Universidade Federal de Santa Maria — UFSM

Dissertação de Mestrado

SISTEMA DISTRIBUÍDO PARA ALOCAÇÃO DE RECURSOS BASEADA EM NECESSIDADES

Cleverton Marlon Possani

Programa de Pós-Graduação em Engenharia de Produção — PPGEP

Santa Maria, RS, Brasil

2006

SISTEMA DISTRIBUÍDO PARA ALOCAÇÃO DE RECURSOS BASEADA EM NECESSIDADES

por

Cleverton Marlon Possani

Dissertação apresentada ao Curso de
Mestrado do Programa de Pós-Graduação em
Engenharia de Produção, área de concentração
em Tecnologia da Informação, da Universidade
Federal de Santa Maria (UFSM, RS), como
requisito parcial para obtenção do grau de
Mestre em Engenharia de Produção

PPGEP

Santa Maria, RS, Brasil

2006

Universidade Federal de Santa Maria
Centro de Tecnologia
PPGEP

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

SISTEMA DISTRIBUÍDO PARA ALOCAÇÃO DE RECURSOS BASEADA EM NECESSIDADES

elaborada por
Cleverton Marlon Possani

como requisito parcial para obtenção do grau de
Mestre em Engenharia de Produção

COMISSÃO EXAMINADORA:

Prof. Dr. Marcelo Pasin
(Presidente/Orientador — Departamento de Eletrônica e Computação — UFSM)

Prof. Dr. Benhur de Oliveira Stein
(Departamento de Eletrônica e Computação — UFSM)

Prof. Dr. Andrea Schwertner Charão
(Departamento de Eletrônica e Computação — UFSM)

Santa Maria, 01 de Novembro de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Possani, Cleverton Marlon

Sistema Distribuído para Alocação de Recursos Baseada em Necessidades / Cleverton Marlon Possani. – Santa Maria: PPGEP, 2006.

79 f.: il.

Dissertação (mestrado) – Universidade Federal de Santa Maria. PPGEP, Santa Maria, BR–RS, 2006. Orientador: Marcelo Pasin.

1. Grade. 2. Alocação de Recursos. 3. Necessidades. 4. Descrição de Recursos. I. Pasin, Marcelo. II. Título.

Endereço: Rua Alfredo Steighlis, n. 317, Bairro São José, Ijuí, RS, 98700-000. Fone (55) 3332-2095. End. Eletr: possani@inf.ufsm.br

AGRADECIMENTOS

Ao apoio financeiro concedido pela Fundação de Apoio à Tecnologia e Ciência (FATEC).

A minha esposa e filhos, que nos momentos mais difíceis estiveram sempre confiantes, dando apoio e acreditando neste trabalho.

Aos meus pais, que sempre depositaram confiança e carinho em todos os momentos da minha vida.

Um agradecimento especial a Milton e Clecir Kozoroski, que de muitas formas auxiliaram para que este trabalho se realizasse.

Ao meu orientador, que se disponibilizou, ajudou, contribuiu como poucos orientadores o fizera.

Ao *Dipartimento di Informatica* da Universidade de Pisa, Itália, pela permissão de uso de seu laboratório.

Aos membros do Laboratório de Sistemas de Computação (LSC) da UFSM.

Ao colega Gerson Antunes Soares, pelas conversas amigas e pelo auxílio prestado, o qual foi de grande importância para a conclusão deste trabalho.

Enfim, a todos que, de alguma forma, contribuíram para a realização deste.

RESUMO

Arquiteturas de sistemas distribuídos possuem um amplo poder de processamento quando seus recursos trabalham através de cooperação, formando organizações virtuais. O aproveitamento de recursos ociosos geograficamente distribuídos é um dos objetivos das grades computacionais. Nestes ambientes, aplicações de usuários necessitam de uma determinada quantidade de recursos para suprir suas necessidades de execução. Ao mesmo tempo, para localizar estes recursos, é necessário uma forma de expressá-los. Este trabalho apresenta o desenvolvimento de um protótipo para prover habilidades de alocação de recursos a aplicações de usuários, utilizando para isso informações sobre descrição de recursos e necessidades. Estas necessidades determinam quais recursos poderão ser utilizados para que a aplicação seja executada com auxílio da grade computacional. Um ambiente de simulação foi desenvolvido para avaliar a performance e descobrir alguns pontos críticos do protótipo.

Palavras-chave: Grade, Alocação de Recursos, Necessidades, Descrição de Recursos.

Distributed System for Alocation of Resource Based on Necessities

ABSTRACT

Distributed architectures systems possesses an ample processing power when its resources work through cooperation, forming virtual organizations. The exploitation of idle geographically distributed resource is one of the objectives of the grid computation. In these environments, users application need a certan amount of resources to fill their execution necessities. At the same time, to find these resources, its necessary a way of expressing them. This work presents the development of an archetype to provide abilities of alocation resource to users application. Using for that informations about resources description and necessities. This necessities determine what kind of resource can be used in order to the application be executed with the help of the grid computation. A simulation environment was developed in order to evaluate the performance and to find some archetype critic points.

Keywords: grid computing, resource allocation, application necessities, resource description.

LISTA DE FIGURAS

Figura 2.1: Taxonomia de Sistemas para Grades 16
Figura 2.2: Estrutura do Gerenciamento de Recursos do Globus (CIRNE, 2003) 19
Figura 2.3: Sintaxe para Descrição de Requisições RSL 19
Figura 2.4: Requerimento de Recursos 19
Figura 2.5: Ações Envolvidas no Matchmaking 21
Figura 2.6: Descrição de um Recurso com ClassAd 21
Figura 2.7: Descrição de uma Tarefa com ClassAd 22
Figura 2.8: Legion: Níveis de Gerenciamento de Recursos (CHAPIN et al., 1999a) . . 23
Figura 2.9: Gramática Utilizada pelo Legion 24
Figura 2.10: Arquitetura do MyGrid 25
Figura 2.11: Cenário do Sistema Cadeo 27
Figura 2.12: Objeto Futuro 29
Figura 2.13: Modelo Básico de Migração de Tarefas 30
Figura 2.14: Modelo Básico de Certificação do ProActive 30
Figura 2.15: Modelo Básico de Políticas de Segurança do ProActive 31
Figura 2.16: Modelo Básico de Comunicação em Grupo 32
Figura 2.17: Modelo Básico de Comunicação em P2P 32

Figura 3.1: Visão da Grade Computacional 34
Figura 3.2: MatchMaker - Busca por Recursos 35
Figura 3.3: Serviço de Pré-casamento Realizado por Todos os DAs 37
Figura 3.4: Serviço de Casamento realizado pelo DAP 37
Figura 3.5: Modelo de Recursos em Grades Computacionais 38
Figura 3.6: Serviço de localização de recursos 39
Figura 3.7: Reserva de Recursos 40
Figura 3.8: União de Recursos 41
Figura 3.9: Algoritmo DAP 42
Figura 3.10: Algoritmo DA 42

Figura 4.1: Gerenciamento de recursos do sistema Cadeo 45
Figura 4.2: Integração do MatchMaker com o sistema Cadeo 45

Figura 5.1: Código SIMJAVA para criação de relacionamento entre DAP e DAs 53
Figura 5.2: Estrutura criada pelo código da figura 5.1 53
Figura 5.3: Simulando MatchMaker com 2 DA vizinhos, totalizando uma busca em 7 DAs 55
Figura 5.4: Distribuição dos custos de comunicação entre 7 DAs 56
Figura 5.5: Simulando MatchMaker com 3 DAs vizinhos 57

Figura 5.6: Distribuição dos custos de comunicação entre 12 DAs 58
Figura 5.7: Analisando tempo de execução das simulações 61
Figura 5.8: Analisando quantidade de recursos casados nas simulações 61
Figura 5.9: Comparando o número de recursos casados com o tempo de execução das simulações 62

Figura 8.1: Distribuição dos custos de comunicação entre 21 DAs 73
Figura 8.2: Distribuição dos custos de comunicação entre 31 DAs 74
Figura 8.3: Distribuição dos custos de comunicação entre 43 DAs 75
Figura 8.4: Distribuição dos custos de comunicação entre 57 DAs 76
Figura 8.5: Distribuição dos custos de comunicação entre 73 DAs 77
Figura 8.6: Distribuição dos custos de comunicação entre 91 DAs 78
Figura 8.7: Distribuição dos custos de comunicação entre 111 DAs 79

LISTA DE TABELAS

Tabela 5.1: Descrição das Características de Custos 54
Tabela 5.2: Custos Relativos à Primeira Simulação 55
Tabela 5.3: Resultados Relativos à Primeira Simulação 55
Tabela 5.4: Custos Relativos à Segunda Simulação 56
Tabela 5.5: Resultados Relativos à Segunda Simulação 57
Tabela 5.6: Resultados Relativos à Terceira Simulação 58
Tabela 5.7: Resultados Relativos à Quarta Simulação 59
Tabela 5.8: Resultados Relativos à Quinta Simulação 59
Tabela 5.9: Resultados Relativos à Sexta Simulação 59
Tabela 5.10: Resultados Relativos à Sétima Simulação 60
Tabela 5.11: Resultados Relativos à Oitava Simulação 60
Tabela 5.12: Resultados Relativos à Nona Simulação 60

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
DA	Domínio Administrativo
DAP	Domínio Administrativo Primário
CA	*Certificate Authority*
CIC	*Communication-Induced Checkpointing*
FIFO	*First In, First Out*
FTP	*File Transfer Protocol*
GRAM	*Grid Resource Allocation Manager*
GRIP	*Grid Resource Information Protocol*
GSI	*Grid Security Infrastructure*
HTTP	*Hypertext Transfer Protocol Overview*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
LDAP	*Lightweight Directory Access Protocol*
NOA	*Number Of Acquaintances*
NOW	*Network Of Workstation*
P2P	*Peer-To-Peer*
PML	*Pessimistic Message Logging*
QoS	*Quality of Service*
RMI	*Remote Method Invocation*
SSH	*Secure Shell*
TTC	*Time To Checkpoint*
TTL	*Time To Live*
TTU	*Time To Update*
WAN	*Wide Area Network*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 12

2 GRADE COMPUTACIONAL 15
2.1 Alocação de Recursos 16
2.1.1 Globus 17
2.1.2 Condor 20
2.1.3 Legion 22
2.1.4 MyGrid 24
2.1.5 Cadeo 26
2.2 Programação em Grade 27
2.2.1 Java 28
2.2.2 Middleware ProActive 29

3 ARQUITETURA DO MATCHMAKER 34
3.1 Idéia Geral do MatchMaker 35
3.2 Arquitetura do Sistema MatchMaker 38
3.2.1 Descrição de Recursos 38
3.2.2 Serviço de Localização 38
3.2.3 Serviço de Casamento 39
3.2.4 Serviço de Reserva 40
3.2.5 Serviço de União 41
3.3 Algoritmo Proposto 42

4 DESENVOLVIMENTO DO MATCHMAKER 44
4.1 MatchMaker 44
4.2 Funcionamento MatchMaker 44
4.3 Classes Implementadas 45
4.3.1 MatchMaker 46
4.3.2 SearchResource 47
4.3.3 ResourceMatch 47
4.3.4 Resource 49
4.3.5 IDApplication 49
4.3.6 Traveller 50

5 ANÁLISE DOS RESULTADOS 52
5.1 Ambiente de Simulação 52
5.1.1 SimJava 52
5.2 Resultados 53

5.2.1 Simulando Grade com 7 Domínios Administrativos 54
5.2.2 Simulando Grade com 13 Domínios Administrativos 56
5.2.3 Simulando Grade com 21 Domínios Administrativos 57
5.2.4 Simulando Grade com 31 Domínios Administrativos 58
5.2.5 Simulando Grade com 43 Domínios Administrativos 59
5.2.6 Simulando Grade com 57 Domínios Administrativos 59
5.2.7 Simulando Grade com 73 Domínios Administrativos 59
5.2.8 Simulando Grade com 91 Domínios Administrativos 60
5.2.9 Simulando Grade com 111 Domínios Administrativos 60
5.2.10 Comparando Simulações 60

6 CONCLUSÃO 63

7 TRABALHOS FUTUROS 64
7.1 União de Recursos 64
7.2 Sistema de Reserva 64
7.3 Algoritmo de Casamento 64
7.4 Sistema de Descrição de Recursos 64
7.5 Agentes de Monitoração 65
7.6 Eleição de um novo *dap* 65

REFERÊNCIAS 66

8 GRÁFICOS DE DISTRIBUIÇÃO DE MENSAGENS 73

1 INTRODUÇÃO

Computação em grade[1] é um modelo de computação baseado na utilização de um número muito grande de computadores interligados em rede (FOSTER, 2002). Segundo este modelo, usuários e computadores se encontram mundialmente espalhados. Aplicações de usuários utilizam simultaneamente múltiplos recursos computacionais, atribuindo a cada um deles uma tarefa diferente. Embora já bem caracterizado, este modelo é um tema científico em ampla discussão neste momento. Este trabalho se concentra em um dos problemas advindos deste modelo, que é a alocação eficiente de recursos computacionais às tarefas das aplicações (RAMAN; LIVNY; SOLOMON, 2004; GALSTYAN; CZAJKOWSKI; LERMAN, 2004).

A alocação eficiente de tarefas a recursos adequados é um típico problema de otimização, e como tal, difícil de ser feita de forma ótima e rápida. No contexto de grades computacionais, esta alocação deve ser feita de forma dinâmica, pois usuários podem iniciar e terminar aplicações e recursos podem ser disponibilizados e indisponibilizados a qualquer momento. Deve ser feita de forma distribuída para possibilitar o crescimento do número de tarefas e de recursos sem prejudicar o seu desempenho. Deve ser feita de forma a englobar diferentes domínios administrativos, pois os recursos pertencem a diferentes organizações. Estas restrições dificultam o desenvolvimento de algoritmos eficientes, o que faz com que existam poucas soluções neste sentido atualmente.

O trabalho descrito nesta dissertação teve a finalidade de desenvolver um algoritmo inovador para alocação de aplicações a recursos de grades computacionais. Este algoritmo é dinâmico de forma a permitir várias alocações simultâneas, sendo que estas podem ser modificadas tanto em nível de oferta de recursos quanto em demanda das aplicações. Ao mesmo tempo, este algoritmo é distribuído, de forma a poder ser desmembrado, adaptando-se a números maiores de recursos e aplicações e à estrutura independente de administração das grades computacionais. Por fim, o algoritmo utiliza primitivas de comunicação assíncrona, de forma a obter concorrência nas interações das diversas instâncias do algoritmo.

Os objetivos deste trabalho resumem-se em:

Projetar um sistema para alocação de recursos em ambiente de grade computacional baseadas em necessidades. Com base em estudos e pesquisas, o primeiro objetivo é realizar um levantamento do funcionamento interno de sistemas para grades computacionais que utilizam a técnica de alocação de recursos baseados em necessidades. Projetar uma estrutura básica de gerenciamento de recursos, oferecendo suporte à descrição de recursos e necessidades de aplicações.

Desenvolver um protótipo de um sistema que permita alocação de recursos em grade. Propor um algoritmo de alocação de recursos para grade computacional. Desenvolver a estrutura básica do protótipo proposto com auxílio de bibliotecas de programação que forneçam uso de

[1] *grid computing*.

comunicação assíncrona entre domínios administrativos.

Validar o sistema proposto. Através de um simulador avaliar o algoritmo proposto. A simulação possibilita o uso de uma grande quantidade de recursos e domínios administrativos, possibilitando a realização de num grande número de testes com o algoritmo.

O problema da alocação de recursos vem sendo estudado por diversas instituições, cada uma delas enfocando algum ou alguns dos seus aspectos. Raman (RAMAN; LIVNY; SOLOMON, 1998) desenvolveu um *framework* de alocação de recursos para o sistema Condor, mas apresenta o inconveniente de trabalhar com um formalismo bilateral. O próprio autor apresenta uma extensão multilateral do modelo utilizado anteriormente (RAMAN; LIVNY; SOLOMON, 2004). Este resolve o problema de co-alocação, permitindo que o sistema trabalhe com várias solicitações de recursos ao mesmo tempo. O trabalho de Chapin (CHAPIN et al., 1999b) não resolve o mesmo tipo de problema, mas apresenta uma solução totalmente baseada em objetos para o sistema Legion. O sistema Globus (FOSTER; KESSELMAN, 1998) implementa um protocolo para alocar recursos através de informações capturadas por serviços de diretório baseados em LDAP. Assim sendo, o algoritmo proposto apresenta características diferenciadas dos sistemas citados, embora algumas das suas idéias, quando complementares, possam ser futuramente nele incorporadas.

Uma das principais características do algoritmo desenvolvido é possibilitar a alocação de forma distribuída e incremental. Assim sendo, ele faz uso de descrições de recursos e de necessidades de aplicações que possam ser desmembradas. Não foi tratado neste trabalho, entretanto, o problema de como expressar recursos e necessidades, nem como medir a qualidade de uma certa alocação. Para tanto, confia-se em outros algoritmos já existentes como o XMatch (ANDREOZZI; MONTESI; MORETTI, 2005) e algoritmos baseados em semântica Web (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003; HARTH et al., 2004). No âmbito deste trabalho, assume-se que existe uma forma de representar recursos e necessidades e de calcular incrementalmente a satisfação dessas necessidades com um subconjunto de recursos.

Para validar o desenvolvimento deste trabalho, foi desenvolvido um protótipo do algoritmo de alocação de recursos, chamado Matchmaker. Este protótipo exprime o algoritmo distribuído em linguagem Java, que é a liguagem mais utilizada atualmente no desenvolvimento de aplicações distribuídas e por conseguinte de aplicações de grade. Em especial, utilizou-se ProActive (INRIA - ProActive, 2006) pelas suas facilidades de criação de objetos distribuídos e de comunicação assíncrona em Java. Para poder verificar a escalabilidade do protótipo, foi desenvolvido em SimJava um simulador de grade computacional (que simula o tempo de comunicação e de busca local de recursos). Desta forma, um grande número de instâncias do algoritmo pode ser testado.

Esta dissertação tem o objetivo de descrever o trabalho desenvolvido. Para tal, sua estrutura é a seguinte.

Capítulo 1: INTRODUÇÃO - Introdução ao texto e descrição dos objetivos do trabalho.

Capítulo 2: GRADE COMPUTACIONAL - Alguns conceitos de grades computacionais são abordados neste capítulo. A utilização da linguagem java é descrita, juntamente com uma abordagem do *middleware* ProActive.

Capítulo 3: ARQUITETURA DO MATCHMAKER - Este capítulo apresenta a idéia geral do MatchMaker, descrevendo a visão do autor sobre a arquitetura comportamental do sistema, juntamente com o algoritmo desenvolvido para o protótipo.

Capítulo 4: DESENVOLVIMENTO DO MATCHMAKER - Este capítulo descreve as principais classes implementadas para o desenvolvimento do protótipo.

Capítulo 5: ANÁLISE DOS RESULTADOS - A análise dos resultados está dividida em uma breve descrição sobre o ambiente de simulação desenvolvido e os resultados obtidos com

a execução do mesmo.

Capítulo 6: CONCLUSÃO - O capítulo 6 destina-se a apresentar a conclusão relativa ao desenvolvimento deste trabalho e os resultados obtidos no capítulo 5. Alguns trabalhos futuros são descritos, visando tornar o protótipo um sistema com usabilidade no mundo real.

2 GRADE COMPUTACIONAL

Grades de Computadores aparecem na literatura como uma infraestrutura de *hardware* ou *software* capaz de agrupar componentes, proporcionando novas funcionalidades a partir de componentes existentes (RANA; MOREAU, 2000). Através destes é possível usufruir de recursos localizados em qualquer parte do planeta, e assim, ter à disposição um poder de processamento e armazenamento potencialmente tão grande quanto se queira.

Para entender melhor a idéia de grade de computadores podemos fazer uma analogia com outra forma de infraestrutura mundialmente dispersa: a rede elétrica (CHETTY; BUYYA, 2002). A rede elétrica oferece energia para seus usuários e estes a utilizam, conforme as suas necessidades. Não importa onde vá, qualquer pessoa nos dias de hoje espera poder extrair uma quantidade razoável de energia de qualquer tomada do planeta (por exemplo, recarregar a bateria de seu celular). No contexto dos computadores, os usuários da grade teriam disponível poder de processamento em quantidade suficiente para atender suas demandas (CERA, 2005).

Nas grades computacionais é possível utilizar, de forma integrada e eficiente, todo o poder computacional representado pelos milhares, senão milhões, de computadores conectados pelas grandes redes de interconexão, sejam elas públicas ou privadas. Assim, corporações com grandes parques computacionais poderiam utilizar os próprios computadores desktop de seus funcionários para realizar processamento de alto desempenho evitando assim, ou ao menos retardando, a compra de caros computadores paralelos de alto poder computacional (LUCCHESE, F., 2006).

Além de poder de processamento, que é uma característica inerente a todo sistema computacional, outros recursos podem ser integrados a uma grade, como por exemplo o poder de armazenamento de dados dos computadores espalhados por uma rede, através da integração, de maneira transparente, de setores ociosos de dispositivos de armazenamento primário ou secundário, da mesma maneira que a grade citada anteriormente procurava integrar em um único sistema o poder computacional de máquinas ociosas. Assim, da mesma forma que a grade citada anteriormente procurava integrar em um único sistema o poder computacional de máquinas ociosos, esta procura integrar, de maneira igualmente transparente, setores ociosos de dispositivos de armazenamento primário ou secundário.

Os recursos das grades abrangem uma gama bastante variada, indo de computadores pessoais, aglomerados e supercomputadores até equipamentos específicos, tais como sensores, microscópios, entre outros. Uma grade computacional é definida com detalhes por Foster (FOSTER; KESSELMAN; TUECKE, 2001) e Buyya (BUYYA et al., 2002).

Jeffery (JEFFERY, K. G., 1999) apresenta uma definição de grades, dividida em três camadas, sendo elas:

- Grade computacional: nível mais baixo, preocupado com a união em larga escala de recursos computacionais e de armazenamento. Esta união de recursos pode ser utilizada

com o objetivo de criar um poder de processamento equivalente ou superior ao de um supercomputador e utilizá-lo no processamento de grandes volumes de dados. Os recursos neste caso podem não ser apenas poder de processamento, mas bases de dados e outros.

- Grade de informação: camada intermediária, permitindo o acesso uniforme às diferentes fontes de informação e tornando possível que os serviços possam ser executados em recursos computacionais distribuídos. Esta camada procura localizar, integrar e gerenciar as diferentes fontes de informação.

- Grade de conhecimento: é a camada mais alta desta definição e proporciona serviços especializados, os quais podem procurar por informações nos repositórios de dados existentes e gerenciar os serviços de informação. A grade de conhecimento pode auxiliar na tomada de decisões e na interpretação das informações manipuladas na grade de informação.

Krauter (KRAUTER; BUYYA; MAHESWARAN, 2002), apresenta uma definição mais abrangente e apresenta uma taxonomia para estes ambientes. A figura 2.1 demonstra a taxonomia proposta.

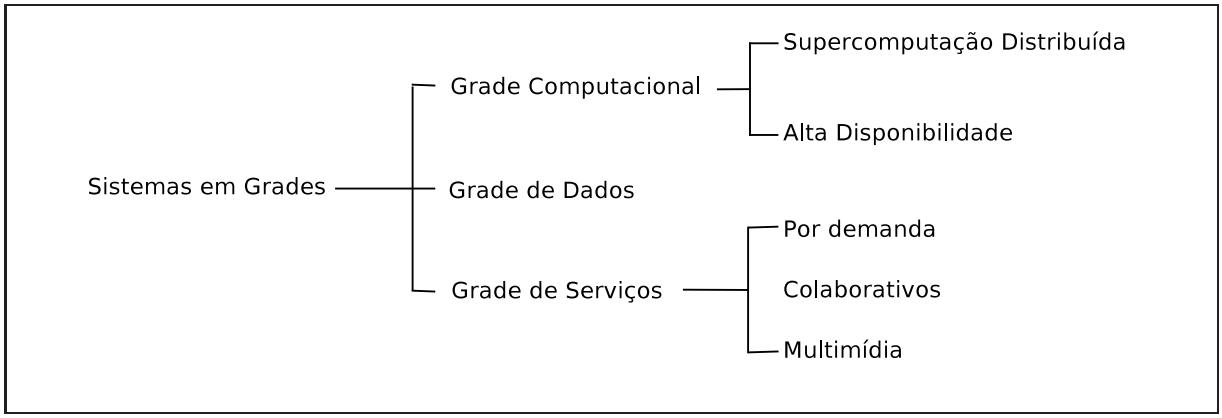

Figura 2.1: Taxonomia de Sistemas para Grades

Muitos esforços para se ter um ambiente de grades computacionais podem ser citados, onde o modelo de informação utilizado está associado a questões de como representar os recursos ou serviços (EEROLA et al., 2003). Pode-se citar Globus (FOSTER; KESSELMAN, 1998), Legion (GRIMSHAW; WULF; TEAM, 1997), Cadeo (CERA, 2005), NetSolve (AGRAWAL et al., 2003), UNICORE (*UNiform Interface COmputer REsources* (ERWIN; SNELLING, 2001)), Ninf (TANAKA et al., 2003), MyGrid (CIRNE et al., 2003), GRACE (*Grid Architecture for Computional Economy* (BUYYA; ABRAMSON; GIDDY, 2000)) entre outros. Esses projetos buscam oferecer gerenciamento dos recursos das grades possibilitando seu uso como plataforma de execução através de uma série de aspectos que influenciam o comportamento dos sistemas.

2.1 Alocação de Recursos

Um dos principais desafios das grades computacionais é alocar recursos para tarefas submetidas por usuários de forma a obter uma maximização do aproveitamento dos recursos conectados à grade. Para que seja realizada a alocação de recursos de maneira eficiente, são necessários que dois mecanismos estejam próximos da optimalidade, são eles:

- Descoberta de recursos

  O problema da localização de recursos é descrito por várias autores como sendo um dos principais componentes das grades compucionais (POLLERES; TOMA; FENSEL, 2005; BAKER; SHADBOLT, 2003), sendo também citado como um dos grandes desafios a serem estudados (GUPTA; AGRAWAL; ABBADI, 2005).

- Casamento[1] de recursos

  O serviço de casamento de recursos possui uma quantidade finita de recursos ociosos e uma quantidade de requerimentos a serem supridos. O casador analisa se um determinado recurso é suficientemente bom para ser aproveitado por uma tarefa. O grande problema deste serviço está associado ao poder de expressão das tarefas e a forma como os recursos são descritos. Para isso, alguns trabalhos (HARTH et al., 2004; PERNAS ANA M.; DANTAS, 2004; TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) propõem o uso de ontologias baseados em semânticas, possibilitando ao sistema uma ampla flexibilidade para serviços de casamentos de recursos.

Segundo Foster (LIU; FOSTER, 2004), é necessário um serviço escalável, eficiente e um mecanismo que expresse características e necessidades para haver um serviço de descoberta e seleção de recursos que satisfaça requisitos de aplicações. Um estudo relacionado aos problemas de alocação de recursos, de alguns sistemas amplamente citados na literatura, é apresentado nos próximas seções.

2.1.1 Globus

O Globus (FOSTER; KESSELMAN, 1998, 1997) é um dos projetos mais referenciados na literatura e consiste de uma infraestrutura de *software* que capacita aplicações a manipular recursos computacionais heterogêneos distribuídos como uma máquina virtual única. O elemento central do sistema Globus é o seu conjunto de ferramentas, que contém programas para segurança, infraestrutura de informação, gerenciamento de recursos e dados, comunicação, detecção de falhas e portabilidade.

Os serviços de Globus são divididos em serviços locais e globais. Cada serviço global é definido em termos dos serviços locais. Assim, para um dado serviço, como por exemplo alocacação de recursos, cada domínio administrativo executa um serviço local. Diversos destes serviços locais se unem para representar um serviço global de alocação de recursos. O conjunto de ferramentas do Globus possui uma estrutura modular que permite que as aplicações possam fazer uso apenas das características que lhes são necessárias. Por exemplo, uma determinada aplicação pode fazer uso da infraestrutura de gerenciamento de recursos e informação sem utilizar a biblioteca de comunicação disponibilizada pelo Globus.

O conjunto de módulos existentes no Globus é grande, sendo que dois deles são destinados ao gerenciamento dos recursos:

- *Grid Resource Allocation Manager* (GRAM) - é o protocolo que permite alocação remota, reserva, monitoração e controle dos recursos computacionais da grade. O gerenciamento de recursos do Globus é implementado de maneira hierárquica e na base desta hierarquia está o GRAM, o qual trabalha com recursos expressos através de uma linguagem de especificação de recursos conhecida como RSL (*Resource Specification Language*).

[1] Neste trabalho o termo **casamento** é utilizado para definir uma relação de associação entre descrição de recursos com requerimentos de aplicações

- *Meta-computing Directory Service* (MDS) - disponibiliza informações da grade. O Globus trabalha com um sistema de diretório de meta-computação, que contém informações sobre sobre diversos aspectos da grade, como a arquitetura dos nós, memória disponível, latência da rede, entre outros.

De maneira integrada, o GRAM pode, a qualquer momento, solicitar informações ao MDS sobre o estado atual dos recursos que compõem a grade e monitorar as tarefas em execução. A figura 2.2 demonstra a arquitetura do GRAM. Nela pode-se perceber três componentes, Gatekeeper, Job Manager e GRAM Reporter, bem como componentes externos que interagem com o GRAM. O **cliente GRAM** é aquele que o utiliza para submeter e controlar execução de tarefas. O cliente GRAM pode ser um escalonador de aplicação ou até o próprio usuário. Para o cliente, a grande vantagem de usar GRAM, é a manipulação uniforme de tarefas, como exemplo, a submissão e controle de tarefas não importando qual é o escalonador de recurso, (*Local Resource Manager*), usado para controlar a máquina. Isso é possível porque as requisições enviadas ao GRAM são sempre escritas em RSL, independetemente de qual escalonador de recurso esteja sendo utilizado. O **Job Manager** é o responsável por converter a requisição em RSL em um formato que o escalonador de recurso em questão entenda. Algumas versões do Job Manager podem ser utilizadas com o Condor, NQE, Codine, LSF, PBS, Unix, Windows, entre outros (CIRNE, 2003).

Ainda na figura 2.2, pode-se perceber o **Gatekeeper**, que consulta o *Globus Security Infrastructure* para identificar o usuário e verificar se ele tem permissão para executar tarefas. Neste caso, uma submissão de tarefas cria um **Job Manager** que é responsável por iniciar e monitorar a tarefa. Requisições sobre o estado da tarefa serão encaminhadas diretamente ao Job Manager.

O **GRAM Reporter** obtém informações de status e carga da máquina junto ao escalonador de recursos, **Local Resource Manager**, e as repassa para o MDS. O **MDS**, por sua vez, torna as informações disponíveis sob demanda para os outros componentes da arquitetura Globus.

2.1.1.1 Resource Specification Language - RSL

A RSL, *Resource Specification Language*, do Globus utiliza um conjunto de símbolos para representar a descrição dos recursos (CZAJKOWSKI et al., 1998). A sintaxe do RSL, representada na figura 2.3, é baseada na sintaxe das especificações do LDAP e no MDS.

A combinação de parâmetros de especificação e condições determinam como é realizada a busca por recursos. O operador **&**, especifica junção, ao contrário do operador |, que expressa disjunção. O operador + serve para unir dois ou mais pedidos em um único pedido composto. A figura 2.4 representa um exemplo de requerimento de recursos no Globus utilizando RSL. A expressão *count=5*, especifica a necessidade de 5 recursos com no mínimo 512 MB de memória, representado por *memory>=512*. Caso o Globus não encontre esses recursos, então pode-se utilizar 10 recursos com capacidade de memória superior a 256, *count=10 memory>256*. Neste requerimento, os atributos *executable* e *count* fazem parte do escalonador do Globus, enquanto o atributo *memory* faz parte do MDS.

2.1.1.2 Recursos inexistentes do Globus

Os serviços oferecidos pelo Globus não podem ser usados isoladamente, havendo a necessidade de *software* adicional. Sistemas como Condor, Legion, AppLeS entre outros, são comumente utilizados para fornecer recursos ao sistema. Seus usuários fazem uso de mecanismos de reserva, monitoração e controle de recursos baseados em informações fornecidas pelo serviço MDS, mas devem implementar seus próprios algoritmos de casamento de recursos.

O serviço GRAM, responsável pela alocação de recursos, não possui suporte a co-alocação

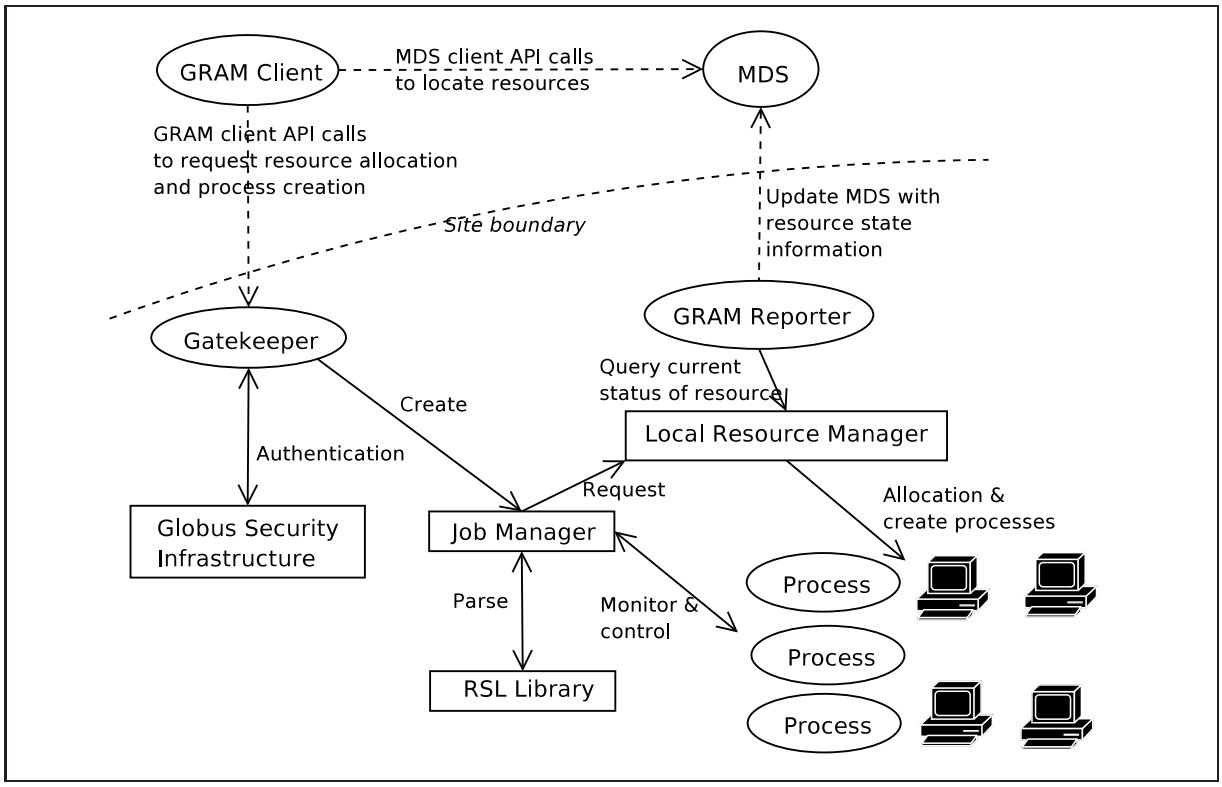

Figura 2.2: Estrutura do Gerenciamento de Recursos do Globus (CIRNE, 2003)

```
specification   := request
request         := multirequest | conjunction | parameter
multirequest    := + request-list
conjunction     := & request-list
disjunction     := | request-list
request-list    := ( request ) request-list | ( request )
parameter       := parameter-name op value
op              := | = | > | < | >= | <= | != |
value           := ([a...Z][0...9][-])+
```

Figura 2.3: Sintaxe para Descrição de Requisições RSL

```
&((executable=myprog)
(|(&count=5)(memory>=512))
(&(count=10)(memory>256)))
```

Figura 2.4: Requerimento de Recursos

de recursos. Caso uma aplicação solicite 10 recursos ao Globus e o sistema alocar apenas 8, não conseguindo os outros 2, a execução é abortada (KON, 2003).

Dentro do propósito deste trabalho, busca-se solucionar o problema de co-alocação com a utilização de níveis de qualidade. A aplicação pode, a qualquer momento, verificar o índice de qualidade obtida pelo sistema. Utilizando o exemplo do parágrafo anterior, caso o sistema não

encontre os 10 recursos, mas apenas 8, a aplicação saberá que o sistema de busca por recursos não conseguiu atingir uma qualidade ótima, porém pode beneficiar-se de uma solução quase ótima, e em um futuro próximo, outros recursos podem ser atribuídos à aplicação totalizando os 10 recursos solicitados.

2.1.2 Condor

O Condor (LITZKOW; LIVNY; MUTKA, 1988; THAIN; TANNENBAUM; LINVY, 2003; LIVNY et al., 1997) é um sistema que oferece localização de recursos e alocação de tarefas de forma automática através do monitoramento dos computadores disponíveis. Inicialmente, ele foi planejado para funcionar em NOWs, com o passar do tempo e as evoluções dos sistemas distribuídos, ele foi se adaptando a novas realidades.

O objetivo principal do sistema é maximizar a utilização de computadores com a menor interferência possível entre as tarefas utilizadas e as atividades do usuário do computador. Para isso, o sistema utiliza computadores em estado de ociosidade, ou seja, computadores que, por um determinado período, estejam pouco ou não estejam sendo utilizados por seus usuários. O sistema identifica os computadores em estado de ociosidade e os insere em um banco de recursos *resources pool*.

O Condor se propõe a oferecer grande quantidade de processamento a médio a longo prazo, ou seja, fornecer um desempenho uniforme a uma aplicação, mesmo que o potencial de desempenho do sistema como um todo seja variável. Tal fato o caracteriza como sendo um sistema capaz de oferecer alta vazão e não alto desempenho (LITZKOW; LIVNY; MUTKA, 1988; FREY et al., 2001; LIVNY et al., 1997).

O sistema também preocupa-se em proporcionar equilíbrio de acesso aos recursos computacionais a todos os seus usuários. Para isso, o Condor faz uso de um algoritmo de balanceamento de carga chamado *Up-Down* (MUTKA; LIVNY, 1987).

Uma evolução do Condor foi o desenvolvimento do Condor-G (FREY et al., 2001), a fim de possibilitar que o Condor pudesse adaptar-se ao conceito de grades de computadores apresentando uma visão mais heterogênea. O Condor-G representa o casamento do sistema ao Globus (FOSTER; KESSELMAN, 1998), onde além dos bancos de recursos o Condor utiliza recursos via Globus. O Condor-G apresenta uma arquitetura centralizada onde o escalonador submete tarefas de aplicações tanto ao banco de recursos quanto aos recursos disponibilizados pelo Globus. Também permite execuções de aplicações em Java (THAIN; LIVNY, 2002). Essa adaptação veio ao encontro da crescente gama de projetos que estão sendo desenvolvidos em Java.

Raman (RAMAN; LIVNY; SOLOMON, 1998) desenvolveu um *framework*, conhecido como Matchmaking, que possibilita ao Condor atribuir recursos a usuários através de um modelo de dados semi-estruturados, definido como *Classified Advertisements* (FREY et al., 2001).

No Matchmaking, agentes são responsáveis por anunciar suas capacidades e suas necessidades através de entidades. Estes anúncios são endereçados a um ponto central, ao *Matchmaking*, que é responsável por realizar pesquisas compatíveis às necessidades dos usuários com as disponibilidades dos recursos (LIU; FOSTER, 2004).

O problema de atribuição de recursos aos usuários do sistema no Matchmaking é visto como um relacionamento entre duas entidades:

- Existem usuários em busca de recursos para executar suas tarefas.
- Existem recursos em busca de tarefas a serem processadas.

O funcionamento interno do sistema Matchmaking é mostrado na figura 2.5. Pode-se perceber duas entidades, recursos e tarefas, que informam ao Matchmaking seus requerimentos. O

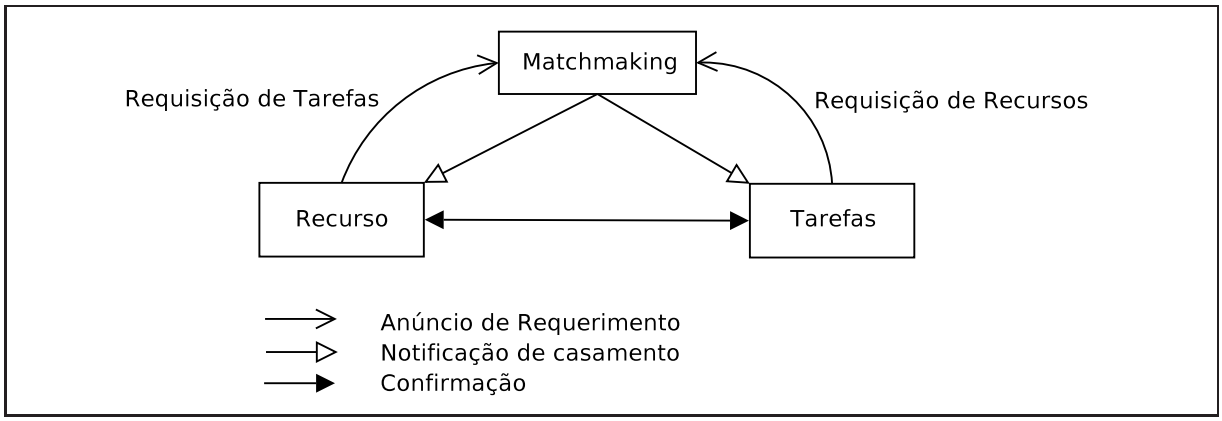

Figura 2.5: Ações Envolvidas no Matchmaking

Matchmaking trabalha com o conceito de que tanto os recursos quanto as tarefas possuem necessidades. Os recursos, quando aptos a serem utilizados pelo sistema, realizam requisições ao Matchmaking em busca de tarefas. As tarefas também realizam requisições ao Matchmaking, solicitando recursos para que sejam processada. Os requerimentos das entidades são expressos através de *ClassAd* (RAMAN; LIVNY; SOLOMON, 1998). As figuras 2.6 e 2.7 representam exemplos de descrição de um recurso e uma tarefa.

```
[
    Type        = "Machine";
    Activity    = "Idle";
    KeybrdIdle  = '00:23:12';
    Disk        = 10000M;
    Memory      = 512M;
    State       = "Unclaimed"
    LoadAvg     = 0.04269;
    Mips        = 104;
    Arch        = "INTEL";
    OpSys       = "LINUX";
    KFlops      = 21893;
    Name        = "foo.cs.wisc.edu";
    Subnet      = "128.105.175";
    Rank        = DayTime() >= '9:00' &&
                  DayTime() <= '17:00' ?
                  1/other.Image : 0;
    Constraint  = other.Type=="Job" &&
                  other.Owner != "riffraff" &&
                  LoadAvg < 0.3 && KeybrdIdle > '00:15'
]
```

Figura 2.6: Descrição de um Recurso com ClassAd

A arquitetura do Matchmaking permite o uso de mecanismos bilaterais, apenas uma entidade pode realizar consultas ou anúncios por vez, não permitindo um gerenciamento de recursos mais avançado. Para resolver este problema, Raman (RAMAN; LIVNY; SOLOMON, 2004) desenvolveu uma extensão do modelo original do Matchmaking, batizando-o de Gangmatching. Esta nova implementação permite ao Condor fazer uso de co-alocação de recursos.

```
[
    Type                = "Job";
    Owner               = "raman";
    Cmd                 = "run_sim";
    WantRemoteSysCalls  = 1;
    WantCheckpoint      = 1;
    Iwd                 = "/usr/local/sim2";
    Args                = "-Q 17 3200 10";
    Memory              = 128;
    Rank                = "KFlops/1E2 + other.Memory/32;
    Constraint          = other.Type == "Machine" &&
                          Arch == "INTEL" &&
                          OpSys == "SOLARIS252" &&
                          Disk >= 10000 &&
                          other.Memory >= self.Memory;
]
```

Figura 2.7: Descrição de uma Tarefa com ClassAd

2.1.2.1 Linguagem ClassAd

A linguagem para expressão de recursos e tarefas utilizadas pelo Condor é a *ClassAd*. Esta é utilizada quando o Condor precisa descobrir e alocar recursos. *ClassAd* é um modelo de dados flexível e extensível que pode representar tanto necessidades de aplicações quanto descrição de recursos. A estrutura de dados é baseada no trabalho de Nestorov (NESTOROV; ABITEBOUL; MOTWANI, 1997), que propõe uma notação de dados semi-estruturados através de níveis hierárquicos. A linguagem ClassAd pode fazer uso de atributos dos tipos: **int**, **real** ou **string**. Também podem utilizar expressões aritméticas e operadores lógicos.

2.1.2.2 Condor versus Grades

O sistema Condor não é exatamente voltado para grades de computadores, mas para *cluster*. Os *frameworks* Matchmaking e Gangmatching foram projetados para trabalharem sob NOWs. Usuários do Condor podem usufruir das grades apenas quando o sistema Globus é utilizado. Toda a estrutura para suporte a grade é oferecida pelo Globus.

2.1.3 Legion

Assim como o Globus, o sistema Legion (GRIMSHAW; WULF; TEAM, 1997; GRIMSHAW et al., 1999; CHAPIN et al., 1999a) visa oferecer serviços básicos para a utilização das grades de computadores, tais como segurança e gerenciamento de recursos e dados. O sistema oferece uma máquina virtual única que será composta por computadores, independentemente da dispersão geográfica dos mesmos. O Legion implementa serviços de mais alto nível utilizando para isso sistemas operacionais, ferramentas de gerenciamento de recursos e mecanismos de segurança já existentes. O sistema busca fornecer, dentre outras coisas, autonomia para os domínios, suporte a heterogeneidade, extensibilidade, facilidade de uso, desempenho, tolerância a falhas e escalabilidade.

O Legion é um sistema baseado em objetos composto por um conjunto de objetos independentes que se comunicam entre si através de invocações remotas de métodos. Essa é uma característica que o difere do Globus já que fornece ao ambiente de grade o encapsulamento de todos os seus componentes em objetos. Essa metodologia tem todas as vantagens normais

de um modelo orientado a objetos, tais como abstração de dados, encapsulamento, herança e polimorfismo (GRIMSHAW et al., 2003).

O Legion oferece um casador de tarefas básico que procura casar descrições de recursos com com as necessidades das tarefas das aplicações. O sistema para gerenciamento de recursos utilizado no Legion é baseado em camadas. Nestas, são utilizadas uma infra-estrutura de gerenciamento de recursos (RMI [2]).

Na figura 2.8 pode-se visualizar a hierarquia dos níveis da arquitetura do RMI, sendo que cada uma pode interagir com os outros níveis através de comunicação entre objetos. Um modelo de mais baixo nível sobre o processo de gerenciamento de recursos adotado pelo Legion pode ser dividido em componentes:

- *Hosts, Vaults* - Recursos básicos;
- *Collection* - Base de informações;
- *Enactor* - Escalonador;
- *Monitor* - Executor de tares.

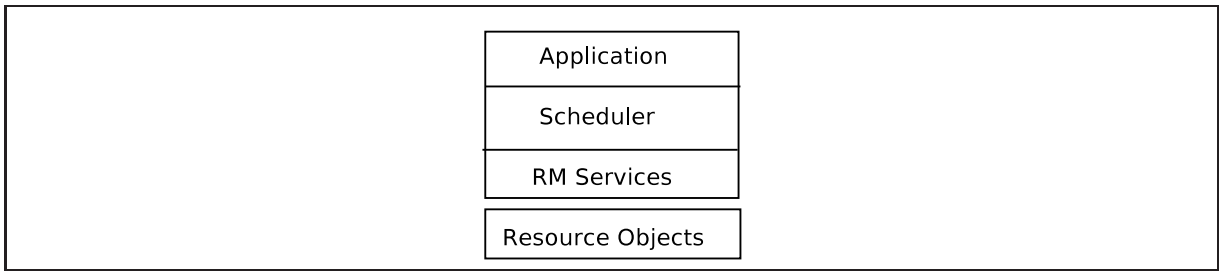

Figura 2.8: Legion: Níveis de Gerenciamento de Recursos (CHAPIN et al., 1999a)

2.1.3.1 MESSIAHS Interface Linguage - MIL

O Legion utiliza uma gramática baseada na linguagem MIL (MESSIAHS Interface linguage) (CHAPIN; SPAFFORD, 1994). Esta linguagem faz uso de quatro tipos básicos de dados: **int**, **boolean**, **float** e **string**.

Na figura 2.9 pode-se visualizar as possíveis expressões que podem ser utilizadas com a gramatica da linguagem MIL. A linguagem é baseada em uma estrutura hierárquica de domínios administrativos.

2.1.3.2 Barreiras do Legion

O projeto Legion foi encerrado em 2001. Uma companhia conhecida como Avaki (AVAKI, 2003) adquiriu os direitos legais do Legion da Universidade de Virginia. Assim, o sistema Legion foi renomeado para Avaki, o qual mantém a arquitetura e alguns servicos de Legion. Alguns de seus servicos e funcionalidades foram removidos, uma vez que Avaki é voltado para aplicações comerciais.

[2]Resource Management Infrastructure

```
int-binop    →  + | - | / | * | mod | & | I | max | min

int-expr     →  int-expr int-binop int-expr | (int-expr) |
                integer | int(float-expr) | id

string-expr  →  string-expr + string-expr | (string-expr) |
                string | id

float-binop  →  + | - | / | * | max | min

float-expr   →  float-expr float-binop float-expr |
                (float-expr) | float | float(int-expr) | id

comp         →  < | > | = | >= | <= | <>

bool-binop   →  and | or | xor

bool-expr    →  bool-expr bool-binop | bool-expr |
                not bool-expr |
                int-expr comp int-expr |
                float-expr commp foat-expr |
                string-expr comp string-expr |
                match(string-expr, string-expr) |
                (bool-expr) | true | false | id
```

Figura 2.9: Gramática Utilizada pelo Legion

2.1.4 MyGrid

O MyGrid (CIRNE et al., 2003; CIRNE; MARZULLO, 2001) é um sistema que possibilita a execução de tarefas de aplicações paralelas em computadores da grade tendo como objetivo ser simples, completo e seguro. Para evitar problemas quanto à definição de quais são os computadores pertencentes à grade, o MyGrid utiliza-se de uma política que baseia-se nas permissões dos usuários do sistema. Para o MyGrid, a grade de um usuário é formada por todos os computadores onde esse usuário possua acesso, independente da localização desses recursos. As aplicações executadas através desse sistema devem seguir o modelo de aplicações do tipo sacola de tarefas (*bag-of-task*) (ANDREWS, 2000), contribuindo para sua distribuição entre computadores geograficamente distribuídos. A arquitetura do MyGrid é flexível, a fim de suportar a dinamicidade dos computadores, uma vez que facilita sua inclusão e exclusão na grade.

Os usuários que desejarem executar suas aplicações com o MyGrid devem inicialmente submeter as tarefas da aplicação para o computador que coordenará a distribuição delas. Esse computador é chamado de máquina base (*home machine*) e costuma ser o próprio computador do usuário. As tarefas serão distribuídas entre os computadores que compõem a grade do usuário, as quais recebem a denominação de máquinas da grade (*grid machines*). As tarefas inicial e final são executadas na máquina base, sendo que a primeira delas inicia o ambiente e transfere dados de entrada, caso necessário. E a última é responsável pela espera dos resultados da tarefa caso existam.

O MyGrid não possui tratamento de busca por recursos baseadas em necessidades de usuário. Ao invés disso ele utiliza todos os recursos disponíveis para aos usuários. O myGrid define o Grid Machine Abstraction como sendo o conjunto mínimo de serviços necessários para que as máquinas possam ser adicionadas ao Grid do usuário. Os serviços necessários são: execução remota; transferência de arquivos da máquina do grid para a máquina base e transferência de arquivo da máquina base para a máquina do Grid.

A figura 2.10 ilustra como acontece a comunicação com as máquinas que fazem parte da

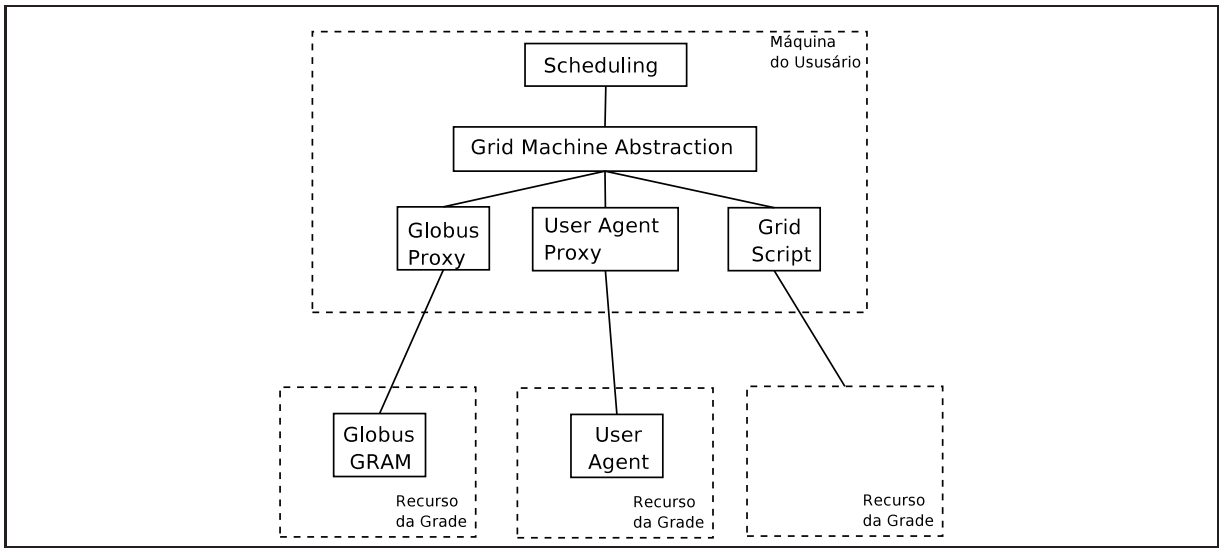

Figura 2.10: Arquitetura do MyGrid

grade. O MyGrid disponibiliza três implementações nativas para a realização da comunicação:

- *Globus Proxy* - O gerenciamento de recursos do MyGrid é oferecido pelo Globus.
- *UserAgent* - Os agentes de usuários costumam ser usados quando é fácil instalar software nas máquinas da grade e esses agentes fornecem uma implementação básica, em Java, dos serviços necessários.
- *GridScript* - Os *scripts* de grade fornecem as mesmas funcionalidades dos agentes de usuário, porém usando *scripts* ao invés de processos Java.

A continuidade do trabalho realizado no MyGrid é o OurGrid (ANDRADE et al., 2003), o qual permite a criação de comunidades em larga escala. O OurGrid trabalha com os conceitos da computação ponto-a-ponto (*peer-to-peer*). Nessas redes, alocar um recurso para requerir um serviço é um favor e os recursos de um ponto da rede são disponibilizados para os outros pontos. O critério para a disponibilização é a existência de créditos em seu histórico de interações, ou seja, um ponto disponibilizará recursos, prioritariamente, com outros pontos que já tenham lhe prestado algum favor. O compartilhamento de recursos em redes P2P também pode ser considerada uma iniciativa para o melhor aproveitamento dos recursos de arquiteturas paralelas. Essa iniciativa é recente, mas tende a evoluir e ser cada vez mais empregada como meio de utilização de recursos.

2.1.4.1 Deficiências do MyGrid

O funcionamento interno do MyGrid não considera as necessidades de seus usuários, significando que o sistema é incapaz de selecionar, dentre um conjunto de recursos disponíveis, os mais adequados à determinadas tarefas. O MyGrid também não se propõe a resolver aspectos gerenciais dos recursos em grades. Para o sistema, é difícil obter boas informações sobre as tarefas a serem executadas e, sem poder contar com tais informações, torna-se difícil encontrar soluções capazes de fornecer um bom escalonamento das cargas de processamento. Para contornar esse problema, o MyGrid utiliza um algoritmo de fila com replicação (*Work Queue with*

Replication - WQR) (PARANHOS; CIRNE; BRASILEIRO, 2003), mas este algoritmo, que oferece bom desempenho, não considera informações das tarefas e nem dos recursos pertencentes a grade.

2.1.5 Cadeo

O Cadeo (CERA, 2005) é um sistema que visa proporcionar um melhor aproveitamento do poder de processamento de recursos ociosos, sendo totalmente desenvolvido em Java, possilita implementar aplicações paralelas e distribuídas de forma simples e intuitiva. O sistema se encarrega de disponibilizar e gerenciar computadores de forma totalmente transparente às aplicações.

A idéia é fazer uso de uma plataforma de execução, composta por um conjunto de recursos que encontram-se temporariamente disponíveis. Estes, integram o sistema e devem estar em estado de ociosidade, ou com baixa utilização, podendo a qualquer momento deixarem de estar disponíveis. Isto faz com que o número de recursos do Cadeo se altere constantemente. Em função dessa característica, a plataforma de execução é dinâmica, ou seja, o conjunto de recursos que a integram é variável.

O sistema oferece transparência na localização das tarefas de aplicações paralelas e distribuídas, onde o mesmo esconde a localização dos computadores que as executam. Para o Cadeo, as aplicações que solicitarem poder de processamento para sua execução serão contempladas por um conjunto de recursos.

Um dos principais módulos do Cadeo é o **alocador**. Ele é responsável pela alocação e controle dos computadores ociosos. É o núcleo central do sistema Cadeo. Todas as máquinas que estão aptas a receber um trabalho são mantidas, através de referência, no alocador. O conjunto de referências a computadores ociosos compõe um aglomerado dinâmico que permanece em posse do alocador até que haja demanda pelos seus recursos. O Cadeo apresenta um único módulo alocador, que centraliza as informações e gerencia a distribuição de todos os computadores que estão disponíveis no sistema.

Ao existir tarefas a serem executadas, o alocador seleciona algumas das máquinas disponíveis para a execução dessa tarefas. A figura 2.11 apresenta um cenário hipotético do funcionamento do Cadeo. Nela podem ser identificados os computadores potencialmente ociosos que integram o sistema, sendo que, neste caso, estes podem se apresentar em dois estados: ociosos ou indisponíveis. Também tem-se o gerenciador, com a identificação do conjunto de computadores ociosos (disponíveis) e uma aplicação paralela composta pelo seu conjunto de tarefas. Para possibilitar a execução de suas tarefas a aplicação solicita ao gerenciador um conjunto de computadores, essa requisição esta indicada na figura 2.11 pela seta no sentido da aplicação para o gerenciador.

O gerenciador tem a função de selecionar recursos, oferecendo equilíbrio na distribuição dos recursos disponíveis entre as aplicações paralelas carentes por computadores. Assim como no MyGrid, o Cadeo não possui tratamento de busca por recursos baseados em necessidades. O que há, são dois módulos distintos:

- **módulo de busca de recursos ociosos** - todos os recursos que se encontram em estado de ociosidade são alocados.
- **módulo de escalonamento** - atribui tarefas a recursos ociosos.

Quando um computador torna-se disponível ao sistema, ele passa a fazer parte de um aglomerado dinâmico. Como o tempo de disponibilidade desse recurso é variável, o aglomerado se apresenta de forma dinâmica, onde os recursos são freqüentemente acrescentados e removidos do aglomerado. Esse aglomerado pode ser particionado em dois ou mais, a fim de atender

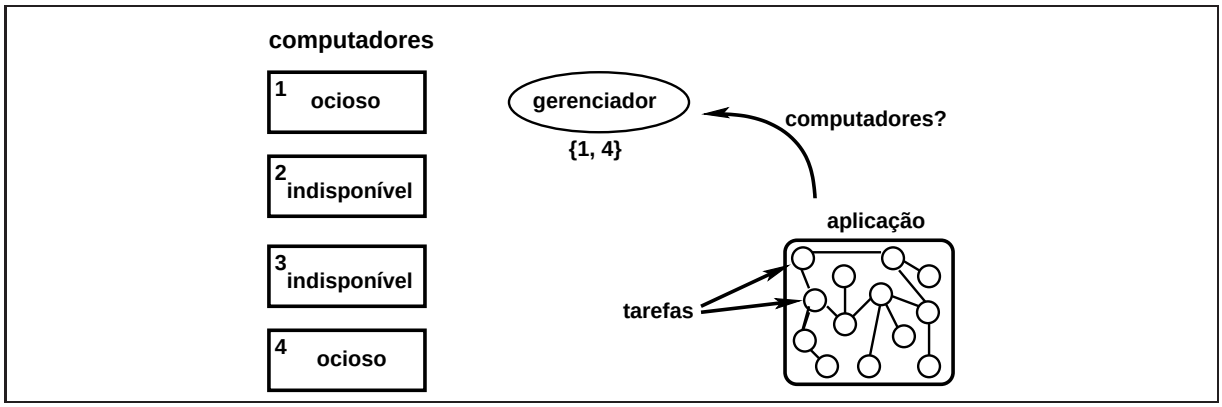

Figura 2.11: Cenário do Sistema Cadeo

solicitações de recursos por aplicações. Cada aplicação que executar suas tarefas através do sistema, terá à sua disposição um aglomerado dinâmico, sendo que a dinamicidade do mesmo é totalmente transparente para a aplicação.

A formação dos aglomerados dinâmicos não usa qualquer meio para selecionar recursos que melhor atendem as necessidades da aplicação. Quando ocorre alteração do estado de ociosidade de um membro do aglomerado dinâmico, outro recurso, se disponível, simplesmente é adicionado ao aglomerado, sem verificar a qualidade deste novo recurso. Para minimizar este problema de adequação de recursos à aplicação, o Cadeo precisa fazer uso de mecanismos que selecionem apenas recursos que melhor satisfazem as necessidades das aplicações. Neste caso, a formação dos aglomerados dinâmicos pode ser realizada através de filtragem de recursos disponíveis, melhorando desta forma a qualidade de serviço oferecido a usuários do sistema.

2.1.5.1 Benefícios do MatchMaker para Cadeo

O sistema proposto propiciará, ao ambiente Cadeo, suporte a alocação de recursos geograficamente distribuídos através das grades computacionais, baseando-se em necessidades de usuários. Todo usuário Cadeo beneficia-se de seleção de recursos disponíveis para execução de suas tarefas. Por ser um sistema de alocação de recursos para grades computacionais onde são consideradas informações de tarefas e recursos, um amplo número de recursos poderão estar aptos a trabalharem nos aglomerados dinâmicos do ambiente Cadeo.

O MatchMaker trabalhará com o núcleo central do Cadeo, o **alocador**, e através de algoritmos de casamento de tarefas poderá selecionar recursos ociosos de forma a obter uma boa relação de aproveitamento da capaciadade dos recursos disponíveis.

2.2 Programação em Grade

A computação distribuída vem sofrendo grandes mudanças nos últimos anos, e estas conduzem a novos avanços significativos na funcionalidade e na qualidade de *middleware*, principalmente os relacionados a rede e aos protocolos. Entretanto, o estilo de programação de sistemas distribuídos mudou muito pouco. Um exemplo é a falta de padronização para programação de ambientes destinados a grades computacionais.

Existem esforços destinados ao desenvolvimento de metodologias para a programação de grades baseadas em serviços e não em objetos (SOBOLEWSKI; KOLONAY, 2006). Porém, a metodologia utilizada neste trabalho é totalmente baseada em objetos, e para tal, faz uso da

linguagem de programação Java.

2.2.1 Java

A linguagem de programação Java vem conquistando, através dos últimos anos, um grande número de programadores. Suas características básicas, como simplicidade do modelo de programação orientada a objetos e a portabilidade entre diferentes plataformas, a tornam um grande atrativo para o desenvolvimento de aplicações.

Java, além da programação seqüencial, suporta programação paralela e distribuída através de um conjunto de APIs. O grande problema, neste caso, está na interface de comunicação entre computadores. Java faz uso de RMI (*Remote Method Invocation*), uma abstração de alto nível do sistema de *socket*. Java necessita serializar objetos para transmiti-los pela rede e este serviço é de baixa eficiência e acaba por comprometer o desempenho da comunicação (CAROMEL; KLAUSER; VAYSSIERE, 1998a). Além deste *overhead* causada pela serialização de objetos o RMI utiliza comunicação síncrona. Isto também é um problema, em relação a desempenho, para a comunicação, fazendo com que o método invocador fique em espera por necessidade.

Algumas bibliotecas foram desenvolvidas para resolver o problema da baixa eficiência durante a execução de programas remotos em Java. Nesse contexto, inserem-se as bibliotecas Java para programação paralela e distribuída, as quais oferecem criação e manutenção de vários fluxos de execução, mecanismos eficientes de sincronização e passagem de mensagem. Como exemplos desse tipo de bibliotecas tem-se o JavaParty (HAUMACHER; MOSCHNY; PHILIPPSEN, 2004; PHILIPPSEN; ZENGER, 1997) e o ProActive (CAROMEL; KLAUSER; VAYSSIERE, 1998b).

O JavaParty busca oferecer mecanismos para implementar aplicações paralelas em Java, focados em sistemas com memória distribuída. JavaParty oferece transparência de localização, ou seja, o JavaParty encarrega-se de mapear a localização de objetos e fluxos de execução, multiprogramação distribuída, e estes podem ser utilizados como se fossem locais. O tratamento dos códigos com a sintaxe do JavaParty é realizado por meio de um pré-processador que gera códigos aptos a realizarem migração, possuírem localização transparente e realizarem serialização mais eficiente (PHILIPPSEN; HAUMACHER; NESTER, 2000). Dessa forma o JavaParty consegue melhoras de desempenho na invocação remota de métodos, além de fornecer uma imagem única do sistema, como uma única máquina virtual.

O ProActive (BAUDE; CAROMEL; MOREL, 2003) é uma biblioteca inteiramente composta por classes Java e apresenta total compatibilidade com o Java tradicional, não sendo necessárias alterações na JVM para o seu funcionamento. O ProActive utiliza um modelo de programação orientado a objetos distribuídos (BADUEL et al., 2006). Ele oferece invocação remota de métodos de forma assíncrona, espera por necessidade, migração, segurança e polimorfismo entre objetos locais e remotos, comunicação em grupo, infra-estrutura P2P e mecanismos de *checkpointing* e tolerância a falhas. Um dos principais objetivos do ProActive é reduzir a distância entre a programação multiprocessada e a programação distribuída. Dessa forma, é possível reutilizar códigos de aplicações com múltiplos processos e executá-las de forma distribuída. Para tornar viável tal objetivo é necessário que os objetos possam ter transparência de localização, a fim de proporcionar polimorfismo entre objetos locais e remotos. No ProActive, a localização de objetos instanciados remotamente é transparente, mas a localização deve ser conhecida no momento da instanciação. Além disso, é necessário que haja transparência nas atividades dos objetos uma vez que no ProActive as invocações de método serão realizadas em uma *thread* existente associada àquele objeto (CAROMEL; KLAUSER; VAYSSIERE, 1998b).

Pelas características apresentadas no parágrafo anterior, optou-se por realizar um estudo mais aprofundado dos benefícios oferecidos pelo uso do ProActive em ambientes de grades

computacionais. Este estudo é descrito na próxima sessão.

2.2.2 Middleware ProActive

O ProActive é um *middleware* para sistemas de grades computacionais que possibilita programação paralela e distribuída fornecendo aspectos avançados em mobilidade e segurança (CAROMEL, 2004; CAROMEL; KLAUSER; VAYSSIERE, 1998b; CAROMEL, 1993), explanada a seguir.

2.2.2.1 Comunicação Assíncrona

Para proporcionar cacterísticas avançadas, o ProActive utiliza-se de uma abstração de objetos ativos. Um objeto ativo é um objeto padrão do Java com um fluxo de execução associado a ele, e este fluxo controlará os serviços oferecidos pelo objeto. Ele é composto pelo objeto padrão do Java e uma *thread*, chamada de corpo (*body*) associada a ele. O corpo é responsável por receber as invocações de método de um objeto ativo e ordená-las em uma lista de requisições pendentes.

Além dos objetos ativos, o ProActive possui objetos futuros que possibilitam assincronismo na invocação de métodos e espera por necessidade, realizando comunicação assíncrona com sincronização automática (BAUDE; CAROMEL; MOREL, 2003).

Sempre quando possível as chamadas a métodos para objetos ativos são assíncronos. Quando isso não é possível uma chamada síncrona e bloqueante ocorre até que o retorno seja recebido. Nos casos de chamadas assíncronas, é retornado imediatamente um objeto futuro.

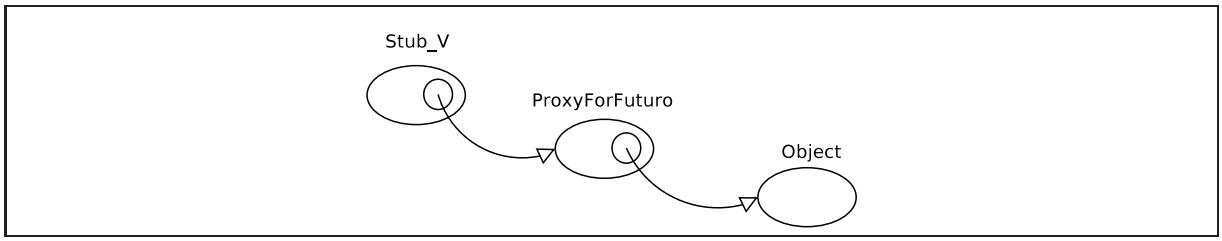

Figura 2.12: Objeto Futuro

2.2.2.2 Migração de Tarefas

Uma das grandes vantagens da utilização do *middleware* ProActive é o serviço de migração de tarefas (BAUDE et al., 2000a), o qual permite que um objeto ativo seja migrado, de forma transparente, entre diferentes JVMs. Além de poder mover os objetos entre diferentes JVMs é preciso que se possa comunicar com o objeto migrado independentemente do local onde ele se encontre. O ProActive mantém um rastreamento da localização dos objetos migrados através de um servidor de localização e todo esse processo ocorre de forma transparente ao programador. A migração oferecida por esta biblioteca é do tipo fraca, uma vez que não se tem acesso à fila de execução da JVM (CERA, 2005). Esta migração pode ser realizada pelo próprio objeto ativo, ou algum evento externo. Em ambos os casos, o ProActive utiliza uma simples primitiva que será chamada para realizar a migração.

Todos os objetos ativos possuem capacidade de migração. Para migrar, um objeto ativo deve ter um método que contenha uma chamada ao método primitivo da migração. As comunicações remotas são realizadas transparentemente através de um *proxy* o qual contem a localização

efetiva dos objetos. O *proxy* também faz repassagem de invocação de objetos móveis (BAUDE et al., 2000b).

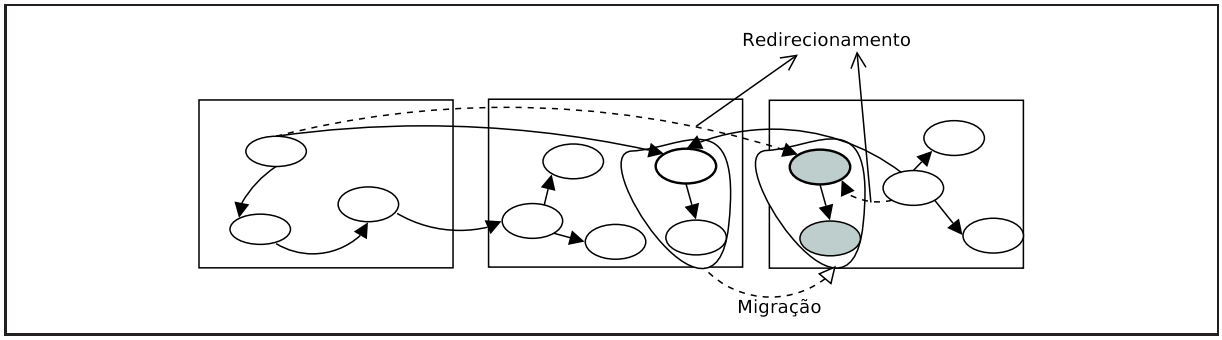

Figura 2.13: Modelo Básico de Migração de Tarefas

2.2.2.3 Segurança

O processo de identificação em sistemas computacionais consiste de um conjunto de procedimentos e mecanismos que permite que agentes externos, usuários, dispositivos, etc, sejam identificados como entidades autorizadas segundo as políticas de seguranca adotadas no sistema (MELLO, 2006).

O ProActive oferece um conjunto de políticas de segurança que vai desde autenticação de comunicação, integridade, confidencialidade até mecanismos de segurança de migração, políticas de segurança hierárquica e políticas de negociação dinâmicas, conforme representado pela figura 2.14. Todas as características são expressadas e usadas transparentemente por aplicações (ATTALI; CAROMEL; CONTES, 2005).

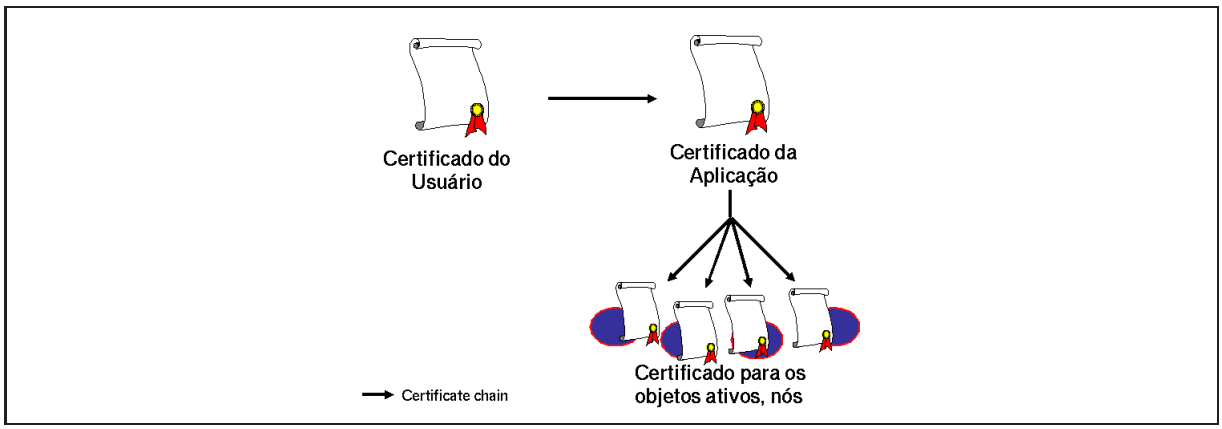

Figura 2.14: Modelo Básico de Certificação do ProActive

No ProActive, a segurança é expressada através de diferentes níveis, de acordo com um conjunto de políticas configuráveis:

- Administradores configuram políticas em nível de domínio - definição de regras gerais.
- Donos de recursos configuram políticas para recursos - pessoas que tem acesso a aglomerados de computadores podem querer oferecer tempos de cpu com algumas regras de acesso.

- Política em nível de Aplicação - aplicações podem ser políticamente configuradas com restrições através de descritores XML

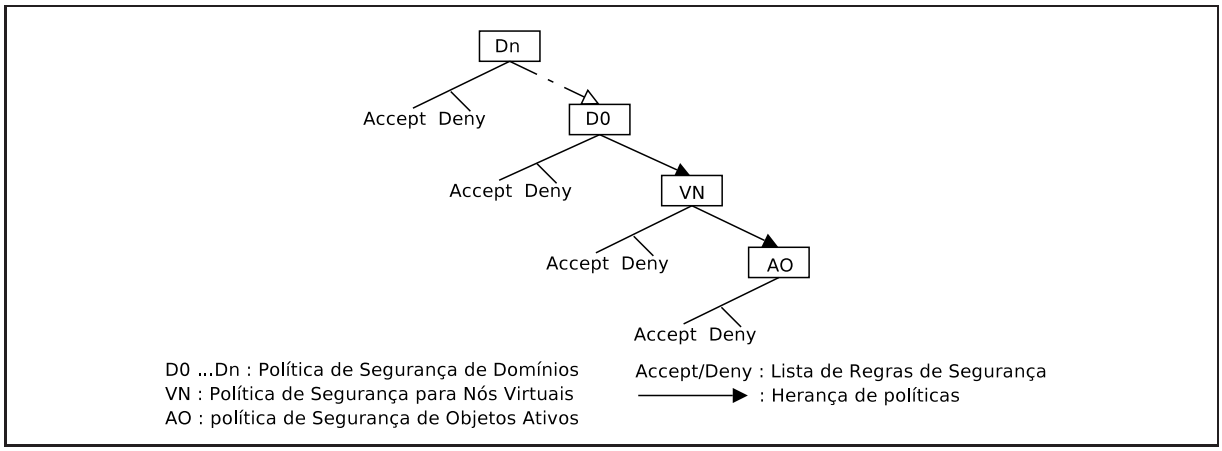

Figura 2.15: Modelo Básico de Políticas de Segurança do ProActive

A arquitetura da segurança relaciona-se com duas abstrações relacionadas à aplicações para grade:

- **Nós** - um nó utiliza diversos objetos em uma entidade lógica. Isso fornece uma abstração para localização física de um conjunto de atividades. Os objetos são limitados a um nó na hora de sua criação ou após uma migração.

- **Nós Virtuais** - para se ter uma distribuição mais flexível, o sistema confia em nós virtuais (NVs). Um NV é identificado com um nome, uma simples String, usado dentro do código fonte do programa, definido e configurado em um descriptor. O usuário pode adicionar políticas para estes nós virtuais. Os nós virtuais são usados dentro de códigos da aplicação (INRIA - ProActive, 2006).

As políticas de segurança são definidas de acordo com todos os casamentos de regras, conforme a Figura 2.15. Os pesquisadores que trabalham no desenvolvimento do ProActive demostram grande preocupação com o quesito segurança. A prova disso é o incremento de novos mecanismos direcionados a resolverem alguns problemas de segurança.

A partir da versão 3.0, o *middleware* oferece suporte ao uso de tuneis SSH, permitindo assim a criptografia do tráfego da rede. Podem ser utilizada conexões RMI e HTTP através destes tuneis. Administradores de redes de domínios administrativos normalmente não se preocupam com conexões externas da sua rede pela porta 22, do SSH. Facilitanto o uso e dando credibilidade às conexões em grade. Isso facilitaria a comunicação entre os *da*s da grade.

2.2.2.4 Comunicação em Grupo

Comunicação em grupo é um aspecto importantíssimo para computação de alto desempenho e para sistemas em grades (BADUEL; BAUDE; CAROMEL, 2002). O mecanismo de comunicação em grupo do ProActive realiza invocação de métodos assíncronos para um grupo de objetos remotos (CAROMEL; KLAUSER; VAYSSIERE, 1998a).

Programadores que fazem uso de RMI, linguagem java, para comunicação remota utilizam o mecanismo de comunicação ponto-a-ponto, um padrão para comunicação cliente-servidor, através de chamadas de métodos síncronos. Na computação em grade o RMI puro não é suficiente, uma comunicação coletiva e assíncrona garante eficiência na interação entre objetos.

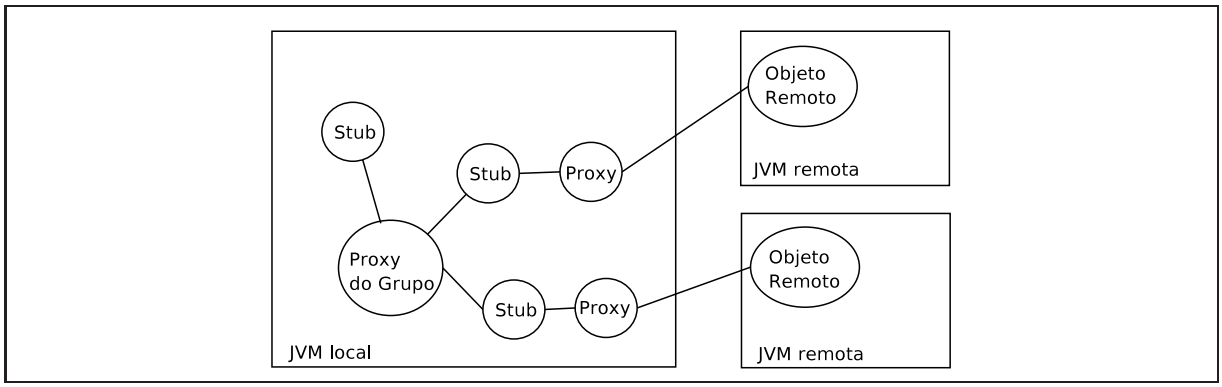

Figura 2.16: Modelo Básico de Comunicação em Grupo

2.2.2.5 Infra-Estrutura P2P

A computação P2P está tornando-se fundamental em ambientes de execução. O potencial uso de, por exemplo, 100.000 nós interconectados para execução de uma simples aplicação é um atrativo convincente, especialmente para grades computacionais.

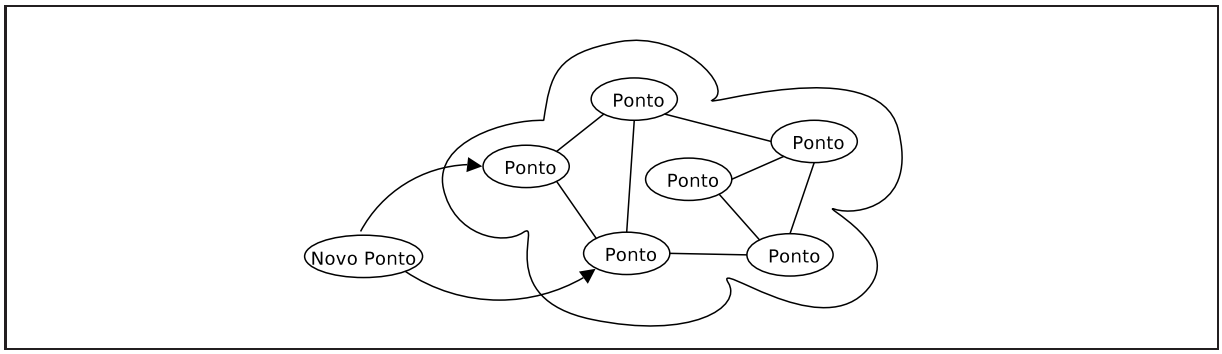

Figura 2.17: Modelo Básico de Comunicação em P2P

Na figura 2.17, a infra-estrutura P2P é implementada utilizando ProActive. Os compartilhamentos de recursos não são feitos pelas JVMs, mas por nós ProActive que trabalham nas extremidades das redes.

Com o uso de objetos ativos, os *Peer* tornam-se entidades independentes que trabalham sob pedidos de usuário através de filas FIFO. O *Peer* é também um cliente que emite pedidos a outros *Peer*. Todas as comunicações entre os *Peer* utilizam comunicação em grupo, mas para enviar respostas a requerimentos de mensagens, a comunicação é do tipo ponto-a-ponto. A infra-estrutura P2P é auto organizada e auto configurada. Existem três parâmetros principais de configuração que o ProActive permite:

- ***Time To Update* (TTU)** - cada *Peer* verifica entre seus pares quando o tempo TTU expira.
- ***Number Of Acquaintances* (NOA)** - é o número mínimo de vizinhos que um *Peer* necessita conhecer para fazer parte a infraestrutura.
- ***Time To Live* (TTL)** - número de ”saltos” entre os *Peer* para as buscas em profundidade no sistema P2P.

2.2.2.6 Tolerância a Falhas

O *middleware* ProActive tem a capacidade de trabalhar com tolerância a falhas através de dois protocolos diferentes:

- CIC - Protocolo *Communication-Induced Checkpointing*;
  - Todo objeto ativo em uma aplicação tolerante a falhas que utiliza CIC tem definido um tempo para realizar o *checkpoint* em TTC *Time To Checkpoint*.
- PML - Protocolo *Pessimistic Message Logging*.
  - Todo objeto ativo em uma aplicação tolerante a falhas que utilize PML tem um TTC para realizar o checkpoint onde todas as mensagens entregues a objetos ativos são armazenadas.

Segundo o instituto INRIA, o protocolo PML possui um *overheard* maior do que o protocolo CIC para realizar o *checkpoint*, porém o tempo de recuperação, caso ocorra falhas é menor.

Fazer uma aplicação ProActive tolerante a falhas é inteiramente transparente. Os objetos ativos utilizam tolerância a falhas usando propriedades do java que podem ser ajustadas no descritor da JVM. O programador pode selecionar, durante o desenvolvimento da aplicação, o protocolo que mais se adaptada a sua aplicação. Os objetos ativos usam a serialização padrão de Java.

3 ARQUITETURA DO MATCHMAKER

Este capítulo tem como objetivo apresentar uma visão abrangente sobre o sistema proposto. Dentro desta, serão expostas as principais características/idéias de funcionamento e sua arquitetura, destinada à alocação de recursos geograficamente distribuídos.

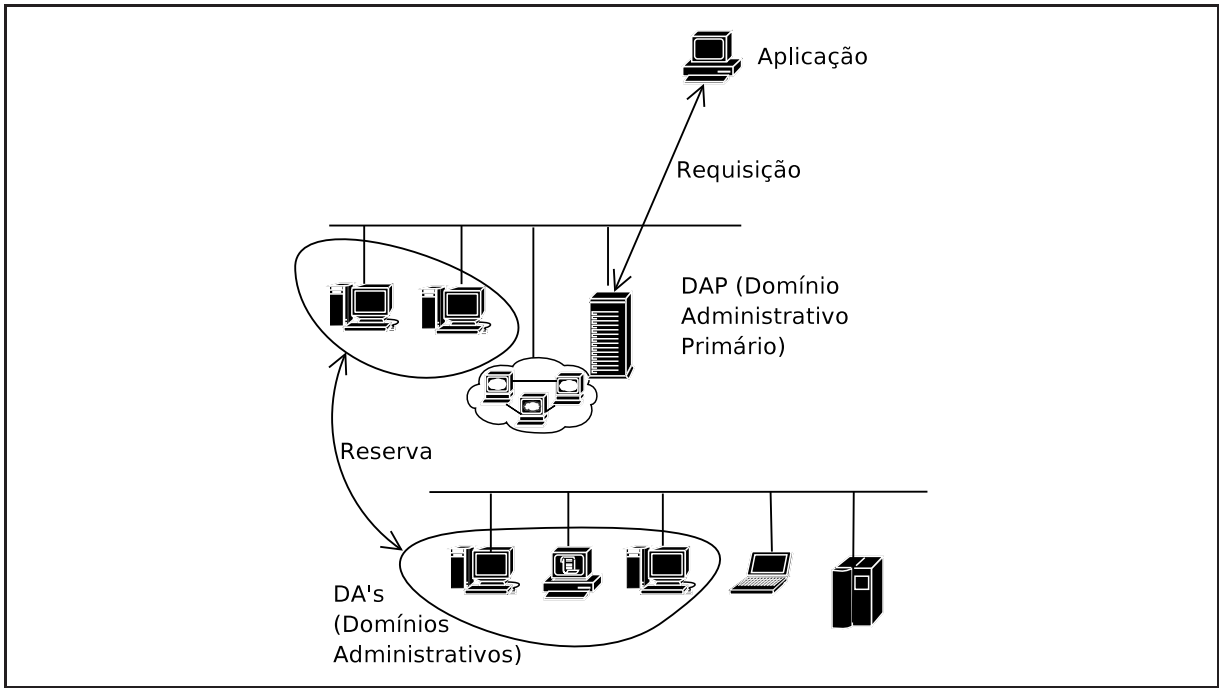

Figura 3.1: Visão da Grade Computacional

A visão da grade computacional referenciada neste trabalho está apresentada na figura 3.1. Nesta existem DAs geograficamente distribuídos, que podem ser representados por qualquer entidade organizacional, como por exemplo, organizações financeiras, industriais, de ensino entre outras, todos interconectados pela internet e associados ao sistema da grade computacional.

O usuário, ao tentar usufruir da grade, deve executar um processo que conecta-se a um DA e requerer recursos para executar sua tarefa. Estes requerimentos possuem informações sobre detalhes específicos de recursos. Todo DA que receber conexões de usuários torna-se uma entidade conhecida como DAP (Domínio Administrativo Primário), e este se encarregará de buscar os recursos específicos para seu usuário.

Com a descrição dos parágrafos anteriores, e os estudos realizados, desenvolveram-se algumas idéias que resultaram no protótipo. A seguir, são apresentados os conceitos incorporados pelo sistema.

3.1 Idéia Geral do MatchMaker

A grade é formada por entidades totalmente heterogêneas, tanto em nível de arquitetura quanto em nível de sistemas, sendo estes fracamente acoplados e geograficamente distribuídos. Para um sistema de alocação de recursos em grades computacionais, faz-se necessário utilizar de tecnologias focadas à dinamicidade do ambiente. Um sistema de gerenciamento de recursos convencional, centralizado, não se adapta bem em sistemas distribuídos (RAMAN; LIVNY; SOLOMON, 1998), como por exemplo a internet. O protótipo, desenvolvido neste trabalho, surge da constante necessidade de pesquisas nas áreas de tecnologias para ambientes de grades computacionais.

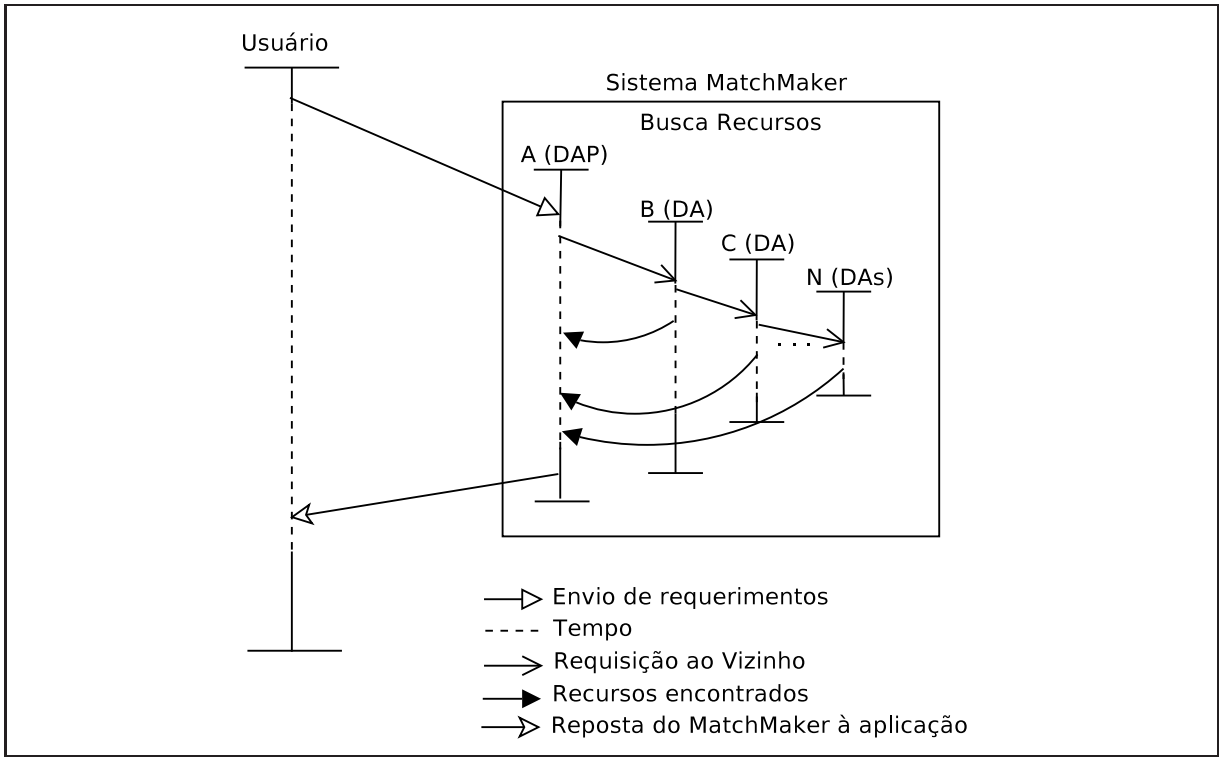

Figura 3.2: MatchMaker - Busca por Recursos

Uma visão de alto nível é demonstrada na figura 3.2. Cada letra representa um domínio administrativo, e pode, por exemplo, ser representado por: inf.ufsm.br, ppgep.ufsm.br, ppee.ufsm.br, inf.ufrgs.br.

A seguir são descritos os processos envolvidos na utilização do MatchMaker, representados na figura 3.2.

- **Usuário** - Grades computacionais normalmente atendem usuários que necessitam executar aplicações com grandes necessidades, seja computacional ou não. Para o MatchMaker, as tarefas dos usuários devem possuir requerimentos. Estes servirão para definir quais recursos poderão auxiliar no processamento da tarefa, como por exemplo, um requerimento pode definir a necessidade de processadores com capacidade iguais ou superiores a 3400 bogomips.

  Quando um usuário conecta a um DA, ele **não** envia a tarefa a ser executada, mas os requerimentos necessários para isso. Após receber o requerimento do usuário, o DA torna-se

um DAP que representará o usuário perante a grade. Um DA pode atender várias solicitações de usuários ao mesmo tempo. Todos os requerimentos contém em seus atributos a URL do seu respectivo DAP. Este endereço é a porta de entrada para o sistema da grade.

- **A** - Na figura 3.1, **A** está representando o processo inicial das operações de alocação de recursos dentro do MatchMaker, a partir de uma solicitação de usuário. Ele torna-se, para todos os membros da grade, um DAP. O processo de iteração entre o usuário e vários DAs podem ocorrer em paralelo. Desta forma, um requerimento pode ser enviado a vários DAs simultaneamente. Todo DAP mantém acesso direto às aplicações dos usuários, possibilitando uma interação de tarefas com recursos.

  No momento em que o DAP recebe requerimentos, uma busca por recursos locais é iniciada. Esta busca local consiste em localizar dentro do seu próprio domínio, entre os recursos disponíveis, os que satisfazem os requerimentos da aplicação. Para isso, cada recurso ocioso deve conter uma representação formal de seus atributos.

  Através de uma busca interna, os recursos encontrados podem ser rapidamente analisados, pois a comunicação se realiza dentro da LAN, sendo desnecessária a comunicação através de WANs.

  A após encontrar um conjunto de recursos para a aplicação deve realizar uma análise para determinar a qualidade destes, perante as necessidades da aplicação. Caso não haja recursos suficientes para que a tarefa seja executada, o DAP envia para os DAs vizinhos, diretamente conectados a ele, uma solicitação de recursos. O pedido contém os mesmos requerimentos que a aplicação enviara ao DAP, que por sua vez aguarda o resultado da busca dos seus vizinhos.

  Como a análise dos recursos é realizada individualmente, recurso por recurso, o desempenho desta atividade está diretamente associada ao desempenho geral do sistema. Sendo assim, os DAs não podem demorar muito para realizar as buscas por recursos dentro de seus próprios domínios, pois o DAP possui um tempo máximo de espera por este processo. O tempo é definido pelo usuário do sistema, que pode determinar alguns segundos, ou até mesmo algumas horas. Quando esse tempo expira, o DAP realiza uma análise sobre todos os recursos até então encontrados. Essa análise determinará se os recursos reservados para o usuário irão satisfazer seus requerimentos ou não. Depois da análise, o DAP comunica-se com a aplicação do usuário, informando o índice de qualidade obtida pelo processo de busca de recursos.

- **B** - As representações **B**, **C** e **N** são consideradas DAs que estão conectadas ao sistema de grade.

  Quando **B**, **C** ou os **N** recebem pedidos de recursos enviados pelo DAP, inicia os serviços de localização e filtragem dos recursos, localmente, dentro de cada DA. Este serviço tem a denominação, neste trabalho, de serviço de pré-casamento. O pré-casamento determina quais os recursos são capazes de satisfazer, ao menos, uma parcela dos requerimentos. Nesta etapa, não é realizada nenhuma análise de qualidade, apenas são repassados ao DAP as propriedades do recurso pré-casado.

  Neste trabalho os termos **casados** e **pré-casados** são processos distintos, sendo representado pelas figuras 3.3 e 3.4.

  Na arquitetura proposta, um conjunto de recursos que passou pelo serviço de pré-casamento somente será analisado pelo DAP. Ele determinará o índice de satisfação do conjunto, através de uma análise que utiliza propriedades dos recursos associadas com as informações

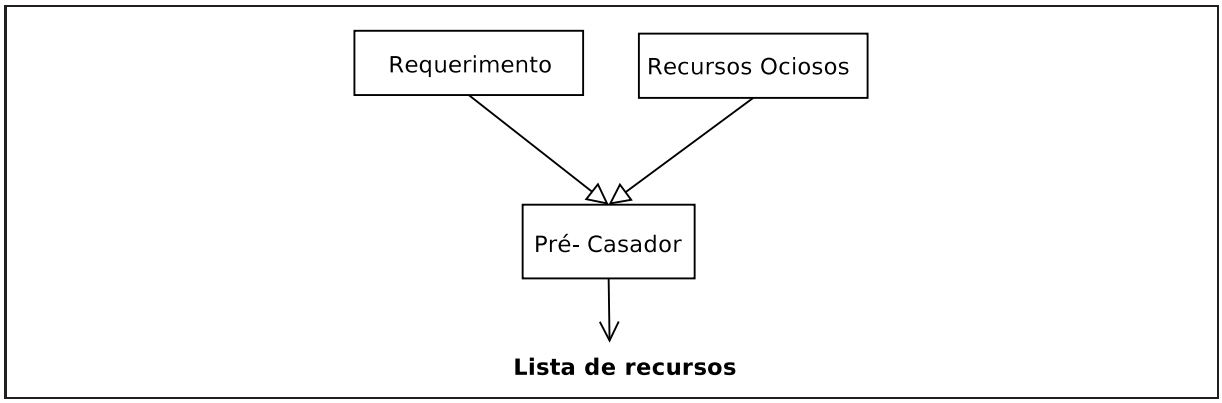

Figura 3.3: Serviço de Pré-casamento Realizado por Todos os DAs

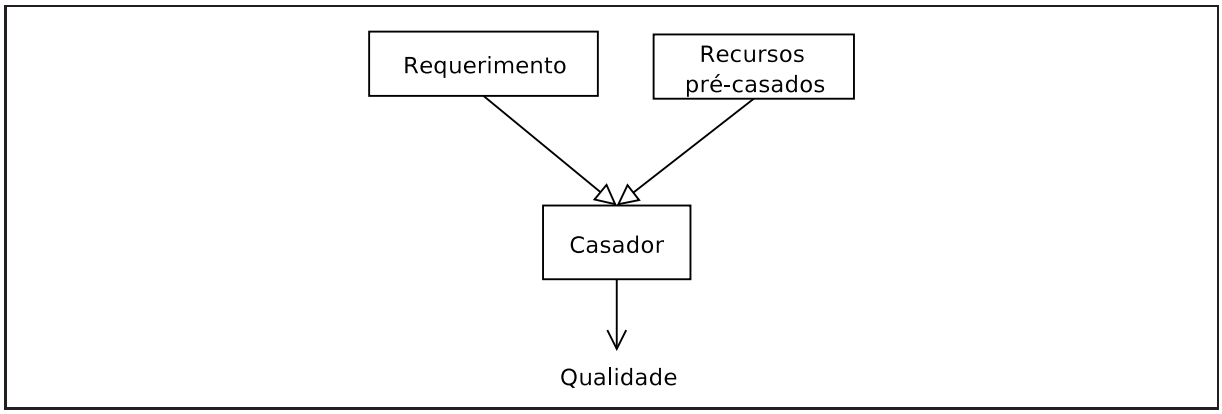

Figura 3.4: Serviço de Casamento realizado pelo DAP

do requerimento do usuário. Isto deve-se ao fato de que apenas o DAP possui informações sobre todos os recursos encontrados.

Todos os recursos pré-casados são primeiramente reservados em seus domínios locais. Após isso, seus atributos são enviados diretamente ao DAP. Através deste processo usufrui-se de:

- **comunicação direta com DAP** - Todos os DAs pertencentes à grade de computadores poderão comunicar-se diretamente com seus respectivo DAP. Para isso, deverá conhecer a URL do destino. Esta URL é obtida quando uma solicitação por recursos é recepcionada por algum DA. Como o sistema é totalmente distribuído, qualquer DA pode tornar-se um DAP, bastando para isso que um requerimento seja realizado por um usuário. Sem a comunicação direta, os DAs deveriam enviar o resultado do serviço de pré-casamento a quem lhe enviou o pedido de recursos. Desta forma, o envio do resultado seria realizado recursivamente até chegar ao DAP, gerando uma sobrecarga desnecessária, devido à hierarquia de distribuição dos DAs.
- **relação de bons vizinhos** - Com comunicação direta, o DAP poderá manter uma relação dinâmica de **bons** vizinhos. Esta relação conterá os DAs que mais contribuirem para o funcionamento geral do sistema. Na teoria, com o uso desta dinamicidade, pode-se atender requisições de usuários mais rapidamente.

- **C** - Idem item B.
- **N** - N representa uma quantidade finita de DAs pertencentes à grade.

3.2 Arquitetura do Sistema MatchMaker

A seguir são detalhados os aspectos relevantes da arquitetura do sistema, sendo dividida em Descrição de Recursos, Serviço de Localização, Casamento de Necessidades, Serviço de Reserva e Serviço de União de Recursos.

3.2.1 Descrição de Recursos

A capacidade de descrever recursos é de vital importância para o protótipo, pois para o sistema determinar quais recursos são suficientemente capazes de atender requisitos de usuários, precisa-se uma forma de expressá-los. Para Brooke (BROOKE et al., 2004), esta habilidade de descrição deve estar disponível aos próprios usuários que desejam utilizar sistemas de alocação de recursos. Esta importância ocorre devido à diversidade de serviços e recursos oferecidos por este novo tipo de rede. A falta de padronização das tarefas de busca e seleção de recursos torna sua utilização, por parte das organizações, complexa, exigindo conhecimento prévio do ambiente e dos requisitos necessários para acesso (PERNAS ANA M.; DANTAS, 2004).

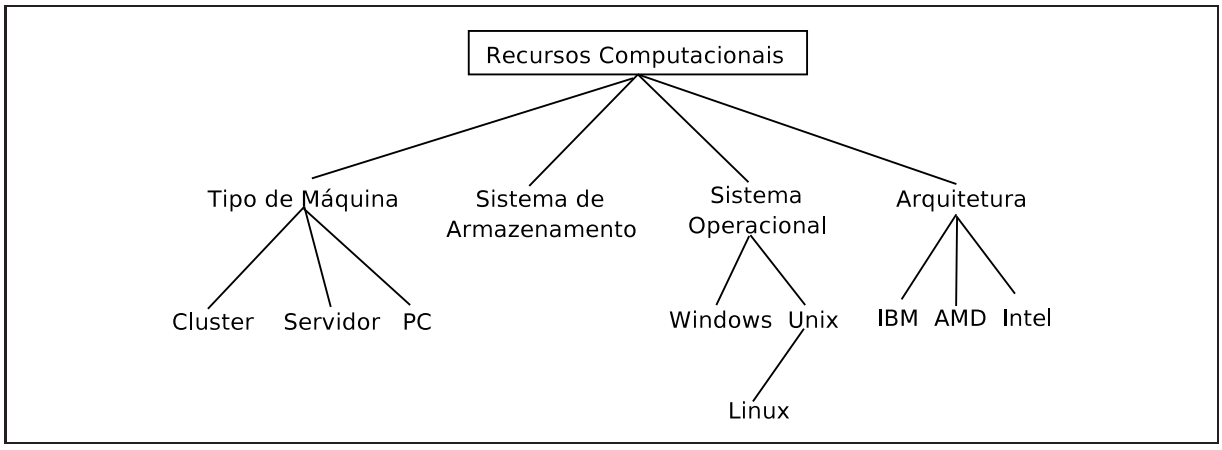

Figura 3.5: Modelo de Recursos em Grades Computacionais

Uma descrição de como os recursos podem ser classificados está representada na figura 3.5, sendo divididos em Tipo de Máquinaa, Sistema de Armazenamento, Sistema Operacional e Arquitetura. Para maiores informações sobre descrição de recursos consultar (ANDREOZZI et al., 2005).

3.2.2 Serviço de Localização

Devido à diversidade de recursos e serviços oferecidos pelas grades computacionais, vários autores (PERNAS ANA M.; DANTAS, 2004; POLLERES; TOMA; FENSEL, 2005; BAKER; SHADBOLT, 2003; TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) propõem o uso de semânticas para diminuir a dificuldade na busca e acesso a recursos computacionais por parte de usuários não familiarizados com as configurações dos domínios administrativos associados às grades.

Neste contexto, as maiores dificuldades estão no sentido de como poder expressar os requerimentos da aplicação. As requisições podem envolver diferentes tipos de recursos e possuir uma inter-dependência entre eles, tornando este processo complexo.

Para o sistema proposto, existem duas etapas bem definidas para o processo de localização dos recursos nos domínios:

- Localização Local - identificada quando um DA ou um DAP realiza a busca por recursos localmente.Esta busca ocorre sob os recursos previamente disponíveis, em estado de ociosidade.
- Localização Global - caracteriza-se pelo mecanismo de propagação dos requerimentos do usuário. Quando o DAP não encontra um número suficiente de recursos locais, capazes de satisfazer o usuário, ele realiza uma solicitação de recursos à todos os seus vizinhos, com o objetivo de encontrar mais recursos. Os DAs vizinhos também realizam solicitação de recursos com seus vizinhos.

O processo de localização local e global pode ser visualizada na figura 3.6.

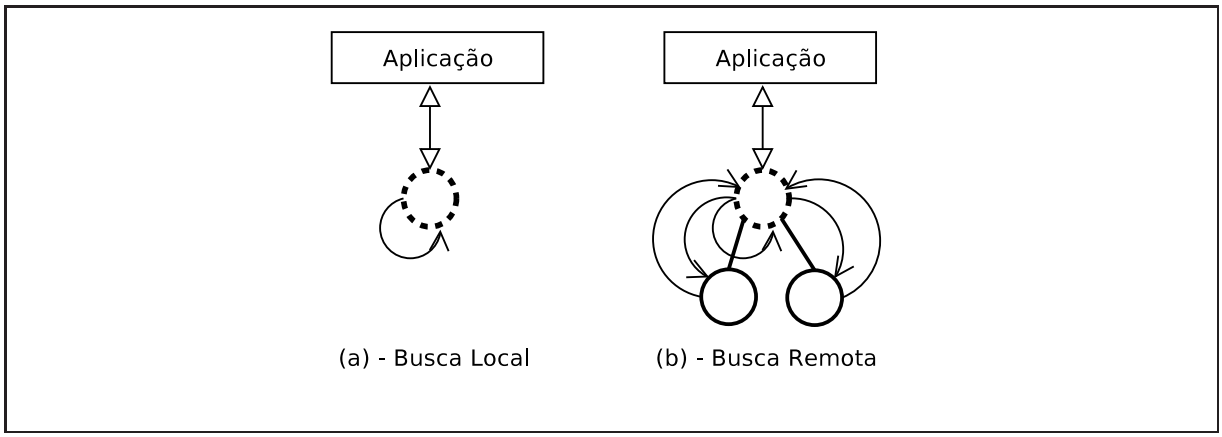

Figura 3.6: Serviço de localização de recursos

3.2.3 Serviço de Casamento

Não basta localizar recursos ociosos, deve-se selecioná-los de forma a obter uma otimalidade nas resoluções das tarefas dos usuários. Serviços de casamentos de recursos convencionais, como os aplicados no Condor (THAIN; TANNENBAUM; LINVY, 2003) e PBS (OpenPBS, 2006), são baseados em atributos simétricos. Nestes sistemas os valores dos atributos são limitados através de comparações dos requerimentos da aplicação, onde tanto o provedor do recurso como o usuário requisitante devem estar de acordo sobre os valores dos atributos. Isto torna o sistema inflexível e dificulta novas entradas de características.

Para o serviço de casamento, este trabalho propõe o uso de ontologia. Este permite uma descrição semântica de um vocabulário, que proporciona um entendimento amplo das características de recursos com necessidades. Além disso pode-se acrescentar o fato de serem extensíveis, pois novos vocabulários podem ser adicionados para descrever um novo domínio de aplicação.

3.2.3.1 Ontologia

Diversas áreas usam ontologias, buscando desenvolver um vocabulário contendo os conceitos relativos ao domínio de aplicação. Em computação, ontologia é citada como uma especifi-

cação formal e explícita de uma conceituação compartilhada, onde, formal refere-se ao fato da ontologia ser interpretável por máquina; conceituação refere-se ao modelo abstrato de algum fenômeno, o qual identifica conceitos relevantes do próprio fenômeno; e compartilhada reflete a noção de que uma ontologia captura o conhecimento apresentado não somente por um único indivíduo, mas por um grupo.

Harth (HARTH et al., 2004) apresenta uma descrição do uso de ontologias na construção de sistemas *MatchMakers*.

Tangmunarunkit (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) apresenta um sistema baseado em ontologias utilizando descrição assimétrica de recursos/requerimentos, propondo um projeto flexível e extensível, com uso de semânticas web, para resolver os problemas de casamentos.

Andreozzi (ANDREOZZI; MONTESI; MORETTI, 2005) apresenta uma linguagem para satisfazer necessidades através de seleção de recursos. Com esta linguagem, usuários de sistemas em grades podem criar requisitos e expressar qual o índice de satisfação desejado para resolver um determinado problema.

3.2.4 Serviço de Reserva

Com o objetivo de aproveitar recursos ociosos de diferentes DAs, o mecanismo de reserva do protótipo visa utilizar uma estrutura dinâmica, devido às possíveis alterações de estados dos recursos, que podem mudar rapidamente de um estado ocioso para ocupado. As decisões de reserva são realizadas localmente, dentro de cada DA, sem considerações ao objetivo global. Dentre os recursos que sofrem o serviço de pré-casamento, os que satisfazem os requisitos saem da lista de recursos disponíveis e são inclusos na lista de reservados.

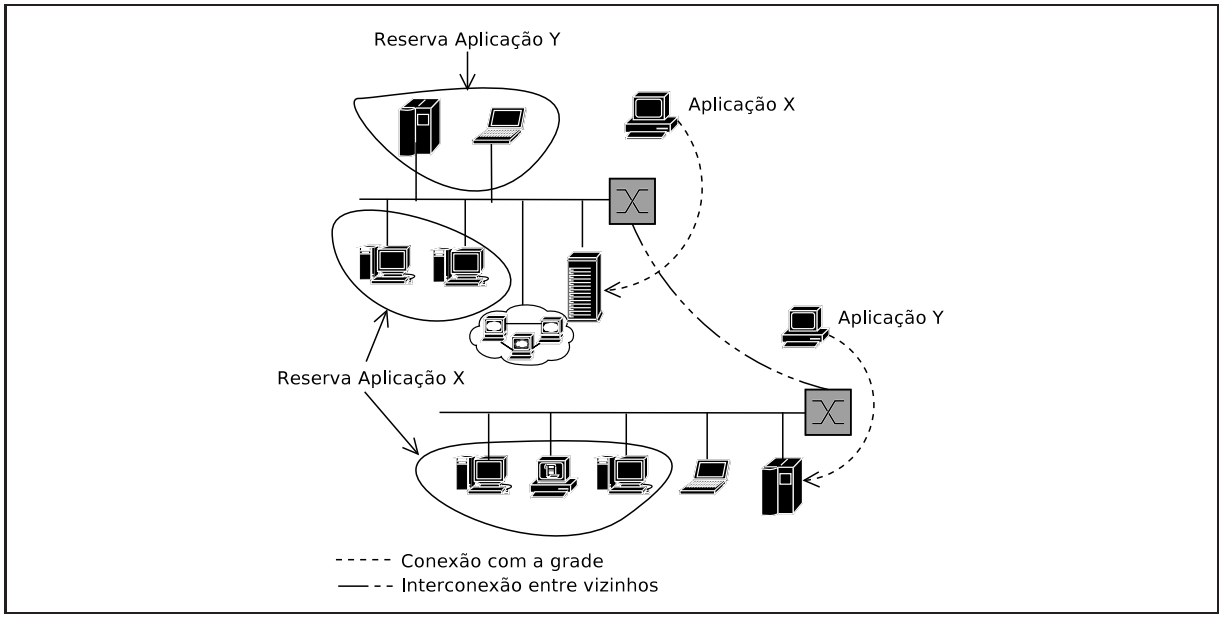

Figura 3.7: Reserva de Recursos

O protótipo controla o tempo de permanência dos recursos na lista de reservados. Quando este tempo vencer, os recursos são incluídos novamente na lista de recursos disponíveis. A partir deste instante, outros requerimentos de recursos poderão utilizá-los. Alguns dos membros da lista de recursos reservados poderão nunca serem ocupados, dentro do seu tempo de permanência. Isso deve-se ao fator que para serem utilizados, o DAP deve analisar quais dos recursos, até

então reservados, irão melhorar a qualidade do casamento. Os recursos não ocupados devem sair da lista de recursos reservados e adicionados aos disponíveis, fornecendo ao sistema:

- Disponibilidade - somente os recursos que serão utilizados permanecem na lista de reservados.
- Dinamicidade - controle do estado de ociosidade.

3.2.5 Serviço de União

O serviço de união procura atender necessidades específicas, cujos requerimentos necessitam a junção de alguns recursos para que seja possível satisfazer usuários da grade. Para isso, deve existir uma cooperação entre os recursos, possibilitando, através do serviço de união, facilidades como:

- Coleta de Informações
- Escalonamento
- Armazenamento de Aplicações em Grade

A união de recursos disponíveis na internet é o que motivou a desenvolvimento das grades computacionais. Segundo Ferreira (FERREIRA; SANTOS; SCHULZE, 2003), a computação em grade é aquela que se utiliza de aplicativos de gerenciamento que permitem compartilhar recursos que estão localizados em domínios distintos, baseados em tecnologias de arquitetura diferentes. Desta forma, têm-se obtido diversas conquistas na união destes recursos, de forma a permitir o seu uso. A formação de união de recursos tem sido amplamente estudadas em diferentes áreas (HE; IOERGER, 2005) (LI; SYCARA, 2002).

Um possível estado de união de recursos é demonstrado na figura 3.8, sendo estes posicionados em diferentes domínios.

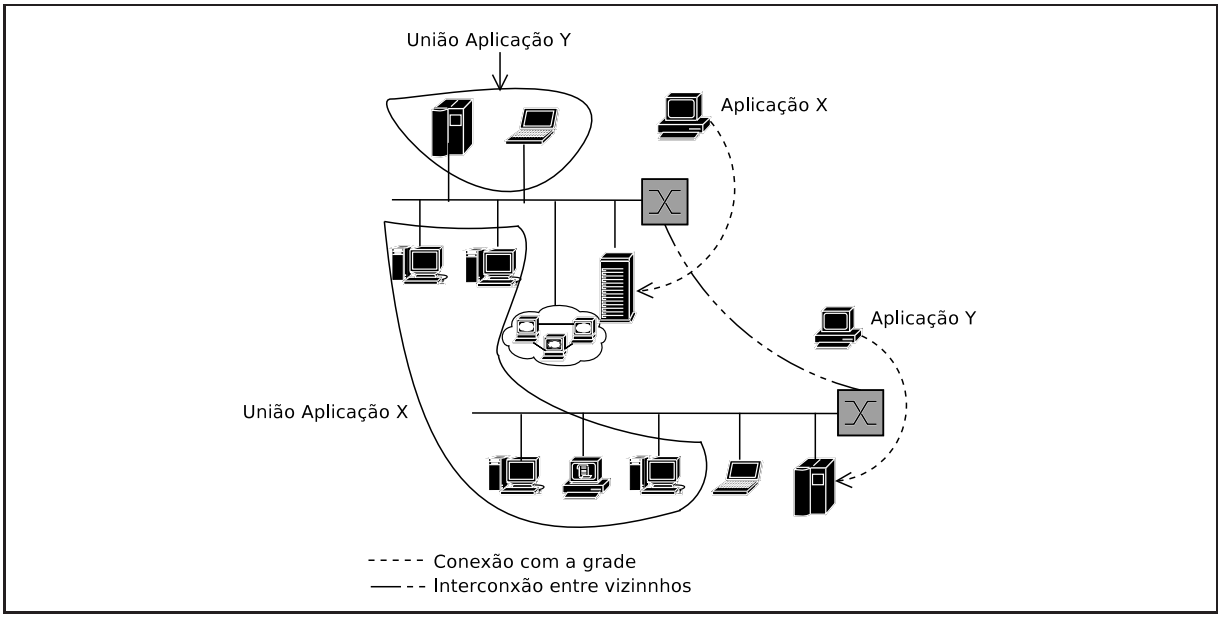

Figura 3.8: União de Recursos

3.3 Algoritmo Proposto

Os algoritmos do DAP e DA são apresentados nas figuras 3.9 e 3.10, representando o estado atual do protótipo. Algumas considerações citadas anteriormente são de propósito para o sistema, mas devido à complexibilidade e conseqüentemente ao tempo, não foram implementados no protótipo. Para melhor entendimento, segue uma descrição das funções do algoritmo desenvolvido.

```
1   ao receber requerimento aplicação
2     dispara temporizador
3     realiza pré-casamento local
4     avalia casamento
5
6     se satisfaz requerimento
7        responde aplicação
8     senão
9        difunde
10
11  ao vencer temporizador
12       responde aplicação
13
14  ao receber resposta
15    faz união dos pré-casados
16    avalia casamento
17
18    se satisfaz requerimento
19       responde aplicação
```

Figura 3.9: Algoritmo DAP

```
1  ao receber requerimento
2     se requerimento novo
3        faz pré-casamento local
4        responde ao DAP
5
6     senão
7        descarta requerimento
8
9     se temporizador é valido
10       difunde
```

Figura 3.10: Algoritmo DA

No algoritmo da figura 3.9 o DAP, ao receber um requerimento, dispara um temporizador que determina o limite máximo de tempo, fornecido com o requerimento, para que os processos

de busca por recursos sejam realizados. Executa também, linha 3, uma busca por recursos dentro do seu domínio através do serviço de pré-casamento, com a finalidade de analisar todos os recursos individualmente, que estejam disponíveis, e reservá-los para a aplicação. O próximo passo do algoritmo é realizar o serviço de casamento, que consiste em fazer união de todos os recursos pré-casados e analisar o grau de qualidade retornado pelo casamento. Desta forma, pode-se determinar se as necessidades da aplicação foram satisfeitas ou não, avisando-a em caso positivo, linha 7. Se as necessidades não foram satisfeitas, o DAP difunde o requerimento à todos os DAs conhecidos, linha 9.

Ao vencer o temporizador, linha 11 da figura 3.9, o DAP informa à aplicação o grau de qualidade atingido no serviço de casamento.

O DAP ao receber respota de um DA, linha 14 da figura 3.9, realiza união dos recursos pré-casados e avalia o casamento, respondendo à aplicação caso suas necessidades forem satisfeitas.

No algoritmo da figura 3.10 o DA recebe um requerimento de um DAP ou de um outro DA. Todo requerimento é analisado, linha 2, para descobrir se é um requerimento novo. Caso este requerimento já tenha sido recebido anteriormente, ele é descartado, linha 7. Não sendo descartado, o DA realiza o serviço de pré-casamento dentro do seu domínio e envia o resultado diretamente ao DAP, linha 4.

Nas linhas 9 e 10 da figura 3.10, o DA verifica se o temporizador ainda é válido e em caso positivo, difunde o requerimento a todos os DAs conhecidos.

4 DESENVOLVIMENTO DO MATCHMAKER

Este capítulo apresenta o modo de operação e o desenvolvimento do MatchMaker. Suas classes e seus métodos foram baseados na arquitetura apresentada no capítulo 3.

4.1 MatchMaker

Visando o aproveitamento de recursos ociosos geograficamente distribuídos através das grades de computadores, desenvolveu-se uma arquitetura baseada na utilização do *framework* ProActive. Através de seus benefícios pode-se, diferentemente do RMI tradicional, portar códigos seqüenciais para uma arquitetura paralela com facilidades. Com isso, é possível tornar transparente o acesso a objetos distribuídos ou aos mapeamentos das atividades concorrentes (MATHIAS et al., 2004).

O MatchMaker provê uma arquitetura de gerenciamento de recursos totalmente transparetente a desenvolvedores de aplicações para grades. Para Baude (BAUDE; CAROMEL; MOREL, 2003) existem três grupos de programadores das grades computacionais:

- Primeiro Grupo - São os usuários que utilizam aplicações através de interfaces gráficas ou com auxílio da Web.
- Segundo Grupo - São os programadores que possuem o conhecimento de como construir aplicações para grades computacionais, utilizando componentes já existentes.
- Terceiro Grupo - São os pesquisadores que constroem novos componentes individuais.

Neste trabalho, utilizou-se a idéia de aproveitar componentes já existentes, *framework* ProActive, para construir novos componentes, que dão suporte à localização, reserva, união, casamento e descrição de recursos. A seguir, são apresentadas o modo de utilização do MatchMaker.

4.2 Funcionamento MatchMaker

O MatchMaker foi projetado para trabalhar juntamente com o sistema Cadeo, agragando capacidades operacionais para suporte as grades de computadores. Com isso o sistema Cadeo ganha novos serviço, como a capacidade de selecionar recursos em domínios remotos através de necessidades do usuário.

O Cadeo possui um módulo central, conhecido como **Alocador**, que é responsável por controlar recursos no seu domínio. O gerenciamento de recursos locais do sistema Cadeo é representado na figura 4.1. No contexto do sistema Cadeo, um recurso é denominado um trabalhador.

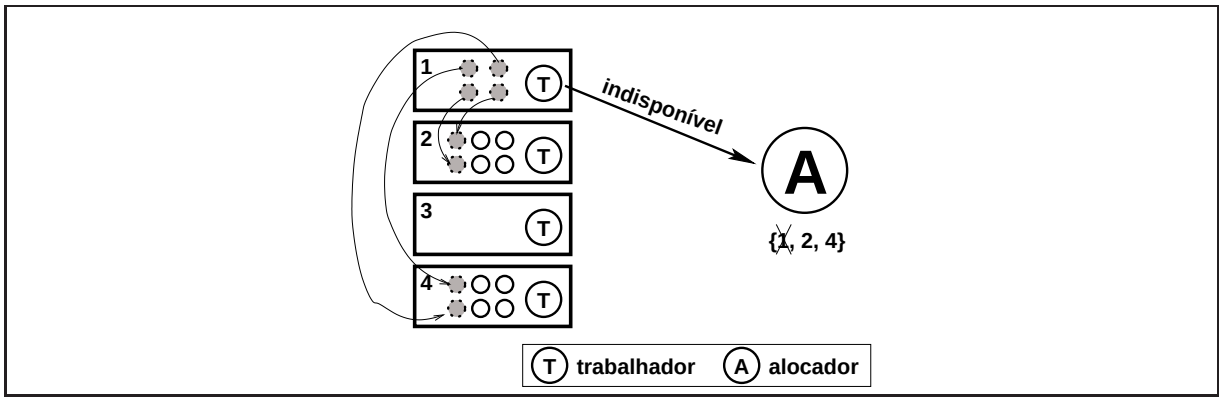

Figura 4.1: Gerenciamento de recursos do sistema Cadeo

O MatchMaker permite fazer a interação entre domínios, possibilitando a criação de aglomerados dinâmicos através da grade. Um conjunto de classes foi desenvolvido visando a simplicidade de utilização. Para o usuário do sistema Cadeo utilizar o MatchMaker é necessário extender a principal classe do MatchMaker, também denominada **MatchMaker**. A integração entre os dois sistemas é descrita na figura 4.2, onde o sistema Cadeo fornece ao MatchMaker uma lista de recursos ociosos e recebe um aglomerado dinâmico.

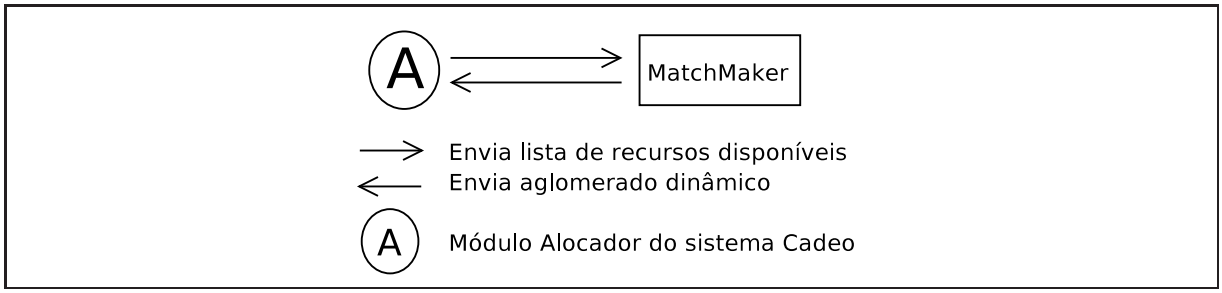

Figura 4.2: Integração do MatchMaker com o sistema Cadeo

A seguir, são apresentadas as classes implementadas para a criação do protótipo MatchMaker.

4.3 Classes Implementadas

Todas as classes implementadas possuem suas próprias funções, características bem definidas. Estas funcionalidades dividem-se em:

- Interface com usuário
- Localização de recursos
- Distribuição de requerimentos
- Descrição de recursos
- Organização virtual

Dentre as classes desenvolvidas destacam-se: MatchMaker, SearchResource, Traveller, ResourceMatch, Resource, IDApplication.

4.3.1 MatchMaker

A classe **MatchMaker** é a principal classe dentro do sistema, ela realiza a interação entre usuários e a grade. A seguir são descritos os métodos desenvolvidos.

- boolean searchMatch(IDApplication id)

  O método **searchMatch** permite a conexão do usuário com a grade. Nesta conexão, o método recebe requerimentos através de um objeto do tipo *IDApplication*, detalhado na sessão 4.2.5. Os requerimentos contêm informações sobre quais as necessidades básicas para que as tarefas do usuário possam ser executadas . Este método retorna ao usuário o resultado obtido no processo de busca por recursos. Caso o MatchMaker encontre recursos suficientes para satisfazerem suas necessidades, o retorno será *true*. Caso contrário, será *false*.

- boolean resourceLocal(IDApplication idLocal)

  Este método é responsável por realizar o processo de busca por recursos localmente, dentro do próprio domínio administrativo. Sempre que uma requisição por recurso for enviada a um DA, o método **resourceLocal** recebe um objeto *IDApplication*. Com posse deste, o DA executa o processo de busca por recursos sabendo por quais procurar.

- searchRemote(Traveller traveller)

  Toda a vez que um DAP não encontra um conjunto suficiente de recursos para satisfazer seu usuário, um pedido por recursos a DAs vizinhos terá que ser feito. O método **searchRemote** tem a função de receber requerimentos provenientes do DAP e de DAs remotos, os quais enviam um objeto do tipo *Traveller*, melhor explicado no item 4.2.6. Após o recebimento do objeto, os DAs realizam o processo de busca local e o serviço de pré-casamento, dentro de seu domínio, e enviam diretamente ao DAP o próprio objeto *Traveller* com todos os recursos encontrados. Envia também, o mesmo requerimento que recebeu, a todos os seus vizinhos.

- resourceRemote(Traveller traveller)

  O método **searchRemote** envia ao DAP o resultado obtido do serviço de pré-casamento invocando método remoto **resourceRemote**.

  O método **resourceRemote** tem a função de, além de recepcionar resultados de buscas locais dos DAs, invocar o serviço de casamento. Toda vez que o DAP recebe objetos *Traveller*, significa que há novos recursos disponíveis para serem utilizados. Cabe ao algoritmo de casamento decidir quais dos novos recursos serão aproveitados pelo sistema.

- float getQualityObtain()

  A qualquer momento o usuário pode verificar como está o processo de busca por recursos. O método **getQualityObtain** retorna o índice de qualidade obtida até o presente momento.

4.3.2 SearchResource

Os processos de busca por recursos são implementados na classe **SearchResource**. Métodos da classe **MatchMaker** acessam métodos da **SearchResource** para que sejam realizadas buscas por recursos, tanto localmente quanto remotamente.

- IDApplication search(IDApplication id)

  O método **search** é responsável por colocar em operação um novo requerimento de usuário. Ele destina-se a realizar o serviço de pré-casamento. Um objeto do tipo *IDApplication* é recebido e fornecerá as informações dos requerimentos do usuário ao serviço de pré-casamento. Quando o método **search** é invocado por um DAP, algumas informações são adicionadas ao objeto *IDApplication*. O objetivo é possilitar que o MatchMaker tenha controle do objeto, através dos seguintes atributos:

  – Hora Inicial - determina o momento exato que a busca por recursos começou a ser realizada na grade.

  – Validade do Objeto - determina a validade de um objeto *IDApplication*, permitindo que a busca por recursos possa ser realizada.

  Toda solicitação de recursos, enviadas a um DAP, significa um novo pedido por parte de usuários.

- boolean findLocal(ApplicationRequeriment ar)

  O método **findLocal** tem acesso a todos os recursos ociosos do domínio. A descrição de cada recurso ocioso é encaminhada, juntamente com os requerimentos do usuário, ao método **matchedResource(rs, ar)** da classe **ResourceMatch**.

- findRemote(Traveller traveller, MatchMaker[] addr)

  O método **findRemote** contêm as operações necessárias para buscar recursos remotos, em DAs vizinhos. Para isso, ele possui todas as URLs dos DAs diretamente conectados, possibilitando tanto ao DAP, quanto aos DAs envolvidos no processo de localização e seleção de recursos, encaminhar pedidos uns aos outros.

4.3.3 ResourceMatch

A classe **ResourceMatch** tem a responsabilidade de prover gerenciamento dos recursos, utilizando vários métodos para isso. A seguir são descritos suas funções.

Devido a dinamicidade com que os recursos podem aparecer ou desaparecer, ociosos ou não, esta classe é definida como uma das mais importantes para o protótipo. Todos os métodos são implementados para que os recursos encontrados pelos DAs e DAPs possam satisfazer as necessidades das tarefas do usuário.

- boolean matched(IDApplication id,IDApplication idLocal)

  O método **matched** tem a função de implementar o algoritmo de casamento. Nele são definidos procedimentos como:

  – Fazer união de recursos;

  – Analisar se a qualidade obtida pela união supre os requerimentos do usuário.

- boolean matchedResource(Resource[] rs, ApplicationRequeriment ar)

  Este método implementa o algoritmo de pré-casamento. Para isso todos os recursos, de maneira individual, devem ser analisados. Com essa verificação, é possível determinar quais deles estão aptos a participarem da execução das tarefas.

  O método **matchedResource** utiliza dois parametros:

  – **Resource[] rs** - através de um objeto **Resource** são capturadas todas as informações, dos recursos ociosos, necessárias para o serviço de pré-casamento.

  – **ApplicationRequeriment ar** - com o objeto **ApplicationRequeriment** são obtidas as informações relevantes sobre o requerimento do usuário.

  Com posse destes dados, o algoritmo se encarrega de determinar se um recurso é suficientemente capaz de atender, ao menos em parte, as necessidades do usuário.

- float matchQuality(IDApplication id)

  O MatchMaker precisa uma maneira de determinar se a soma dos recursos, até então encontados, satisfazem ou não os requerimentos do usuário. O método **matchQuality** tem como objetivo, analisar todos os recursos pré-casados. Esta análise determina um índice de qualidade dos recursos encontrados, possibilitando ao sistema, saber a qualquer momento se foi satisfeita as necessidades do usuário ou não.

- boolean union(Vector r2)

  Dependendo das necessidades das tarefas do usuário, alguns recursos isoladamente não poderão satisfazê-los. Neste caso, alguns recursos devem ser agrupados de modo a aumentar sua característica de computação. O método *union* implementa o algoritmo de agrupamento de recursos. Pode-se citar como exemplo a união de dois ou mais recursos, para juntos oferecerem uma maior capacidade de armazenamento ou ainda unir dois enlaces de redes para aumentar a vazão das informações.

- void addResource(Resource rs)

  O método **addResource** responsabiliza-se por adicionar recursos pré-casados em vetores dinâmicos. Cada usuário do sistema possui o seu próprio vetor, que contêm todos os recursos para ele reservados. Este método adiciona recursos individuais, mas para adição de um conjunto de recursos há nesta classe uma sopreposição de método que implementa o método **void addResource[] rs**.

- void removeResource(Resource r)

  Como os recursos podem a qualquer momento mudar de estado, o sistema deve dinamicamente adicionar ou remover recursos. O método **removeResource** tem a função de remover recursos que sofreram alteração do estado de ociosidade para ocupados. O método **removeResource** também pode receber um conjunto de recursos a serem removidos. Para isso, o conceito de sopreposição de métodos é utilizado.

- boolean validTimer(IDApplication id)

  Este método é invocado toda vez que um objeto **IDApplication** é recebido por um DA. Ele analisa se o tempo que o sistema tem para encontrar recursos se esgotou ou não. Caso o tempo tenha acabado, o DA simplesmeste descarta o objeto.

- void addRoute(String nodeURL)

  Para o funcionamento geral do sistema, o MatchMaker deve conhecer a URL de todos os recursos pré-casados. Como exemplo de uso, pode-se citar a necessidade do algoritmo de escalonamento em conhecer todas as *URL*s dos recursos disponíveis.

4.3.4 Resource

Muitos tipos de recursos podem ser utilizados nas grades computacionais, a classe **Resource** tem por finalidade permitir a adoção de vários algoritmos de descrição de recursos.

Neste trabalho os recursos foram classificados e divididos nos seguintes métodos:

- ResourceAttribute[]

  Os recursos são representados por sua descrição. A classe **ResourceAttibute** permite que especificações de diferentes recursos sejam utilizadas através de objetos. Usuários podem expressar suas necessidades de várias maneiras. Isso não afeta as definições dos tipos de atributos, pois os requerimentos e os atributos não precisam ser iguais, ficando sob responsabilidade do algorítmo de casamento esta tarefa. Vários serviços podem querer consultar informações sobre objetos do tipo *Resource*, como por exemplo: (1) serviço de casamento e pré-casamento e (2) serviços de *status* da grade.

- ResourceCluster[]

  Este método é invocado toda vez que recursos do tipo *Cluster* são utilizados pelo serviço de pré-casamento.

- ResourceNode[]

  Todos os recursos do tipo *Node* são armazenados em um objeto da classe **Resource**, que juntamente com atributos fornecem informações para o serviço de pré-casamento.

4.3.5 IDApplication

Toda vez que um usuário queira utilizar a grade para executar tarefas, ele deve criar um objeto do tipo *IDApplication* e seus requerimentos devem ser adicionados neste objeto. No momento em que o usuário se conecta em um DA, ele envia este objeto com informações que irão auxiliar o Matchmaker a buscar os melhores recursos para suas tarefas.

Esta classe tem como principal objetivo oferecer, tanto ao usuário quanto ao MatchMaker, um ponto central de referência. Todos os recursos pré-casados são adicionados em um objeto deste tipo. Quando o MatchMaker executa o serviço de casamento, obtendo assim a qualidade dos recursos encontrados, irão permanecer no objeto apenas os recursos que possam ser agrupados com outros, afim de melhorar a qualidade do casamento.

Cada usuário possui um objeto *IDApplication* único, que contêm as necessidades das tarefas e as propriedades de todos os recursos casados. Para isso, alguns métodos são utilizados, conforme descrição a seguir.

- IDApplication(String url, ApplicationRequeriment ar, float qualityNeed)

  O método construtor **IDApplication** tem a função de criar um objeto, com informações definidas durante a inicialização, que é encaminhado a um DA na rede. Alguns dados só são definidos quando o objeto chegar ao MatchMaker. Os três parâmetros deste método são explanados a seguir:

  - o primeiro parâmetro, *String url*, define a URL do usuário. Toda vez que o DAP se comunica com o usuário, o atributo URL é utilizado. Além disso, qualquer DA poderá acessar o usuário quando o seu DAP venha perder conexão com a grade, melhor discutido em "Eleição de um novo DAP", ítem 6.1.6.
  - no segundo parâmetro um objeto do tipo *ApplicationRequeriment* é parte integrante de *IDApplication*.O objeto *ApplicationRequeriment* contêm todos os requerimentos do usuário.
  - o terceiro parâmetro, *float qualityNeed*, define o índice de qualidade requerida pelo usuário. Um índice de qualidade muito alta exigirá muito trabalho para o MatchMaker, pois ele terá que se comunicar com mais vizinhos em busca de mais ou melhores recursos. O serviço de casamento precisa acessar esta informação para poder classificar os recursos pré-casados, satisfatórios ou não.

- void setTimerLife(long timerLife)

  O método **setTimerLife** permite que o usuário defina o tempo máximo que o MatchMaker tem para desempenhar suas funções de localização e alocação de recursos.

- void setTimerInitial(long time)

  Para que um usuário envie requisições ao MatchMaker, ele deve enviar um objeto *IDApplication*. No momento em que o sistema recebe o objeto de um usuário, é configurado, através do método **setTimerInitial**, o horário inicial do processo de busca por recursos. Este procedimento permite a qualquer DA, em qualquer momento, verificar se o objeto ainda é valido para que se possa continuar a busca pelos recursos locais.

- void setQuality(float quality)

  Este método é utilizado apenas pelo MatchMaker. Toda vez que o sistema recebe o resultado da busca por recursos, de algum DA, é verificado o índice de qualidade obtida com a chegada dos novos recursos. Após cada processo de análise da qualidade, o método **setQuality** seta o atributo *qualityObtain* com o índice alcançado. Ao final do processo de busca por recursos, o usuário poderá consultar o atributo *qualityObtain* para saber qual o índice de qualidade obtida pelo MatchMaker.

- void setDomainAddr(String[] domain)

  Este método adiciona em um vetor dinâmico as URLs dos domínios administrativos que o objeto *Traveller* passou.

- void addResource(Resource n)

  Este método permite que todos os recursos pré-casados, localmente no DAP, sejam adicionados em um vetor dinâmico.

4.3.6 Traveller

Com o objetivo de criar um objeto que seja leve, pequeno o suficiente para trafegar pela grade e grande o bastante para conter todas as informações sobre os requerimentos de usuários, criou-se a classe **Traveller**. Um objeto deste tipo é criado toda vez que uma busca interna em um DAP não satisfazer os requerimentos do usuário. Neste caso, o sistema envia um pedido por recursos a seus vizinhos remotos através deste objeto.

- Traveller(String url, ApplicationRequeriment ar)

  Este é um método construtor que tem a finalidade de configurar o requerimento do usuários através de um objeto do tipo **ApplicationRequeriment**, juntamente com a URL do DAP. Todo DA ao receber um objeto do tipo **Traveller** envia um retorno para o DAP com o respectivo resultado da sua busca pelos recursos.

  O objeto Traveller contêm os seguintes atributos:

  - Requerimentos dos usuários - para executar o serviço de pré-casamento, todo DA necessita visualizar todos os dados de requerimento do usuário.
  - URL do DAP - com a URL do DAP, todos os DAs podem enviar seu objeto Traveller diretamente ao DAP.
  - Vetor de Recursos - todo recurso que satisfazer os requesitos do usuário, detectado através do serviço de pré-casamento, são adicionados em um vetor dinâmico.

- void setValid(boolean valid)

  No momento em que o tempo de vida do objeto *Traveller* acaba, deve-se configurá-lo como inválido, finalizando assim a busca por recursos. Neste momento, o objeto é encaminhado diretamente ao DAP.

- void addResource(Resource rs) -

  Durante o processo de busca por recursos, o objeto *Traveller* fica percorrendo diferentes DAs. Todas as descrições dos recursos pré-casados são adicionadas em um vetor dinâmico implementado pelo método **addResource**.

- void setDomainAddr(String domain)

  Este método adiciona em um vetor dinâmico as URLs dos domínios administrativos que o objeto *Traveller* passou.

5 ANÁLISE DOS RESULTADOS

Este capítulo visa demonstrar os resultados obtidos através de ambientes simulados da implementação. Em virtude do comportamento das grades, desenvolveu-se um simulador utilizando domínios administrativos hierárquicos. A próxima sessão descreve o processo de simulação utilizado.

5.1 Ambiente de Simulação

A simulação vem sendo utilizada nas mais diversas áreas, como por exemplo na simulação de sistemas urbanos, sistemas de controle aeroespaciais, sistemas de treinamento militar, sistemas ecológicos, sistemas de redes de comunicação de computadores e outros (BALIEIRO, M. O. S., 2005). O uso de simulação em sistemas para grades computacionais possibilita a realização de inúmeros testes e análises, sem a necessidade de possuir um grande número de recursos físicos. Assim como no protótipo, utilizou-se a linguagem Java, mas com auxílio da biblioteca SimJava.

5.1.1 SimJava

Simjava é um pacote de simulação de eventos discretos, totalmente escrito em Java, baseada na biblioteca SIM++ (CUBERT; FISHWICK, 2006) para linguagem C++. O SimJava consiste em uma coleção de objetos (Sim_entity), cada qual executando sua própria *thread* dentro da JVM.

Cada sistema simulado é considerado como um conjunto de processos que interagem através de entidades referenciadas pelo SimJava. Estas entidades comunicam-se uma com a outra passando eventos. As entidades da biblioteca são conectadas através de portas (Sim_port), que se comunicam enviando e recebendo eventos (Sim_event).

As relações criadas entre o DAP e os DAs são demonstradas através da figura 5.1, as quais utilizam **Sim_port's**. As linhas da figura são explicadas a seguir:

- **linha 1** - cria o DAP para a simulação, passando como referência um objeto *simulation* que contém todos os custos relativos ao funcionamento da simulação;
- **linha 2** - laço de repetição de 1 até o número total de vizinhos.
- **linha 3** - adiciona ao simulador os DAs especificados pelos números de vizinhos. Estes estarão diretamente conectados ao DAP;
- **linha 6** - adiciona ao simulador novos DAs que estarão conectados aos DAs diretamente conectados ao DAP;

- **linha 9** - realiza as conexões entre o DAP e DAs;
- **linha 13** - as conexões entre DAs vizinhos são definidas.

```
1  Sim_system.add(new DAP("DAP", 1, DAP.SRC_OK, simulation));

2  for(int i=1 ; i<=numNeighbour ; i++) {
3      Sim_system.add(new DA("Filho_"+1, 2, DA.SINK_OK));
4  }

5  for(int i=1 ; i<=(numNeighbour*numNeighbour) ; i++) {
6      Sim_system.add(new DA("Neto_"+1, 2, DA.SINK_OK));
7  }

8  for(int i=1 ; i<=numNeighbour ; i++) {
9      Sim_system.link_ports("DAP", "out"+i, "Filho_"+i, "in"+i)
10 }

11 for(int i=0 ; i<numNeighbour ; i++) {
12     for(int j=1 ; j<=numNeighbour ; j++) {
13         Sim_system.link_ports("Filho_"+(i+1), "out_"+j, "Neto_"+getnumber(), "in_"+1);
14     }
15 }
```

Figura 5.1: Código SIMJAVA para criação de relacionamento entre DAP e DAs

Cada objeto simulado possui sua própria *thread*, desta forma a performance da simulação está atrelada ao bom funcionamento dos controles das *threads* do sistema operacional. A figura 5.2 demonstra o processo de *linking* entre os DAs, realizado pelo simulador.

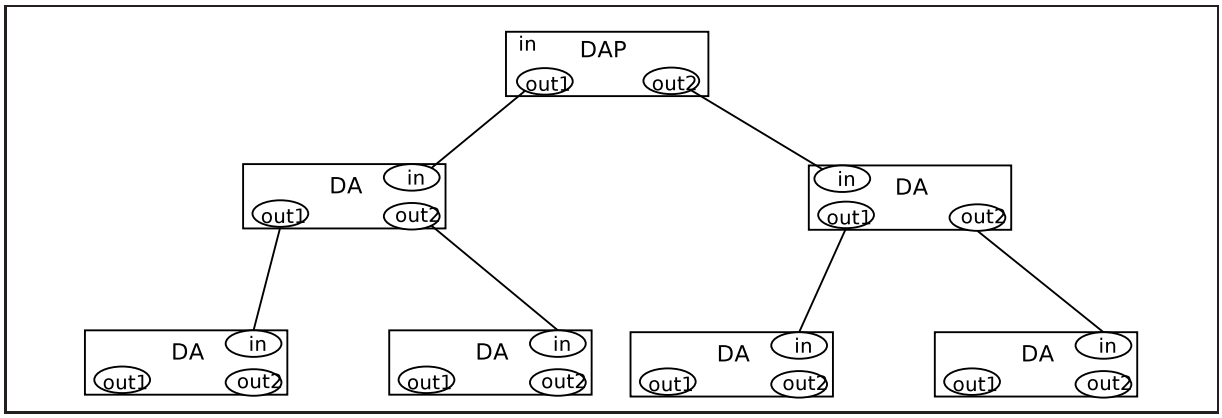

Figura 5.2: Estrutura criada pelo código da figura 5.1

5.2 Resultados

Esta seção apresenta alguns cenários de testes da simulação do protótipo. As informações apresentadas têm o intuito de demonstrar quais os maiores custos que efetam a performance do

sistema de alocação de recursos. A tabela 5.1 apresenta algumas características simuladas, estas devem ser consideradas fatores críticos, e desta forma, podem afetar o desempenho geral do sistema se seus custos forem altos.

Tabela 5.1: Descrição das Características de Custos

Característica	Descrição
Número de Recursos existentes	Descrevem quantos recursos farão parte de cada DA.
Envio do objeto a um DA	Custo referente ao envio do objeto *Traveller* para outros DAs ou para o DAP.
Análise dos recursos	Tempo necessário para execução do serviço de casamento.
Análise da qualidade	Custo relativo a análise da qualidade dos recursos encontrados.
Porcentagem de recursos pré-casados	Determina quantos recursos o serviço de pré-casamento consegue reservar.
Número de vizinhos	Determina quantos vizinhos o DAP possui.

As simulações realizadas são referentes ao sistema proposto e segue algumas características informativas, tabela 5.1, necessárias para ser possível analisar as mudanças de comportamento do ambiente simulado. Para melhor análise, as simulações foram divididas em etapas. A seguir são detalhadas as simulações realizadas.

Para tornar os testes realizados o mais realístico possível, utilizou-se custos aleatórios. Os custos de comunicação entre DAs foram escolhidos através de uma distribuição normal e distribuição uniforme. Segundo a *Internet Traffic Report* o custo médio de comunicação entre domínios na internet é de 200 milisegundos (Traffic, 2006). O número de recursos que cada DA contém também foi escolhido aleatóriamente, porém somente com uma distribuição uniforme. Uma descrição destes custos é explanada a baixo:

- Custos variados relativos à comunicação entre os DAs. Considera-se o custo de comunicação de um DAP para DA, DA para DA, e DA para DAP. Todas as comunicações podem variar em decorrer do tempo com média de 200 milisegundos. Toda comunicação envolve o envio de um objeto. O objeto pode conter requerimentos de usuários e ou informações sobre características de recursos reservados.

- Quantidade variável do número de recursos ociosos que cada DA contém. Estes podem variar através de uma distribuição uniforme entre 10 e 110.

5.2.1 Simulando Grade com 7 Domínios Administrativos

Na primeira simulação são utilizados dois vizinhos conectados ao DAP, sendo que cada um possui mais dois vizinhos, totalizando 7 domínios. A figura 5.3 contém a representação da grade simulada.

5.2.1.1 Descrição do Cenário

Na figura 5.3 pode-se visualizar o DAP com 48 recursos ociosos que serão analisados em busca dos que poderão satisfazer as necessidades da aplicação. Neste cenário, os recursos existentes no DAP não satisfazem o usuário. Sendo assim, o DAP envia aos seus DAs vizinhos um

requerimento de recursos. Os vizinhos do DAP contém 16 recursos disponíveis cada um, sendo que eles realizam o serviço de pré-casamento. Após isso, enviam ao DAP o resultado da busca pelos recursos e se possível enviam aos seus vizinhos o mesmo requerimento que recebeu do DAP. Os DAs identificados pelo número de seus recursos, 22, 99, 18 e 37 na figura 5.3 também realizam o serviço de pré-casamento. Após os DAs reservarem os recursos pré-casados, eles enviam ao DAP informações sobre o estado da busca por recursos. Toda mensagem recebida pelo DAP, originárias dos DAs, faz com que seja realizado o serviço de casamento, que determina se a soma de todos os recursos reservados é satisfatória ou não.

A tabela 5.2 demonstra os parâmetros utilizados.

Tabela 5.2: Custos Relativos à Primeira Simulação

Característica	Quantidade/Custo
Número de Recursos existentes	de 10 a 110
Envio do objeto para cada DA	média de 200 milisegundos
Análise por recurso	2
Análise da qualidade	2
Porcentagem de recursos pré-casados	10
Número de vizinhos	2

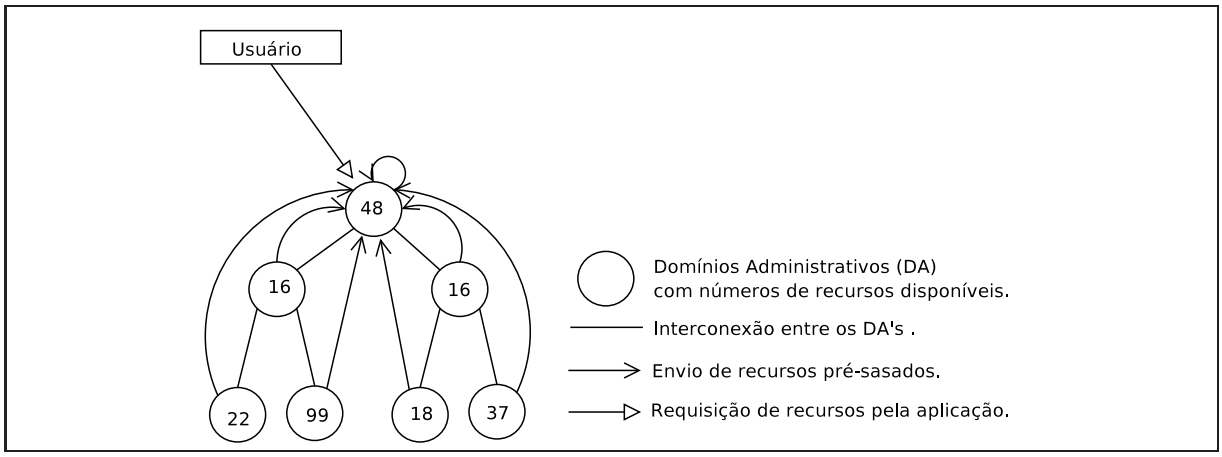

Figura 5.3: Simulando MatchMaker com 2 DA vizinhos, totalizando uma busca em 7 DAs

Tabela 5.3: Resultados Relativos à Primeira Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**787**	**870**
Total de recursos disponíveis	256	256
Recursos casados	21	21
Número de DA	7	7

A tabela 5.3 demonstra os resultados obtidos na primeira simulação. Nela podemos perceber a utilização de 7 DAs, contendo um total de 256 recursos osiosos. O tempo total de busca de recursos é de 787 milisegundos quando utilizado uma distribuição normal dos custos de

comunicação e de 870 milisegundos quando utilizado uma distribuição uniforme dos custos de comunicação.

A média de distribuição dos custos que envolvem a comunicação entre DAs é apresentadas no gráfico da figura 5.4.

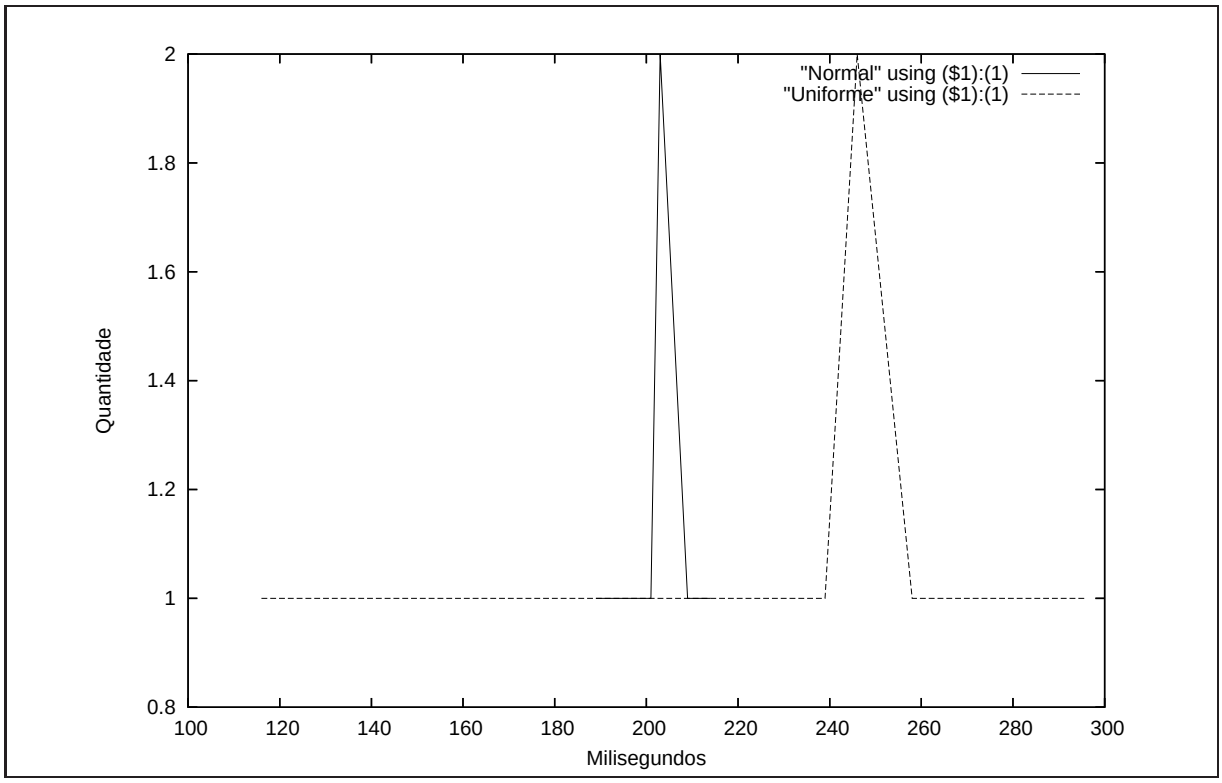

Figura 5.4: Distribuição dos custos de comunicação entre 7 DAs

5.2.2 Simulando Grade com 13 Domínios Administrativos

Na segunda simulação são utilizados três vizinhos conectados ao DAP, sendo que cada um possui mais três vizinhos, totalizando 13 domínios. Esta representação pode ser analizada como um estrutura hierárquica ou um gráfo qualquer. A figura 5.5 contém a representação da grade simulada.

Tabela 5.4: Custos Relativos à Segunda Simulação

Característica	Quantidade/Custo
Número de Recursos existentes	de 10 a 110
Envio do objeto para cada DA	média de 200 milisegundos
Análise por recurso	2
Análise da qualidade	2
Porcentagem de recursos pré-casados	10
Número de vizinhos	3

A seguir é descrito o cenário da segunda simulação.

5.2.2.1 Descrição do cenário

Neste cenário de simulação, o DAP após receber um requerimento do usuário, realiza o processo de alocação de recursos dentro do seu próprio domínio, contendo 18 recursos ociosos. Como os recursos pré-casados não são suficientes para satisfazer os requerimentos do usuário, o DAP encaminha um pedido a seus DAs vizinhos. Estes possuem respectivamente 95, 95 e 31 recursos ociosos. Os DAs após realizarem o pré-casamento local, enviam ao DAP um objeto com a descrição dos recursos reservados. Após os envios das mensagens para o DAP os DAs encaminham os mesmos requerimentos que receberam a outros DAs vizinhos se o temporizador não venceu.

A tabela 5.5 demonstra o tempo total consumido para realizar a alocação de recursos. As comunicações entre DAs totalizaram **24 mensagens**.

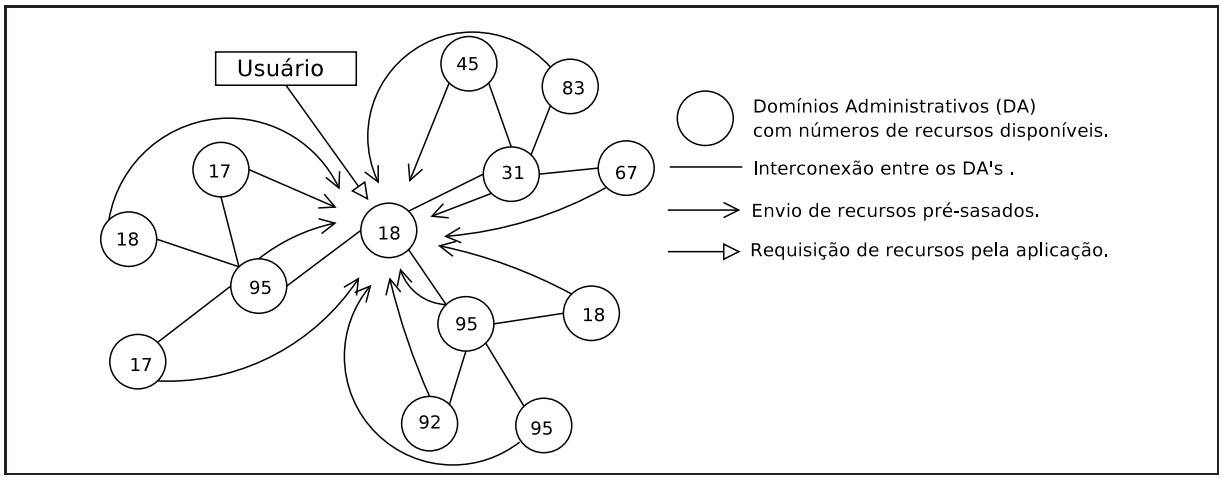

Figura 5.5: Simulando MatchMaker com 3 DAs vizinhos

Tabela 5.5: Resultados Relativos à Segunda Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**768**	**899**
Total de recursos disponíveis	691	691
Recursos casados	67	67
Número de DA	13	13

A média de distribuição dos custos que envolvem a comunicação entre DAs é apresentadas no gráfico da figura 5.6.

5.2.3 Simulando Grade com 21 Domínios Administrativos

Representar graficamente uma grade computacional com um número grande de domínios é complexo. Por isso, a partir desta seção são descritos apenas por tabelas as informações coletadas com as simulações, por estas utilizarem uma grande quantidade de recursos e domínios administrativos.

Nas tabelas 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 e 5.12 pode-se visualizar o comportamento do MatchMaker utilizando 21, 31, 43, 57, 73, 91, 111 domínios administrativos. Os parâmetros

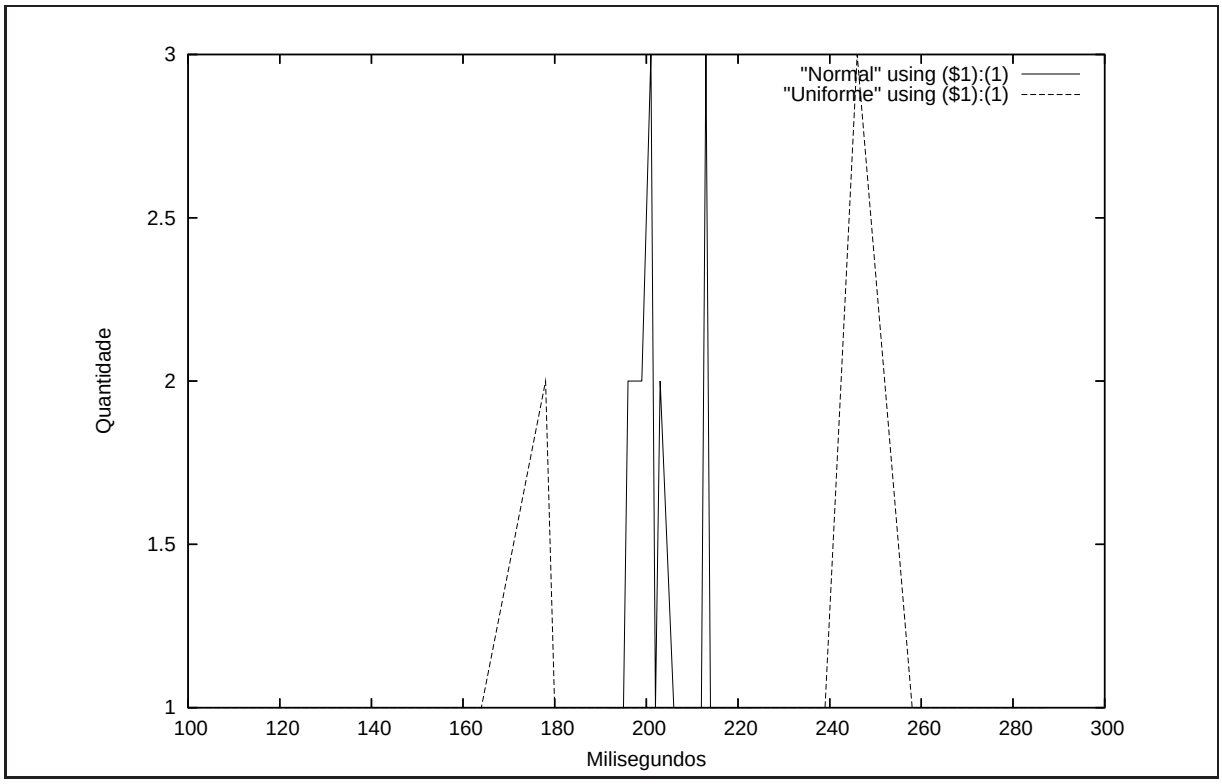

Figura 5.6: Distribuição dos custos de comunicação entre 12 DAs

utilizados para a simulação são os mesmos encontrados na tabela 5.4, com exceção ao número de vizinhos.

Na terceira simulação as comunicações entre DAs totalizaram **40 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.1 do anexo.

Tabela 5.6: Resultados Relativos à Terceira Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**786**	**935**
Total de recursos disponíveis	858	858
Recursos casados	85	85
Número de DAs	21	21

Quando utilizada uma simulação que faz uso de 4 DAs vizinhos, obtem-se uma grade formada por **21 DAs**, tabela 5.6, encontra-se um montante de **858 recursos ociosos**. Destes recursos ociosos, apenas 10% são definidos como sendo recursos pré-casados, conforme tabela 5.4, totalizando **85 recursos casados**.

5.2.4 Simulando Grade com 31 Domínios Administrativos

Na quarta simulação as comunicações entre DAs totalizaram **60 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.2 do anexo.

A tabela 5.7 faz referência à simulação utilizando **31 DAs**, através de 5 DAs diretamente conectados ao DAP. Neste, a grade possui um total de **1.753 recursos ociosos**. Conforme especificado na tabela 5.4 demonstra, a simulação consegue fazer uso de **175 recursos casados**.

Tabela 5.7: Resultados Relativos à Quarta Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**775**	**1010**
Total de recursos disponíveis	1.753	1.753
Recursos casados	175	175
Número de DA	31	31

5.2.5 Simulando Grade com 43 Domínios Administrativos

As comunicações entre DAs totalizaram **84 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.3 do anexo.

Tabela 5.8: Resultados Relativos à Quinta Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**780**	**1014**
Total de recursos disponíveis	2.332	2.332
Recursos casados	233	233
Número de DA	43	43

Na tabela 5.8 pode-se visualizar a utilização de **43 DAs** através de 6 DAs diretamente conectados ao DAP. Com esta configuração tem-se **2.332 recursos ociosos**, sendo que destes **233 casados**.

5.2.6 Simulando Grade com 57 Domínios Administrativos

As comunicações entre DAs totalizaram **112 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.4 do anexo.

Tabela 5.9: Resultados Relativos à Sexta Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**823**	**944**
Total de recursos disponíveis	2.882	2.882
Recursos casados	288	288
Número de DA	57	57

Visualiza-se na tabela 5.9 um total de **57 DAs**, isso deve-se ao fato do DAP possuir 7 vizinhos. Com 57 DAs obteve-se **2.882 recursos ocisos**, sendo destes **288 recursos casados**.

5.2.7 Simulando Grade com 73 Domínios Administrativos

As comunicações entre DAs totalizaram **144 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.5 do anexo.

Na tabela 5.10 visualiza-se os resultados de simulação com a utilização de **73 DA**. Neste, o DAP contém 8 vizinhos, sendo que a grade possui **4.150 recursos ociosos**. O DAP conseguiu reunir **415 recursos casados**.

Tabela 5.10: Resultados Relativos à Sétima Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**767**	**960**
Total de recursos disponíveis	4.150	4.150
Recursos casados	415	415
Número de DA	73	73

5.2.8 Simulando Grade com 91 Domínios Administrativos

As comunicações entre DAs totalizaram **180 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.6 do anexo.

Tabela 5.11: Resultados Relativos à Oitava Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**747**	**1001**
Total de recursos disponíveis	4.479	4.479
Recursos casados	447	447
Número de DA	91	91

Visualiza-se na tabela 5.11 a utilização de 9 DAs vizinhos diretamente conectados ao DAP, possibilitando a utilização de **91 DAs**. Os 91 DAs fornecem ao DAP um total de **4.479 recursos ociosos**. Do total de recursos, **447** serão utilizados.

5.2.9 Simulando Grade com 111 Domínios Administrativos

As comunicações entre DAs totalizaram **220 mensagens**. A distribuição das mensagens está representada no gráfico da figura 8.7 do anexo.

Tabela 5.12: Resultados Relativos à Nona Simulação

	Distribuição Normal	Distribuição Uniforme
Tempo de execução em milisegundos	**758**	**968**
Total de recursos disponíveis	6.338	6.338
Recursos pré-casados	633	633
Número de DA	111	111

Na última simulação, tabela 5.12, os resultados obtidos demonstram a utilização de **6.338 recursos ociosos**, divididos em **111 DAs**. Conforme a tabela 5.4 define, utilizaram-se 10% dos recursos ociosos para definir os **633 recursos casados**.

5.2.10 Comparando Simulações

As evoluções das simulações são apresentadas nos gráficos das figuras 5.7 e 5.8.

Analisando o desempenho do tempo de execução das simulações no gráfico da figura 5.7, percebe-se que o aumento do número de DAs não é um fator determinante para o aumento do tempo de execução. Isso deve-se ao fato de que as operações de pré-casamento, casamento e envio e recepção de mensagens serem realizadas de forma paralela na grade.

No gráfico da figura 5.8, ao contrário do que demonstra o gráfico da figura 5.7, o aumento do número de vizinhos é importante para o bom funcionamento do MatchMaker, pois aumenta o número de recursos casados. A quantidade de recursos casados é importante, pois os recursos podem deixarem de estar disponíveis a qualquer momento.

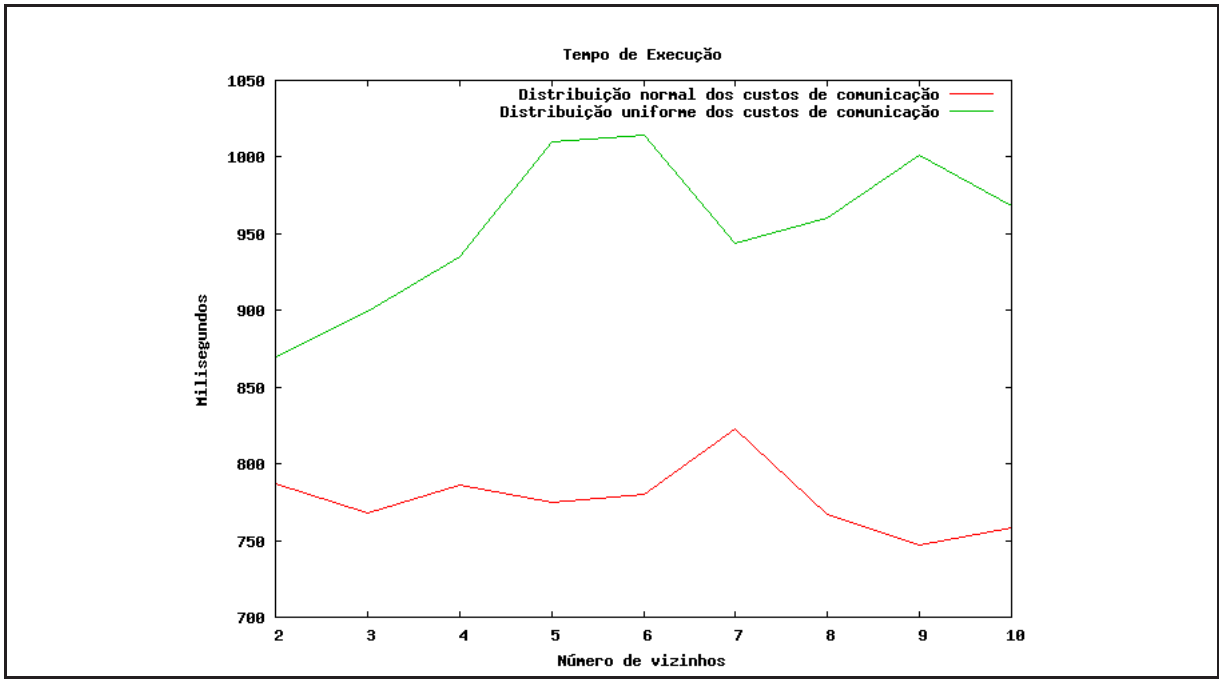

Figura 5.7: Analisando tempo de execução das simulações

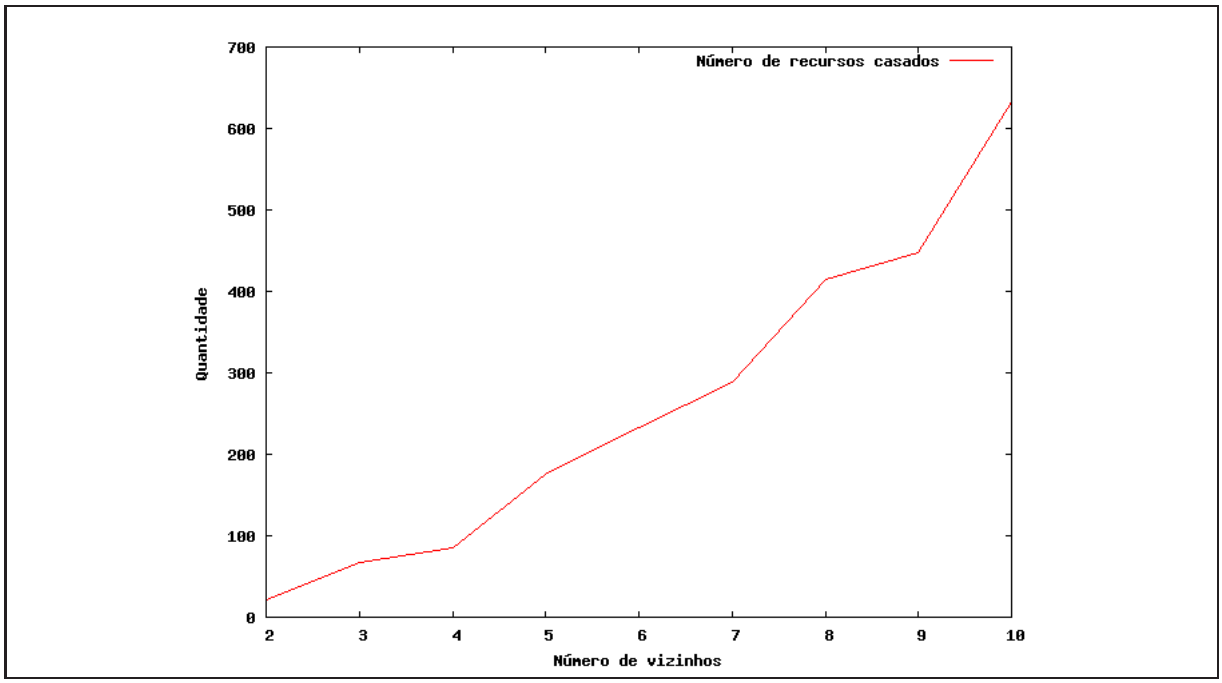

Figura 5.8: Analisando quantidade de recursos casados nas simulações

Uma comparação da influência da quantidade do número de recursos é apresentada no gráfico da figura 5.9. O gráfico indica uma tendência que demonstra que o aumento do número de DAs possibilita ao MatchMaker trabalhar com grandes quantidades de recursos sem comprometer o tempo de execução do processo de busca por recursos.

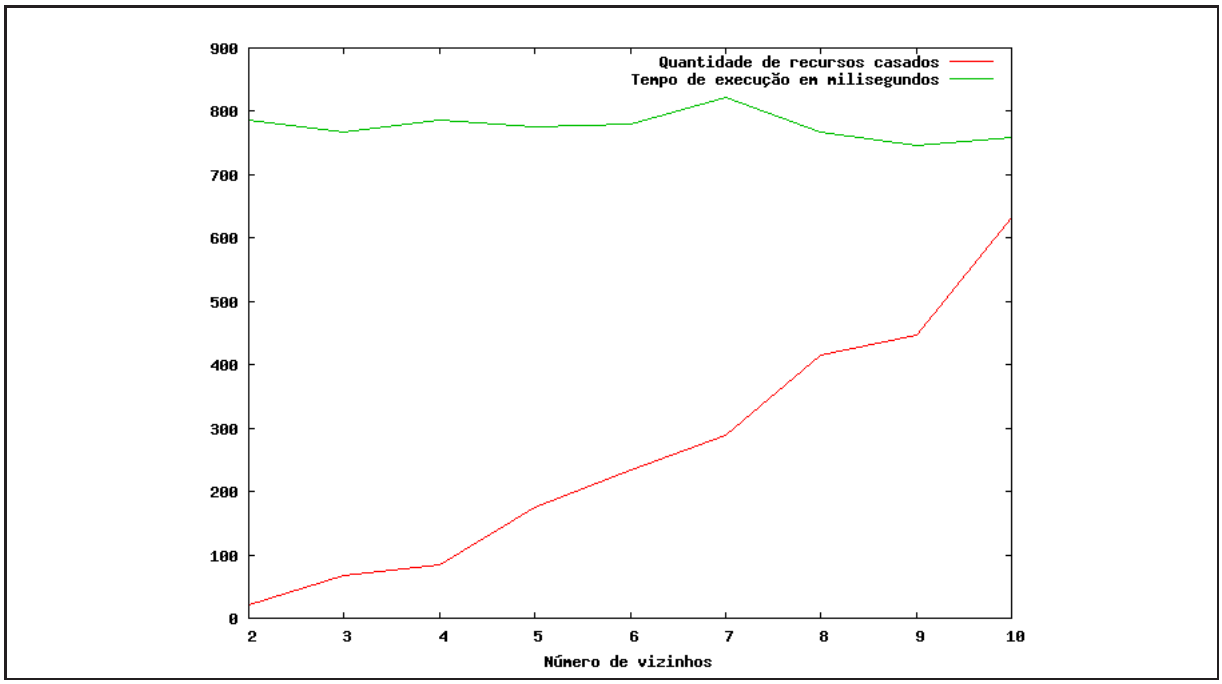

Figura 5.9: Comparando o número de recursos casados com o tempo de execução das simulações

6 CONCLUSÃO

Sistemas em grades possibilitam o compartilhamento de um amplo número de recursos geograficamente distribuídos que permitem a execução de aplicações com grandes dependências por recursos. A utilização de grades computacionais ainda possui grandes problemas desafiadores a serem resolvidos. Vários trabalhos destacam deficiências na exploração de recursos ociosos, na descrição de recursos, na expressão de necessidades, etc.

Este trabalho destaca algumas tecnologias implementadas em nível de *middleware*, que podem ser utilizadas para desenvolver sistemas para grades computacionais. Uma ênfase é dada para o *middleware* ProActive, demonstrando a existência de vários serviços que podem ser ofertados na construção de sistemas para grades.

O capítulo 6 apresenta os resultados de simulação realizados durante a fase de avaliação do algoritmo apresentado no capítulo 3.

Conclui-se que não existe uma convergência na literatura sobre questões técnicas para descrição de recursos e para determinação de como expressar as necessidades de usuários para com suas aplicações.

Conclui-se também, através das simulações, que o grande fator crítico da grade simulada é o custo de comunicação entre domínios.

Este trabalho apresentou um protótipo de um sistema, denominado MatchMaker, que possibilita o estudo e análise do comportamento de um ambiente para grades computacionais, tendo como principal objetivo a avaliação do processo de alocação de recursos que será destinado ao sistema Cadeo. O protótipo desenvolvido oferece suporte para que se possa implementar e avaliar algoritmos destinados à descrição de recursos e expressão de necessidades.

Para que se possa utilizar o MatchMaker em grades computacionais reais são necessários o aperfeiçoamento e implementação de alguns métodos, como os apresentados em trabalhos futuros.

7 TRABALHOS FUTUROS

Esta seção destina-se a apresentar alguns trabalhos que podem ser realizados para melhorar o protótipo.

7.1 União de Recursos

Como trabalho futuro observa-se a necessidade de implementar e avaliar algoritmos de união de recursos. Estes permitem uma melhor taxa de utilização de recursos disponíveis na grade computacional. Hu (HU, 2005) propõe um método de gerenciamento de recursos em grades computacionais baseados em agentes de união de recursos.

7.2 Sistema de Reserva

Um sistema de reserva de recursos possibilita ao MatchMaker aumentar sua flexibilidade em relação à alocação de recursos. Foster (FOSTER; ROY; SANDER, 2000) descreve que um sistema de reserva aumenta o desempenho da aplicação em ambientes de grades. Elmroth (ELMROTH; TORDSSON, 2004) apresenta um algoritmo de realização de reserva avançada de recursos.

7.3 Algoritmo de Casamento

Implementar e avaliar algoritmos de casamento baseados em semânticas para selecionar recursos com base nos requerimentos de usuários, fornecendo ao MatchMaker um método flexível de seleção de recursos. Vários trabalhos confirmam a eficiência do uso de semânticas (PERNAS ANA M.; DANTAS, 2004; POLLERES; TOMA; FENSEL, 2005; BAKER; SHADBOLT, 2003; TANGMUNARUNKIT; DECKER; KESSELMAN, 2003).

7.4 Sistema de Descrição de Recursos

A descrição dos recursos pertecentes às grades computacionais é um dos fatores premordiais para sistemas do tipo MatchMaker. O projeto NorduGrid (EEROLA et al., 2003) implementa um modelo de informações para grade computacional baseado em LDAP. O trabalho de Andreozzi (ANDREOZZI et al., 2005) apresenta um modelo de descrição de recursos de nível conceitual.

7.5 Agentes de Monitoração

Agentes móveis podem auxiliar no gerenciamento de processos de monitoração de recursos ociosos e recursos já em uso pelo MatchMaker. Vários trabalhos utilizam agentes móveis ((FOSTER; JENNINGS; C., 2004) (BELLIFEMINE; POGGI A.AND RIMASSA, 2001) (CAO et al., 2002)).

7.6 Eleição de um novo *dap*

No protótipo apresentado, o *dap* é a única entidade que se comunica com o usuário. Caso este venha a falhar, o usuário ficará sem condições de continuar operando suas tarefas dentro da grade. Neste contexto, entende-se que o uso de uma política de eleição de um novo *dap* é de grande valia para o sistema MatchMaker, incluindo habilidades de tolerância a falhas. Para maiores informações sobre processos de eleição consultar (ANTONOIU; SRIMANI, 1996).

REFERÊNCIAS

AGRAWAL, S.; DONGARRA, J.; SEYMOUR, K.; VADHIYAR, S. NetSolve: past, present, and future - a look at a grid enabled server. In: BERMAN, F.; FOX, G.; HEY, A. (Ed.). **Grid Computing**: making the global infrastructure a reality. New York, NY, USA: Wiley, 2003. p.615–624.

ANDRADE, N.; CIRNE, W.; BRASILEIRO, F.; ROISENBERG, P. OurGrid: an approach to easily assemble grids with equitable resource sharing. In: FEITELSON, D. G.; RUDOLPH, L.; SCHWIEGELSHOHN, U. (Ed.). **Job Scheduling Strategies for Parallel Processing**. [S.l.]: Springer Verlag, 2003. p.61–86. Lect. Notes Comput. Sci. vol. 2862.

ANDREOZZI, S.; BURKE, S.; FIELD, L.; FISHER, S.; KóNYA, B.; MAMBELLI, M. **GLUE Schema Specification version 1.2**. http://infnforge.cnaf.infn.it/glueinfomodel/uploads/Spec/GLUEInfoModel_1_2_final.pdf - último acesso em junho 2006.

ANDREOZZI, S.; MONTESI, D.; MORETTI, R. XMatch: a Language for Satisfaction-based Selection of Grid Services. **Scientific Programming Journal, Special Issue on Grids and Worldwide Computing, IOS Press**, [S.l.], v.13, n.4, p.299?316, 2005.

ANDREWS, G. R. **Foundations of multithreaded, parallel, and distributed programming**. Reading, Massachusetts: Addison-Wesley, 2000.

ANTONOIU, G.; SRIMANI, P. K. A Self-Stabilizing Leader Election Algorithm for Tree Graphs. **Journal of Parallel and Distributed Computing**, [S.l.], v.34, n.2, p.227–232, 1996.

ATTALI, I.; CAROMEL, D.; CONTES, A. Deployment-Based Security for Grid Applications. In: THE INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE (ICCS 2005), ATLANTA, USA, MAY 22-25, 2005. **Anais...** Springer Verlag, 2005. (LNCS).

AVAKI, C. **Avaki Data Grid 3.0**. 2003.

BADUEL, L.; BAUDE, F.; CAROMEL, D. Efficient, Flexible, and Typed Group Communications in Java. In: JOINT ACM JAVA GRANDE - ISCOPE 2002 CONFERENCE, 2002, Seattle. **Anais...** ACM Press, 2002. p.28–36.

BADUEL, L.; BAUDE, F.; CAROMEL, D.; CONTES, A.; HUET, F.; MOREL, M.; QUILICI, R. **Grid Computing: software environments and tools**. [S.l.]: Springer-Verlag, 2006.

BAKER, D.; SHADBOLT, J. **The Evolution of the Grid**. http://citeseer.ist.psu.edu/535794.html - último acesso em janeiro de 2006.

BALIEIRO, M. O. S. **PROTOCOLO CONSERVATIVO CMB PARA SIMULAÇÃO DISTRIBUÍDA**. www.dct.ufms.br/mestrado/dissertacoes/marta.pdf - último acesso em junho 2006.

BAUDE, F.; CAROMEL, D.; HUET, F.; VAYSSIERE, J. Communicating Mobile Active Objects in Java. In: HPCN EUROPE 2000, 2000. **Proceedings...** Springer, 2000. p.633–643. (LNCS, v.1823).

BAUDE, F.; CAROMEL, D.; HUET, F.; VAYSSIERE, J. Communicating Mobile Active Objects in Java. In: HPCN EUROPE 2000, 2000. **Proceedings...** Springer, 2000. p.633–643. (LNCS, v.1823).

BAUDE, F.; CAROMEL, D.; MOREL, M. From Distributed Objects to Hierarchical Grid Components. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS (DOA), CATANIA, SICILY, ITALY, 3-7 NOVEMBER, 2003, Springer Verlag. **Anais...** Lecture Notes in Computer Science: LNCS, 2003.

BELLIFEMINE, F.; POGGI A.AND RIMASSA, G. JADE: a fipa2000 compliant agent development environment. In: AGENTS '01: PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2001, New York, NY, USA. **Anais...** ACM Press, 2001. p.216–217.

BROOKE, J.; FELLOWS, D.; GARWOOD, K.; GOBLE, C. Semantic Matching of Grid Resource Descriptions. In: EUROPEAN ACROSS GRIDS CONFERENCE, 2004. **Anais...** [S.l.: s.n.], 2004. p.240–249.

BUYYA, R.; ABRAMSON, D.; GIDDY, J. An Economy Driven Resource Management Architecture for Global Computional Power Grids. In: THE 2000 INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS (PDPTA 2000), 2000, Las Vegas, USA. **Anais...** [S.l.: s.n.], 2000.

BUYYA, R.; ABRAMSON, D.; GIDDY, J.; STOCKINGER, H. **Economic Models for Resource Management and Scheduling in Grid Computing**. 2002.

CAO, J.; SPOONER, D.; TURNER, J.; JARVIS, S.; KERBYSON, D.; SAINI, S.; NUDD, G. Agent-Based Resource Management for Grid Computing. In: CCGRID '02: PROCEEDINGS OF THE 2ND IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2002, Washington, DC, USA. **Anais...** IEEE Computer Society, 2002. p.350.

CAROMEL, D. Toward a method of object-oriented concurrent programming. **Communications of the ACM**, [S.l.], v.36, n.9, p.90–102, Sept. 1993.

CAROMEL, D. **Oasis Group at INRIA Sophia - Antipolis. ProActive, the Java libary for parallel, distributed, concurrent computing with security and mobility**. Disponibilizado em http://www-sop.inria.fr/sloop/javall/index.html. Atualizado em maio de 2004.

CAROMEL, D.; KLAUSER, W.; VAYSSIERE, J. Towards Seamless Computing and Metacomputing in Java. In: CONCURRENCY PRACTICE AND EXPERIENCE, 1998. **Anais...** Wiley & Sons: Ltd., 1998. v.10, n.11–13, p.1043–1061. http://www-sop.inria.fr/oasis/proactive/.

CAROMEL, D.; KLAUSER, W.; VAYSSIERE, J. Towards Seamless Computing and Metacomputing in Java. In: CONCURRENCY PRACTICE AND EXPERIENCE, 1998. **Anais...** Wiley & Sons: Ltd., 1998. v.10, n.11–13, p.1043–1061. http://www-sop.inria.fr/oasis/proactive/.

CERA, M. C. **Suporte ao Controle e Alocação Dinâmica de Computadores em Java**. 2005. Dissertação (Mestrado em Ciência da Computação) — Dept. of Informatics, University of Santa Maria, Brazil.

CHAPIN, S. J.; KATRAMATOS, D.; KARPOVICH, J.; GRIMSHAW, A. S. The Legion Resource Management System. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). **Job Scheduling Strategies for Parallel Processing**. [S.l.]: Springer Verlag, 1999. p.162–178.

CHAPIN, S. J.; SPAFFORD, E. H. Support for Security in Distributed Systems Using Messiahs. In: NIST-NCSC NATIONAL COMPUTER SECURITY CONFERENCE, 17., 1994. **Proceedings...** [S.l.: s.n.], 1994. p.339–347.

CHAPIN, S.; KATRAMATOS, D.; KARPOVICH, J.; GRIMSHAW, A. Resource management in Legion. **Future Generation Computer Systems**, [S.l.], v.15, n.5–6, p.583–594, 1999.

CHETTY, M.; BUYYA, R. Weaving Computational Grids: how analogous are they with electrical grids? **Computing in Science and Engineering**, [S.l.], v.4, n.4, p.61–71, July/Aug. 2002.

CIRNE, W. Grids Computacionais: arquiteturas, tecnologias e aplicações. In: ESCOLA REGIONAL DE ALTO DESEMPENHO - ERAD2003, 3., 2003, Santa Maria, RS. **Anais...** [S.l.: s.n.], 2003. p.103–134.

CIRNE, W.; MARZULLO, K. OpenGrid: a user-centric approach for grid computing. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBACPAD 2001), 13., 2001, Pirenópolis, GO. **Anais...** [S.l.: s.n.], 2001.

CIRNE, W.; PARANHOS, D.; COSTA, L.; SANTOS-NETO, E.; BRASILEIRO, F.; SAUVé, J.; SILVA, F. A. B. da; BARROS, C. O.; SILVEIRA, C. Running Bag-of-Tasks Applications on Computational Grids: the MyGrid approach. In: ICCP'2003 - INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 2003, Kaohsiung, Taiwan, ROC. **Proceedings...** [S.l.: s.n.], 2003.

CUBERT, R.; FISHWICK, P. **SimPack/Sim++ Simulation Toolkit**. http://www.cs.sunysb.edu/ algorith/implement/simpack/implement.shtml - último acesso em maio 2006.

CZAJKOWSKI, K.; FOSTER, I.; KARONIS, N.; KESSELMAN, C.; MARTIN, S.; SMITH, W.; TUECKE, S. A Resource Management Architecture for Metacomputing Systems. **Lecture Notes in Computer Science**, [S.l.], v.1459, p.62–??, 1998.

EEROLA, P.; EKELOF, T.; ELLERT, M.; HANSEN, J. R.; KONSTANTINOV, A.; KONYA, B.; NIELSEN, J. L.; OULD-SAADA, F.; SMIRNOVA, O.; WAANANEN, A. **The NorduGrid architecture and tools**. 2003.

ELMROTH, E.; TORDSSON, J. A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection. In: PARA, 2004. **Anais...** [S.l.: s.n.], 2004. p.1061–1070.

ERWIN, D. W.; SNELLING, D. F. UNICORE: A Grid computing environment. **Lecture Notes in Computer Science**, [S.l.], v.2150, p.825–??, 2001.

FERREIRA, E. C.; SANTOS, A. F.; SCHULZE, B. Distribuição de Processos em Ambiente de Grid. **Relatório de Pesquisas do LNCC**, [S.l.], 2003.

FOSTER, I. The Grid: A New Infrastructure for the 21st Century Science. **Physics Today**, [S.l.], February 2002.

FOSTER, I.; JENNINGS, N. R.; C., K. Brain Meets Brawn: why grid and agents need each other. In: AAMAS '04: PROCEEDINGS OF THE THIRD INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.8–15.

FOSTER, I.; KESSELMAN, C. Globus: a metacomputing infrastructure toolkit. **11 International Journal of Supercomputer Applications and High Performance Computing**, [S.l.], p.115–128, 1997.

FOSTER, I.; KESSELMAN, C. Globus: a toolkit-based grid architecture. In: FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid**: blueprint for a future computing infrastructure. San Francisco, CA, USA: MORGAN-KAUFMANN, 1998. p.259–278.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: enabling scalable virtual organizations. **Lecture Notes in Computer Science**, [S.l.], v.2150, p.1–??, 2001.

FOSTER, I.; ROY, A.; SANDER, V. **A quality of service architecture that combines resource reservation and application adaptation**. 2000.

FREY, J.; TANNENBAUM, T.; LIVNY, M.; FOSTER, I.; TUECKE, S. Condor-G: a computation management agent for multi-institutional grids. In: High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium, 2001, San Francisco, CA, USA. **Anais...** IEEE Computer Society Press, 2001. p.55–63.

GALSTYAN, A.; CZAJKOWSKI, K.; LERMAN, K. **Resource Allocation in the Grid Using Reinforcement Learning**. 2004.

GRIMSHAW, A.; FERRARI, A.; KNABE, F.; HUMPHREY, M. Wide Area Computing: resource sharing on a large scale. **IEEE Computer**, Seattle, Washington, USA, v.35, n.2, p.29–37, May 1999.

GRIMSHAW, A. S.; NATRAJAN, A.; HUMPHREY, M. A.; LEWIS, M. J.; NGUYEN-TUONG, A.; KARPOVICH, J. F.; MORGAN, M. M.; FERRARI, A. J. From Legion to Avaki: the persistence of vision. In: BERMAN, F.; FOX, G.; HEY, A. (Ed.). **Grid Computing**: making the global infrastructure a reality. New York, NY, USA: Wiley, 2003. p.265–298.

GRIMSHAW, A. S.; WULF, W. A.; TEAM, C. T. L. The Legion vision of a worldwide virtual computer. **Communications of the ACM**, New York, NY, USA, v.40, n.1, p.39–45, 1997.

GUPTA, A.; AGRAWAL, D.; ABBADI, A. E. Distributed Resource Discovery in Large Scale Computing Systems. **The 2005 Symposium on Applications and the Internet (SAINT'05)**, [S.l.], p.320–326, 04 2005.

HARTH, A.; DECKER, S.; HE, Y.; TANGMUNARUNKIT, H.; KESSELMAN, C. A semantic matchmaker service on the grid. In: WWW ALT. '04: PROCEEDINGS OF THE 13TH INTERNATIONAL WORLD WIDE WEB CONFERENCE ON ALTERNATE TRACK PAPERS & POSTERS, 2004, New York, NY, USA. **Anais...** ACM Press, 2004. p.326–327.

HAUMACHER, B.; MOSCHNY, T.; PHILIPPSEN, M. **JavaParty, a distributed companion to Java**. Disponibilizado em http://wwwipd.ira.uka.de/JavaParty/.

HE, L.; IOERGER, T. R. Forming resource-sharing coalitions: a distributed resource allocation mechanism for self-interested agents in computational grids. In: SAC '05: PROCEEDINGS OF THE 2005 ACM SYMPOSIUM ON APPLIED COMPUTING, 2005, New York, NY, USA. **Anais...** ACM Press, 2005. p.84–91.

HU, H. Grid Computing based on Social Agent Union. In: PARALLEL AND DISTRIBUTED COMPUTING AND NETWORKS, 2005. **Anais...** [S.l.: s.n.], 2005. p.301–306.

INRIA - ProActive. **ProActive Security Mechanism**. http://www-sop.inria.fr/oasis/proactive/doc/api/org/objectweb/proactive/doc-files/Security.html - último acesso em maio 2006.

JEFFERY, K. G. **METADATA**: the future of information systems. http://www.eurocris.org/en/taskgroups/cerif/articles/new_12 - último acesso em julho 2006.

KON, F. **Introdução aos sistemas de computação em grade**: globus, legion, globe e condor. 2003.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software Practice and Experience**, [S.l.], v.32, n.2, p.135–164, 2002.

LI, C.; SYCARA, K. Algorithm for combinatorial coalition formation and payo division in an electronic marketplace. In: IN PROCEEDINGS OF THE FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS(AAMAS), 2002. **Anais...** [S.l.: s.n.], 2002. p.120–127.

LITZKOW, M. J.; LIVNY, M.; MUTKA, M. W. Condor : a hunter of idle workstations. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS (ICDCS '88), 8., 1988, Washington, D.C., USA. **Anais...** IEEE Computer Society Press, 1988. p.104–111.

LIU, C.; FOSTER, I. A Constraint Language Approach to Grid Resource Selection. **14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)**, [S.l.], p.7–14, Summer 2004.

LIVNY, M.; BASNEY, J.; RAMAN, R.; TANNENBAUM, T. Mechanisms for High Throughput Computing. **SPEEDUP Journal**, [S.l.], v.11, n.1, June 1997.

LUCCHESE, F. **Princípios de Grid Computing**. http://www.gridcomputing.com.br/tiki-read_article.php?articleId=4 - último acesso em maio 2006.

MATHIAS, E. N.; KOSLOVSKI, G. P.; RECKZIEGEL, J. F.; PASIN, M. Uma Avaliação do uso de ProActive no Processamento de Alto Desempenho. In: V WORKSHOP DE COMPUTAçãO DE ALTO DESEMPENHO, WSCAD 2004, 2004, Foz do Iguaçu, PR, Brasil. **Anais...** SBC, 2004. p.204–207.

MELLO, E. R. **Um tutorial sobre SPKI/SDSI - Simple Public Key Infrastructure / Simple Distributed Security Infrastructure**. www.das.ufsc.br/seguranca/artigos/tutorial-spki.pdf - último acesso em maio 2006.

MUTKA, M.; LIVNY, M. Scheduling Remote Processing Capacity In A Workstation-Processing Bank Computing System. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 7., 1987, Berlin, Germany. **Anais...** [S.l.: s.n.], 1987. p.2–9.

NESTOROV, S.; ABITEBOUL, S.; MOTWANI, R. Inferring Structure in Semistructured Data. In: WORKSHOP ON MANAGEMENT OF SEMISTRUCTURED DATA, 1997. **Anais...** [S.l.: s.n.], 1997.

OpenPBS. **Portable Batch System**. http://www.openpbs.org/ - último acesso em junho 2006.

PARANHOS, D.; CIRNE, W.; BRASILEIRO, F. Trading Cycles for Information: using replication to schedule bag-of-tasks applications on computational grids. In: EURO-PAR 2003: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING, 2003, Klagenfurt, Austria. **Anais...** [S.l.: s.n.], 2003.

PERNAS ANA M.; DANTAS, M. A. Ontologias Aplicadas a Descrição de Recursos em Ambientes Grid. **INFOCOMP Journal of Computer Science**, [S.l.], v.3, n.2, p.26–31, 2004.

PHILIPPSEN, M.; HAUMACHER, B.; NESTER, C. More efficient serialization and RMI for Java. **Concurrency: Practice and Experience**, [S.l.], v.12, p.495–518, 2000.

PHILIPPSEN, M.; ZENGER, M. JavaParty – transparent remote objects in Java. **Concurrency: Practice and Experience**, [S.l.], v.9, n.11, p.1225–1242, nov 1997. Special Issue: Java for computational science and engineering – simulation and modeling II.

POLLERES, A.; TOMA, I.; FENSEL, D. Modeling Services for the Semantic Grid. In: SEMANTIC GRID: THE CONVERGENCE OF TECHNOLOGIES, 2005. **Anais...** Internationales Begegnungs- und Forschungszentrum (IBFI): Schloss Dagstuhl: Germany, 2005. n.05271. (Dagstuhl Seminar Proceedings). <http://drops.dagstuhl.de/opus/volltexte/2005/394> [date of citation: 2005-01-01].

RAMAN, R.; LIVNY, M.; SOLOMON, M. **Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching**. 2004.

RAMAN, R.; LIVNY, M.; SOLOMON, M. H. Matchmaking: distributed resource management for high throughput computing. In: HPDC, 1998. **Anais...** [S.l.: s.n.], 1998. p.140–.

RANA, O. F.; MOREAU, L. Issues in Building Agent based Computational Grids. In: THIRD WORKSHOP OF THE UK SPECIAL INTEREST GROUP ON MULTI-AGENT SYSTEMS (UKMAS'2000), 2000, Oxford, UK. **Anais...** [S.l.: s.n.], 2000. p.11.

SOBOLEWSKI, M.; KOLONAY, M. Federated Grid Computing with Interactive Service-oriented Programing. **SAGE Publications**, [S.l.], v.14, n.1, p.55–66, 2006.

TANAKA, Y.; NAKADA, H.; SEKIGUCHI, S.; SUZUMURA, S.; MATSUOKA, S. Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing. **Journal of Grid Computing**, [S.l.], v.1, n.1, p.41–51, 2003.

TANGMUNARUNKIT, H.; DECKER, S.; KESSELMAN, C. Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, 2003. **Anais...** [S.l.: s.n.], 2003. p.706–721.

THAIN, D.; LIVNY, M. Error Scope on a Computational Grid: theory and practice. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING (HPDC-11 2002), 11., 2002, Edinburgh, Scotland, UK. **Anais...** [S.l.: s.n.], 2002. p.199–208.

THAIN, D.; TANNENBAUM, T.; LINVY, M. Condor and the Grid. In: BERMAN, F.; FOX, G.; HEY, A. (Ed.). **Grid Computing**: making the global infrastructure a reality. New York, NY, USA: Wiley, 2003. p.299–336.

Traffic. **Frequently asked questions**. http://www.internettrafficreport.com/ - último acesso em novembro 2006.

8 GRÁFICOS DE DISTRIBUIÇÃO DE MENSAGENS

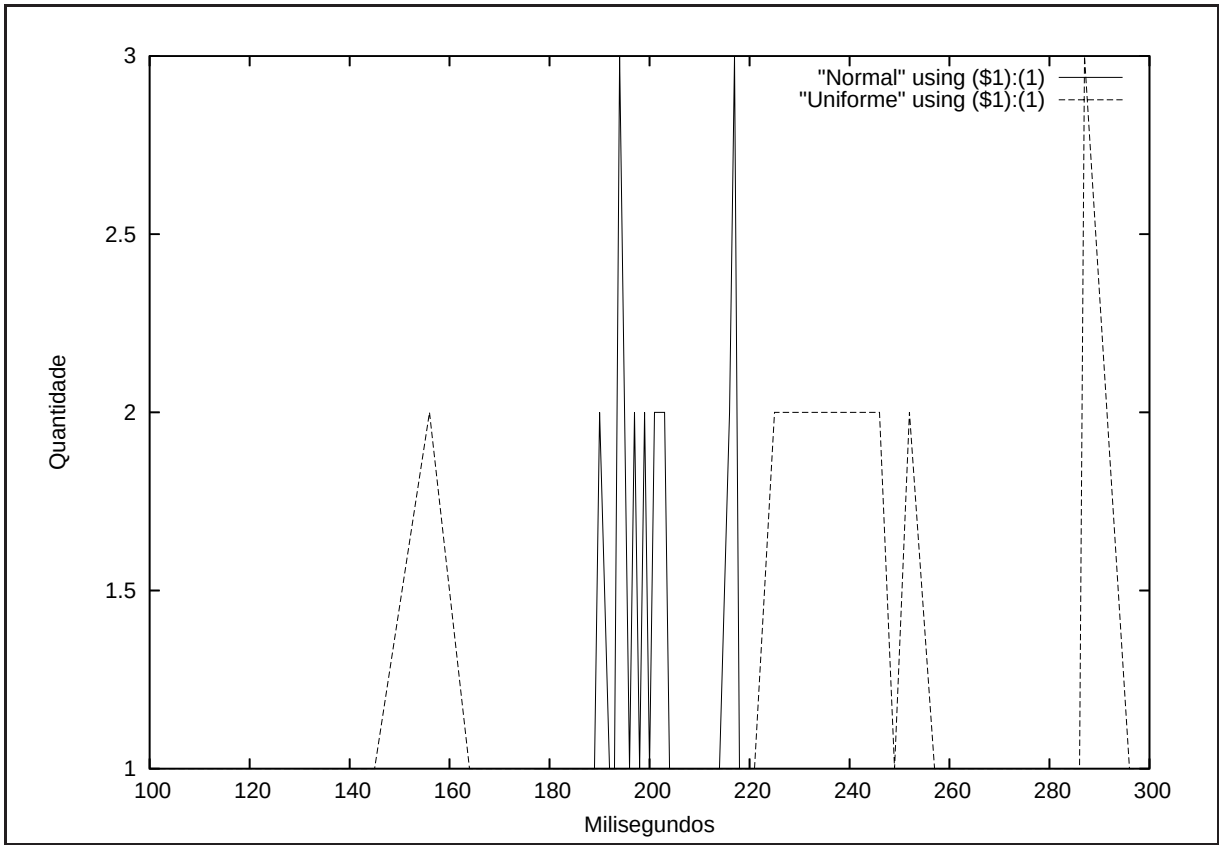

Figura 8.1: Distribuição dos custos de comunicação entre 21 DAs

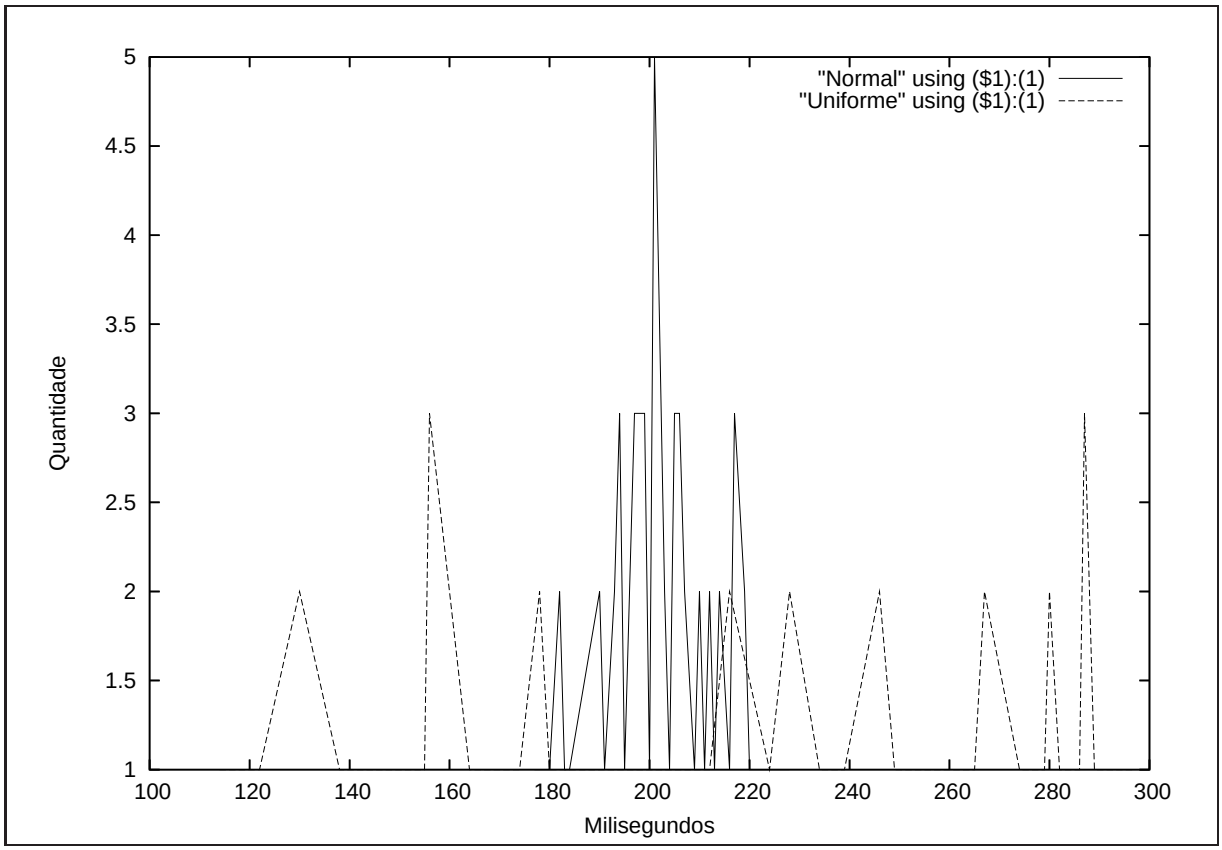

Figura 8.2: Distribuição dos custos de comunicação entre 31 DAs

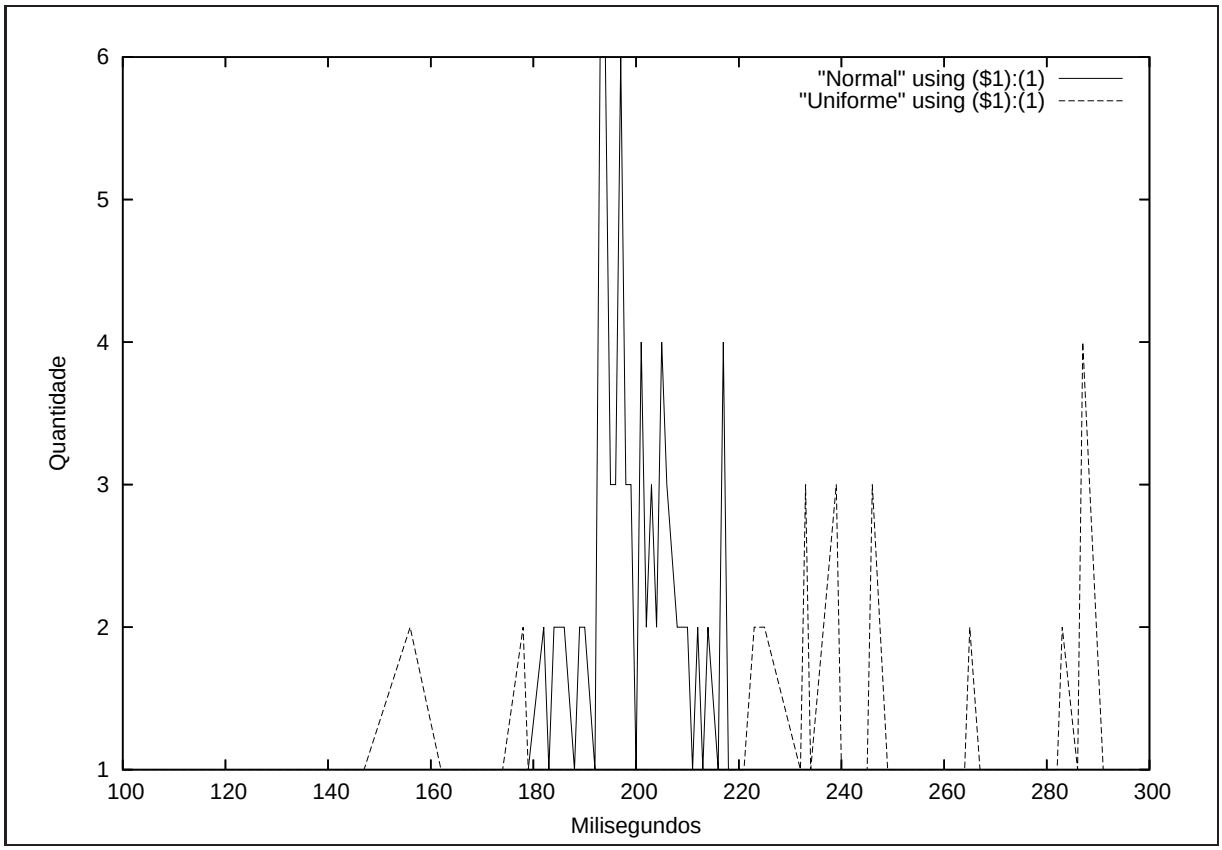

Figura 8.3: Distribuição dos custos de comunicação entre 43 DAs

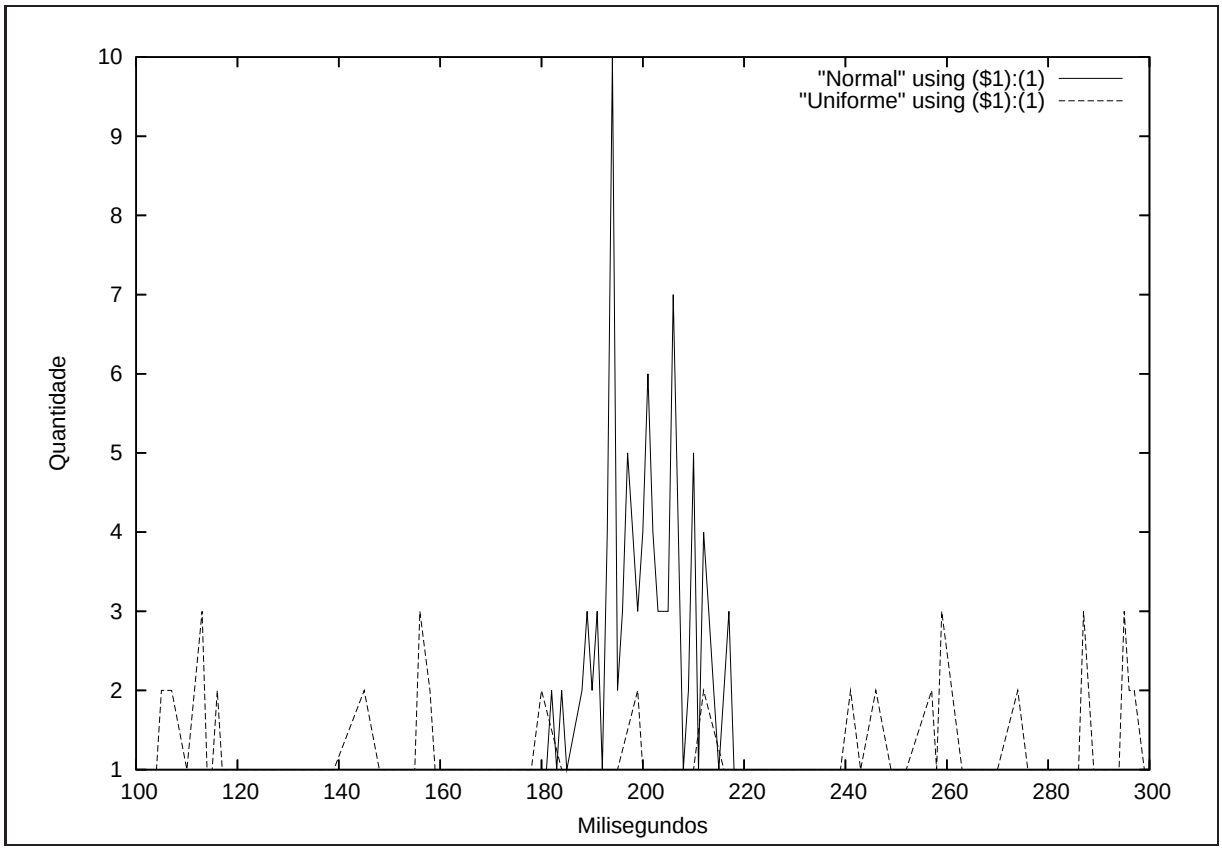

Figura 8.4: Distribuição dos custos de comunicação entre 57 DAs

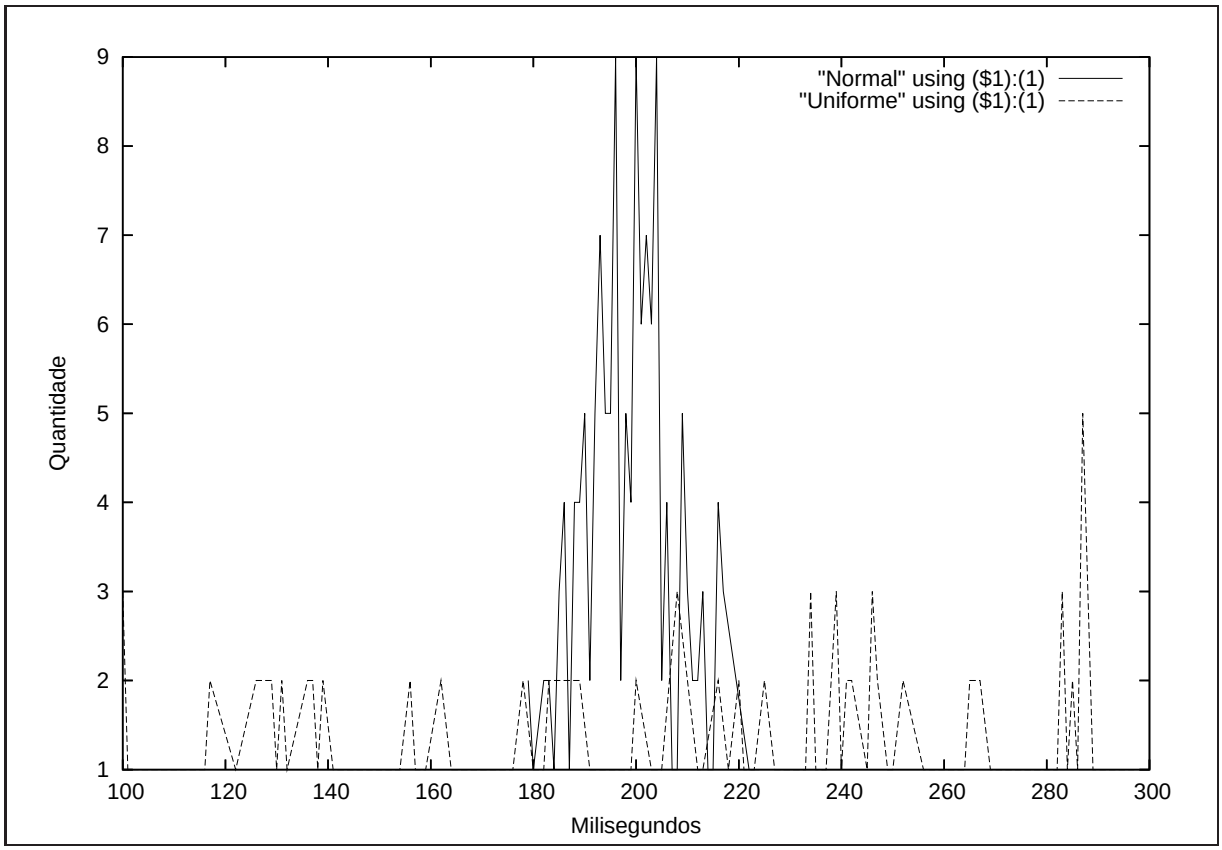

Figura 8.5: Distribuição dos custos de comunicação entre 73 DAs

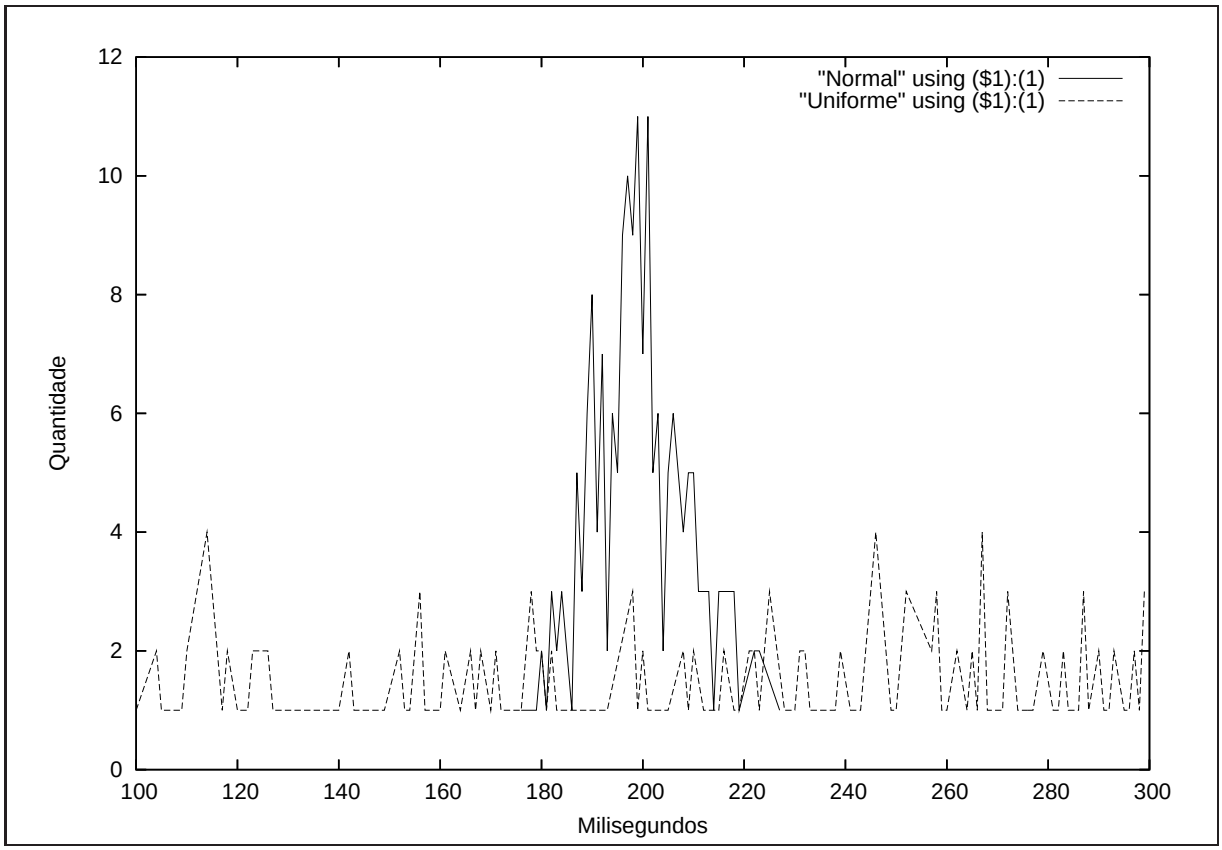

Figura 8.6: Distribuição dos custos de comunicação entre 91 DAs

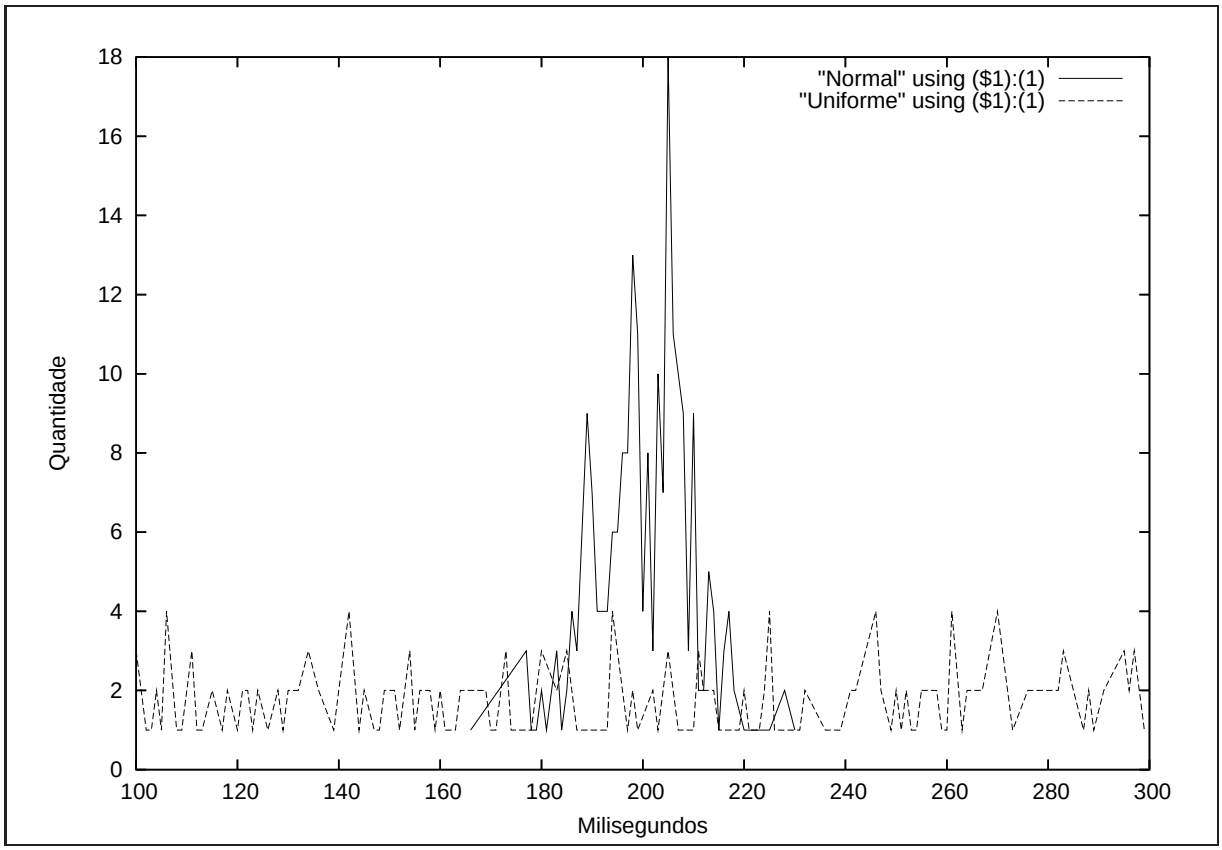

Figura 8.7: Distribuição dos custos de comunicação entre 111 DAs