Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Dissertação de Mestrado

Ad Hoc Grid : uma grade computacional entre-pares auto-organizável

Pablo Gustavo Soares Tibúrcio

Campina Grande, Paraíba, Brasil
Setembro, 2009

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Ad Hoc Grid : uma grade computacional entre-pares auto-organizável

Pablo Gustavo Soares Tibúrcio

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Marco Aurélio Spohn
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

G554a
2009

Tibúrcio, Pablo Gustavo Soares.
Ad Hoc Grid: uma grade computacional entre-pares auto-organizável / Pablo Gustavo Soares Tibúrcio. – Campina Grande, 2009.
70f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientador: Prof. Dr. Marco Aurélio Spohn.

1. Redes de Computadores. 2. Grades Computacionais. 3. Auto-organização. I. Título.

CDU 004.7(043)

"Ad Hoc Grid: UMA GRADE COMPUTACIONAL ENTRE-PARES AUTO-ORGANIZÁVEL"

PABLO GUSTAVO SOARES TIBURCIO

DISSERTAÇÃO APROVADA EM 18.09.2009

MARCO AURÉLIO SPOHN, Ph.D
Orientador(a)

FRANCISCO VILAR BRASILEIRO, PH.D
Examinador(a)

GUSTAVO HENRIQUE MATOS BEZERRA MOTTA, DR.
Examinador(a)

CAMPINA GRANDE - PB

"A ciência nos dá o Conhecimento, e a religião nos dá o Sentido. Ambos são pré-requisitos para uma existência decente."

Pe. cosmólogo Michael Heller.

Resumo

O paradigma de computação em grades tem evoluído bastante nos últimos anos, estimulando o desenvolvimento de um crescente número de *middlewares* para computação distribuída. Além de revolucionar a disciplina da computação distribuída, a comunidade de grades computacionais vem aperfeiçoando seus padrões e protocolos. Um dos desafios desse avanço, e que tornará a grade ainda mais atrativa, é fazer com que ela seja um mecanismo cada vez mais simples de ser implantado, mantido e utilizado, além de possibilitar a interação entre as diversas soluções de grade hoje existentes. Uma nova abordagem de grades *ad hoc* vem sendo estudada, com o objetivo de viabilizar algumas dessas características. Neste trabalho, foi desenvolvida uma versão auto-organizável (*ad hoc*) de grades entre-pares baseada no *middleware* do OurGrid. Dentre as características de uma grade ad hoc, foram destacadas a independência de controle e a independência estrutural. A independência de controle garante que a grade suporte funções administrativas sem um ponto central de coordenação. Já a independência estrutural garante a auto-organização da grade.

Abstract

The grid computing paradigm has been quite evolved in recent years, stimulating the development of a growing number of middlewares for distributed computing. In addition to revolutionizing the discipline of distributed computing, the community of computational grids is improving its standards and protocols. One of the challenges of this progress, that will make the grid even more attractive, is to make it a mechanism increasingly easy to be deployed, maintained and used, and to enable the interaction between the various solutions of grid now available. A new approach to *ad hoc* grids has been studied, aiming to make some of these characteristics become viable. In this work, it was developed a self-organizing version (*ad hoc*) of a peer-to-peer grids based on the OurGrid middleware. Among the characteristics of an *ad hoc* grid, were highlighted the structural independence and control independence. The control independence ensures that the grid supports administrative functions without a central point of coordination, whereas the structural independence ensures the self-organization of the grid.

Agradecimentos

Primeiramente agradeço a Deus o criador da vida e que dá sentido a tudo que fazemos.

Agradeço aos meus pais que sempre zelam por mim, mesmo na distância. No silêncio sábio do meu pai, em seus ensinamentos e abraços sinceros. Nas conversas com minha mãe, que também tanto me ensinaram e me confortaram.

Aos meus irmãos Ingrid, David e Gel por sempre estarem ao meu lado, por me apoiarem e proporcionarem alegria, muitas vezes com atos singelos.

A minha pequena notável namorada Isabela, pelo suporte afetivo, emocional, gramatical (...), e que apesar da distância, se fez próxima em todos os momentos. Pequeno grande suporte para mim.

Ao meu amigo e orientador Professor Spohn, por ter contribuído intensamente no meu crescimento intelectual, e o melhor, de forma dedicada, descontraída e bastante eficiente. Certamente um exímio exemplo a seguir no decorrer de minha carreira acadêmica.

Ao Professor Fubica, primeiramente pelo agradabilíssimo ambiente disponibilizado para o meu trabalho, o LSD. Agradeço também pelas contribuições dadas ao meu trabalho.

Aos grandes amigos e irmãos do Grupo de Oração Água Viva que sempre me incentivaram e ajudaram a seguir com perseverança, através de conversas, orações e momentos de descontração. Verdadeiras expressões do amor de Deus na minha vida.

Aos amigos do Apto. 04 Fernando Fagundes, Daniel Lucena, Daniel Leite, Fábio Gualberto e Jackson Porciúncula, com quem compartilhei momentos de discussão, desentendimento, ódio, afeto mas sobre tudo de companheirismo, momentos quais me renovaram as forças pra continuar o mestrado.

À todos os amigos do LSD e da UFCG, em especial, José Flávio e Elmano com quem firmei grande amizade, e as amigas da Sala Dona Bica, Ayla Débora, Lívia Sampaio e Eliana que sempre me incentivaram no mestrado, com suas dicas, conversas e ensinamentos. À Andréia Mendonça, Espedito, Marcelo Iury que me acolheram tão bem em Campina Grande, e hoje também são grandes amigos.

A paciência de Vinicius e Tomás ao prestarem o suporte técnico durante a implementação e os experimentos do *ad hoc* Grid.

A Moisés Rodrigues pela ajuda na implementação e nos testes do *ad hoc* Grid.

Aos amigos UFAL/UFCG, Thiago Sales, Leandro Sales, Ivo Augusto, Marcos Fábio, Glauber Vinícius, prof. Hyggo Almeida, Luíz Augusto, Eduardo Barros, Mário Hozano, Ig Bittencour e o prof. Iedo Teodóro, pelas viagens, conversas, saídas, encomendas *from China*, dentre outros momentos interessantes.

À todos meus familiares que contribuem de forma positiva na minha vida.

Às instituições HP e por conseguinte à Capes que forneceram-me bolsa de estudos neste período do mestrado.

Conteúdo

Lista de Siglas e Abreviaturas

OG - *OurGrid*

RP - *Rendezvous Point*

LMG - *Grupo Multicast Local*

P2PMG - *Grupo Multicast entre-pares*

NoF - *Network of Favors*

RMI - *Remote Method Invocation*

XMPP - *Extensible Messaging and Presence Protocol*

OV - *Organização Virtual*

XORP - *eXtensible Open Router Platform*

SDF - *Site Description File*

GDF - *Grid Description File*

IGMP - *Internet Group Management Protocol*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

IP - *Internet Protocol*

P2P - *Peer-to-Peer*

LAN - *Local Area Network*

PIM - *Protocol Independent Multicast*

MSOPF - *Multicast Open Shortest Path First*

DVMRP - *Distance Vector Multicasting Routing*

Mbone - *Multicast Backbone*

SM - *Sparse Mode*

DM - *Dense Mode*

DS - *Discovery Service*

SSM - *Source Specific Mode*

SDM - *Sparse-Dense Mode*

SNMP - *Simple Network Management Protocol*

MRTG - *Multi Router Traffic Grapher*

SWAN - *Sandboxing Without A Name*

ICENI - *Imperial College e-Science Networked Infrastructure*

SOA - *Service-oriented Architecture*

SLA - *Service Level Agreement*

NMS - *Network Management System*

SMI - *Structure of Management Information*

OID - *Object Identifier*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

O conceito de grade computacional tem sido bastante estudado nos últimos anos, elevando o nível de aperfeiçoamento dessa tecnologia. Como consequência disso, *middleware* cada vez mais robustos que possibilitam a criação e o gerenciamento de uma grade computacional vem sendo desenvolvidos. Outro resultado bastante relevante é a criação e padronização de tecnologias que permitem a interoperabilidade, portabilidade e reutilização de serviços em grades. Além desses avanços, um novo conceito de grades *ad hoc* vem sendo estudado, a fim de tornar uma grade mais simples de ser implantada, mantida e utilizada.

Smith *et al.* [51] apresentam uma definição simples de uma grade *ad hoc*: "*uma formação espontânea de computadores heterogêneos em uma comunidade, sem uma estrutura fixa predefinida e com o mínimo de funções administrativas*". Os autores destacam que uma grade *ad hoc* difere das grades tradicionais, por ter uma grande quantidade de máquinas não dedicadas, solicitando operações não intrusivas ao *middleware* da grade *ad hoc*. Apesar de ser composta em sua maioria por máquinas não dedicadas e transientes, ela também pode ter máquinas dedicadas e de alto desempenho.

Amin *et al.* [9] definem uma grade *ad hoc* da seguinte forma: uma arquitetura de computação distribuída que oferece soluções de grade independente da arquitetura, tecnologia e controle, e que suporta modalidades de utilização esporádicas e *ad hoc*. Para os autores supracitados, as principais diferenças são que uma grade tradicional tem pontos de entrada bem definidos e um índice central de informações da grade para a descoberta de serviço, enquanto que uma grade *ad hoc* não tem nenhum ponto de entrada e nem mesmo um índice central de informações da grade.

Assim como a própria definição de grade computacional "*pode significar diferentes coisas para indivíduos diferentes*" [37], uma grade *ad hoc* ainda não possui um padrão, nem mesmo uma arquitetura ou um conjunto de regras que a definam. Os trabalhos desenvolvidos nessa área [9; 51; 41] apresentam apenas algumas características importantes que uma grade *ad hoc* deve ter. Dentre essas características, as que buscamos abordar neste trabalho são: uma grade auto-organizável e descentralizada.

1.1 Motivações

Atualmente grande parte dos projetos de grades computacionais é desenvolvida e mantida por centros de pesquisa, liderados por grandes instituições privadas ou governamentais. Dentre elas podemos destacar a Comissão Européia que fundou o projeto EGEE [35], a NASA com o projeto NASA's Information Power Grid (IPG) [39] e a IBM juntamente com o Departamento de energia dos EUA, responsáveis pelo Science Grid [2]. Com esses projetos, as instituições estão interessadas, em especial, em unir o poder de computação de seus grandes centros. Para isso, elas mantêm seus ambientes de grade computacional, formado por uma infra-estrutura estática e centralizada [29; 40].

Esta estrutura é baseada no modelo tradicional de grades computacionais (vide Capítulo 2) que é composto, em sua grande maioria, por máquinas dedicadas. Sendo assim, para manter essas comunidades ativas e oferecer seus recursos da grade, as instituições devem manter uma equipe responsável por configurar, gerenciar e monitorar suas comunidades. Apesar do modelo de grade tradicional ser eficiente para comunidades que tem um tipo de colaboração bem definido, e que possuem uma arquitetura definida com controle de acesso e privilégios específicos para os seus usuários, ele não se adapta bem a certos tipos de comunidades cujos componentes mudam constantemente. Essa mudança pode ser na estrutura física, como por exemplo, na quantidade de máquinas, de recursos, etc, ou na parte lógica, como no tipo de colaboração, políticas de privacidade, dentre outras. Essa comunidade é referenciada por alguns autores como uma grade esporádica ou *ad hoc* [9; 53].

Em uma pesquisa realizada em 2003 [25] através de questionários preenchidos por usuários de grades computacionais espalhados por todo o mundo, concluiu-se que em média

75% das falhas em grades computacionais estavam relacionadas a configuração do ambiente. Após dois anos essa pesquisa foi repetida e esta ainda foi a principal reclamação entre os usuários com cerca de 60% do total. Portanto, um dos grandes potenciais desta nova abordagem de grades computacionais *ad hoc* está no fato de que ela elimina a necessidade de configuração inicial e re-configuração por parte de um administrador. Essa característica possibilita a redução das falhas na configuração do ambiente, bem como reduz o custo do gerenciamento de grades.

Considerando as diferentes abordagens de grades computacionais, este trabalho tem como foco, fazer uma breve apresentação do paradigma tradicional de grades e apresentar os novos paradigmas, enfatizando as grades *ad hoc*, suas principais definições e características segundo diferentes autores, destacando aquelas que serão utilizadas na solução desenvolvida nesta dissertação (o *ad hoc* Grid). Será então apresentada a arquitetura do *ad hoc* Grid baseada nas características de uma grade *ad hoc*. Em seguida, o *ad hoc* Grid será implementado e validado e, por fim, será feita a avaliação do impacto da solução no desempenho da grade.

1.2 Objetivos do Trabalho

Com base nas discussões conduzidas anteriormente, esta dissertação tem por objetivo principal o desenvolvimento e a avaliação de uma solução auto-organizável (i.e., *ad hoc*) de grade entre pares baseado no *middleware* do OurGrid (OG) [21].

Com isso, pretende-se oferecer uma versão do OurGrid auto-organizável e descentralizado, facilitando sua instalação e manutenção. Pretende-se ainda que essa abordagem proporcione à grade uma maior autonomia, visto que ela pode ser reestruturada automaticamente em caso de falhas.

1.3 Relevância

"*Grades emergentes podem ajudar a diminuir a distância entre as tecnologias de grade e os usuários*" [41].

Com a abordagem dada neste trabalho, buscou-se prover uma solução que possibilite a construção de uma grade computacional Peer-to-Peer (P2P) auto-organizável. Tendo em

vista que o OG provê os elementos básicos para a execução de tarefas em uma grade computacional, este trabalho dedicou-se apenas aos aspectos referentes à infra-estrutura da grade, ou seja, na forma como seus componentes são organizados e mantidos. Para torná-la auto-organizável, o serviço de descoberta de recursos (i.e. `CorePeer`) foi substituído por um grupo *multicast* entre-pares. Assim, quando os pares da grade (`Peers`) entram na comunidade eles se associam ao grupo *multicast* entre-pares e, a partir deste grupo, eles obtêm informações sobre os outros pares ativos que fazem parte da comunidade. Sabendo-se que no OurGrid cada `Peer` é responsável pelos `Workers`[1] de um domínio administrativo, o `Peer` passará a gerenciar seus `Workers` através de um grupo *multicast* local. Portanto, a infraestrutura da grade terá um grupo *multicast* local para formar um *site* e um grupo *multicast* entre-pares para formar toda a comunidade da grade.

Desta forma, as principais contribuições deste trabalho são a arquitetura proposta para a auto-organização da grade baseada em grupos *multicast* e a implementação desta arquitetura baseada no *middleware* do OG. Outra contribuição é a avaliação do *overhead* gerado pela auto-organização do sistema. Esta última contribuição deve ser destacada, pois em outros trabalhos onde foram desenvolvidas grades computacionais auto-organizáveis [19; 5], a característica de auto-organização está intrínseca a arquitetura do sistema, e portanto, se torna difícil fazer uma avaliação do custo adicional para manter a auto-organização. Na solução apresentada neste trabalho, a auto-organização é uma característica adicionada ao OurGrid.

1.4 Estrutura da Dissertação

Além da Introdução, o presente trabalho está estruturado da seguinte forma:

No Capítulo 2, é apresentada uma revisão dos principais paradigmas de grades computacionais, dando ênfase às grades *ad hoc*.

No Capítulo 3 é abordada a estrutura e o funcionamento do OurGrid em suas duas versões, a 3.3 e a 4.1.

No Capítulo 4 é apresentada a arquitetura para o *ad hoc* Grid e em seguida são apresentados os detalhes de implementação.

[1] Membros da comunidade que executam as tarefas

No Capítulo 5 são apresentados e discutidos os resultados acerca dos experimentos realizados com o *ad hoc* Grid.

No Capítulo 6 são apresentados os trabalhos relacionados.

Por fim, no Capítulo 7 são feitas as considerações finais, além de apontar as dificuldades encontradas no desenvolvimento desse trabalho e sugestões para trabalhos futuros.

Capítulo 2

Classificação de Grades Computacionais

Este capítulo apresenta uma revisão sobre os principais paradigmas de grades computacionais desde a sua concepção, com a abordagem tradicional, até as abordagens mais atuais (denominadas grades emergentes), dando destaque às grades *ad hoc*, que serão utilizadas como motivação para o desenvolvimento deste trabalho.

2.1 Grades Tradicionais

O conceito de grade computacional foi criado em meados da década de 90, para definir uma infra-estrutura de computação distribuída proposta para o avanço da ciência e das engenharias [28]. Esse conceito, juntamente com suas tecnologias, foram projetadas para possibilitar o compartilhamento de recursos entre as diversas comunidades científicas, permitindo a elas aumentar seu poder computacional. Embora esta idéia seja simples, o grande desafio está na forma com que o compartilhamento dos recursos serão coordenados diante das diferentes modalidades de uso e das diferentes comunidades. Essas comunidades são denominadas Organizações Virtuais (OV) [36]. Mais especificamente, uma OV é um conjunto dinâmico de indivíduos e/ou instituições definido por um conjunto de regras de compartilhamento de recursos.

Com evolução das tecnologias de grade, elas deixaram de ser utilizadas apenas pela comunidade científica, e passaram a ser utilizadas também em atividades empresariais como, por exemplo, na integração de aplicações empresariais e em colaborações *business-to-business* pela Internet [36]. Podemos comparar essa transição da utilização das grades com-

putacionais da comunidade científica para área comercial, com a Internet, que deixou de ser apenas uma estrutura acadêmica e passou a ser utilizada pelas empresas e pela população em geral. Diferente da Internet, que hoje está bem estabelecida e padronizada, existem ainda diversas arquiteturas de grades que atendem a diferentes requisitos (soluções) e diferentes escalas (tamanho). Essas diferenças de requisito e escala são duas características que na literatura são utilizadas como parâmetro para categorizar as grades tradicionais. Segundo Hariri *et al.* [48] do ponto de vista de escala, uma grade pode ser dividida em quatro categorias:

- Grades de produção Nacional: São grades que unem recursos de rede, dados e processamento, através de uma nação a fim de prover uma infraestrutura de computação distribuída unificada. Como exemplos dessas grades podemos citar o D-Grid [26] e o China-Grid [1];

- Grades de produção Comunitária: Tem a estrutura semelhante às grades nacionais, mas seus recursos estão espalhados entre diferentes domínios administrativos e diferentes domínios geográficos [4];

- Grades de produção Empresarial: Nesta categoria de grades computacionais os recursos disponíveis fazem parte da uma empresa/instituição. Geralmente seu poder computacional é de baixo custo, composto pelos *desktops* pessoais da empresa [13];

- Grades de produção Voluntária: Neste tipo de grade os usuários podem colaborar doando os recursos de suas máquina através da Internet[44; 46];

Em um trabalho mais recente, Kurdi *et al.* [41] identificam mais duas categorias:

- Intra-Grade ou *Campus Grid*: São grades onde seus recursos são restritos a determinados usuários de uma organização.

- Grades Pessoais: São grades computacionais que tem o escopo mais reduzido dentre as grades computacionais. Esse tipo de grade possibilita que usuários individuais possam controlar e personalizar o seu ambiente [55].

Segundo o ponto de vista de requisitos, as grades podem ser divididas em:

- Grades computacionais: Este tipo de grade fornece como principal recurso o processamento (ciclos da CPU). Dependendo do tipo de *hardware* esse tipo de grade ainda pode ser subdividido em:
  - Desktop: Constituído pelos recursos de diferentes *desktops*;
  - Servidor: Formado por recursos disponíveis em servidores;
  - Equipamentos: Formado por recursos adquiridos de equipamentos específicos como, por exemplo, equipamentos eletrônicos.
- Grades de dados: São utilizadas para o armazenamento, recuperação e sincronização de grande volume de dados, através dos recursos de armazenamento disponibilizados pela grade.
- Grades de serviço ou grades utilitárias: Disponibilizam recursos de armazenamento e processamento que podem ser comprados sob demanda.
- Grades de acesso: Constituídas por recursos distribuídos de entrada e saída. Esse tipo de grade foi projetado com o intuito de disponibilizar um ambiente de rede, computação e com recursos de interação, possibilitando a interação humana entre os diferentes grupos de usuários que a utilizam.

Essa classificação de grades tradicionais segundo Kurdi *et al.* [41] desenvolveu-se em algumas fases distintas ao longo do tempo. Ela é utilizada para se referir a grades de primeira, segunda e terceira geração. A primeira geração teve início no começo da década de 90 como um modelo de metacomputação, onde os recursos de supercomputadores e os dados dos usuários podiam ser compartilhados. A segunda geração foi marcada pelo uso de *middlewares* computacionais com objetivo de unir diferentes tecnologias de grade [28]. A terceira geração surge com a união das tecnologias da Web e de grade.

Na Seção a seguir serão apresentados alguns modelos de grades atuais, que alguns autores denominam **grades emergentes**.

2.2 Grades Emergentes

Muitos trabalhos na área de grades computacionais têm apresentado novas abordagens [23; 9; 55]. Apesar de ainda não se encontrar na literatura uma classificação bem definida para essas novas abordagens de grade, Kurdi *et al.* [41] propõem uma classificação para elas. Os autores definem essas novas abordagens como grades emergentes nas quais as grades possuem características pervasivas e auto-organizáveis [33]. Dessa forma, eles propõem a divisão das grades emergentes em quatro grupos (vide Figura 2.1):

- Acessibilidade;
- Centrado ao usuário;
- Interatividade;
- Gerenciamento.

Cada um desses grupos é dividido em subgrupos de acordo com suas principais diferenças entre as grades tradicionais. Por exemplo, o grupo de acessibilidade é subdividido em grade *ad hoc*, móvel e sem fio. Nas grades *ad hoc*, sua principal diferença entre uma grade tradicional é que ela não possui pontos de entrada predefinidos. As grades móveis diferem das tradicionais por possibilitarem a mobilidade dos clientes e/ou dos serviços. Já as grades sem fio suportam conexões sem fio entre os componentes de uma grade. A Tabela 2.1 apresenta todas as subdivisões e suas principais diferenças das grades tradicionais. .

A seção a seguir apresenta as principais características de uma grade *ad hoc*, que serão utilizadas como base para a elaboração da arquitetura e para implementação do *ad hoc* Grid.

2.3 Grades *Ad hoc*

Conforme foi apresentado na Seção anterior, as grades tradicionais são divididas em diferentes categorias de acordo com os requisitos a que ela se propõe e ao seu tamanho. Apesar dessas diferenças que as fazem distinguir em categorias, elas compartilham algumas características em comum. Um exemplo disto é que, em todas as comunidades, seus usuários têm um objetivo em comum. Seja nas grades voluntárias, onde os usuários decidem doar de

Grupo	**Subgrupo**	**Diferenças entre uma grade tradicional**
Acessibilidade	Ad hoc	Não tem pontos de entrada definidos
	Móvel	Suportam mobilidade dos clientes e/ou dos serviços
	Sem fio	Suportam conexões sem fio
Interatividade	Com interatividade	Suportam interação em tempo real entre os usuários
	Contexto-cientes	Suportam interação com os vizinhos para adaptar seu comportamento
Centrado ao usuário	Pessoais	De propriedade particular
	Personalizadas	Possui portais customizados
Gerenciamento	Autonômicas	Utiliza idéias do funcionamento do corpo humano para o auto-gerenciamento
	Inteligentes	Utiliza tecnologias de conhecimento para auto-gerenciamento
	Orgânicas	Utiliza idéia de colônia para auto-gerenciamento

Tabela 2.1: Classificação das grades emergentes (Adaptada de Kurdi [41]).

forma altruísta seus recursos computacionais, ou nas grades empresariais, onde os usuários executam aplicações específicas da instituição da qual fazem parte.

Outra característica em comum entre as grades tradicionais é que todas têm um ponto central responsável por coordenar o acesso à grade, a permissão dos usuários e a manutenção dos recursos disponíveis na grade. Todas essas características restringem a utilização da grade a uma comunidade fixa com uma estrutura bem definida e uma entidade administrativa centralizada que é geralmente composta por um grupo de profissionais treinados para coordenar as permissões de acesso em grade. Esta atividade de administração também inclui as atividades de monitoramento e manutenção dos recursos da grade.

Dada essas características semelhantes das grades tradicionais, é perceptível que elas não atendem bem a certos tipos de comunidade que necessitam de uma grade provisória e

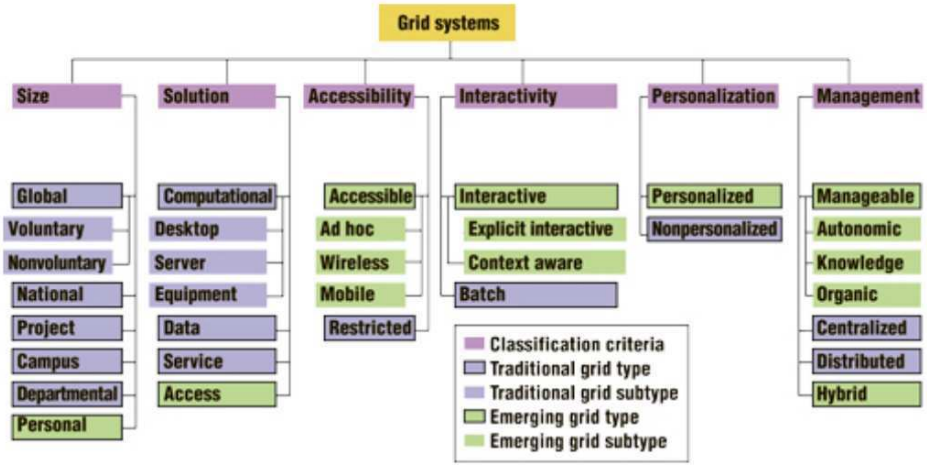

Figura 2.1: Classificação de grades tradicionais e emergentes (Kurdi [41]).

por um período curto de utilização. Como exemplo típico deste tipo de comunidade, pode-se citar dois laboratórios de pesquisa que necessitam simular experimentos de uma determinada aplicação e compartilhar os resultados entre eles. Para esta comunidade, por exemplo, seria bastante custoso, estabelecer e manter uma grade computacional com todas as regras de uma OV. O *overhead* neste caso pode, de fato, ser até maior que os benefícios que a grade proporcionaria. Dessa forma, a proposta de uma grade *ad hoc* é justamente possibilitar a construção de uma infra-estrutura de compartilhamento de recursos provisória e dinâmica.

A Figura 2.2 apresenta um exemplo de arquitetura de duas grades *ad hoc*. A grade *ad hoc* A (Ad hoc Grid A) é composta por máquinas de duas organizações diferentes. Fazem parte desta grade máquinas dedicadas de alto desempenho e também máquinas não dedicadas e provisórias. A grade *ad hoc* B (Ad hoc Grid B) é composta apenas por máquinas não dedicadas e provisórias, assemelhando-se às grades pessoais. Já a grade *ad hoc* A se assemelha bastante ao exemplo citado anteriormente onde dois laboratórios de pesquisa desejam compartilhar seus recursos em um determinado momento.

Portanto, alguns autores [9] definem que uma grade *ad hoc* deve ser capaz de formar uma comunidade, independente da sua estrutura, independente de controle e da tecnologia adotada. Estas três características são destacadas a seguir:

- Independência de controle: Possibilita que a grade tenha suas políticas de segurança e de acesso sem um ponto central de coordenação. Para tornar isso possível, cada

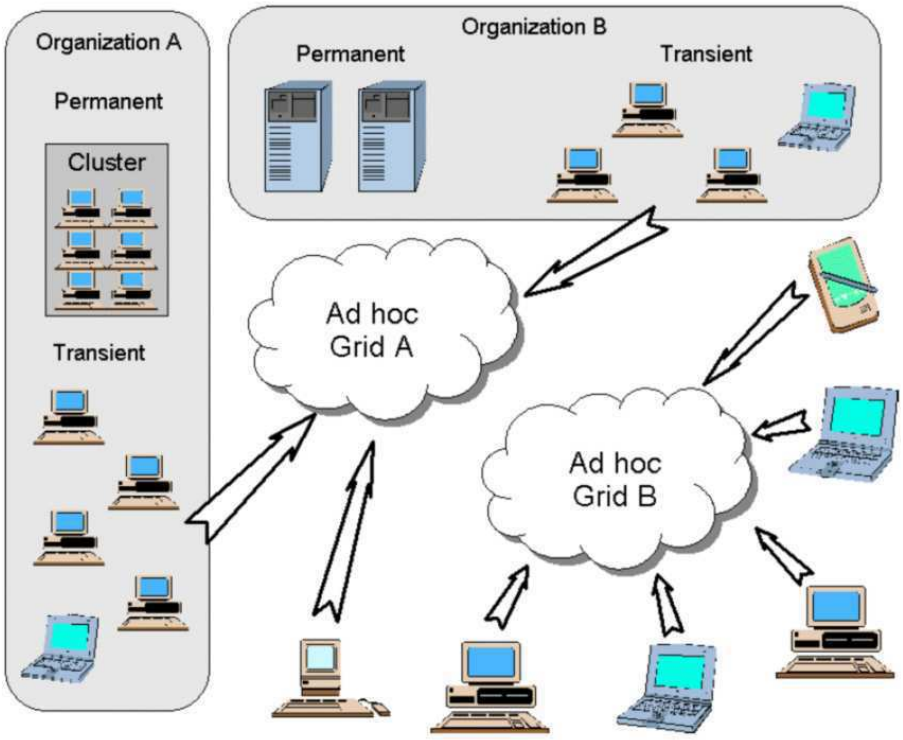

Figura 2.2: Arquitetura de uma grade *ad hoc* (Smith [51]).

componente da grade deve ser responsável por manter essas políticas sobre os seus recursos;

- Independência de estrutura: Permite que a grade se auto organize sem que exista um ponto central para o descobrimento de recursos, ou seja, a grade não tem nenhum ponto de entrada bem definido. Qualquer componente da grade pode ser um ponto de entrada. Esta é uma característica importante, pois exclui a existência de um ponto central de falha na grade;

- Independência de tecnologia: Esta característica permite a grade suportar diferentes tecnologias de grades e diferentes protocolos.

Essas características se tornam mais evidentes quando apresentadas através de requisitos que o ambiente de grade deve ter para suportar uma grade *ad hoc*. Dentre esses requisitos estão o serviço de implantação automático, o descobrimento de recursos, a tolerância a falhas e a possibilidade de migração de serviço [29]. A seguir, serão descritos cada um dos requisitos mais detalhadamente.

2.3.1 Serviço de Implantação Automática

Nas grades tradicionais, a etapa de implantação dos serviços, bem como as atividades de monitoramento e definição das políticas de permissões da grade são de responsabilidade do administrador. A implantação dos serviços nessas grades é feita com a instalação destes em cada máquina da grade. Esta implantação geralmente exige a edição de arquivos de configuração e permissão de acesso. Por causa da estrutura dinâmica de uma grade *ad hoc*, ela necessita de uma forma diferente de implantação de seus serviços, que exija uma menor ou nenhuma sobrecarga administrativa [29].

2.3.2 Tolerância a Falhas

Uma grade *ad hoc* possui, por natureza, máquinas provisórias e por este motivo não há como garantir por quanto tempo uma máquina ficará disponível na grade. Além disso, essas máquinas estão susceptíveis a falhas. Assim, uma grade *ad hoc* deve implantar mecanismos de tolerância a falhas a fim de tratar a perda de conexão de uma máquina ou uma falha desta, proporcionando transparência na aplicação da grade.

2.3.3 Migração de Serviço

Como foi apresentado na Seção anterior, na estrutura das grades tradicionais, o serviço de descoberta de recursos fica centrado em uma única máquina da grade.

Nas grades *ad hoc*, como sua arquitura deve permitir que qualquer máquina colabore de forma provisória (ou seja, ela pode juntar-se ou deixar a grade a qualquer momento) uma aplicação da grade não pode supor que o serviço de descoberta de recursos estará sempre disponível.

Uma solução para este problema de indisponibilidade é fazer com que algumas máquinas da grade garantam a disponibilidade de seus recursos por um certo tempo [10]. Mas esta solução se torna inviável quando todas as máquinas da grade *ad hoc* são transientes. Logo, quando não há como garantir a disponibilidade de um serviço, deve existir um mecanismo de migração de serviço de uma máquina que deixou a grade, para outra que está disponível.

2.3.4 Descobrimento de Recursos

O descobrimento de recursos numa grade *ad hoc* é um dos requisitos mais importantes, porque ela possui uma estrutura que é modificada dinamicamente. Neste tipo de grade o descobrimento de recursos deve ser otimizado a fim de que possa detectar as mudanças frequentes na estrutura da grade. Desta maneira, quando uma nova máquina junta-se à grade, seus recursos compartilhados devem ser descobertos por alguma entidade da grade.

Apesar de alguns trabalhos classificarem o OurGrid como uma grade *ad hoc* [41], sua arquitetura ainda não provê os serviços de implantação automática, de migração de serviço e alguns aspectos no descobrimento de recursos. Isto porque, sua configuração inicial é feita manualmente pelos administradores da grade, bem como a migração de seus serviços. Com relação ao descobrimento de recursos, o OurGrid possui um serviço centralizado (`CorePeer`), o que dificulta a dinâmica de uma grade *ad hoc*. O próximo Capítulo detalha o funcionamento do OurGrid e apresenta suas principais características.

Capítulo 3

OurGrid

Neste Capítulo é apresentada a arquitetura do OurGrid e detalhado o funcionamento de cada um de seus componentes. Na Seção 3.1 são apresentados os detalhes referentes à versão 3.3 e na Seção 3.2 são abordadas as modificações incorporadas na versão 4.1.

3.1 OurGrid 3.3

OurGrid é uma grade computacional aberta, *free-to-join* e cooperativa, onde laboratórios de pesquisa podem doar seus recursos computacionais ociosos e, em troca, obter recursos ociosos de outros laboratórios [21]. Esses recursos podem ser utilizados pelos usuários para a execução de aplicações paralelas *Bag-of-Tasks*, ou seja, aplicações em que as tarefas são independentes entre si.

A arquitetura do OG na sua versão 3.3 é composta por quatro componentes principais: `Peer`, `CorePeer`, `MyGrid` e `Worker` [1]. O `MyGrid` é a interface do OurGrid, através da qual os usuários interagem com o sistema. Ele implementa um escalonador das aplicações[2] e monitora a sua execução. O `Peer` é responsável pela gerência dos recursos de um *site*[3] local. O `UserAgent` é o componente responsável pela execução das tarefas submetidas pelo `MyGrid`. E o `CorePeer`, é o componente utilizado para a formação da grade computacional. Ele monitora todos os `Peers` ativos das comunidades e possibilita a troca de informações entre estes.

[1] Também denominado UserAgent.

[2] No OurGrid, aplicações são representadas por um conjunto de tarefas.

[3] Instância do OurGrid em uma determinada instituição.

O OG utiliza a rede de favores (Network of Favours - NoF), um mecanismo *peer-to-peer* de reputação para incentivar os laboratórios a doarem seus recursos ociosos [12]. Esse mecanismo permite que um `Peer` com recursos ociosos escolha para quem ele deve doá-los, dentre um conjunto de `Peers` que requisitam seus recursos. Esta escolha é feita a partir de um cálculo de reputação que envolve o histórico da troca de favores entre os `Peers`. A NoF é um mecanismo simples e autônomo, pois é calculado e mantido apenas pelo `Peer` de cada site.

A Figura 3.1 a seguir apresenta a arquitetura do OG através de um exemplo de uma comunidade OG estabelecida. Neste exemplo, a comunidade do OG é formada por três *sites* distintos: *lncc.br*, *lsd.ufcg.edu.br* e *foo.bar*. Cada site encontra-se em um domínio administrativo diferente e são unidos na comunidade OG através de seus `Peers`. Os sites são compostos também pelo `UserAgent`, responsáveis por executar as tarefas. Eles podem habilitar o módulo Sandboxing A Name (SWAN) [18], que provê um ambiente de execução baseado na tecnologia de máquina virtual Xen [14]. O SWAN isola as tarefas que vem de outro site em uma *sandbox*, protegendo os recursos da máquina que executa essas tarefas. Além do `UserAgent`, dois *sites* são compostos pelo `MyGrid`, através do qual os usuários submetem suas tarefas.

3.2 OurGrid 4.1

A versão 3.3 do OurGrid utiliza o modelo de comunicação Remote Method Invocation (RMI) [24], que possibilita a manipulação dos objetos remotos de forma transparente ao programador. O grande problema deste modelo, no contexto do OG é que se faz necessário programar utilizando *multi-threaded*, o que tem apresentado alguns problemas na programação concorrente [8].

Portanto, a principal modificação no OG 4.1 foi a substituição desse modelo de comunicação pelo Extensible Messaging and Presence Protocol (XMPP) [47]. Esse modelo utiliza uma arquitetura baseada em troca de mensagens através de um servidor XMPP. Desta forma, os componentes do OG se comunicam através de mensagens XMPP assíncronas, que são enfileiradas no receptor e consumidas por uma única *thread*.

Devido ao caráter heterogêneo dos sites do OG, foi desenvolvido um portifólio de segu-

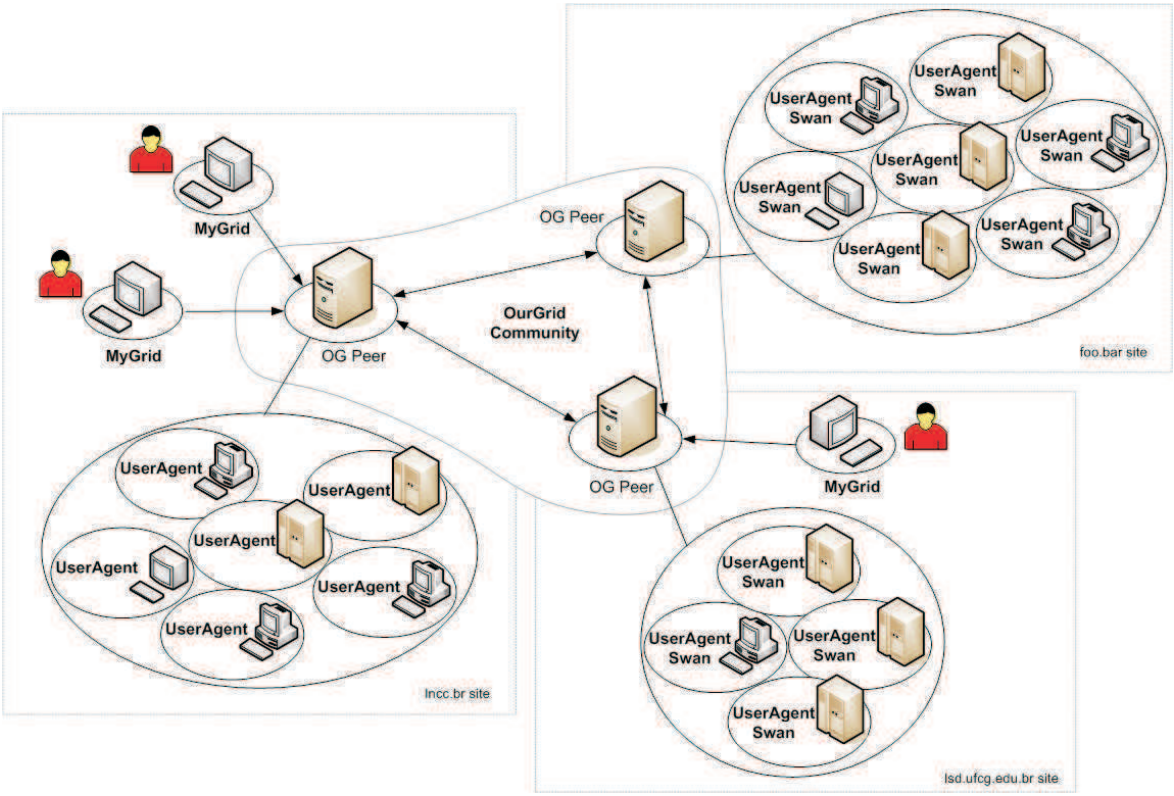

Figura 3.1: Arquitetura do OurGrid (Cirne *et al.* [21]).

rança que possibilita o administrador de um site escolher os mecanismos de segurança mais adequados às suas necessidades.

Outras modificações foram realizadas na versão 4.1 do OurGrid [45], entretando estas foram as modificações consideradas de maior relevância para a escolha da versão utilizada no trabalho. Além disso, a versão 4.1 do OG ainda estava sendo desenvolvida quando se iniciou a implementação do *ad hoc* Grid.

Capítulo 4

Solução Proposta

Muitos trabalhos foram desenvolvidos na área de redes *ad hoc*, focando em uma infra-estrutura de comunicação móvel e sem fio [50]. Na abordagem de grades computacionais *ad hoc*, o foco principal passa a ser nas características esporádicas, mutáveis e auto-adaptativas de sua estrutura e de seu controle, ao invés da mobilidade. Conforme apresentado no Capítulo 2, três características importantes de uma grade *ad hoc* devem ser destacadas: a **independência de tecnologia**, a **independência de controle** e a **independência estrutural** [9].

A independência de tecnologia faz com que a grade suporte diferentes tipos de protocolos e tecnologias de grades. A independência de controle retrata a capacidade da grade de suportar funções administrativas sem um ponto central de coordenação. Por fim, a independência estrutural implica na capacidade de auto-organização da grade. A independência de controle e a independência estrutural serão o foco maior da versão *ad hoc* do OurGrid.

O modelo de comunicação utilizado para a organização da grade será baseado em grupos *multicast*. A descrição detalhada da arquitetura do *ad hoc* Grid será apresentada na Seção a seguir.

4.1 Arquitetura

A arquitetura do *ad hoc* Grid é formada pelos três componentes principais do OurGrid: `Peer`, `MyGrid` e `Worker`, como apresentados na Figura 3.1 do Capítulo 3. Com exceção do Discovery Service (ou `CorePeer`) a estrutura do OurGrid será mantida, ou seja, os componentes da grade no *ad hoc* Grid continuam se comunicando através de RMI, para

submissão e execução das tarefas, mudando apenas a forma como os componentes serão criados e organizados durante sua existência. No decorrer desta Seção serão apresentadas as funcionalidades que foram adicionadas à versão 3.3 do OurGrid para que ele se tornasse auto-organizável.

Embora a versão atual do OurGrid em produção esteja na 4.1 [45], no início da elaboração deste trabalho, ela ainda estava sendo implementada. Por este motivo, foi escolhida a sua última versão mais estável, a 3.3. Portanto, sempre que for feita alguma referência ao OurGrid apenas, deve ser considerado que o texto se refere à versão 3.3.

Na arquitetura do OG, a configuração de cada componente é feita manualmente, pelo administrador do *site*. Para a instalação do Discovery Service, deve ser definido pelo administrador um endereço e uma porta (RMI) que será utilizada como ponto de encontro entre os `Peers` da comunidade. Desta forma, todo aquele que quiser fazer parte da comunidade do OG (através de seu *site*) deve configurar o `Peer` a fim de que ele conheça o endereço e porta do Discovery Service (DS). Nesta etapa, já é possível identificar um componente que é uma entidade centralizada na comunidade e que está sujeito a falhas, portanto, se o Discovery Service falhar, toda a comunidade será comprometida. Assim, todos os *sites* da comunidade são compostos por um `Peer` que também deve ser instalado por um administrador em seu domínio administrativo local.

Os componentes da grade que efetivamente executam as tarefas (`Workers`) devem também ser instalados em cada máquina da grade. Após a instalação do `Worker` nas máquinas que irão executar as tarefas, elas devem ser associadas ao `Peer` de seu respectivo *site*, através de um arquivo de configuração (SDF - arquivo de descrição do *site*), com o endereço, nome, porta e configuração da máquina (memória, processador, sistema operacional, etc) (vide código 4.1).

Código Fonte 4.1: Exemplo do arquivo de configuração do um *site*

```
1 #
2 # Peer Description File Example
3 #
4 gumdefaults:
5         site : lsd.ufcg.edu.br
6         type : ualinux
7         os : linux
```

```
8           processorfamily : IA32
9           mem : 640
10          remExec : ssh -x $machine $command
11          copyFrom : scp $machine:$remotefile $localfile
12          copyTo : scp $localfile $machine:$remotefile
13          port : 3080
14
15  gum:
16          name: machine1
17          mem : 256
18
19  gum:
20          name: machine2
21          mem : 256
```

Por fim, cada usuário que desejar executar suas tarefas na grade deve instalar o `MyGrid` e indicar quais *sites* da comunidade (representados pelo endereço e porta do `Peer`) ele deseja interagir. Da mesma forma, o usuário deve conhecer quais `Peers` estão ativos, adicionando-os em seu arquivo de configuração semelhante ao SDF (vide código 4.2).

Código Fonte 4.2: Exemplo do arquivo de configuração do uma comunidade

```
1  #
2  # Grid Description File Example
3  #
4  peer:
5           label: LSD Public Peer
6           name : peer.lsd.ufcg.edu.br
7           port : 3081
8
9  peer:
10          label: Foo Public Peer
11          name : peer.foo.bar
12          port : 3081
13
14 peer:
15          label: LNCC Public Peer
16          name : peer.lncc.br
```

```
17          port : 3081
```

Como se pode observar no arquivo de configuração 4.1, toda vez que for necessário retirar ou colocar outro `Worker` no *site*, é indispensável que o administrador modifique seu arquivo de configuração, incluindo ou retirando o `Worker`, e configure novamente o `Peer` com esse novo arquivo. Assim, deve ser configurado o `MyGrid`, acrescentado ou retirado o `Peer` que o usuário for utilizar. Na arquitetura do *ad hoc* Grid, um dos objetivos é tornar estes processos de instalação e configuração automáticos e garantir o restabelecimento da grade, caso algum desses componentes falhem.

Além disso, para tornar o sistema totalmente descentralizado, os principais componentes do OG foram integrados em uma única aplicação. Isto é, o usuário ainda interage com o `MyGrid`, mas agora, o `Peer` e o `Worker` podem estar ativos na mesma máquina.

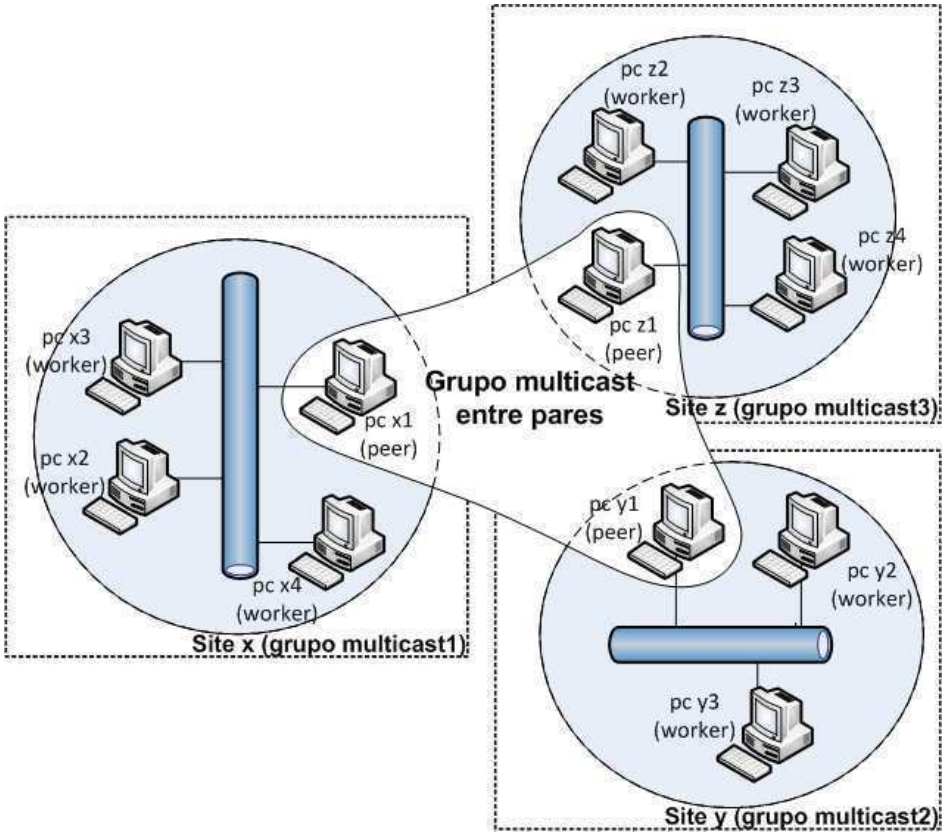

Figura 4.1: Exemplo de uma comunidade no *ad hoc* Grid

A partir do modelo de comunicação adotado, via grupos *multicast*, a grade será organizada da forma explicitada na Figura 4.1. Cada *site* da grade será formado por um grupo

multicast local, e dependendo de quando a aplicação for iniciada pelo usuário, ela deve assumir a função de `Peer` (o Algoritmo 1 apresenta o pseudo-código para o procedimento de inicialização da grade). Sendo assim, a primeira instância da aplicação que for executada na rede local tornar-se-á `Peer`. Para saber se já existe um `Peer` local, o nó associa-se a um grupo *multicast* local pré definido[1] (linha 2 do Algoritmo 1) usado para a comunicação com o `Peer` local. Neste trabalho, foram abordadas duas soluções para detectar a existência de um `Peer` local (o pseudo código no Algoritmo 1 mostra apenas um desses casos):

- **Método ativo**: Após associar-se ao grupo *multicast* local, a máquina envia uma mensagem para este grupo, solicitando confirmação do `Peer`. Caso não receba nenhuma confirmação após n tentativas, com um tempo de espera t_{ack} entre cada tentativa, ele se auto-elege o `Peer` desse *site*. Se já houver um `Peer`, ele será apenas `Worker`.

- **Método passivo**: Neste método, o `Peer` se encarrega de fazer o anúncio de sua existência. Esse anúncio será feito periodicamente, a cada t_{alive} segundos. Desta forma, quando uma máquina se associar a um grupo *multicast* local, ela aguarda $n * t_{alive}$ segundos, para receber o anúncio do `Peer`. Caso não receba esse anúncio, ele se auto-elege `Peer`. Caso contrário ele será apenas `Worker`.

Embora essas duas abordagens resolvam o mesmo problema (detectar a existência de um `Peer` no *site*), os tempos de convergência variam. No método ativo, o tempo de descoberta no *melhor caso*[2] é o tempo de *Round-trip time* (RTT) e, no *pior caso*[3], o tempo é $n * t_{ack}$. Já no método passivo, o tempo de descoberta de um `Peer` no *melhor caso* é próximo de zero. No *pior caso* o tempo de descoberta do `Peer` é de $n * t_{alive}$.

Nesta avaliação, o *melhor caso* em ambos os métodos tem em geral a mesma ordem de grandeza e, portanto, a diferença não é significativa. No entanto, no *pior caso* verifica-se uma grande disparidade entre os tempos de descoberta, pois geralmente $t_{ack} \ll t_{alive}$. A solução adotada neste trabalho foi uma combinação desses dois métodos. Assim, quando um nó inicia o *ad hoc* Grid, ele envia uma mensagem para o grupo local requisitando uma resposta do `Peer`, ao mesmo tempo em que aguarda uma mensagem *alive* do `Peer`.

[1] O endereço do grupo *multicast* local (LMG é derivado do endereço de rede local

[2] quando já existe um Peer no *site* e nenhuma mensagem é perdida

[3] quando não há um Peer na rede

	i	Endereço IP do Membro
Data:	*peer*	Peer Local
	LMG	Grupo Multicast Local

```
1  begin
       /* Associa-se ao grupo multicast local                          */
2      JoinGroup(LMG)
       /* Verifica a existência de algum Peer:  método ativo!          */
3      for j=1 to n do
           /* Consulta o grupo, e espera t_ack s                       */
4          peer = SendMessage(LMG,QueryPeer)
5          if peer != null then
6              break
7      if peer==null then
           /* Nenhum Peer no grupo local.  Este nó deve assumir o
              papel de Peer.                                           */
8          peer = i
           /* Inicia a thread Peer!                                    */
9          Peer.start()
           /* Anuncia o status do Peer no LMG                          */
10         SendMessage(LMG, PeerAnnouncement(i))
11     else
           /* Inicia a thread que verifica a existência de um Peer local! */
12         CheckPeer.start()
       /* Existe um Peer.Pronto para iniciar um Worker.                */
13     Worker.start()
       /* Informa para o Peer local sua existência, mandando todas
          as informações necessárias a respeito deste nó.              */
14     SendPeer(peer, i.info)
15 end
```

Algoritmo 1: Start-Up.

É importante destacar que esses mecanismos de descoberta do `Peer` podem não detectar um `Peer` apesar de ele existir. Isto pode ocorrer quando o `Peer` estiver sobrecarregado e demorar a confirmar sua existência (extrapolando o t_{ack}). Quando isso ocorrer, o *site* ficará com múltiplos `Peers` ativos. Embora inconsistente, este é um estado transiente do sistema, e é resolvido de forma simples com a escolha do `Peer` que tiver o menor IP (vide Algoritmo 2). Após esta escolha, o `Peer` anuncia no grupo local que ele é de fato o `Peer` deste *site*.

```
          i          Endereço IP do Membro
Data:     LMG        Grupo Multicast Local
          peer       ID do peer Local
1 begin
2   while true do
3     ScheduleTimer(Peer-Report, n × t_rep)
4     if não receber nenhum anuncio de Peer /* Inicia a thread
          PeerFailureDetector                                  */
5     else
6       if tiver mais de um peer AND i maior que algum membro do grupo local then
            /* Neste caso há dois Peers no mesmo grupo local.
               Para o peer local!                              */
7           Peer.stop()
8     then
9 end
```

Algoritmo 2: CheckPeer *thread.*

Ainda que pouco provável, múltiplas máquinas podem se associar a um grupo *multicast* local simultaneamente. Neste caso, se o grupo não tiver nenhum `Peer`, essas máquinas serão eleitas como `Peers`. Este estado transiente do sistema é resolvido da mesma forma que o estado citado anteriormente (vide Algoritmo 2).

Uma máquina que não for definida como `Peer` terá que comunicar sua existência ao `Peer` local (linha 15, Algoritmo 1). Sendo assim, o `Peer` local mantém uma lista com o endereço de todas as outras máquinas (`Worker`) do grupo local.

Considerando que os `Peers` são entidades que podem falhar (vide Algoritmo 3 que representa o pseudo-código do `Peer`), sempre que sua lista de `Workers` for atualizada,

ele a envia para o grupo *multicast* local (LMG). Assim, todos os membros do LMG quando receberem essa mensagem devem guardar esta lista de `Workers` localmente. Esta lista servirá para que, quando um membro identificar que o `Peer` de seu *site* falhou, ele deve iniciar o seu serviço como `Peer`, já com a lista atualizada de todos os `Workers` do *site*. Desta mesma forma, a contabilidade da rede de favores (NoF) é mantida. Ou seja, sempre que a NoF é atualizada, ela é enviada para os membros do LMG. Então todos os membros persistem o valor da NoF mais atual, e se futuramente ele for um `Peer`, ele iniciará este serviço com a NoF mais atual.

Esta abordagem para a manutenção da NoF é válida enquanto existir algum componente da grade em um site local. Isto porque, o site sempre terá um `Peer` que mantém a contabilidade da NoF mais recente. Caso um site, através do `Peer` doe algum recurso para outro site (i.e. inicie a contabilidade da NoF) e posteriormente deixe de existir (todos os nós falharem), a contabilidade da NoF deste site será persistida apenas nos nós que faziam parte do site naquele momento. Se logo em seguida um novo nó iniciar sua atividade neste site, a contabilidade da rede de favores será nula, embora o site já tenha trocado favores com outro site.

Para resolver este problema, a abordagem adotada neste trabalho mantém a NoF distribuída entre os `Peers` dos outros *sites*, ou seja, se um *site* deixar de existir, sua NoF será mantida em outro site até que este volte a funcionar.

Embora seja pouco provável, toda a grade pode deixar de existir em algum instante, e portanto, a NoF pode se tornar inconsistente. Para estes casos, pode ser utilizado uma solução onde são definidas épocas diferentes para cada NoF persistida. Desta forma, seria possível identificar em que época a NoF recuperada foi persistida, e caso um nó entre na rede com uma NoF de época mais atual, essa NoF passa a ser a utilizada. Esta solução não foi aplicada à implementação atual do *ad hoc* Grid, mas segue como referência para trabalhos futuros.

`Peers` de diferentes *sites* interagem entre si através do grupo *multicast* P2P (P2PMG) (linhas 2 e 3 do Algoritmo 3). Desta forma, ao invés de consultar um servidor central (e.g., `Core Peer`), através do grupo *multicast* P2P, o `Peer` obtém o acesso a outros *sites* da grade.

Quando um usuário submete uma tarefa para a grade (*i.e.*, através do `MyGrid`), a apli-

	i	Endereço IP do Membro
Data:	*LMG*	Grupo Multicast Local
	P2P-MG	Grupo Multicast P2P

begin

`/* Associa-se ao grupo multicast P2P */`

JoinGroup(P2P-MG)

`/* Envia o status do Peer para todos os Peers via P2P-MG */`

SendMessage(P2P-MG, *peer.info)*

`/* Sempre que a lista de workers ou a NoF for atualizada, envia para o grupo local uma mensagem (i.e., a contabilidade da rede de favores e a lista de workers ativos)! */`

Schedule(LMG, PeerReport, t_{rep}*)*

`/* Inicia a` *thread para enviar e receber informações de/para outros Peers!* `*/`

Peer.start(Thread.P2P-MG)

`/* Inicia a` *thread que responde aos membros locais que este Peer está ativo* `*/`

Peer.start(Thread.LMG)

end

Algoritmo 3: Peer *thread.*

cação solicita ao `Peer` local os `Workers` disponíveis (Algoritmo 4). O `Peer` por sua vez envia sua requisição para todos os `Peers` ativos através do grupo *multicast* P2P (isto é feito pela thread *P2P-MG*, linha `5` no Algoritmo 3). De fato, isso pode ser considerado uma melhoria comparado com a OG, pois enquanto que *multicast* envia apenas uma mensagem para atingir todos os `Peers` ativos na grade, no OG é enviada uma mensagem *unicast* para todos os `Peers` individualmente. Assim que um `Peer` escuta a requisição de outro `Peer` através do grupo *multicast* P2P, ele envia a sua lista de `Workers` disponíveis. A partir daí, as tarefas são escalonadas como no OG [21].

Data: *i* Endereço IP do Membro
peer ID do peer Local

```
1 begin
2     while true do
          /* Aguarda uma nova tarefa                                  */
3         Worker.accept(task)
4         ret=task.run()
5 end
```

Algoritmo 4: Worker *thread.*

4.2 Recuperação de Falhas do Peer

Um dos grandes desafios de se trabalhar com sistemas distribuídos é o de detectar com precisão se uma determinada entidade do sistema falhou, ou se ela está apenas demorando a responder, por problemas de transmissão ou por uma sobrecarga no sistema. Em sistemas assíncronos (*time-free*), essa detecção de falhas é não confiável, pois não é possível definir um atraso máximo para a entrega de mensagens. No entanto, é possível construir sistemas distribuídos confiáveis mesmo que os detectores de falhas sejam não confiáveis [20]. Para isso, deve ser criado um mecanismo de detecção de falha distribuído.

Um tipo de falha que será tratada nesta arquitetura é o *crash* de um `Peer`. Esta falha é caracterizada quando um processador (no caso, o `Peer`) simplesmente para de executar em um determinado ponto no tempo [31]. Neste caso, para incorporar a tolerância à falha de um `Peer`, serão necessárias duas etapas: (i) a detecção da falha e a (ii) recuperação do sistema.

Como descrito anteriormente, para garantir a confiabilidade do sistema, o detector de falhas deve ser distribuído de tal forma que cada entidade da grade tenha um módulo de detecção de falha não confiável [20]. A função de cada módulo de detecção de falha é manter o estado de uma ou mais entidades do sistema e informar esse estado a outras entidades, quando solicitado. Este estado pode ser *alive* ou *crashed*. Na abordagem utilizada neste trabalho, as entidades responsáveis por detectar a falha serão os `Workers`, e os `Peers` serão as entidades monitoradas.

Há diversas estratégias para implementar a detecção de falha não confiável. Neste trabalho serão abordadas duas implementações [49]: o modelo *heart-beat* e o modelo *interrogation*.

No modelo *heart-beat* o `Peer` manda periodicamente (a cada t_{alive} segundos) uma mensagem para o grupo *multicast* local notificando que ele está funcionando. Cada `Worker` aguarda t_{alive}, e se nesse tempo não receber nenhuma mensagem do `Peer`, ele armazena o estado do `Peer` como *crashed*. Caso ele receba posteriormente alguma mensagem do `Peer` ele altera seu estado para *alive*.

No modelo *interrogation* os `Workers` mandam uma mensagem (a cada t_{alive} segundos) para o `Peer`, para verificar se ele está funcionando. Se o `Peer` não responder em um intervalo de tempo de t_{ack} segundos, ele armazena o estado do `Peer` como *crashed*, caso contrário, ele altera o estado do `Peer` para *alive*.

Esses dois métodos são semelhantes ao método ativo e passivo apresentados na Seção anterior, que servem para verificar se a já existe um `Peer` no *site* local. Assim, para a detecção de falha do `Peer`, foi utilizada também uma combinação desses dois métodos. Quando um `Worker` identifica que o `Peer` falhou, ele inicia um *status* provisório `PeerCandidate` e envia uma mensagem para o seu grupo *multicast* local, requisitando o IP de cada componente e o status do `Peer` para este componente (vide Algoritmo 5). Desta forma, cada componente que tiver o *status* provisório `PeerCandidate`, inicia uma *thread* e aguarda $n * t_{ack}$ segundos para receber uma mensagem de resposta com o IP de outro componente e o status do `Peer`.

Se nestes $n * t_{ack}$ segundos ele receber alguma mensagem de um componente com IP mais baixo, e informando que o status do `Peer` é *crash* ele termina essa *thread* e retira seu status provisório de `PeerCandidate`. Caso nestes $n * t_{ack}$ segundos ele receba apenas

Data:

IP	Endereço IP do Membro
Remote IP	Endereço IP do Membro que enviou a mensagem
peer	Peer Local
LMG	Grupo Multicast Local
PeerStatus	Status do Peer
PeerCandidate	Status de Falso Peer

```
1  begin
2      PeerCandidate=true
3      for j=1 to n do
           /* Consulta o grupo requisitando o IP e o PeerStatus, e espera
              t_ack s                                                    */
4          Remote IP = SendMessage(LMG,Request IP)
5          if Remote IP != null then
6              if Remote IP < IP Remote AND PeerStatus=crash then
7                  PeerCandidate=false
8      if PeerCandidate then
           /* Nenhum Peer no grupo local.  Este nó com o menor IP
              deve assumir o papel de Peer.                          */
9          peer = i
           /* Inicia a thread Peer!                                  */
10         Peer.start()
           /* Anuncia o status do Peer no LMG                        */
11         SendMessage(LMG, PeerAnnouncement(i))
           /* Encerra a thread que verifica a existência de um Peer local! */
12         CheckPeer.stop()
       /* Encerra a thread que verifica a existência de um Peer local! */
13 end
```

Algoritmo 5: Peer Failure Detector.

mensagens de componente com um IP mais alto, e informando que o status do `Peer` é *crash* ele termina essa *thread*, retira seu *status* provisório de `PeerCandidate` e inicia o serviço de `Peer`. Sendo assim, caso ocorra a falha de um `Peer`, apenas o componente que tiver o menor IP assumirá o papel de `Peer`. Esta *thread* `PeerFailureDetector` está descrita

no Algoritmo 5.

4.3 Implementação

Nesta Seção são apresentados os detalhes de implementação do *ad hoc* Grid. Ele foi implementado utilizando a linguagem de programação Java™2 Platform Standard Edition 5.0 [32] para facilitar a integração com o código do OurGrid, já que este também foi desenvolvido em Java.

O desenvolvimento do *ad hoc* Grid foi dividido em quatro componentes principais: inicialização, comunicação *multicast*, detecção de falha do Peer e manutenção da NoF, ilustrados na Figura 4.2.

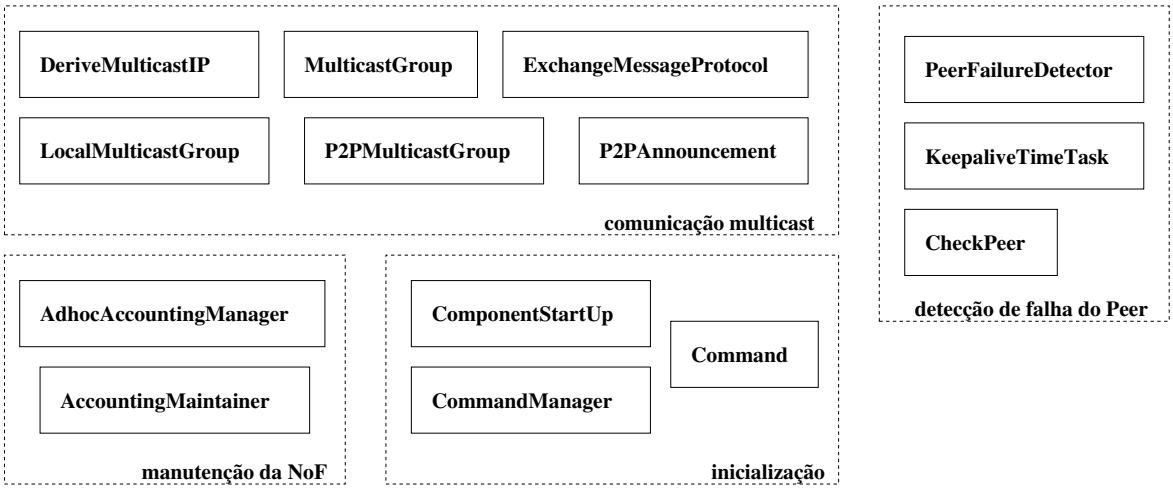

Figura 4.2: Estrutura do *ad hoc* Grid.

Nas Seções a seguir são descritos os componentes que compõem a estrutura do *ad hoc* Grid.

4.3.1 Comunicação *Multicast*

Para possibilitar a troca de mensagens entre as entidades do OurGrid, foi utilizado o mecanismo de *socket*. A linguagem Java oferece suporte a dois modos de utilização de *sockets*: um modo orientado a conexão, que funciona sobre o protocolo TCP e o modo orientado a datagrama que funciona sobre o protocolo UDP. Visto que a troca de men-

sagens no *ad hoc* Grid é feita através de grupos *multicast*, foi utilizada a biblioteca `java.net.MulticastSocket` do Java que oferece suporte a *multicast*.

Como descrito na Seção anterior, a arquitetura do *ad hoc* Grid é formada por dois grupos *multicast*. O grupo *multicast* local (`LocalMulticastGroup`) é formado pelos componentes da grade que desejam fazer parte de um *site*, que podem ser o `Peer`, o `Worker` e o `MyGrid`. O grupo *multicast* entre-pares (`P2PMulticastGroup`) é formado pelo `Peer` de cada *site*. A Figura 4.3 apresenta o diagrama das classes responsáveis pela formação dos grupos *multicast*. A classe `MulticastGroup` implementa as principais funcionalidades de um grupo *multicast*. Um objeto desta classe é construído recebendo como parâmetro o endereço e a porta que serão utilizados pelo grupo *multicast* e o `loopbackMode` que indica se a *interface* deve receber ou não, os pacotes enviados por ela. Através desta classe um componente pode associar-se, ou deixar o grupo *multicast*, bem como enviar e receber as mensagens para este grupo.

O endereço do grupo *multicast* entre-pares (P2PMG) é um endereço fixo, e serve de ponto de encontro para os `Peers` de uma grade. Portanto, todos os sites que desejarem fazer parte de uma mesma comunidade devem utilizar o mesmo endereço para o P2PMG. Já o endereço do grupo multicast local (LMG) é derivado do endereço da rede local. Desta forma, todas as máquinas de uma mesma rede local que entrarem na grade terão o mesmo endereço do LMG e, portanto, farão parte de um mesmo site.

Como apresentado na Figura 4.3 as classes (`LocalMulticastGroup`) (`P2PMulticastGroup`) herdam os métodos e atributos da classe `MulticastGroup` para criar o grupo, enviar e receber mensagens. Como o recebimento das mensagens *multicast* via *sockets* é bloqueante, foi definido um *timeout*, que é o tempo máximo que o grupo deve esperar por uma mensagem. Se neste intervalo de tempo o grupo não receber mensagem alguma, ele define a mensagem como `null` e aguarda uma nova mensagem.

A recepção de mensagens em ambos os grupos deve ser constante, para que o grupo se mantenha ativo, portanto, estas classes criam uma *thread* responsável por receber as mensagens do grupo. Esta *thread* recebe as mensagens *multicast* até que seja satisfeita uma condição de parada. Esta condição de parada é satisfeita quando o componente abandona o grupo *multicast*.

O diagrama da Figura 4.4 apresenta as classes responsáveis pela troca de men-

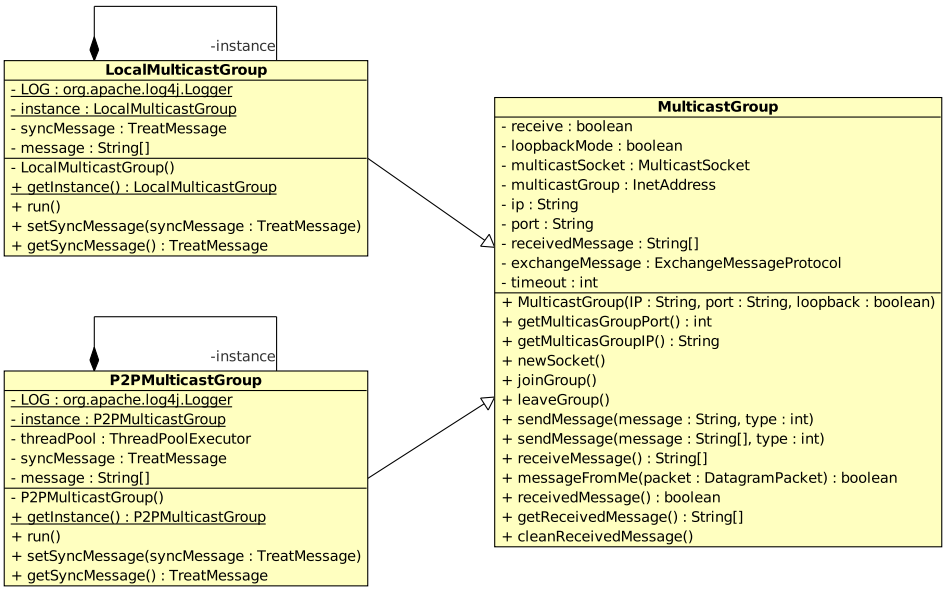

Figura 4.3: Diagrama das classes responsáveis pela formação dos Grupos *multicast*.

sagens *multicast*. Assim, sempre que uma mensagem for enviada através da classe `MulticastGroup`, ela é primeiramente organizada em um pacote através da classe `ExchangeMessageProtocol`. Da mesma forma, quando um grupo receber uma mensagem ela é extraída deste pacote através da classe `ExchangeMessageProtocol`. Para especificar a forma pela qual essas mensagens são organizadas em pacotes, foi definido um protocolo de troca de mensagens que será apresentado na seção a seguir.

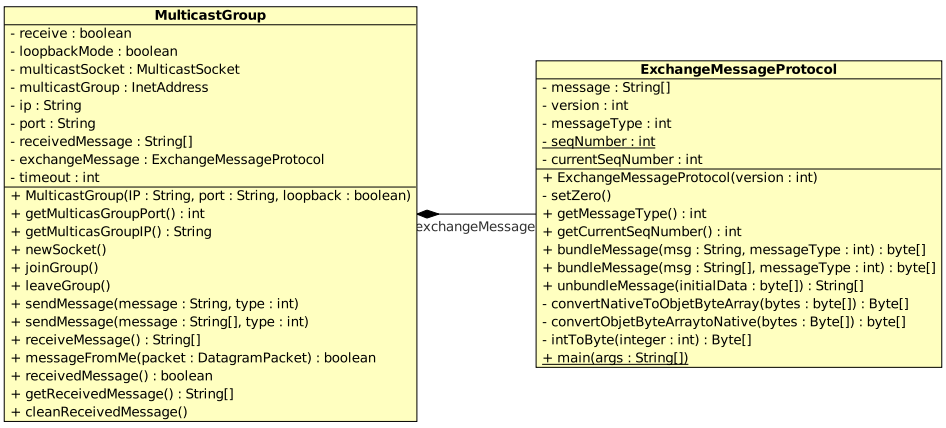

Figura 4.4: Diagrama das classes responsáveis pela troca de mensagens *multicast*.

ID do protocolo	tipo de mensagem	tam. do quadro	*payload*	fim da msg.
2bytes	4bytes	2bytes	2032bytes	4bytes

Tabela 4.1: Estrutura da mensagem *multicast*

4.3.2 Protocolo de Troca de Mensagens

As atividades de auto-organização do *ad hoc* Grid foram baseadas em troca de mensagens através de grupos *multicast*. Portanto, foi elaborado um protocolo de alto nível, a fim de definir os tipos de mensagens trocadas entre as entidades dos grupos, a estrutura da mensagem e o tamanho máximo do pacote.

A estrutura da mensagem foi dividida em cinco campos: o ID do protocolo, o tipo de mensagem, o tamanho do campo de dados, o campo de dados (*payload*) e o último campo que marca o fim da mensagem. A tabela 4.1, apresenta a configuração da mensagem com o tamanho de cada um de seus campos.

O campo ID é um identificador que o diferencia dos outros protocolos que trafegam na rede. O campo tipo de mensagem, identifica que mensagem está sendo enviada no pacote, o tamanho do quadro que define o tamanho da mensagem, o campo *payload* que, contém a mensagem que será enviada e o campo fim da mensagem que marca seu término.

As mensagens que são definidas pelo protocolo e que podem ser enviadas/recebidas através dele são:

1. **LMGHasPeer** - Esta mensagem é utilizada para requisitar ou informar se o grupo *multicast* local tem um `Peer`. Ela pode assumir os seguintes valores:

   request : Para solicitar o anúncio do `Peer` (método passivo);

   response : Para responder uma solicitação de anúncio, se o componente for um `Peer` do LMG (método passivo);

   alive : Para anunciar a existência de um `Peer` no LMG (método ativo).

2. **P2PMGAnnouncement** - Esta mensagem é utilizada para o `Peer` anunciar sua existência no P2PMG, podendo assumir os seguintes valores:

   IdValue : Para solicitar o anúncio do `Peer` (método passivo);

Name : Para responder uma solicitação de anúncio, se o componente for um `Peer` do LMG (método passivo);

URL : Para anunciar a existência de um `Peer` no LMG (método ativo).

3. **IP** - Mensagem utilizada para requisitar ou informar o IP dos membros do LMG;

   request : Solicitar o IP dos membros do LMG;

   responde : Para enviar o IP aos membros do LMG.

4. **NoF** (*Network of Favors*) - Mensagem utilizada para disseminar a rede de favores no grupo *multicast* local.

5. **WDF** (*Worker Description File*) - Mensagem utilizada para enviar o WDF para o `Peer` do *site* local (quando um `Worker` inicia seu serviço).

6. **SDF** (*Site Description File*) - Mensagem utilizada para enviar a lista dos `Workers` para o LMG.

7. **PDF** (*Peer Description File*) - Mensagem utilizada para enviar a descrição do `Peer` para o `Mygrid` e pode assumir os seguintes valores:

   peerLabel : Para enviar o apelido do `Peer`;

   peerName : Para enviar o nome do `Peer`.

   peerPort : Para enviar a porta (RMI) de acesso do `Peer`.

4.3.3 Manutenção da Rede de Favores

O propósito da rede de favores no OurGrid é incentivar os `Peers` de diferentes *sites* a cooperarem, ou seja, doarem seus recursos a outro *site* (`Peer`), para que futuramente possa utilizar os recursos desse *site*. Um favor é contabilizado quando ocorre a alocação de um processador de um *site* para um `Peer` de outro *site*, e o valor deste favor é o trabalho realizado pelo `Peer` que disponibilizou o processamento em seu *site* [12; 11]. Esta contabilidade de favores é persistida pelo `Peer` sempre que seu valor é atualizado.

A diferença em manter a NoF no *ad hoc* Grid é que o `Peer` deixa de ser um componente centralizado e passa a ser um componente dinâmico no decorrer do tempo. Neste caso, entende-se que houve dinâmica sempre que ocorrer alguma falha no `Peer` de um *site*, e outro componente da grade assumir o papel de `Peer`. Portanto, no *ad hoc* Grid, qualquer componente do *site* pode futuramente se tornar um `Peer`. A fim de garantir que um componente que venha a ser um `Peer` tenha a NoF mais atual, toda vez a NoF é persistida no `Peer`, ela é enviada para o *site* local, e os componentes que a recebem também persistem esse valor. Assim, quando um componente assumir o papel de `Peer` em seu *site*, ele já terá a NoF mais atual.

A Figura 4.5 apresenta o diagrama das classes responsáveis por manter a NoF de um *site*. Ela é mantida através da classe `AdhocAccountingManager` que utiliza a classe `LocalMulticastGroup` para enviar a NoF ao LMG.

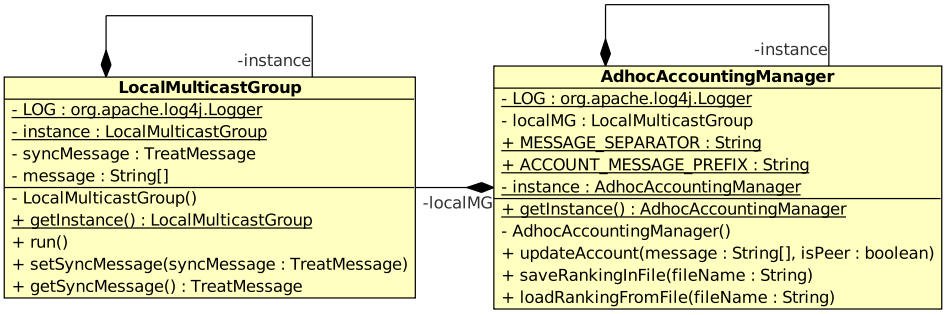

Figura 4.5: Diagrama das classes responsáveis por manter a NoF.

4.3.4 Inicialização dos Componentes

O *ad hoc* Grid é iniciado através do *script* adhocgrid.sh (vide Capítulo A). Este *script* é uma interface que recebe os parâmetros de inicialização, verifica se são válidos e executa o *ad hoc* Grid através da classe `CommandManager`. Há duas opções de parâmetros que podem ser passados para o `CommandManager`, iniciar o *ad hoc* Grid com o `MyGrid` (parâmetro $-mg$) ou sem ele. Por fim, na classe `CommandManager` é instanciado o `ComponentStartUp` que, de fato, inicia os componentes da grade. O diagrama da Figura 4.6 apresenta as classes responsáveis pela inicialização do *ad hoc* Grid.

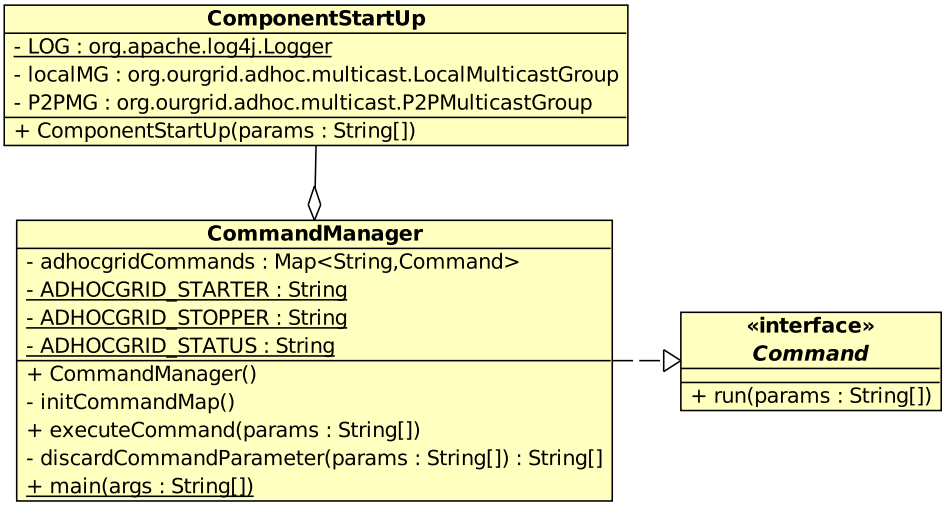

Figura 4.6: Diagrama das classes responsáveis pela inicialização do *ad hoc* Grid.

Capítulo 5

Avaliação Experimental

Neste Capítulo, é apresentado o método utilizado na execução dos experimentos, bem como, os resultados obtidos e as discussões. Na Seção 5.1 é apresentada a configuração do ambiente para os experimentos. Na Seção 5.2 são descritos os cenários utilizados. Na Seção 5.3 é avaliado o comportamento de cada componente do sistema segundo a arquitetura proposta. Na Seção 5.4 é avaliado o impacto da abordagem de grade *ad hoc* na execução dos *jobs*. Por fim, na Seção 5.5 é apresentado o *overhead* gerado pela grade *ad hoc* na rede.

5.1 Preparação do Ambiente de Execução dos Experimentos

Para a execução dos experimentos com o *ad hoc* Grid, foi necessária a configuração de alguns equipamentos de rede bem como, a configuração de aplicações responsáveis por coletar os dados referentes aos experimentos. Os experimentos foram conduzidos em um ambiente controlado no Laboratório de Sistemas Distribuídos (LSD) da Universidade Federal de Campina Grande (UFCG). As máquinas tinham configurações semelhantes com processadores Pentium IV e com 1 $gigabyte$ de meméria RAM. Foram utilizadas 30 máquinas do laboratório, dividindo-as em duas subredes como ilustrado na Figura 5.1.

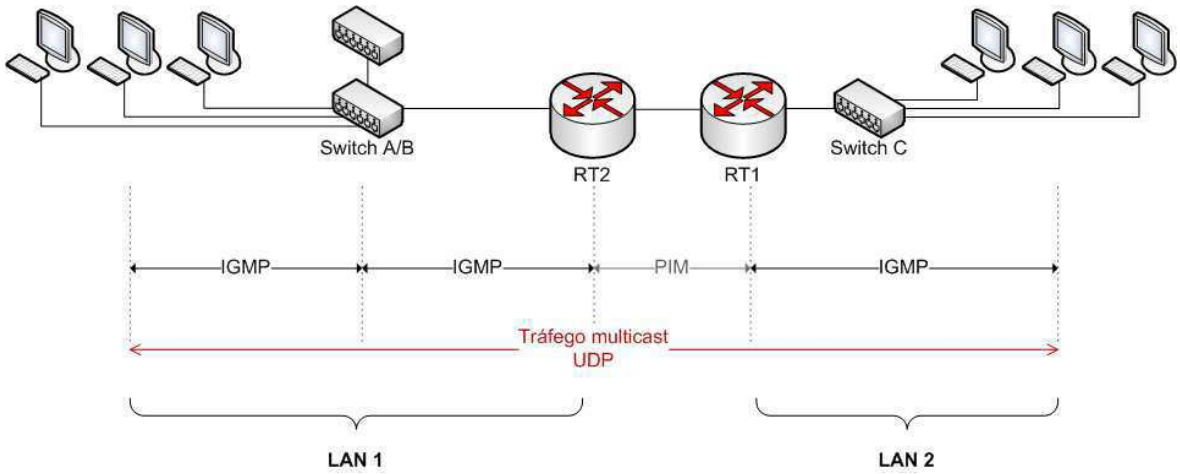

Figura 5.1: Configuração da rede para os experimentos.

5.1.1 Considerações sobre *Multicast*

A comunicação *multicast* permite a transmissão de informação simultaneamente de um nó para um grupo de destinatários sendo também conhecida como difusão seletiva. Diferente do uso de *broadcast*, onde a informação é enviada para todos os nós da rede, com *multicast* é possível que cada nó escolha se deseja ou não participar de uma transmissão.

Essa difusão seletiva pode ser feita tanto em *hardware* da rede local como também na camada IP da arquitetura Internet TCP/IP[22].

Quando ela é feita em *hardware*, existe um conjunto de endereços reservados para uso com difusão seletiva. Assim, quando um grupo de máquinas deseja se comunicar, elas escolhem o endereço *multicast* que irão utilizar na comunicação. As interfaces de rede reconhecem o endereço selecionado, e todas as máquinas pertencentes ao grupo, passam a receber uma cópia de cada quadro enviado para o endereço *multicast*.

Na arquitetura TCP/IP (*Multicast* IP), a forma de endereçamento é uma abstração do *multicast* em *hardware*. Nessa camada, os datagramas IP são enviados para múltiplos pontos (grupo *multicast*) ao mesmo tempo, usando um endereço *multicast*. Este grupo *multicast* é um conjunto de máquinas que é identificado por um endereço IP único. Além disso, é construída uma árvore de distribuição *multicast* para este grupo. Os membros deste grupo *multicast* são dinâmicos, ou seja, as máquinas podem entrar ou sair dos grupos a qualquer momento. Não existem limitações para a quantidade de membros de um grupo, e qualquer

membro pode participar de um grupo ou mais, bem como, enviar datagramas a um grupo sem fazer parte dele. O protocolo utilizado pelos membros para fazer parte do grupo *multicast* é chamado Internet Group Management Protocol (IGMP).

Um grupo *multicast* é especificado por um endereço IP de classe D[1] e por uma porta UDP. Há uma série de endereços IP de classe D reservados para o uso pelos protocolos de roteamento, descobrimento de topologia e manutenção dos protocolos [7].

A difusão seletiva pode ser usada em uma rede local ou através da internet. Neste último caso, como o *multicast* não é compreendido em todos os roteadores da Internet, é utilizada a topologia do MBone. Nesta topologia, os roteadores *multicast* utilizam um protocolo de roteamento *multicast* para descobrir sua própria topologia e assim identificar a rota adequada dos datagramas. Atualmente, o protocolo de roteamento mais utilizado para isto é o Protocol Independent Multicast (PIM). Este protocolo é denominado independente, pois não depende de nenhum outro protocolo de roteamento unicast para o descobrimento da topologia. Além do PIM, em algumas partes do MBone, os roteadores utilizam outros protocolos de roteamento, dentre eles o Multicast Open Shortest Path First (MSOPF)[43] e o Distance Vector Multicasting Routing (DVMRP)[54].

Neste trabalho, foi utilizado o protocolo PIM para configurar os roteadores RT1 e RT2 representados na Figura 5.1. Esse protocolo cria árvores de distribuição *multicast* de tal forma, que os pacotes de dados enviados para um grupo *multicast* chegam para todos os receptores que se associaram a este grupo. Estas árvores de distribuição *multicast* ficam armazenadas em um roteador que é denominado Rendezvous Point (RP) para onde a fonte deve enviar seu tráfego. Quando o RP recebe este tráfego ele o encaminha para os receptores que fazem parte da árvore de distribuição. Por fim, quando o roteador a um salto do receptor aprende o caminho da fonte, ele manda uma mensagem diretamente para a fonte requisitando associar-se a este grupo. Desta forma, é criada uma árvore de distribuição com base na fonte.

Existem ainda diversas implementações do protocolo PIM, dentre elas o Sparse Mode (SM)[27], Dense Mode (DM)[6], Source Specific Mode (SSM)[15] e o Bidirectional Mode (ou Sparse-Dense Mode - SDM)[34].

O protocolo PIM Sparce Mode é otimizado para grupos que estão distribuídos em diferentes regiões da Internet e nos quais uma pequena parte das máquinas e roteadores irão se

[1] De 224.0.0.1 até 239.255.255.255

associar a uma seção *multicast*. Sendo assim, essa implementação assume que a maior parte das subredes não desejam receber pacotes *multicast*.

Por outro lado, protocolo PIM Dense Mode é otimizado para grupos com membros próximos. A implementação DM inunda a rede com pacotes *multicast* e apara os ramos da árvore de distribuição *multicast* que não possuem membros no grupo. Diferente do PIM-SM ele considera que grande parte das subredes desejam receber os pacotes *multicast* enviados. Já no PIM-SSM, os únicos pacotes que são entregues a um receptor são aqueles que se originam de um endereço de fonte específico, solicitado pelo receptor. Por fim, o PIM-SDM que ao invés de utilizar árvores de distribuição baseada na fonte, o fluxo de dados para o RP na árvore compartilhada é bidirecional.

A implementação utilizada na configuração do ambiente para a comunicação *multicast* foi PIM-SM, como será apresentado na Seção 5.1.3 deste Capítulo.

5.1.2 Configuração dos *Switches*

A primeira etapa teve como objetivo a configuração do ambiente para fornecer um serviço de transporte *multicast*. Para isto, foi necessário configurar os Switches do laboratório para habilitar o IGMP (Internet Group Management Protocol)[16]. IGMP é um protocolo utilizado para gerenciar os membros de um grupo *multicast*. A Figura 5.1 apresenta a configuração das duas redes do LSD com três *switches* da 3com. Os dois Switches do modelo 4250T já estavam com a função habilitada, no entanto, o outro *switch* 3824 não suportava esta função. Esse problema foi resolvido com a atualização do *firmware*, o que possibilitou a configuração do IGMP.

Outra característica que podemos encontrar nos Switches gerenciáveis e que ajuda a eliminar o congestionamento de tráfego *multicast* em uma rede local, é o IGMP *snooping*. Quando um *switch* recebe uma mensagem *multicast*, ele tipicamente distribui essa informação para todas suas portas. Com a função IGMP *snooping* habilitada, o *switch* mantém uma tabela com o endereço de todas as máquinas que pertencem a um grupo *multicast*. Quando uma mensagem é enviada para este grupo o switch direciona a informação apenas para a porta das máquinas que fazem parte da tabela daquele grupo.

Quando esta função foi habilitada nos *switches* do LSD, observou-se que as mensagens *multicast* chegavam com muito atraso para o grupo, muitas vezes excedendo o *time-out* do

ad hoc Grid, causando perda de pacotes para a aplicação. Com o intuito de resolver este problema, foram feitos vários testes com os equipamentos, atualizando os *firmwares*, mas o problema não foi solucionado.

Por isso, a avaliação do *overhead* gerado na rede que será apresentado na Seção 5.5, pode ser ainda melhorada se os *switches* de rede suportarem o IGMP *snooping*, e funcionarem de forma correta.

5.1.3 Configuração dos Roteadores XORP

Esta etapa de configuração dos roteadores foi realizada para possibilitar a transmissão do fluxo *multicast* entre as rede locais LAN1 e LAN2. Como apresentado na Seção 5.1.1 deste Capítulo, existem várias implementações do protocolo PIM, sendo cada uma delas mais adequada para um determinado ambiente. Em particular, o Sparse Mode que é mais eficiente em ambientes onde poucas máquinas inter-redes farão parte de um mesmo grupo *multicast* e o Dense Mode, mais adequado para ambientes em que grande parte das máquinas inter-redes farão parte do mesmo grupo. Buscando a compatibilidade com o padrão que atualmente é utilizado no MBone, a escolha mais adequada seria o PIM-SM.

No entanto, deve-se avaliar se esta escolha também é mais adequada para o *ad hoc* Grid. Como apresentado na arquitetura do *ad hoc* Grid, ela possui dois grupos *multicast*, o LMG e o P2PMG. Apesar de o LMG ser composto por todas as máquinas da rede (que forem participar da comunidade), ele é um grupo *multicast* que está limitado à rede local. Já o P2PMG é composto pelos `Peers` de cada *site*. Sendo assim, cada rede local que fizer parte da comunidade *ad hoc* Grid terá apenas uma máquina participando do P2PMG. Portanto, o PIM-SM também é o mais adequado para a arquitetura do *ad hoc* Grid.

Para configurar os rotadores RT1 e RT2 (vide Figura 5.1) para rotear pacotes *multicast* utilizando o protocolo PIM-SM, foi utilizado o eXtensible Open Router Platform (XORP) [52]. XORP é atualmente a única plataforma de código aberto que suporta o roteamento *multicast* nas plataformas baseadas em Unix com kernel 2.4.x ou 2.6.x. A interação do usuário com o XORP é semelhante à interação com uma interface de um roteador dedicado. O passo mais importante para a configuração desta plataforma é a definição do arquivo de configuração que determina o estado inicial do roteador. No Apêndice B, é apresentado o arquivo utilizado para a configuração do RT1, explicado seus detalhes. Nele, são definidas

as interfaces utilizadas no roteamento, os protocolos, o mecanismo de encaminhamento de pacote. A configuração do XORP no RT2 utilizou a mesma configuração do RT1 adaptando apenas as interfaces para este roteador.

5.1.4 Estatísticas do Tráfego *Multicast*

Para o monitoramento do tráfego *multicast* na rede foi utilizado o protocolo SNMP (Simple Network Management Protocol) [17]. O SNMP foi desenvolvido para permitir que dispositivos de rede que utilizam o protocolo IP (Internet Protocol) possam ser gerenciados remotamente, através de um conjunto de operações simples.

Este protocolo utiliza o modelo *manager*/*agent*, onde um servidor (*manager*) com a função de NMS (Network Management System) comunica-se com o agente de gerência de rede (*agent*[2]) através do protocolo de gerência. O servidor é responsável por buscar as informações no agente, ou receber *traps* mensagens enviadas pelo agente para informar alterações em seu estado. O agente é um *software* que é executado no dispositivo gerenciado, podendo ser um programa separado, ou incorporado em *hardware* para prover informações ao NMS.

O protocolo SNMP utiliza UDP na camada de transporte para a comunicação entre NMS e agente como apresentado na Figura 5.2.

Cada objeto gerenciado através do SNMP é definido pelo SMI (Structure of Management Information). Além de definir um objeto, o SMI especifica quais os tipos de dados estão associados a ele. A definição do objeto pelo SMI é feita através de um campo OID (Object Identifier) que identifica o objeto, o tipo de dados deste objeto e o seu tipo de codificação.

No contexto deste trabalho, o SNMP foi utilizado para verificar qual o tráfego de pacotes *multicast* gerado por cada um dos componentes da grade. Desta forma, a requisição desta informação foi feita através do comando a seguir:

```
snmpwalk -v 2c -c public IPsource 1.3.6.1.2.1.31.1.1.1.4
```

O comando *snmpwalk* é utilizado para se recuperar uma árvore de informações de um agente SNMP, que neste caso foi especificado pelo OID `1.3.6.1.2.1.31.1.1.1.4` que recupera a informação do fluxo *multicast* gerado por este IPsource. O parâmetro `-v 2c` identifica a versão do SNMP e o `-c public` identifica o grupo ao qual o objeto pertence.

[2]Instalado no dispositivo gerenciado

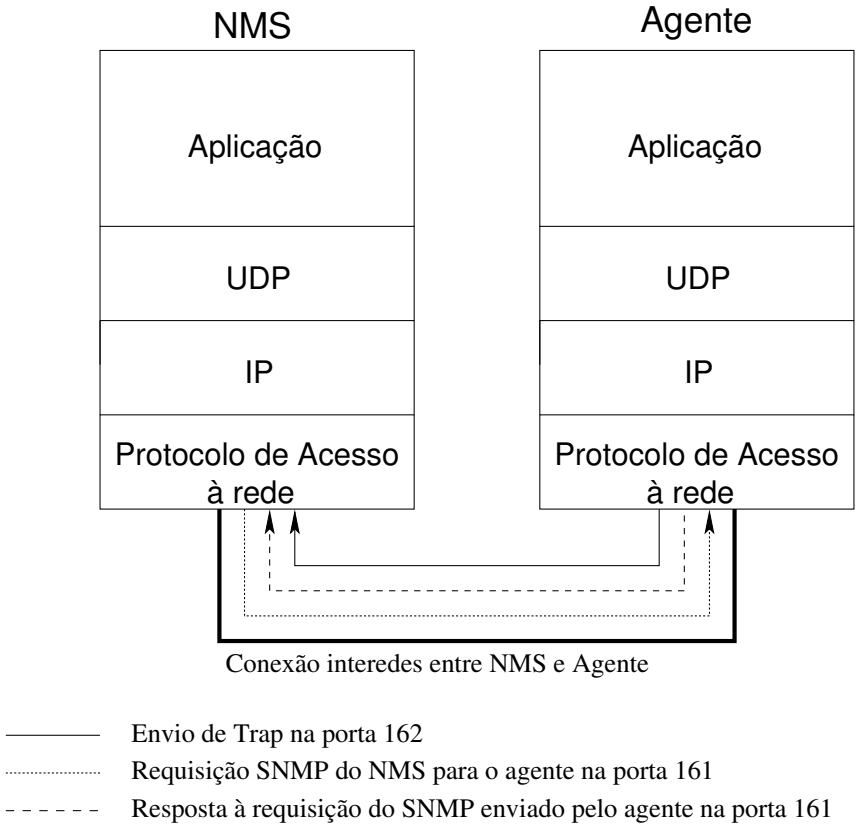

Figura 5.2: Comunicação entre NMS e agente (Adaptado de Douglas *et al.* [42]).

A partir desses comandos, foram implementados *scripts* que monitoram o fluxo *multicast* de todos os componentes da grade, e geram as estatísticas desses dados para os experimentos.

5.2 Cenários dos Experimentos

Para a execução dos experimentos do *ad hoc* Grid foram utilizados *jobs* que simulam o comportamento do NodeWiz-R [3]. Cada *job* é composto por 5 tarefas, que por sua vez simulam diferentes cenários para o NodeWiz-R.

Portanto, foram definidos três cenários diferentes para a execução dos experimentos. O primeiro cenário inicia o *ad hoc* Grid nas redes LAN1 e LAN2 e mantém a configuração da grade até o fim dos experimentos. No segundo cenário, é inserida falha (executando um *kill* no processo do *ad hoc* Grid) em um componente da grade escolhido aleatoriamente, a cada 10 minutos. Após 5 minutos este componente é restabelecido na grade. Desta forma, esse cenário visa simular o comportamento dinâmico de uma grade *ad hoc*. Por fim, no terceiro

cenário, é inserida falha a cada 1 segundo, simulando uma dinâmica mais forte.

Em todos os cenários, o *ad hoc* Grid é iniciado em 25 máquinas da rede LAN1 e em 5 máquinas da rede LAN2. Após estabelecida a comunidade, são submetidos 4 *jobs* a cada 20 minutos através de uma instância do `MyGrid` na rede LAN1 e 2 *jobs* a cada 20 minutos a partir da rede LAN2. A quantidade de *jobs* e o intervalo de tempo em que eles são submetidos foi escolhido de tal forma que, no cenário 1 todas as tarefas destes *jobs* possam ser executadas na grade simultâneamente, sem ter que aguardar para ser executada. Isto porque o cenário 1 foi elaborado para verificar o *overhead* causado apenas pela trocas de mensagens *multicast*, e não tem o intuito de verificar o quanto, uma falha de algum componente compromete o desempenho global da grade. Já nos cenários 2 e 3, o objetivo foi avaliar quanto a dinâmica da grade *ad hoc* influencia em seu desempenho.

Cada cenário foi executado durante uma hora, por 10 vezes, e a partir desses resultados foram retirados o tempo médio de execução, o *overhead* das mensagens *multicast* e a disponibilidade dos `Peers` da comunidade. Portanto, cada *job* foi executado 180 vezes para cada cenário.

5.3 Comportamento dos Componentes

Na primeira etapa, foram executados testes de alguns cenários, monitorando o comportamento de cada um dos componentes da grade no decorrer de sua existência. É importante destacar que nesta primeira avaliação não foram utilizados os cenários discutidos na Seção anterior. Isto porque, o objetivo deste experimento é verificar se a arquitetura da grade se comporta da forma como foi projetada. Para cada cenário foram executadas dez rodadas, obtendo-se o tempo médio de descoberta do `Peer` e, eventualmente, a identificação de mais de um `Peer` por grupo local.

Com o objetivo de automatizar a realização desses testes, foi implementado um módulo de testes funcionais para o sistema. Esse módulo foi implementado em java e monitora o funcionamento de cada componente da grade, da seguinte forma:

- Junta-se ao grupo *multicast* local;
- Junta-se ao grupo *multicast* entre pares;

- Verifica o tipo de mensagem recebida através do grupo *multicast* local e a partir dela armazena o status do componente, o seu IP e o *time-stamp* da mensagem;

- Verifica o tipo de mensagem recebida do grupo *multicast* entre pares e verifica se o mesmo é realmente o `Peer` em seu respectivo grupo *multicast* local.

A partir desse módulo de testes funcionais foram obtidos os resultados para cada cenário que será apresentado a seguir.

O primeiro cenário testado, foi o caso mais simples onde há apenas uma máquina na grade. Ela assume o papel de `Peer` e `Worker`. Depois de instanciado, o *ad hoc* Grid levou em média $10s$ para descobrir que não havia nenhum `Peer` no grupo e iniciar os três componentes, ou seja, o pior caso do método ativo de descoberta do `Peer` ($n * t_{ack}$), com $n = 3$ e $t_{ack} = 1s$ mais o tempo para iniciar o `Peer` e o `Worker`. Apesar do t_{ack} ter sido definido como uma constante (por questões de simplificação da implementação), na próxima etapa desta implementação seu valor será adaptativo (adotando uma média móvel ponderada sobre o RTT). Assim, é possível calcular o tempo de ack com uma margem de segurança.

O segundo cenário testado foi com duas máquinas, cada uma delas instanciando o *ad hoc* Grid em momentos distintos. A primeira instanciação do *ad hoc* Grid inicia os três componentes da grade e, em seguida, a outra instância é iniciada. O tempo médio para a descoberta do `Peer` foi de $1s$. Simulando a falha deste `Peer` (*i.e.*, parando a execução deste processo), a outra máquina identifica a falha do `Peer` e assume o seu papel. Como cada máquina mantém localmente a lista de `Workers` ativos e a contabilidade da rede de favores, esses dados são recuperados sem a necessidade de consulta ao grupo local.

Para validar a implementação do *CheckPeer*, o terceiro cenário foi testado com duas máquinas que iniciam o *ad hoc* Grid ao mesmo tempo e assim, assumem o papel de `Peer` simultaneamente. Neste caso, a máquina com o IP mais baixo pára a execução do `Peer` em média após um período de $2s$, e a máquina com IP mais alto mantém seu estado de `Peer`.

O último cenário foi reproduzido com todas as máquinas da rede, com a inicialização do *ad hoc* Grid aleatória. A ocorrência de várias instâncias do `Peer` simultaneamente é pouco provável, pois depende da sincronia tanto da inicialização do *ad hoc* Grid em cada máquina quanto, dos envios de mensagens. Há ainda a possibilidade de sobrecarga do `Peer`, e de ele não responder em tempo hábil, causando a inicialização de outro `Peer`. Essa situação,

entretanto, já foi reproduzida e o resultado é semelhante ao descrito no cenário anterior.

Em síntese, esta primeira etapa de testes mostrou que o comportamento da solução *ad hoc* foi, de fato adequado com a arquitetura proposta.

5.4 *Overheard* do *ad hoc* Grid na Execução dos *Jobs*

Na segunda etapa dos testes foi medido o *overhead* gerado pelo *ad hoc* Grid. O primeiro tipo de teste teve como intuito medir o *overhead* de processamento causado pela solução *ad hoc*. Isto ocorre porque além de executar as tarefas (no caso dos `Workers`) e/ou gerenciar os `Workers` (no caso do `Peer`) a máquina terá que dedicar uma parte de seu processamento para as tarefas de recepção e envio das mensagens *multicast* utilizadas para a auto-organização da grade. Para isto, foram executados os mesmos *jobs* no OurGrid e no *ad hoc* Grid e medidos os seus respectivos tempos de execução. Como descrito na Seção 5.2 foi executado o *job* em três cenários diferentes.

O tempo médio de execução do *job* no cenário 1 foi de $742,42$ segundos, aumentado o tempo de execução em $1,48\%$ com relação à execução no OurGrid ($731,6$ segundos). No cenário 2 o tempo de execução aumentou em média $1,64\%$. No cenário 3, onde a entrada e saída de nós é mais constante que nos dois primeiros cenários, o tempo médio de execução dos *jobs* foi de $778,6$ segundos, ou seja $6,43\%$ a mais que no OurGrid. Os tempos de execução foram obtidos através de 180 experimentos para cada cenário, com nível de confiança de 95%. A Tabela 5.1 apresenta os intervalos de confiança para os experimentos realizados.

	OurGrid	**Cenário 1**	**Cenário 2**	**Cenário 3**
Int. de confiança	726,38-736,83	733,84-750,97	735,47-751,71	763,12-794,08

Tabela 5.1: Intervalo de confiança de 95% para o tempo de execução dos *jobs*

Aplicando o método estatístico *t-test* [38] para comparar os intervalos de confiança dos experimentos, foi constatado que o tempo de execução dos *jobs* nos cenários não apresentam diferença significativa.

5.5 *Overheard* do *ad hoc* Grid na Rede

A próxima etapa foi medir o *overhead* gerado na rede pela troca de mensagens *multicast*. Para isso foram implementados *scripts* em *bash*, para coletar a estatística do tráfego *multicast* na rede como descrito na Seção 5.1.4. Inicialmente foi medido o *overheard* entre as redes LAN1 e LAN2, apresentado na Seção 5.5.1 e o *overheard* na rede local, apresentado na Seção 5.5.2.

5.5.1 *Overheard* Inter-Rede

O gráfico da Figura 5.3 mostra o fluxo de pacotes *multicast* no roteador RT1 do LSD (vide Figura 5.1 da Seção 5.1). Como esta era a única aplicação *multicast* que era executada no laboratório do LSD no momento dos experimentos, todo o fluxo de pacotes *multicast* do roteador foi produzido pelo *ad hoc* Grid. Este gráfico apresenta o tráfego referente à comunicação *multicast* entre os `Peers` das duas redes LAN1 e LAN2. Durante os experimentos, a taxa média de transmissão de pacotes foi de $11,73$ pacotes/s com um valor máximo de $13,24$ pacotes/s no início dos experimentos. Isso se deve ao fato de que no primeiro minuto além da troca de mensagem com o grupo, são enviadas mensagens IGMP para o cadastro de cada componente no grupo *multicast*.

Como as duas redes LAN1 e LAN2 estão interligadas através dos roteadores RT1 e RT2, é possível concluir que o fluxo *multicast* no roteador RT2 foi equivalente ao fluxo no RT1, visto que ele realizou apenas o encaminhamento das mensagens *multicast* de uma rede para outra.

5.5.2 *Overheard* na Rede Local

Para medir o *overhead* na rede local LAN1 e LAN2, o fluxo *multicast* foi analisado em ambas as redes como descrito na Seção 5.1.4. O gráfico da Figura 5.4 apresenta a taxa de transferência de pacotes por segundo durante a execução dos experimentos. A taxa de transferência média na rede LAN1 foi de $8,43$ pacotes por segundo. Já na rede local LAN2, a média foi de $1,49$ por segundo. Essa diferença de fluxo *multicast* entre as duas redes ocorre devido a diferença na quantidade de máquinas nas duas redes. Enquanto na rede LAN1 havia

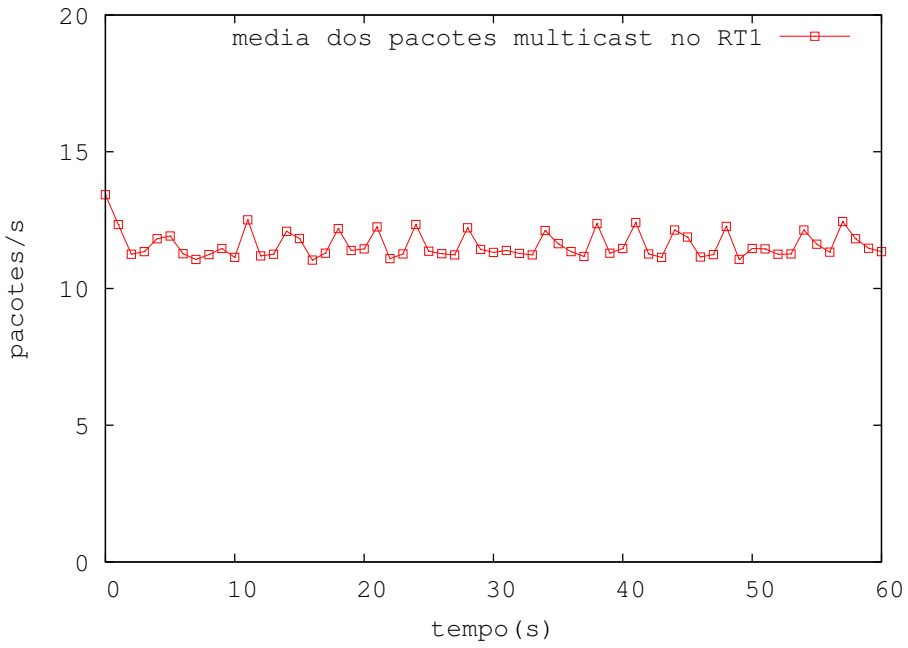

Figura 5.3: Fluxo gerado pela troca de mensagens *multicast* no roteador RT1.

25 máquinas, a rede LAN2 possuía apenas 5 máquinas. Dividindo-se a quantidade de pacotes por máquina, cada uma gera em média $0,33$ pacotes por segundo.

A partir desses valores, pode-se observar que o *overhead* gerado na rede pelo *ad hoc* Grid é muito pequeno. Considerando a media de $0,33$ pacotes por segundo, produzido por cada máquina, e o tamanho das mensagens de 2044 $bytes$, o tráfego médio gerado por cada componente na rede é de $674,52$ $bytes/s$.

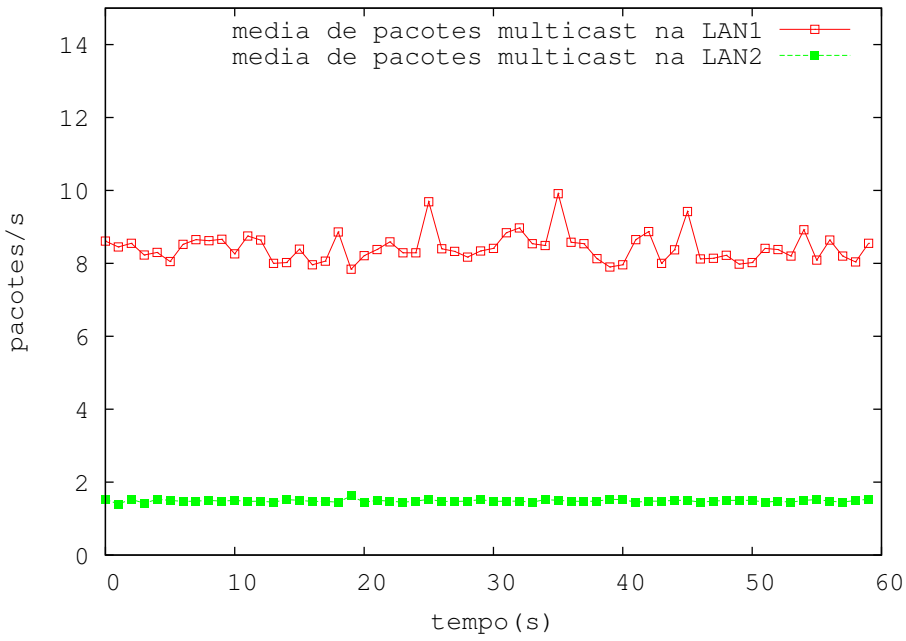

Figura 5.4: Fluxo gerado pela troca de mensagens *multicast* na LAN1 $\times$ LAN2.

Capítulo 6

Trabalhos Relacionados

Apesar da abordagem de grades computacionais ser uma área de pesquisa bastante ativa, poucos trabalhos desenvolvidos nessa área têm se dedicado ao estudo de grades auto-organizáveis e adaptativas (*ad hoc*) [10]. Neste Capítulo, é discutida a infra-estrutura de duas grades que apresentam algumas dessas características.

6.1 *The Organic Grid*

The Organic Grid [19] é uma grade descentralizada, baseada em escalonamento autônomo através de agentes móveis em uma rede P2P. Ela é uma grade de *desktops* de larga escala e possibilita que a organização da computação seja adaptada a diferentes tipos de aplicações.

Os agentes móveis de sua estrutura encapsulam o poder computacional e o comportamento da grade. Esses agentes se comunicam para obter uma visão global das mudanças do sistema e se adaptarem ao novo ambiente. Os agentes adotam algoritmos inspirados na biologia para, que de forma, independente possam escalonar as tarefas submetidas à grade aos recursos que estão disponíveis, maximizando a utilização global.

A grade é organizada em uma estrutura de árvore que é modificada constantemente para se adaptar às novas condições do sistema. Assim, cada agente representa um nó na árvore, e quando lhe é requisitado um recurso, ele o disponibiliza para o agente que requisitou e coloca este agente na árvore como nó filho. A topologia resultante da rede é uma árvore onde a máquina que iniciou a grade é a raiz.

Cada nó da árvore pode ter um número máximo de nós filhos, que são selecionados, com

base em seu desempenho. Dessa maneira, se um nó estiver extrapolado a quantidade máxima de nós filhos, o nó mais lento é removido. O desempenho do nó filho é medido através do intervalo de tempo entre o envio de dois resultados.

A Figura 6.1 apresenta uma boa configuração de uma árvore no *Organic Grid*. A qualidade da configuração da árvore é referente à proximidade dos nós com melhor desempenho (*FAST*) à raiz.

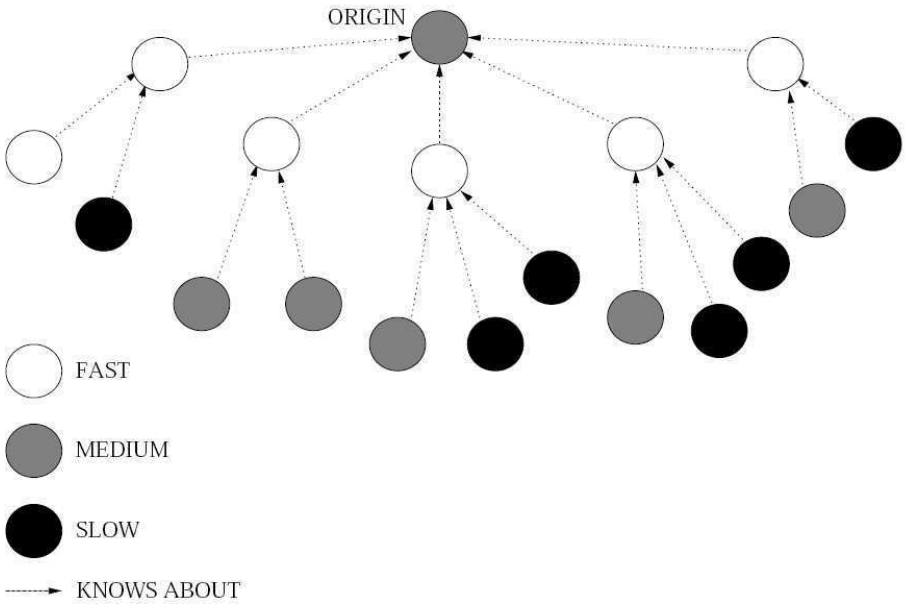

Figura 6.1: Configuração exemplo de uma árvore do Organic Grid (Chakravarti *et al.* [19]).

Avaliando o *The Organic Grid* através dos requisitos que o ambiente de grade deve ter para suportar uma grade *ad hoc* (apresentado no Capítulo 2, Seção 2.3), ele apresenta o serviço de migração de serviço, através de seus agentes móveis têm autonomia para mover-se entre os nós de uma árvore caso a disponibilidade de seus recursos seja alterada.

Os tipos de falhas considerado pelo *The Organic Grid* são quando um nó e/ou um *link* da árvore falham. Desta forma, a tolerância a falhas é implantada da seguinte forma. Cada nó da árvore mantém um lista dos ancestrais de seu nó pai (l), e caso seu nó pai demore a responder, ele o substitui por um nó ancestral da lista (l). O descobrimento de recursos é implantado através da manutenção da árvore pelos agentes. Cada recurso que é descoberto é inserido na árvore como um nó filho.

Por fim, o serviço de implantação automático desta grade pressupõem uma configuração

inicial para que ela entre em funcionamento. Essas configurações iniciais são denominadas *friend list*, que é uma lista dos nós que um agente deve procurar ao iniciar seu serviço. Portanto, pode-se observar que a configuração inicial desta grade ainda depende de um estado inicial pré definido. No *ad hoc* este serviço é automático, dado que ele possui um P2PMG que funciona como ponto de encontro entre os `Peers` da grade e um LMG que serve como ponto de encontro entre os membros de um *site* local. Através desses grupos *multicast* os componentes se comunicam e trocam as informações necessárias para a configuração incial da grade.

6.2 ICENI *Grid Middleware*

O ICENI *Grid Middleware* [30] é baseado em uma arquitetura orientada a serviço (SOA) que permite aos seus usuários compartilharem os recursos ociosos. A Figura 6.2 apresenta a arquitetura orientada a serviços, na qual o ICENI é baseado. Essa arquitetura é composta por três ações principais:

- Anúncio: O provedor de serviço (*Service Provider*) disponibiliza o serviço para o intermediador de serviços (*Service Broker*);
- Descoberta: O consumidor do serviço (*Service Consumer*) encontra um serviço através do intermediador de serviços;
- Interação: O consumidor do serviço e o provedor de serviço interagem entre si.

O intermediador de serviços no ICENI é representado por uma OV, onde usuários autorizados podem consultar e acessar seus recursos. Sempre que um recurso é anunciado e descoberto por algum consumidor, as interações são controladas por um acordo de nível de serviço (SLA), que define quais entidades possuem acesso ao serviço e por quanto tempo ele pode utilizá-lo.

O tempo de vida de um serviço do ICENI é definido em três passos. O primeiro passo é a criação do serviço a partir de uma fábrica de serviços (`IceniServiceFactory`). O próximo passo é o anúncio desse serviço em uma OV específica através do SLA (`IceniServiceAdvertizingManager`). Logo que é anunciado em uma OV, o serviço pode ser descoberto através de consultas ao provedor de serviços.

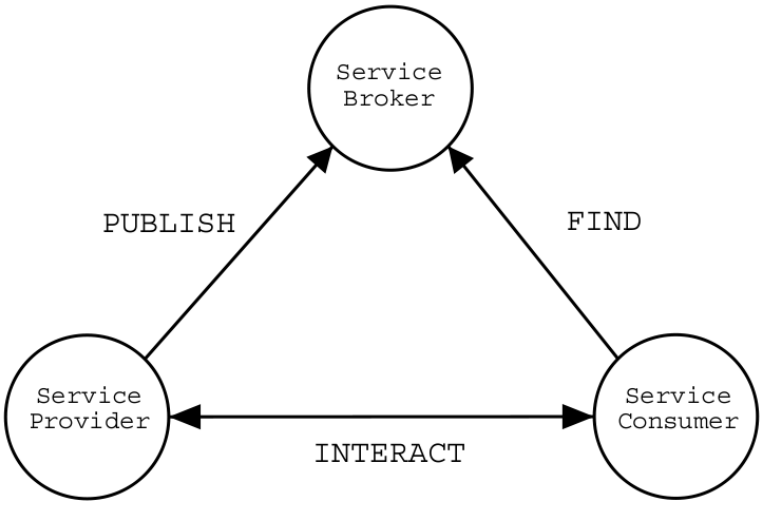

Figura 6.2: Arquitetura orientada a serviço do ICENI *Grid Middleware* (Furmento *et al.* [30]).

O ICENI *Grid Middleware* tem uma arquitetura semelhante à do OurGrid onde o provedor de serviço pode ser comparado com o `Worker` que disponibiliza seus recursos a grade, o consumidor de serviços com o `MyGrid`, e a interação do consumidor e do provedor que no OurGrid que é feita através do `Peer`. Desta forma, assim como no OurGrid, sua arquitetura não provê os serviços de implantação automática e de migração de serviço, essenciais em um ambiente de grade *ad hoc*.

Capítulo 7

Conclusão

Neste capítulo são relacionadas as considerações finais do trabalho, incluindo as conclusões obtidas através dos resultados dos experimentos, as principais dificuldades encontradas e sugestões para trabalhos futuros.

Nesta dissertação foi apresentada uma nova solução para grades computacionais P2P auto-organizáveis (i.e. *ad hoc*). Nesta solução, um grupo *multicast* age como um ponto de encontro para todos os `Peers`, provendo as informações necessárias para o serviço de descoberta do `Peer`. Há apenas um `Peer` por *site*, o qual é gerenciado através de um grupo *multicast* local. Todos os membros que não são `Peer` de um site são gerenciados pelo `Peer` local. Para garantir que o *site* local seja mantido mesmo com a falha do `Peer`, os `Peers` enviam periodicamente seu registro da NoF para todos os membros locais. No caso de falha, o membro do grupo local com maior IP assume o papel de `Peer`.

Além disso, mostrou-se que esta solução mantém a auto-organização da grade sem causar grande impacto no tempo de execução dos *jobs*, nem no tráfego da rede.

Nesta nova abordagem, foi possível prover alguns serviços para o OurGrid se tornar uma grade *ad hoc*. Dentre eles, a implantação automática, que foi possibilitada através do anúncio de cada novo componente da grade em um grupo *multicast* estabelecendo, desta forma, a comunidade e a migração de serviço do `Peer`, caso este falhe.

Dentre as dificuldades encontradas para a execução deste trabalho, está a configuração do ambiente *multicast* para os experimentos. Apesar deste tipo de comunicação estar bastante consolidado, através dos diferentes protocolos e dos diversos equipamentos que utilizam esta abordagem, a configuração dos equipamentos disponíveis para a execução dos experimentos

foi custosa e demorada. Isto porque, os *switches*, mesmo após reconfigurados não apresentaram um comportamento adequado ao utilizar o IGMP Snooping. Além disso, os roteadores do laboratório não são dedicados e, portanto, o roteamento *multicast* teve que ser implantado via *software* (i.e. XORP). Este *software* apresentou algumas falhas no momento da compilação que foram reportadas à comunidade que a desenvolve, e também exigiu um bom desempenho da máquina onde foi compilado.

Outra dificuldade encontrada, foi a adaptação do OurGrid 3.3 à versão *ad hoc*. Isto porque havia pouca documentação disponível para determinadas partes do código, exigindo assim um maior esforço no entendimento e desenvolvimento da nova solução. Soma-se a isto o fato de que durante a implementação da versão *ad hoc*, o OG 3.3 não estava mais em produção.

Alguns aspectos desse trabalho podem ser desenvolvidos e/ou aprimorados futuramente. Sugere-se, por exemplo, a investigação da consistência da rede de favores (NoF) no *ad hoc* Grid.

Outra sugestão é a implementação do tunelamento das mensagens *multicast* para o grupo MGP2P. Isto deve ser feito para que o *ad hoc* Grid possa funcionar entre redes locais distintas sem a necessidade de que os roteadores que conectam essas redes transmitam pacotes *multicast*.

Bibliografia

[1] China grid. http://www.chinagrid.com/, 2009.

[2] National energy research scientific computing center. Berkley Lab, 2009. http://www.nersc.gov/news/newsroom/IBMgrids032202.php.

[3] NodeWiz-R, 2009. http://redmine.lsd.ufcg.edu.br/wiki/nodewiz.

[4] Particle physics data grid. http://www.ppdg.net/, 2009.

[5] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[6] A. Adams, J. Nicholas, and W. Siadak. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised). RFC 3973 (Experimental), January 2005.

[7] Z. Albanna, K. Almeroth, D. Meyer, and M. Schipper. IANA Guidelines for IPv4 Multicast Address Assignments. RFC 3171 (Best Current Practice), August 2001.

[8] Francisco Brasileiro Aliandro Lima, Walfredo Cirne and Daniel Fireman. *A Case for Event-Driven Distributed Objects*, volume 4276/2006. Springer Berlin / Heidelberg, 2006.

[9] Kaizar Amin, Gregor von Laszewski, and Armin R. Mikler. Toward an architecture for ad hoc grids. In *12th International Conference on Advanced Computing and Communications (ADCOM)*, Ahmedabad Gujarat, India, 2004.

[10] Kaizar Abdulhusain Amin. *An integrated architecture for ad hoc grids*. PhD thesis, Denton, TX, USA, 2006. Adviser-Mikler, Armin R.

[11] N. Andrade, M. Mowbray, W. Cirne, and F. Brasileiro. When can an autonomous reputation scheme discourage free-riding in a peer-to-peer system? *IEEE International Symposium on Cluster Computing and the Grid CCGrid 2004*, pages 440–448, 2004.

[12] Nazareno Andrade, Francisco Brasileiro, Walfredo Cirne, and Miranda Mowbray. Discouraging free-riding in a peer-to-peer cpu-sharing grid. *Proceedings of 13th IEEE International Symposium on High-Performance Distributed Computing =(HPDC13)*, pages 4–9, 2004.

[13] Stephen Elbert Andrew Chien, Brad Calder and Karan Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, Volume 63, Issue 5:597–610, 2003.

[14] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[15] S. Bhattacharyya. An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational), July 2003.

[16] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), October 2002. Updated by RFC 4604.

[17] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[18] E. Cavalcanti, L. Assis, M. Gaudencio, W. Cirne, and F. Brasileiro. Sandboxing for a free-to-join grid with support for secure site-wide storage area. In *Proc. First International Workshop on Virtualization Technology in Distributed Computing VTDC 2006*, pages 11–11, 2006.

[19] A.J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(3):373–384, 2005.

[20] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *ACM*, 43(2):225–267, March 1996.

[21] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[22] Douglas E. Comer. *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture*. Prentice Hall, 2000.

[23] D. De Roure, N.R. Jennings, and N.R. Shadbolt. The semantic grid: Past, present, and future. *Proceedings of the IEEE*, 93(3):669–681, 2005.

[24] Troy Bryan Downing. *Java RMI: Remote Method Invocation*. Wiley, 1998.

[25] A. Duarte, F. Brasileiro, W. Cirne, and J.A. Filho. Collaborative fault diagnosis in grids through automated tests. In *Proc. 20th International Conference on Advanced Information Networking and Applications AINA 2006*, volume 1, page 6pp., 18–20 April 2006.

[26] Federal Ministry Education and Research. D-grid initiative. http://www.d-grid.de/, 2009.

[27] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2117 (Experimental), June 1997. Obsoleted by RFC 2362.

[28] Ian Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, volume 1. Morgan Kaufmann Publishers, 1 edition, 1998.

[29] Thomas Friese, Matthew Smith, and Bernd Freisleben. Hot service deployment in an ad hoc grid environment. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 75–83, New York, NY, USA, 2004. ACM.

[30] Nathalie Furmento, Jeffrey Hau, William Lee, Steven Newhouse, and John Darlington. Implementations of a service-oriented architecture on top of jini, jxta and ogsi. *Lecture Notes in Computer Science*, 3165:90–99, 2004.

[31] Felix C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM*, 31(1):1–26, March 1999.

[32] James Gosling. *The Java Language Specification*. Addison Wesley, third edition edition, 2005.

[33] NGG2 Group. Next generation grids 2: Requirements and options for european grids research 2005-2010 and beyond. Technical report, Expert Group Report, 2004.

[34] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. Bidirectional Protocol Independent Multicast (BIDIR-PIM). RFC 5015 (Proposed Standard), October 2007.

[35] HealthGrid. Egee. enabling grids for e-science in europe. Executive Summary, 2009. http://eu-egee.com/.

[36] S. Tuecke I. Foster, C. Kesselman. The anatomy of the grid: Enabling scalable virtual organizations. *Int'l J. Supercomputer Applications*, 15(3), 2001.

[37] Bart Jacob, Michael Brown, Kentaro Fukui, and Nihar Trivedi. *Introduction to Grid Computing*. RedBooks, December 2005.

[38] Raj Jain. *The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley, 1991.

[39] Bill Johnston. Nasa information power grid (ipg) infrastructure. Gloriad, August. http://www.gloriad.org/gloriad/projects/project000053.html.

[40] Gregor von Laszewski Kaizar Amin and Armin R. Mikler. Grid computing for the masses: An overview. *Lecture Notes in Computer Science*, 3033/2004:464–473, 2004.

[41] H. Kurdi, M. Li, and H. Al-Raweshidy. A classification of emerging and traditional grid systems. *IEEE distributed systems online*, 9(3):1–1, 2008.

[42] Douglas R. Mauro and Kevin J. Schmidt. *Essential SNMP*. O'Reilly, 2001.

[43] John Moy. Multicast routing extensions for ospf. *Communications of the ACM*, 37:61–66, 1994.

[44] University of California. Seti@home. http://setiathome.ssl.berkeley.edu/, 2009.

[45] Federal University of Campina Grande. Ourgrid community web site. http://www.ourgrid.org/, 2009.

[46] Vijay Pande and Stanford University. Folding@home. http://folding.stanford.edu/, 2009.

[47] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004.

[48] Manish Parashar Salim Hariri. *Tools and environments for parallel and distributed computing*. Wiley, 2004.

[49] Nicole Sergent, Xavier Défago, and André Schiper. Failure detectors: implementation issues and impact on consensus performance. Technical Report SSC/1999/019, 1999.

[50] Samba Sesay, Zongkai Yang, and Jianhua He. A survey on mobile ad hoc wireless network. *Information Technology Journal*, 2:168–175, 2004.

[51] M. Smith, T. Friese, and B. Freisleben. Towards a service-oriented ad hoc grid. In *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on*, pages 201–208, 5-7 July 2004.

[52] The XORP Team. extensible open router platform. http://www.xorp.org/, 2009.

[53] G. von Laszewski, J. Gawor, C.J. Pena, and I. Foster. Infogram: a grid service that supports both information queries and job execution. In *Proc. 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002*, pages 333–342, 2002.

[54] D. Waitzman, C. Partridge, and S.E. Deering. Distance Vector Multicast Routing Protocol. RFC 1075 (Experimental), November 1988.

[55] Lijuan Xiao Zhiwei Xu and Xingwu Liu. Personal grid. *Lecture Notes in Computer Science*, 4672/2008:536–540, 2008.

Apêndice A

Script de Inicialização do *ad hoc* Grid

Código Fonte A.1: *Script* de inicialização do *ad hoc* Grid

```
1  #!/bin/bash
2  #
3   # ------------------------------------------------------
4   #
5   # Package:              Adhoc Grid
6   #
7   # Description:  This shell script takes care of starting and stopping Ad
        Hoc Grid.
8   #
9   # Copyright (c) 2007-2009 Universidade Federal de Campina Grande
10  #
11  # ------------------------------------------------------
12
13
14 OK=0
15 NOK=1
16
17 echo "Ad hoc Grid"
18
19 rm adhocgridConfig/.mygrid/*.*
20
21 print_commands() {
22
23          echo "Adhoc grid comands are:"
```

```
24          echo "  start                                   Start Adhoc grid
                on the local machine"
25          echo "  stop                                    Stop Adhoc on the
                 local machine"
26          echo "          -mg                                     Start Adhoc
                 Grid with MyGrid"
27          echo "          -mgGui                                  Start Adhoc
                 Grid with MyGrid GUI"
28  }
29
30
31
32  start() {
33          cd /home/pablotiburcio/workspace/adhocgrid/bin
34          java -classpath .:lib/aspectjrt.jar:lib/cglib.jar:lib/easymock.
                jar:lib/easymockclassextension.jar:lib/junit.jar:lib/log4j.jar
                :lib/servlet.jar $commandmanager $startcommand $COMMAND_ARGS
35
36  }
37
38  stop() {
39          java -classpath .:lib/aspectjrt.jar:lib/cglib.jar:lib/easymock.
                jar:lib/easymockclassextension.jar:lib/junit.jar:lib/log4j.jar
                :lib/servlet.jar org.ourgrid.adhoc.ComponentStartUp
40
41  }
42
43  usage(){
44
45  if [ $# -eq 0 ]; then
46          echo "Usage: adhocgrid command [command-options-and-arguments]"
47          echo
48          print_commands
49          echo
50          echo "command-options-and-arguments depends on the specific
                command"
51          echo
```

```
52 else
53         case "$1" in
54                 start) echo "Usage: Adhoc grid start"
55                 ;;
56                 stop) echo "Usage: Adhoc grid stop"
57                 ;;
58                 *) echo "no implemented"
59         esac
60 fi
61 exit $ERR_STD_MISUSE
62 }
63
64 usage_commands(){
65
66
67         UNK_CMD=$1
68
69         echo "Adhoc grid comands are:"
70         echo "  start                                   Start Adhoc grid
              on the local machine"
71         echo "  stop                                    Stop Adhoc grid
              on the local machine"
72         echo "  status                                  Show status of
              Adhoc"
73         echo "          -mg                                     Start Adhoc
               Grid with MyGrid"
74         echo "          -mgGui                                  Start Adhoc
               Grid with MyGrid GUI"
75
76         exit $ERR_STD_MISUSE
77
78 }
79
80
81 nargs=$#
82
83 if [ $nargs -eq 0 ]; then
```

```
84          COMMAND_NAME="help"
85  fi
86
87  if [ "$1" = "start" ]; then
88          if [ $nargs -eq 1 ]; then
89                  COMMAND_NAME=$1
90                  COMMAND_ARGS="-no"
91          else
92                  if [ $nargs -eq 2 ]; then
93                          if [ "$2" = "-mg" ] || [ "$2" = "-mgGui" ]; then
94                                  COMMAND_NAME=$1
95                                  COMMAND_ARGS=$2
96                                  shift
97                          else
98                                  usage $1
99                          fi
100                 else
101                         usage $1
102                 fi
103         fi
104 fi
105
106 if [ "$1" = "stop" ] || [ "$1" = "status" ]; then
107         if [ $nargs -eq 1 ]; then
108                 COMMAND_NAME=$1
109         else
110                 usage $1
111         fi
112 fi
113
114 if [ -z "$COMMAND_NAME" ]; then
115                 usage_commands $1
116 fi
117
118 commandmanager="org.ourgrid.adhoc.CommandManager"
119 startcommand="start"
120
```

```
121 ret=$?
122
123 if [ $ret -eq $OK ]; then
124
125         case "$COMMAND_NAME" in
126                 start)
127                         start
128                 ;;
129                 stop)
130                         #stop
131                 ;;
132                 status)
133                         #status
134                 ;;
135         esac
136 fi
```

Apêndice B

Arquivo de Configuração do XORP

Código Fonte B.1: Arquivo de configuração do XORP

```
1  /* $XORP: xorp/rtrmgr/config.boot.sample,v 1.23 2005/03/09 22:50:41
       pavlin Exp $ */
2
3  interfaces {
4
5      restore-original-config-on-shutdown: false
6
7      interface eth0 {
8          default-system-config
9          disable: false
10     }
11     interface eth1 {
12         default-system-config
13         disable: false
14     }
15 }
16
17 fea {
18     unicast-forwarding4 {
19         disable: false
20     }
21 }
22
23 plumbing {
```

```
24      mfea4 {
25          interface eth0 {
26              vif eth0 {
27                  disable: false
28              }
29          }
30          interface eth1 {
31              vif eth1 {
32                  disable: false
33              }
34          }
35          interface register_vif {
36              vif register_vif {
37                  /* Note: this vif should be always enabled */
38                  disable: false
39              }
40          }
41          traceoptions {
42              flag all {
43                  disable: false
44              }
45          }
46      }
47  }
48
49  protocols {
50      igmp {
51          interface eth0 {
52              vif eth1 {
53                  disable: false
54              }
55          }
56          interface eth1 {
57              vif eth1 {
58                  disable: false
59              }
60          }
```

```
61          traceoptions {
62              flag all {
63                  disable: false
64              }
65          }
66      }
67  }
68
69  protocols {
70      pimsm4 {
71          disable: false
72          interface eth0 {
73              vif eth0 {
74                  disable: false
75              }
76          }
77          interface eth1 {
78              vif eth1 {
79                  disable: false
80              }
81          }
82          interface register_vif {
83              vif register_vif {
84                  /* Note: this vif should be always enabled */
85                  disable: false
86              }
87          }
88
89          static-rps {
90              rp 150.165.85.189 {
91                  group-prefix 224.0.0.0/4 {
92                      rp-priority: 192
93                      hash-mask-len: 30
94                  }
95              }
96          }
97
```

```
98          traceoptions {
99              flag all {
100                 disable: false
101             }
102         }
103     }
104 }
105
106 protocols {
107     fib2mrib {
108         disable: false
109     }
110 }
```