="6" value="0.0" attrtype="C" />
  <setz id="7" value="1.0" attrtype="C" />
  <setz id="8" value="1.0" attrtype="C" />
  <setz id="9" value="1.0" attrtype="C" />
  <setz id="10" value="1.0" attrtype="C" />
  <setz id="11" value="1.0" attrtype="C" />
  <setz id="12" value="1.0" attrtype="C" />
  <setz id="13" value="1.0" attrtype="C" />
  <setz id="14" value="1.0" attrtype="C" />
  <setz id="15" value="1.0" attrtype="C" />
  <setz id="16" value="1.0" attrtype="C" />
  <setz id="17" value="1.0" attrtype="C" />
  <setz id="18" value="1.0" attrtype="C" />
```

```

    <setz id="19" value="1.0" attrtype="C" />
    <setz id="20" value="1.0" attrtype="C" />
    <setz id="21" value="0.0" attrtype="C" />
    <setz id="22" value="0.0" attrtype="C" />
    <setz id="23" value="0.0" attrtype="C" />
    <setz id="24" value="0.0" attrtype="C" />
    <setz id="25" value="0.0" attrtype="C" />
    <setz id="26" value="0.0" attrtype="C" />
    <setz id="27" value="0.0" attrtype="C" />
    <setz id="28" value="0.0" attrtype="C" />
    <setz id="29" value="0.0" attrtype="C" />
    <setz id="30" value="0.0" attrtype="C" />
    <setz id="31" value="0.0" attrtype="C" />
    <setz id="32" value="0.0" attrtype="C" />
    <setz id="33" value="0.0" attrtype="C" />
    <setz id="34" value="0.0" attrtype="C" />

    <train-start>1</train-start>
    <train-end>10</train-end>
    <output-field>21</output-field>
    <query-point id="1" value="guto" />
    <query-point id="3" value="/tmp/tabuada/tabuada 1" />
    <query-point id="4" value="1" />
</ibl>

```

Onde:

- **<dimension>**: O número de características levadas em consideração;
- **<neighborhood>**: Define o número de pontos próximos ao ponto de consulta que serão considerados para obter a saída do IBL;
- **<sigma>**: Caracteriza uma função radial que define a distância dos pontos que serão considerados ao calcular a vizinhança de pontos similares ao ponto de consulta;
- **<table-name>**: O nome da tabela base utilizada para predições;
- **<username>**: Usuário do banco de dados;
- **<passwd>**: Senha do usuário do banco de dados;
- **<driver>**: Driver JDBC;

-
- `<url>`: Url JDBC para a base de dados;
 - `<train-select-add>`: Restrições adicionais para a tabela;
 - `<pkname>`: O nome da chave primária de `<table-name>`
 - `<distance-field-name>`: O nome do campo onde será armazenado o valor da distância;
 - `<distance-field>`: O índice do campo que armazena o valor da distância;
 - `<setz id="0" value="0.0" attrtype="C" />`: Definição de cada campo, o valor de relevância e o tipo de atributo (C - contínuo; N - nominal);
 - `<train-start>`: Onde iniciar o treinamento, o valor inicial de `app_gid`;
 - `<train-end>`: Onde finalizar o treinamento, o valor final de `app_gid`;
 - `<output-field>`: O campo utilizado para realizar as predições;
 - `<query-point id="1" value="user" />`: Informações sobre a consulta, *id* 1 é o usuário que submeteu a aplicação;
 - `<query-point id="3" value="/nfs/application" />`: Informações sobre a consulta, *id* 3 é o nome da aplicação (caminho incluído);
 - `<query-point id="4" value="1" />`: Informações sobre a consulta, *id* 4 são os parâmetros da aplicação.

Os itens `<train-start>`, `<train-end>`, `<output-field>` e os três `<query-point>` são modificados para cada característica a ser predita.

H.4 Saída do IBL (resultado)

A saída do IBL é o valor predito do campo definido na *tag* `<output-field>` no arquivo de configuração, baseando-se nas *tags* `<query-point>`.

Tabelas da Base de Dados

I.1 Visão Geral

O MidHPC utiliza o banco de dados *PostgreSQL*. A base de dados do MidHPC é chamada de MIDHPCDB (para referência no *postgres*) e é composta pelas seguintes tabelas:

- *Computer*: Armazena os computadores que executam (ou executaram) o módulo Scheduler;
- *RTT*: A distância em milissegundos entre dois Brokers Globais;
- *Application*: Informações sobre aplicações, armazena as cargas previstas e realizadas;
- *Process*: Processos de aplicações. Permite verificar em quais computadores uma determinada aplicação executou;
- *History*: Dados obtidos pelo monitor de processos;
- *System*: Armazena informações sobre o computador que executa o Scheduler.

I.2 Descrição das tabelas

A seguir são detalhadas as tabelas que compõem a base de dados. Apresentado tipo e descrição de cada campo.

I.2.1 Tabela: *Computer*

Tabela I.1: Descrição dos campos da tabela *Computer*

Campo	Tipo	Chave	Descrição
comp_key	inteiro	primária	Identificador
ip	string		Endereço IP
name	string		Nome
qtd_memPr	<i>float</i>		Quantidade de memória principal
qtd_memSw	<i>float</i>		Quantidade de memória <i>swap</i>
trhd	<i>float</i>		<i>Throughput</i> de leitura em disco
twhd	<i>float</i>		<i>Throughput</i> de escrita em disco
coef_la	<i>float</i>		Coeficiente a de desempenho da memória principal
coef_lb	<i>float</i>		Coeficiente b de desempenho da memória principal
coef_o2a	<i>float</i>		Coeficiente a de desempenho da memória <i>swap</i>
coef_o2b	<i>float</i>		Coeficiente b de desempenho da memória <i>swap</i>
coef_o2c	<i>float</i>		Coeficiente c de desempenho da memória <i>swap</i>
comp_alive	inteiro		Se está executando (1) ou não (0)
bl_ip	string		Endereço do Broker Local
bg_ip	<i>string</i>		Endereço do Broker Global

I.2.2 Tabela: *RTT*

Tabela I.2: Descrição dos campos da tabela *RTT*

Campo	Tipo	Chave	Descrição
rtt_key	inteiro	primária	Identificador do RTT
from_ip	<i>string</i>		IP do Broker Global de origem
to_ip	<i>string</i>		IP do Broker Global de destino
rtt	<i>float</i>		Valor de RTT (em milissegundos)

I.2.3 Tabela: *Application*

Tabela I.3: Descrição dos campos da tabela *Application*

Campo	Tipo	Chave	Descrição
app_gid	inteiro	primária	Identificador da aplicação
app_user	<i>string</i>		Nome do usuário que iniciou a aplicação
app_group	<i>string</i>		Grupo do usuário que iniciou a aplicação

app_name	<i>string</i>		Nome da aplicação (caminho)
app_params	<i>string</i>		Parâmetros da aplicação
app_distance	<i>float</i>		Distância ao ponto de consulta (atualizado pelo IBL)
mips_avg	<i>float</i>		Média de Mips executados por cada processo da aplicação
bytes_in_avg	<i>float</i>		Média de bytes recebidos (rede) por cada processo da aplicação
bytes_out_avg	<i>float</i>		Média de bytes enviados (rede) por cada processo da aplicação
blocks_out_avg	<i>float</i>		Média de blocos de memória escritos por cada processo da aplicação
blocks_in_avg	<i>float</i>		Média de blocos de memória lidos por cada processo da aplicação
pagefaults_avg	<i>float</i>		Média de pagefaults por cada processo da aplicação
pagereclaims_avg	<i>float</i>		Média de pagereclaims por cada processo da aplicação
mips_stdv	<i>float</i>		Desvio padrão de Mips executados por cada processo da aplicação
bytes_in_stdv	<i>float</i>		Desvio padrão de bytes recebidos (rede) por cada processo da aplicação
bytes_out_stdv	<i>float</i>		Desvio padrão de bytes enviados (rede) por cada processo da aplicação
blocks_in_stdv	<i>float</i>		Desvio padrão de blocos de memória lidos por cada processo da aplicação
blocks_out_stdv	<i>float</i>		Desvio padrão de blocos de memória escritos por cada processo da aplicação
pagefaults_stdv	<i>float</i>		Desvio padrão de pagefaults por cada processo da aplicação
pagereclaims_avg	<i>float</i>		Desvio padrão de pagereclaims por cada processo da aplicação
mips_avg_ibl	<i>float</i>		Média predita de Mips executados por cada processo da aplicação
mips_stdv_ibl	<i>float</i>		Desvio padrão predito de Mips executados por cada processo da aplicação
bytes_in_avg_ibl	<i>float</i>		Média predita de bytes recebidos (rede) por cada processo da aplicação
bytes_in_stdv_ibl	<i>float</i>		Desvio padrão predito de bytes recebidos (rede) por cada processo da aplicação

bytes_out_avg_ibl	<i>float</i>		Média predita de bytes enviados (rede) por cada processo da aplicação
bytes_out_stdv_ibl	<i>float</i>		Desvio padrão predito de bytes enviados (rede) por cada processo da aplicação
blocks_in_avg_ibl	<i>float</i>		Média predita de blocos de memória lidos por cada processo da aplicação
blocks_in_stdv_ibl	<i>float</i>		Desvio padrão predito de blocos de memória lidos por cada processo da aplicação
blocks_out_avg_ibl	<i>float</i>		Média predita de blocos de memória escritos por cada processo da aplicação
blocks_out_stdv_ibl	<i>float</i>		Desvio padrão predito de blocos de memória escritos por cada processo da aplicação
pagefaults_avg_ibl	<i>float</i>		Média predita de pagefaults por cada processo da aplicação
pagefaults_stdv_ibl	<i>float</i>		Desvio padrão predito de pagefaults por cada processo da aplicação
pagereclaims_avg_ibl	<i>float</i>		Média predita de pagereclaims por cada processo da aplicação
pagereclaims_stdv_ibl	<i>float</i>		Desvio padrão predito de pagereclaims por cada processo da aplicação

I.2.4 Tabela: *Process*

Tabela I.4: Descrição dos campos da tabela *Process*

Campo	Tipo	Chave	Descrição
proc_key	inteiro	primária	Identificador do Processo
proc_gid	inteiro	estrangeira	<i>Gid</i> da aplicação que gerou o processo
proc_pid	<i>string</i>		<i>Pid</i> do processo
proc_comp_id	inteiro	estrangeira	ID do Scheduler que executa o processo
proc_time	timestamp		Hora que foi criado o processo
proc_alive	inteiro		Ativo (1) ou Inativo (0)
proc_mcost	inteiro		Custo de migração (tamanho do arquivo)

I.2.5 Tabela: *History*

Tabela I.5: Descrição dos campos da tabela *History*

Campo	Tipo	Chave	Descrição
hist_key	inteiro	primária	Identificador do histórico
hist_time	inteiro longo		Hora do sistema
hist_id	inteiro		Identificador de evento (abrir arquivo, socket etc)
utime	inteiro		Tempo gasto em <i>usertime</i>
stime	inteiro		Tempo gasto em <i>systemtime</i>
pagefaults	inteiro		Quantidade de <i>pagefaults</i>
pagereclaims	inteiro		Quantidade de <i>pagereclaims</i>
blocks_in	inteiro		Uso do disco (leitura)
blocks_out	inteiro		Uso do disco (escrita)
addr_in	inteiro longo		Endereço de onde foram lidos os dados
bytes_in	inteiro		Quantidade de dados lidos
addr_out	inteiro longo		Endereço para onde foram escritos os dados
bytes_out	inteiro		Quantidade de dados escritos
descr	<i>string</i>		Descrição do evento (<i>open</i> , <i>read</i> , <i>write</i> , <i>close</i>)
proc_id	inteiro	estrangeira	Identificador do processo monitorado

I.2.6 Tabela: *System*

Tabela I.6: Descrição dos campos da tabela *System*

Campo	Tipo	Chave	Descrição
system_key	inteiro	primária	Identificador
comp_id	inteiro	estrangeira	ID do computador monitorado
time_at	<i>float</i>		Tempo decorrido desde o início do monitoramento
usertime	<i>float</i>		Processador em <i>usertime</i>
systemtime	<i>float</i>		Processador em <i>systemtime</i>
userload	<i>float</i>		Percentual de utilização em <i>usertime</i>
maxuserload	<i>float</i>		<i>userload</i> máximo desde o início do monitoramento
systemload	<i>float</i>		Percentual de utilização em <i>systemtime</i>
maxsystemload	<i>float</i>		<i>systemload</i> máximo desde o início do monitoramento
bytesin	inteiro		Tráfego de rede recebido
maxbytesin	inteiro		Valor máximo de <i>bytesin</i> desde início do monitoramento

bytesout	inteiro		Tráfego de rede enviado
maxbytesout	inteiro		Valor máximo de <i>bytesout</i> desde início do monitoramento
disk_throughput_read	inteiro		<i>Throughput</i> de leitura em disco
disk_throughput_write	inteiro		<i>Throughput</i> de escrita em disco
disk_bytes_read	inteiro		Quantidade de bytes lidos do disco
disk_bytes_write	inteiro		Quantidade de bytes escritos no disco

* Dados coletados a partir da variação entre duas verificações

Referências

- Adve, S. V.; Gharachorloo, K. (1996). Shared memory consistency models: A tutorial. *IEEE Computer*, v.29, n.12, p.66–76.
- Agarwal, A.; Bianchini, R.; Chaiken, D.; Chong, F. T.; Johnson, K. L.; Kranz, D.; Kubiawicz, J. D.; Lim, B.-H.; Mackenzie, K.; Yeung, D. (1999). The MIT alewife machine. *Proc. of the IEEE, Special Issue on Distributed Shared Memory*, v.87, n.3, p.430–444.
- Aha, D. W.; Kibler, D.; Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, v.6, n.1, p.37–66.
- Ahuja, S.; Carriero, N.; Gelernter, D. (1986). Linda and friends. *IEEEEC*, v.19, n.8, p.26–34.
- Amza, C.; Cox, A. L.; Dwarkadas, S.; Keleher, P.; Lu, H.; Rajamony, R.; Yu, W.; Zwaenepoel, W. (1996). Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, v.29, n.2, p.18–28.
- Andrade, N.; Cirne, W.; Brasileiro, F.; Roisenberg, P. (2003). Ourgrid: An approach to easily assemble grids with equitable resource sharing.
- Apon, A.; Baker, M. (2000). Network technologies. *Cluster Computing White Paper*, p. 13–26. University of Portsmouth, UK.
- Back, T.; Fogel, D. B.; Michalewicz, Z., editores (1999). *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, UK.
- Beguelin, A.; Gueist, A.; Dongarra, J.; Jiang, W.; Manchek, R.; Sunderam, V. (1994). *PVM: Parallel Virtual Machine: User's Guide and tutorial for Networked Parallel Computing*. MIT Press.

-
- Bisiani, R.; Ravishankar, M. (1990). Plus: a distributed shared-memory system. *ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture*, p. 115–124, New York, NY, USA. ACM Press.
- Bokhari, S. (1979). Dual processor scheduling with dynamic reassignment. *IEEE Trans. Software Eng.*, v.SE-5, n.4, p.326–334.
- Bokhari, S. (1981). A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. Software Eng.*, v.SE-7, n.6, p.335–341.
- Bote-Lorenzo, M. L.; Dimitriadis, Y. A.; Gómez-Sánchez, E. (2004). Grid characteristics and uses: A grid definition. *Lecture Notes in Computer Science*, v.2970/2004, p.291 – 298.
- Buyya, R. (1999). *High Performance Cluster Computing - Architecture and Systems*, v. 1. Prentice Hall.
- Carter, J. B.; Bennett, J. K.; Zwaenepoel, W. (1995). Techniques for reducing consistency-related communication in distributed shared-memory systems. *ACM Trans. Comput. Syst.*, v.13, n.3, p.205–243.
- Casavant, T. L.; Kuhl, J. G. (1988). A Taxonomy of Scheduling in General-purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, v.14, n.2, p.141–154.
- Costa, P. V. C.; Zorzo, S. D.; Guardia, H. C. (2003). Progrid: A proxy-based architecture for grid operation and management. *15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 03)*, p. 1–2.
- Cox, A. L.; Dwarkadas, S.; Lu, H.; Zwaenepoel, W. (1997). Evaluating the performance of software distributed shared memory as a target for parallelizing compilers. *Proc. of the 11th Int'l Parallel Processing Symp. (IPPS'97)*, p. 474–482.
- Dodonov, E. (2006). Modelo para a avaliação e predição do comportamento de processos aplicado a sistemas distribuídos heterogêneos. *Tesde de Qualificação de Doutorado – ICMC/SC-USP*.
- Dodonov, E.; Mello, R. F. (2007). A model for automatic on-line process behavior extraction, classification and prediction in heterogeneous distributed systems. *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, p. 899–904. IEEE Computer Society.

- Dodonov, E.; Mello, R. F.; Yang, L. T. (2005). A network evaluation for lan, man and wan grid environments. *EUC*, v. 3824 de *Lecture Notes in Computer Science*, p. 1133–1146. Springer.
- Dodonov, E.; Mello, R. F.; Yang, L. T. (2006). Adaptive technique for automatic communication access pattern discovery applied to data prefetching in distributed applications using neural networks and stochastic models. *The International Symposium on Parallel and Distributed Processing and Application – ISPA*, p. 292–303.
- Dodonov, E.; Sousa, J. Q.; Guardia, H. C. (2004). Gridbox: securing hosts from malicious and greedy applications. *Proceedings of the 2nd workshop on Middleware for grid computing*, p. 17–22, New York, NY, USA. ACM Press.
- Ferreira, E. A.; Mello, R. F. (2004). Avaliação de desempenho de protocolos e bibliotecas de comunicação entre processos. *International Information and Telecommunication Technologies Symposium (I2TS 2004)*, p. 8.
- Fleisch, B. D.; Popek, G. J. (1989). Mirage: A coherent distributed shared memory design. *SOSP12*, p. 211–223.
- Foster, I.; Kesselman, C. (1999). The Globus project: a status report. *Future Generation Computer Systems*, v.15, n.5–6, p.607–621.
- Foster, I.; Williams, D.; Middleton, D. (2003). The earth system grid ii: Turning climate datasets into community resources. Esg tech report, American Geophysical Union.
- Foster, I. T.; Kesselman, C. (2000). Computational grids. *VECPAR*, p. 3–37.
- Goldchleger, A. (2005). Integrate: Um sistema de middleware para computação em grade oportunista. Dissertação (Mestrado), Instituto de Matemática e Estatística da Universidade de São Paulo.
- Goldchleger, A.; Kon, F.; Goldman, A.; Finger, M.; Bezerra, G. C. (2002). Integrate: object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, p. 1–12.
- Hagersten, E.; Landin, A.; Haridi, S. (1992). DDM - a cache-only memory architecture. *IEEE Computer*, v.25, n.9, p.44–54.
- Harchol-Balter, M.; Downey, A. B. (1997). Exploiting Process Lifetimes Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, v.15, n.3, p.253–285.

-
- Heinrich, M.; Kuskin, J.; Ofelt, D.; Heinlein, J.; Baxter, J.; Singh, J. P.; Simoni, R.; Gharachorloo, K.; Nakahira, D.; Horowitz, M.; Gupta, A.; Rosenblum, M.; Hennessy, J. (1994). The performance impact of flexibility in the Stanford FLASH multiprocessor. *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, p. 274–285, San Jose, California.
- Hinterding, R. (2000). Representation, mutation and crossover issues in evolutionary computation. *Proc. of the 2000 Congress on Evolutionary Computation*, p. 916–923, Piscataway, NJ. IEEE Service Center.
- IEEE (1993). *9945-2: 1993 (ISO/IEC) [IEEE/ANSI Std 1003.2-1992 and IEEE/ANSI 1003.2a-1992] Information Technology-Portable Operating System Interface (POSIX®) — Part 2: Shell and Utilities*. IEEE CS.
- Ishii, R. P.; Mello, R. F.; Senger, L. J.; Santana, M. J.; Santana, R. H. C. (2005). Scheduling based on the impact over process communication of parallel applications. *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, p. 7. IEEE Computer Society.
- Iskra, K. A.; van der Linden, F.; Hendrikse, Z. W.; Overeinder, B. J.; van Albada, G. D.; Sloot, P. M. A. (2000). The implementation of dynamite: an environment for migrating pvm tasks. *SIGOPS Oper. Syst. Rev.*, v.34, n.3, p.40–55.
- Janakiraman, G. J.; Santos, J. R.; Subhraveti, D. (2005). Cruz: Application-transparent distributed checkpoint-restart on standard operating systems. *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, p. 260–269, Washington, DC, USA. IEEE Computer Society.
- Karl, W.; Leberecht, M.; Schulz, M. (1999). Supporting shared memory and message passing on clusters of PCs with a smile. *Communication, Architecture, and Applications for Network-Based Parallel Computing*, p. 196–210.
- Kerrigan, T. (2008). Tscp benchmark. disponível em: <http://www.tckerrigan.com/chess/tscp>. acesso em: 24 de janeiro de 2008.
- Kim, B.-J. (2005). Comparison of the existing checkpoint systems.
- Kleinrock, L. (1976). *Queueing Systems - Volume II: Computer Applications*. John Wiley & Sons.
- Korpela, E.; Werthimer, D.; Anderson, D.; Cobb, J.; Leboisky, M. (2001). Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, v.3, n.1, p.78–83.

- Kyiv National Taras Shevchenko University (2007). Checkpoint for linux. <http://www.cluster.kiev.ua/eng/?chpx>.
- Lee, C.; Talia, D. (2003). Grid programming models: Current tools, issues and directions. Berman, F.; Fox, G.; Hey, T., editores, *Grid Computing: Making the Global Infrastructure a Reality*, p. 555–578. Wiley.
- Lenoski, D.; Laudon, J.; Gharachorloo, K.; Weber, W.-D.; Gupta, A.; Hennessy, J.; Horowitz, M.; Lam, M. S. (1992). The stanford dash multiprocessor. *Computer*, v.25, n.3, p.63–79.
- Li, A.; Hermannsson, G.; Wittie, L. (1995). Performance modeling of eagersharing distributed memory systems.
- Liao, C.; Martonosi, M.; Clark, D. W. (1998). Performance monitoring in a myrinet-connected shrimp cluster. *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, p. 21–29, New York, NY, USA. ACM Press.
- Matsuoka, S.; Shinjo, S.; Aoyagi, M.; Sekiguchi, S.; Usami, H.; Miura, K. (2005). Japanese computational grid research project: Naregi. *Proceedings of the IEEE*, p. 522 – 533.
- Mello, R. F.; Andrade Filho, J. A.; Senger, L. J.; Yang, L. T. (2007). RouteGA: A grid load balancing algorithm with genetic support. *The IEEE 21th International Conference on Advanced Information Networking and Applications (AINA 2007)*, p. 885–892. IEEE Computer Society.
- Mello, R. F.; Senger, L.; Yang, L. (2005). Automatic text classification using an artificial neural network. *High Performance Computational Science and Engineering*, v.1, p.1–21.
- Mello, R. F.; Senger, L. J. (2004). A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments. *International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, p. 222–227. IEEE Computer Society.
- Mello, R. F.; Senger, L. J. (2006). Model for simulation of heterogeneous high-performance computing environments. *7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006*, p. 11.
- Mello, R. F.; Senger, L. J.; Yang, L. T. (2006a). Performance evaluation of route: A load balancing algorithm for grid computing. *RITA – Revista de Informática Teórica e Aplicada*, v.13, n.1, p.87–108.
- Mello, R. F.; Senger, L. J.; Yang, L. T. (2006b). A routing load balancing policy for grid computing environments. *The IEEE 20th International Conference on Advanced*

-
- Information Networking and Applications (AINA 2006)*, p. 153–158. IEEE Computer Society.
- Mello, R. F.; Trevelin, L. C.; Paiva, M. S.; Yang, L. T. (2004). Comparative study of the server-initiated lowest algorithm using a load balancing index based on the process behavior for heterogeneous environment. *Networks, Software Tools and Application, ISSN 1386-7857*. Kluwer.
- Message Passing Interface Forum (1993). MPI: A Message Passing Interface. *Proceedings of Supercomputing '93*, p. 878–883. IEEE Computer Society Press.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
- MPI Forum (1997). *MPI-2: Extensions to the Message Passing Interface*. Message Passing Interface Forum.
- Nery, B. R.; Mello, R. F.; Carvalho, A. C. P. L. (2006). Escalonamento de processos utilizando técnicas de aco. *XXV Concurso de Trabalhos de Iniciação Científica*, p. 1–15.
- Object Management Group (2002). Corba v3.0 specification.
- Osman, S.; Subhraveti, D.; Su, G.; Nieh, J. (2002). The design and implementation of zap: A system for migrating computing environments.
- Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley, Reading, Massachusetts.
- Paranhos, D.; Cirne, W.; Brasileiro, F. (2003). Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. *Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing*, p. 1–14.
- Plank, J. S.; Beck, M.; Kingsley, G.; Li, K. (1995). Libckpt: Transparent Checkpointing under Unix. *Proceedings of USENIX Winter1995 Technical Conference*, p. 213–224, New Orleans, Louisiana/U.S.A.
- Protic, J.; Tomasevic, M.; Milutinovic, V. (1996). Distributed shared memory: concepts and systems. *IEEE Parallel and Distributed Technology*, p. 63–79.
- Quinn, M. J. (2004). *Parallel Programming in C with MPI and OpenMP*. McGraw Hill.
- Ramachandran, U.; Khalidi, M. Y. A. (1989). An implementation of distributed shared memory. *SEDMS1*, p. 21–38.

- Reinhardt, S. K.; Larus, J. R.; Wood, D. A. (1994). Tempest and typhoon: User-level shared memory. *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA '94)*, p. 325–337.
- Reis, V. Q.; Santana, M. J.; Mello, R. F.; Santana, R. H. C. (2005). Uma política de escalonamento orientada à redução do tempo de resposta de aplicacoes paralelas. *VI Workshop em Sistemas Computacionais de Alto Desempenho*, Rio de Janeiro, RJ, BR. Sociedade Brasileira de Computação.
- Sankaran, S.; Squyres, J. M.; Barrett, B.; Lumsdaine, A.; Duell, J.; Hargrove, P.; Roman, E. (2003). The lam/mpi checkpoint/restart framework: System-initiated checkpointing. *Proceeding of LACSI Symposium*.
- Schoinas, I.; Falsafi, B.; Lebeck, A. R.; Reinhardt, S. K.; Larus, J. R.; Wood, D. A. (1994). Fine-grain access control for distributed shared memory. *ASPLOS6*, p. 297–307.
- Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C. (2006). Aprendizado baseado em instâncias aplicado à predição de características de execução de aplicações paralelas. *Revista de Informática Téorica e Aplicada*, v.to be published, p.1–26.
- Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C.; Yang, L. T. (2005). Improving scheduling decisions by using knowledge about parallel applications resource usage. *High Performance Computing and Communications: First International Conference (HPCC), LNCS 3726*, v. 3726, Sorrento, Italy.
- Sethi, R. (1989). *Programming languages: concepts and constructs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Shankar, A. (2005). Core restart: process checkpointing and restarting (using dumped core). Disponível em: <http://www.geocities.com/asimshankar/checkpointing/> - Acesso em: 21/01/2008.
- Shen, C.-C.; Tsai, W.-H. (1985). A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Trans. Comput.*, v.C-34, n.3, p.197–203.
- Shivaratri, N. G.; Krueger, P.; Singhal, M. (1992). Load distributing for locally distributed systems. *IEEE Computer*, v.25, n.12, p.33–44.
- Sterling, T. (2000). An introduction to pc clusters for high performance computing. *Cluster Computing White Paper*, p. 3–12. University of Portsmouth, UK.
- Stevens, R.; Woodward, P.; DeFanti, T.; Catlett, C. (1997). From the i-way to the national technology grid. *Commun. ACM*, v.40, n.11, p.50–60.

-
- Stockinger, H. (2007). Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, v.1, p.1–15.
- Stone, H. (1978). Critical load factors in two-processor distributed systems. *IEEE Trans. Software Eng.*, v.SE-4, n.3, p.254–258.
- Stone, H.; Bokhari, S. (1978). Control of distributed processes. *Computer*, v.11, n.7, p.97–106.
- Wilson, D. R.; Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, v.6, p.1–34.
- Zhang, Y.; Wong, D.; Zheng, W. (2005). User-level checkpoint and recovery for lam/mpi. *SIGOPS Oper. Syst. Rev.*, v.39, n.3, p.72–81.
- Zhou, S.; Stumm, M.; McInerney, T. (1990). Extending distributed shared memory to heterogeneous environments. *ICDCS10*.