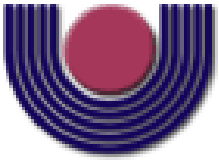

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Informática

Curso de Bacharelado em Informática

Monitoramento de Recursos Distribuídos em Grades Computacionais

Gustavo Mantovani

CASCAVEL

2009

GUSTAVO MANTOVANI

MONITORAMENTO DE RECURSOS DISTRIBUÍDOS EM GRADES COMPUTACIONAIS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel.

Orientador: Prof. Msc. Luiz Antonio Rodrigues

CASCAVEL

2009

GUSTAVO MANTOVANI

MONITORAMENTO DE RECURSOS DISTRIBUÍDOS EM GRADES COMPUTACIONAIS

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Informática*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Msc. Luiz Antonio Rodrigues (Orientador)

Colegiado de Informática, UNIOESTE

Prof. Dr. Clodis Boscarioli

Colegiado de Informática, UNIOESTE

Prof. Msc. Guilherme Galante

Colegiado de Informática, UNIOESTE

Cascavel, 10 de Dezembro de 2009.

AGRADECIMENTOS

Primeiramente a Deus pelo dom da vida e pelos sonhos que se concretizam, como este que agora se torna realidade.

Ao meu orientador, professor Luiz Antonio Rodrigues, por toda disponiblidade, colaboração e ajuda dispensada ao longo da realização deste trabalho.

Aos membros da banca, professores Guilherme Galante e Clodis Boscarioli, pelas idéias e sugestões que influenciaram no resultado final do trabalho.

Aos colegas que estiver presentes em todas as etapas desta caminhada, em especial ao Orlando Luiz Pelossi Junior que perdeu tardes me auxiliando na configuração da ferramenta OurGrid.

À equipe do OurGrid da Universidade de Campina Grande, em especial ao professor Rodrigo de Almeida Vilar de Miranda pelo suporte e auxílio prestados.

À minha família, por tudo que me ensinaram durante a vida me tornando uma pessoa de caráter, em especial à minha mãe e à namorada por terem aguentado meus momentos de *stress* e mesmo assim estiveram sempre ao meu lado dando força.

A todos vocês, meu Muito Obrigado!!

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BoT	*Bag of Tasks*
GMA	*Grid Monitoring Architecture*
Gmetad	*Ganglia Meta Daemon*
Gmetric	*Ganglia Metric Client*
Gmond	*Ganglia Monitoring Daemon*
JDF	*Job Description File*
Monalisa	*MONitoring Agents in a Large Integrated Services Architecture*
R-GMA	*Relational Grid Monitoring Architecture*
RRDtoll	*Round Robin Database*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
UFCG	Universidade Federal de Campina Grande
XDR	*Extreme Data Representation*
XML	*Extensible Markup Language*
SGBD	Sistema Gerenciador de Banco de Dados
SO	Sistema Operacional
WQR	*Work Queue with Replication*

Sumário

Resumo

Os sistemas de Grades computacionais surgiram na década de 90 como uma alternativa que melhoraria o desempenho de aplicações paralelas, por permitir a utilização simultânea de diversos recursos disponíveis em um ambiente distribuído. Recursos como ciclos de processamento, armazenamento e compartilhamento de dados, entre outros, podem ser utilizados de forma compartilhada gerando um ambiente computacional de alto desempenho. A fim de aperfeiçoar o uso das Grades Computacionais surgiram sistemas capazes de monitorar os recursos disponíveis nos nós da Grade e disponibilizar essas informações aos usuários do sistema. É imprescindível que haja uma forma de visualizar o estado global do ambiente para viabilizar a detecção de falhas e o gerenciamento do seu desempenho. Dado isto, o presente trabalho tem por objetivo contribuir com as pesquisas na área de Grades computacionais, testando ferramentas *open-source* capazes de monitorar os recursos distribuídos em um ambiente de Grade e mostrando como o escalonador desta pode prevenir erros e ser otimizado utilizando os dados coletados por estas ferramentas de monitoramento.

Palavras-chave: Grades Computacionais, Ambiente Distribuído, Sistemas de Monitoramento Distribuído.

Capítulo 1

Introdução

A necessidade de utilizar uma quantidade muito elevada de recursos computacionais apresentada por algumas áreas de conhecimento, que têm por objetivo realizar cálculos extremamente complexos e repetitivos, fez com que projetistas de sistemas computacionais desenvolvessem uma forma de compartilhamento de recursos que suprisse tais necessidades [1].

O notável desenvolvimento das tecnologias de redes de computadores facilitou a interconectividade e a comunicação entre computadores. Desde então, pesquisadores vêm propondo novas arquiteturas de organização dessas redes, a fim de possibilitar o compartilhamento de seus recursos. O objetivo passa a ser interconectar diversos computadores e fazer com que estes trabalhem em conjunto, somando seus recursos, de forma que se configurem como um ambiente computacional único [2].

A partir dessa idéia, surgiram os clusters computacionais, cuja estrutura consiste em um conjunto de computadores chamados nós, interconectados e funcionando como um supercomputador com alta capacidade de processamento.

Após o sucesso dos clusters e o surpreendente aumento no número de computadores conectados através da internet, passou-se a pensar em um novo modelo computacional denominado de grades computacionais – *Grid Computing*. A idéia era propor uma tecnologia capaz de interligar diversos nós geograficamente dispersos, compartilhando seus recursos, e fazendo com que estes trabalhassem como um supercomputador, tornando possível a criação de um cluster de clusters [2].

No entanto, os nós de uma Grade podem estar operando com plataformas de hardware e sistemas operacionais variados e distribuídos em diferentes instituições [4], o que torna difícil o monitoramento de seus recursos.

Neste sentido, esta pesquisa se propõe a monitorar os recursos de uma Grade utilizando ferramentas de monitoramento distribuído. Tratam-se de sistemas de monitoramento utilizados em ambientes computacionais de alto desempenho, capazes de fornecer informações referentes aos recursos disponíveis em cada nó conectado à Grade, visando aumentar a usabilidade dos mesmos.

1.1 Justificativa

Um dos grandes desafios presentes nos sistemas computacionais distribuídos é o monitoramento dos nós, nos quais a heterogeneidade de *hardware* e de *software* é cada vez mais freqüente, dificultando o seu gerenciamento.

Uma intensa atividade de pesquisas que tem por objetvio buscar formas eficientes de escalonar aplicações paralelas em ambientes de Grades computacionais já foram realizadas [28], [29], [30]. O interesse dos pesquisadores é despertado uma vez que os resultados obtidos vêem indicando melhoras significativas no desempenho dos escalonadores das Grades computacionais.

Sendo assim, o presente projeto visa intensificar as pesquisas na área de Grades computacionais, aplicando ferramentas capazes de monitorar os recursos distribuídos nas máquinas, o que pode prevenir falhas, auxiliar no gerenciamento e melhorar o desempenho das aplicações paralelas, mostrando como é possível otimizar o escalonador de tarefas utilizando informações obtidas pelo sistema de monitoramento.

1.2 Objetivos

O objetivo geral deste trabalho é monitorar a utilização de recursos distribuídos em uma Grade computacional, utilizando ferramentas de monitoramento distribuído e mostrar como estes serviços podem ser úteis na otimização do escalonador da Grade.

1.2.1 Objetivos Específicos

- Estudar a tecnologia de computação em Grade;

- Instalar e configurar um ambiente de Grade Computacional;
- Instalar e configurar a ferramenta GANGLIA na Grade e realizar testes a fim de avaliar seu desempenho;
- Monitorar a utilização dos recursos disponíveis na Grade.
- Mostrar como as informações obtidas com um sistema de monitoramento podem auxiliar o escalonador da Grade na tomada de decisões.

1.3 Organização do Texto

Este trabalho está organizado da seguinte forma.

O Capítulo 2 aborda Grades computacionais e seus principais conceitos, características e benefícios de se compartilhar recursos. É visto também a arquitetura básica de uma Grade computacional, as tecnologias de Grades mais conhecidas, sendo explanada de forma mais aprofundada a tecnologia OurGrid que será utilizada no decorrer dessa monografia.

O Capítulo 3 refere-se ao monitoramento de recursos distribuídos em Grades computacionais. Primeiramente é apresentada uma introdução ao monitoramento de recursos, seguida dos principais requerimentos que o sistema deve prover para ter um funcionamento pleno. Por fim, são citadas as principais ferramentas de monitoramento de Grades sendo que o sistema Ganglia é explicado de forma mais completa por ser o escolhido para efetuar os testes no presente trabalho.

O Capítulo 4 apresenta os resultados dos testes realizados com as ferramentas. Inicialmente é mostrado o ambiente onde a Grade foi implantada e como funciona o sistema OurGrid. Após isto é explicado como foi feita a instalação e configuração da ferramenta de monitoramento no ambiente distribuído. Por fim, a partir dos erros gerados na execução das tarefas, é mostrado como um sistema de monitoramento poderia ser util na prevenção destes erros e na otimização do escalonador.

Finalmente, no Capítulo 5 encontram-se as considerações finais do trabalho e as perspectivas de trabalhos futuros.

Capítulo 2

Grades Computacionais

A tecnologia de computação em Grade surgiu em meados da década de 90, desenvolvida inicialmente em universidades e institutos de pesquisa. Logo, ganhou o mundo empresarial, tornando-se parte da estratégia das mais famosas empresas na área de tecnologia, como IBM, HP, Sun, NEC, Microsoft e Oracle [3].

A proposta era viabilizar a conexão de diversos clusters e nós independentes geograficamente dispersos, conectados através da Internet, criando a ilusão de um único computador virtual facilmente gerenciável [4]. Tal proposta visava fornecer uma plataforma mais barata e acessível que os supercomputadores, capaz de executar aplicações distribuídas fazendo uso de recursos aglomerados [3].

Empresas e organizações necessitam processar enormes quantidades de dados e possuem dezenas de computadores ociosos. A tecnologia de Grade propõe resolver o problema de falta de processamento fazendo uso dos recursos dessas máquinas, sem necessitar de um supercomputador [4].

O conceito de Grade Computacional é análogo ao das redes de energia elétrica, que visa fornecer energia de forma eficiente, atendendo solicitações de qualquer usuário que necessite dela. A Grade deverá fornecer recursos computacionais aos usuários que nela estejam conectados da mesma forma que a rede elétrica, podendo estes estar situado em qualquer ponto do planeta [8].

A Grade deve se portar perante o usuário como um recurso computacional unificado, sendo responsabilidade dos programas gerenciadores distribuir a execução das aplicações entre os recursos disponíveis, executando de forma rápida e segura. A idéia é que o usuário não tenha que se preocupar com a alocação dos recursos de sua aplicação, isto é, a distribuição deverá ocorrer automaticamente.

Atualmente, os pesquisadores da área estudam formas de aperfeiçoar o gerenciamento das aplicações paralelas e dos recursos oferecidos pela grade, aumentando a eficiência do serviço e sua usabilidade, prevendo que esta pode ser uma das grandes evoluções da computação [4].

Devido ao fato de se tratar de um assunto relativamente recente, ainda não existe um conceito padrão para Grades Computacionais, nem mesmo uma padronização de *Software*. No mercado são encontradas diferentes soluções.

2.1 Compartilhamento de Recursos

Uma Grade é uma coleção de máquinas compartilhando diversos tipos de recursos. O compartilhamento de recursos ociosos em computadores pessoais gera um ambiente computacional extremamente potente, semelhante a um supercomputador. Os principais tipos de recursos a serem compartilhados são:

- Ciclos de Processamento: São os recursos mais solicitados em uma Grade. Ciclos ociosos de alguma máquina podem ser utilizados para executar processamento externo [9]. Os processadores variam em termos de arquitetura e velocidade de processamento. Sendo assim, é possível alocar um processo para ser executado em apenas uma máquina da Grade ou dividir a tarefa em partes menores para que estas sejam executadas paralelamente. Grades que têm como principal funcionalidade a utilização de ciclos de processamento são comumente chamadas de *grids computacionais* [4].

- Armazenamento: A capacidade de armazenamento é outro recurso que compartilhado tem um rendimento muito elevado, pois utilizar o espaço disponível em cada máquina como se fosse um único sistema de arquivos aumenta a capacidade de armazenamento como um todo e auxilia na localização de um determinado arquivo, o qual pode estar em um único nó ou fragmentado entre diversos nós da Grade. Grades que utilizam a capacidade de armazenamento como principal recurso compartilhado são comumente chamadas de *data grids* [16].

• Software: Como alguns softwares possuem licenças caras torna-se inviável sua instalação em todas as máquinas da Grade. Assim, as requisições para utilizar estes softwares devem ser direcionadas para as máquinas que já possuam estas licenças.

• Comunicação: A velocidade com que as máquinas se comunicam pode influenciar no desempenho da Grade. Desta forma, pode-se fazer uso de um compartilhamento de conexões. Algumas aplicações precisam processar uma grande quantidade de dados e estes normalmente não se encontram na máquina onde a aplicação será executada. Nesta situação torna-se necessário o envio desses dados através da Grade junto com seu executável, sendo que a velocidade desta transmissão influenciará no desempenho. Máquinas com conexões ociosas podem ser utilizadas para enviar parte destes dados, aumentando a taxa de envio e evitando a contenção. Grades que têm como principal finalidade fornecer um alto desempenho no quesito comunicação são conhecidas como *network grids.*

2.2 Arquitetura de uma Grade Computacional

Como citado, Grades são ambientes de computação fortemente distribuídos e formados pela integração de nós independentes dispersos. A arquitetura de uma Grade define as funções de seus componentes fundamentais e quais as interações que ocorrem entre eles.

A arquitetura de uma Grade é constituída de serviços e protocolos implementados e separados em uma série de camadas, ilustradas na Figura 2.1. Cada uma dessas camadas está detalhada nas próximas seções.

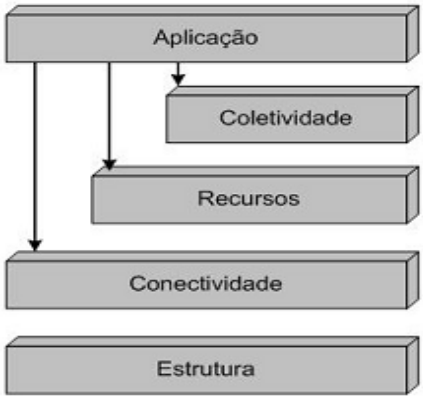

Figura 2.1: Modelo arquitetural de uma Grade Genérica [15].

2.2.1 Estrutura: Interfaces para controle local

A camada de estrutura é responsável pelo compartilhamento de recursos e também por mecanismos que controlam o acesso a estes recursos. Para que sejam utilizados ciclos de clock na Grade é necessário que os programas sejam inicializados e a execução de seus processos seja monitorada. No caso de recursos de armazenamento, é necessário que os dados sejam inseridos ou retirados dos dispositivos de armazenamento. Este controle é feito na camada de estrutura.

Os recursos precisam ter implementados dois mecanismos fundamentais para seu melhor aproveitamento na Grade [15]. Um mecanismo de consulta, que permite ao usuário obter informações sobre a estrutura, estado e capacidade dos recursos. E, um mecanismo de gerenciamento, que torna possível algum tipo de controle sobre a qualidade do serviço.

Para cada tipo de recurso compartilhado na Grade, a camada de estrutura deve implementar algumas funções específicas [15]:

- Recursos Computacionais: Necessitam funções capazes de inicializar os programas e monitorar sua execução. Mecanismos de gerenciamento sobre os recursos alocados a uma determinada aplicação também são importantes. E, mecanismos de consulta capazes de fornecer informações sobre características de hardware e software.

- Recursos de Armazenamento: São necessárias funções que permitam adicionar e retirar dados das bases disponíveis. Necessitam também de mecanismos de gestão, que permitem um controle e fornecem informações sobre os recursos alocados para transferência de dados, como espaço, taxa de transferência do disco, taxa de transferência da rede e ciclos de processamento.

- Recursos de Rede: Necessitam funções que tornem possível um gerenciamento sobre os recursos alocados para transferências de rede e disponibilizem informações sobre a taxa de transferência.

2.2.2 Conectividade: Comunicação segura

A camada de conectividade possui a responsabilidade de definir os protocolos de comunicação e autenticação. É nesta camada que ocorre a transferência dos dados e a filtragem dos mesmos, determinando quais dados chegarão à camada inferior.

Os serviços de autenticação, implementado por alguns protocolos da camada de conectividade, possibilitam uma comunicação criptografada tornando segura a identificação de usuários e recursos, aumentando assim a confiabilidade na comunicação. Já os protocolos de comunicação são responsáveis pelo transporte e roteamento das informações.

Algumas características básicas devem ser alcançadas pelas soluções de autenticação [15].

- Assinatura única: A autenticação dos usuários deve ser feita apenas uma vez, obtendo assim acesso aos múltiplos recursos da Grade, definidos na camada de Estrutura, não é necessário efetuar uma autenticação para cada tipo de recurso.
- Delegação: Um programa executado está apto a utilizar os mesmos recursos que o seu usuário. E, um programa pode também delegar alguns direitos de acesso a outro programa em execução.
- Integração com diversas soluções de segurança local: Cada recurso é capaz de prover soluções de segurança local. As soluções de segurança da Grade devem ser capazes de interagir com essas diversas soluções locais.
- Relações confiáveis entre usuários: Para que um usuário possa utilizar os recursos de vários fornecedores em conjunto, o sistema de segurança não deve exigir que cada um dos fornecedores interaja com os outros na configuração do ambiente de segurança. Por exemplo, se um usuário está apto a utilizar os recursos A e B, o usuário deve ser capaz de utilizar os recursos A e B juntos, sem necessitar que os administradores dos sistemas interajam [15].

2.2.3 Recursos: Compartilhamento simples de recursos

Na camada de recursos é implementado um conjunto de regras responsáveis pela negociação entre a camada superior e a inferior, para que estas interajam de forma correta e segura. É feito também um monitoramento de quanto tempo cada usuário utiliza um determinado recurso [15].

São citadas duas classes de protocolos implementados na camada de recursos como sendo os principais:

- Protocolo de informação, designado a obter informações sobre o estado dos recursos.

- Protocolos de gerenciamento, responsáveis pelo ajuste das regras de acesso aos recursos.

2.2.4 Coletividade: Coordenando recursos múltiplos

A camada de coletividade tem como função implementar soluções para que uma porção de recursos trabalhe de forma correta, diferente da camada de recursos que trata cada recurso separadamente. Para tal função, ela disponibiliza algumas facilidades de compartilhamento [15].

- Serviços de diretório: Fazem com que as organizações virtuais da Grade descubram mais facilmente a existência e as propriedades dos recursos compartilhados. Podendo ser estes recursos localizados por nome, tipo, carga, disponibilidade entre outros atributos [17].

- Serviços de alocação e agendamento: Permite que os usuários da Grade aloquem recursos para uma finalidade específica e agendem tarefas em recursos apropriados.

- Serviços de monitoramento e diagnóstico: Monitora e dá suporte caso ocorram falhas nos recursos. Por exemplo: quando múltiplos usuários estão tentando acessar o mesmo recurso, esta utilização indevida pode gerar uma sobrecarga no sistema.

- Serviços de replicação de dados: Dá suporte e gerencia os recursos de armazenamento, a fim de tornar mais eficiente o acesso aos dados. Dentre os recursos que podem ser monitorados estão o tempo de resposta, a confiabilidade e o custo [18].

Estes exemplos ilustram apenas alguns serviços dentre a grande variedade de protocolos encontrados na camada de coletividade.

Alguns componentes desta camada podem ser específicos para determinada organização virtual ou aplicação. Por exemplo, um protocolo implementado para alocar um conjunto específico de recursos de rede que uma organização possui. Outros componentes de coletividade podem ser mais generalizados, por exemplo, um serviço que gerencia uma coleção de dispositivos de armazenamento para diversas organizações virtuais. Em geral, o

importante é que os protocolos sejam baseados em padrões abertos, evitando assim problemas de integração.

2.2.5 Aplicação

O desenvolvimento das aplicações tem que estar de acordo com os serviços implementados nas outras camadas da arquitetura. As ferramentas dessa camada devem possibilitar que o usuário efetue suas aplicações de forma eficiente no ambiente de grade e também mantenham conexão com a Grade, principalmente se estiverem executando alguma aplicação.

2.3 Plataformas de Grades

Existe atualmente várias ferramentas de *Software* como Globus [15], [20], Condor [21] e OurGrid [22], sendo desenvolvidas por diversos grupos de pesquisa e instituições comerciais para operarem em plataformas de Grades. Esta seção descreverá as plataformas mais conhecidas e utilizadas. No entanto, serão explicadas mais detalhadamente as características do OurGrid, que é a plataforma escolhida para testar os *softwares* de monitoramento.

2.3.1 Globus

O projeto Globus foi iniciado em 1997 com o objetivo de desenvolver um pacote de software de código aberto, disponível a qualquer usuário que deseja fazer uso de suas funcionalidades. Com o surgimento do *Globus Toolkit* [15], [20], o projeto obteve uma forte expansão, contando com a participação de renomadas empresas da área da computação, como IBM e HP, as quais tinham interesse na tecnologia de Grades.

Globus provê um conjunto de serviços desenvolvidos a fim de facilitar a utilização da computação em Grades, podendo estes ser usados na submissão e no controle de aplicações, movimentação de dados, descoberta de recursos e na segurança da Grade.

Os serviços supracitados foram desenvolvidos em linguagem C, tendo seu código fonte aberto para que colaboradores possam reparar erros ou até mesmo acrescentar linhas de código, com o objetivo de enriquecer o sistema [19].

O sistema oferece um ambiente seguro, possibilitando aos clientes acessar remotamente recursos computacionais e de armazenamento em uma simples máquina virtual. Trata-se de uma infra-estrutura de hardware e software capaz de prover a usuários dispersos acesso independente e consistente aos recursos computacionais [20].

Para os usuários, os serviços do sistema tornam-se transparentes depois de serem instalados e configurados. As funções são executadas de forma automática, fazendo uso de comandos específicos do Globus para tal. Segundo Cirne e Neto [3], um dos motivos que levou o Globus a obter uma grande aceitação é que os serviços oferecidos são razoavelmente independentes, tornando possível a utilização de apenas parte dos serviços em uma determinada solução.

2.3.2 Condor

O projeto Condor teve início em 1985 e foi desenvolvido na Universidade de Wisconsin, dando seqüência a um outro projeto chamado Remote Unix, sendo certamente o projeto mais antigo na área da computação em Grade. O Condor tem como usuário alvo aquele que necessita de um grande poder computacional por um longo período de tempo, como dias, meses ou até anos [31].

Sua arquitetura é formada por diversos *Condor Pools* que são grupos de computadores interconectados. O principal elemento do Condor Pool é conhecido como *Matchmaker*, que tem a responsabilidade de alocar tarefas, levando em consideração as necessidades de cada tarefa e os recursos de cada máquina. O usuário é quem especifica as necessidades de cada tarefa no momento em que as submete. Já as restrições de uso das máquinas são especificadas pelo seu proprietário no momento em que a mesma é incluída no sistema. São essas restrições que determinam como cada máquina será utilizada pelo *Condor Pool*, dando a opção ao proprietário de fornecer recursos apenas quando a máquina estiver ociosa e cancelar as aplicações em execução do Condor quando precisar usar a máquina [21].

Uma funcionalidade bastante interessante do Condor é um mecanismo chamado de *checkpointing*, usado para salvar o estado de execução de uma determinada tarefa após esta ter sido interrompida por algum motivo específico. O *checkpointing* resolve o problema da

interrupção permitindo que a tarefa seja executada em outra máquina a partir do ponto onde ela parou, tornando possível a execução em etapas.

É fácil entender como o Condor funciona [21]:

- Os proprietários incluem suas máquinas no sistema especificando suas restrições e anunciando seus recursos.
- Quando um usuário submete uma aplicação o *Matchmaker* aloca a tarefa às máquinas que possuem recursos disponíveis.
- No caso da máquina ser solicitada pelo seu proprietário, a tarefa é alocada em outro nó.

2.3.3 OurGrid

A plataforma OurGrid é um sistema de Grade desenvolvido no Brasil desde 2004 pela Universidade Federal de Campina Grande (UFCG) em parceria com a HP Brasil, tendo por base os conceitos de *sites* que executam aplicações modeladas como *BoT – Bag of Tasks* [22].

O sistema foi baseado em quatro grandes objetivos que nortearam os desenvolvedores do OurGrid [22]:

• Rapidez: O tempo de resposta de uma tarefa executada na Grade deve ser mais eficiente do que o tempo de resposta de uma tarefa executada utilizando os recursos locais, de outra forma não haverá necessidade de se utilizar a Grade.

• Simplicidade: É necessário que a utilização da Grade seja simples, pois os usuários querem gastar o mínimo de esforço possível para se adaptar a tecnologia que irá resolver o seu problema, tendo mais tempo para estudar a solução apresentada.

• Escalabilidade: Necessita ser escalável para disponibilizar da melhor forma o imenso poder computacional ocioso nas máquinas conectadas e espalhadas pelo mundo todo. A escalabilidade não é apenas uma questão técnica, ela também tem suas ramificações administrativas. Pois a Grade é aberta e de livre acesso não sendo viável ter de passar por permissões humanas para definir quem tem acesso a o quê, quando e como.

• Segurança: A segurança é um fator fundamental, pois trata-se uma conexão automática *peer-to-peer* de acesso gratuito que permitirá que qualquer pessoa acesse

os recursos e executem códigos nas máquinas, possibilitando assim atitudes maldosas por parte de alguns usuários mal intencionados.

Estes objetivos não são fáceis de atingir. Para tanto, os desenvolvedores do OurGrid simplificaram o problema, executando no sistema apenas aplicações no formato *BoT*.

BoT são aplicações paralelas, onde as tarefas são independentes e podem ser finalizadas com sucesso sem ter de se comunicar com outros integrantes da Grade. Apesar das *BoT* serem aplicações simples, são utilizadas em diversos cenários, como mineração de dados, pesquisas massivas (quebra de chave), simulações e renderização de imagens.

Tendo em vista a escalabilidade do projeto, qualquer pessoa pode fazer parte deste gratuitamente, executando suas aplicações paralelas. O poder computacional da plataforma é obtido por meio de recursos ociosos nas máquinas participantes, sendo que quem disponibiliza mais recursos tem uma prioridade maior na fila de acesso a recursos para rodar suas aplicações.

A opção por esta ferramenta para o desenvolvimento do trabalho deu-se pelo fato da mesma ser 100% nacional, por suportar as ferramentas a serem testadas e por já ser previamente conhecida e estudada, facilitando assim o desenvolvimento do trabalho.

2.3.3.1 Arquitetura do OurGrid

A arquitetura do OurGrid é dividida em três componentes [22].

- *Broker:* Funciona como uma interface entre o usuário que necessita de recursos e a Grade. É responsável pela submissão e sincronização das tarefas na Grade.

- *Peer:* É responsável por encontrar e controlar os recursos disponíveis na Grade e disponibilizá-los ao *Broker*. Todas as requisições feitas por um usuário através do *Broker*, são encaminhadas para o *Peer*, que por sua vez, encontra e disponibliza os recursos necessários.

- *Worker:* Tem por objetivo executar as tarefas submetidas pelo *Broker* nas máquinas finais da Grade. É o software em execução dentro das máquinas ociosas, fornecendo uma interface segura para execução de aplicações e submissão de arquivos.

A arquitetura do OurGrid é ilustrada na Figura 2.2. Neste exemplo, cada site representa uma instituição ligada a grade. Os *Peers* são alocados em cada um dos sites e disponibilizam aos *Brokers* a informação de quais *Workers* estão disponíveis. Através dos *Brokers*, o usuário pode submeter suas tarefas e coletar os resultados.

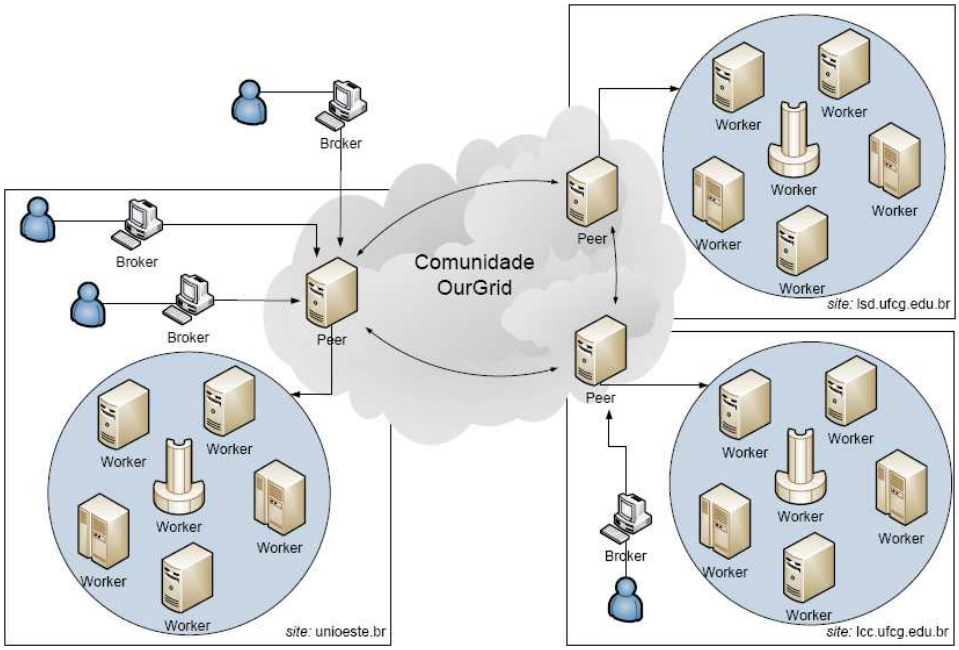

Figura 2.2: Modelo arquitetural do OurGrid [21].

2.3.3.2 Execução de Aplicações no OurGrid

Previamente foi dito que para executar uma aplicação no OurGrid esta deve estar modelada como *BoT*. As tarefas independentes de uma aplicação são chamadas de *Job* que em português significa trabalho e é submetido ao OurGrid por meio da interface do *Broker*.

Para submeter um *Job* a Grade, é necessário detalhar o que a aplicação deverá fazer. Isto é feito em um arquivo descritor de tarefas chamado de *JDF - Job Description File*. O JDF pode ser criado em qualquer editor de texto e descreverá um único Job.

Cada tarefa possui quatro cláusulas [22]:

- *Label:* Cláusula onde deve ser descrito o nome do Job.

- *Init:* É a cláusula inicial que descreve as ações necessárias a fim de preparar o *Worker* que irá receber a tarefa. É utilizada normalmente para definir quais os arquivos que serão transferidos da máquina do usuário para o *Worker*.

• *Remote:* É a cláusula remota que descreve o conjunto de comandos que serão executados pelo *Worker*, gerando assim a saída da tarefa.

• *Final:* É a cláusula final que descreve quais arquivos serão transferidos do *Worker* para a máquina do usuário.

O Figura 2.3 ilustra um *Job* composto pelas quatro clausulas supracitadas sendo submetidas ao OurGrid. Esta tarefa, denominada em *label* como myjob, envia por meio da cláusula *init* um program.txt que será executado em um *Worker* da Grade. Na cláusula *remote* estão descritos os comandos que o *Worker* irá executar, o arquivo de saída que será gerado e o número da tarefa (task) que está sendo executada. Por fim, a cláusula final envia à máquina do usuário o arquivo resultante.

```
job :
     label : myjob
task:
     init: put grid/program.txt program.txt
           store grid/mytask.jar mytask.jar
     remote: java –jar Grid.jar program.txt saida.txt 0
     final: get saida.txt saida.txt
```

Figura 2.3: Uma aplicação sendo submetido à Grade [22].

Uma vez que a grade pode ser composta por inúmeros elementos de computação, heterogêneos e espalhados na Internet, é imprescindível que um mecanismo de monitoramento automatizado seja utilizado para verificar o estado da grade. Neste sentido, no próximo capítulo será detalhado quais requisitos são exigidos para um sistema de monitoramento e as principais ferramentas disponíveis atualmente para esta tarefa.

Capítulo 3

Monitoramento de Recursos

O monitoramento de recursos é um serviço que pode ser útil em um ambiente de Grade computacional. Mantendo um conhecimento geral sobre a disponibilidade de recursos e sua utilização corrente, este serviço é capaz de apresentar ao cliente informações como os ciclos de clock, capacidade de armazenamento, entre outros disponíveis em cada máquina da Grade.

As informações referentes a um recurso podem ser classificadas em estáticas ou dinâmicas. Uma informação dita estática é alterada com pouca freqüência, como o nome do sistema operacional que está instalado em um determinado nó. Já uma informação dinâmica é modificada freqüentemente, como a carga do processador.

O controle do estado e de quanto cada recurso vem sendo utilizado auxilia na tomada de decisões por parte do escalonador das aplicações e também para calcular quanto um determinado usuário está usufruindo dos recursos disponíveis. A partir do valor calculado é possível, por exemplo, que o administrador da Grade possa cobrar do usuário pelos recursos consumidos.

Os serviços de monitoramento devem ser desenvolvidos de forma a interferir o mínimo possível no processamento da máquina que será monitorada e ao mesmo tempo deve ser preciso nas informações prestadas. Na maioria das ferramentas o monitoramente se dá em três etapas [7]:

- *Recolhimento:* É nesta etapa que o sistema captura dados referentes aos recursos disponíveis em cada máquina da Grade.

- *Armazenamento:* Após ter sido efetuado o recolhimento dos dados, estes são armazenados em arquivos com formatos distintos dependendo de cada ferramenta. Os mais

comuns são arquivos texto, XML (*eXtensible Markup Language*), ou até mesmo em bancos de dados.

- *Apresentação:* Esta etapa consiste na apresentação dos dados para o usuário final da Grade. Existem várias formas de disponibilizar as informações ao usuário, podendo usar desde páginas web até softwares desenvolvidos especificamente para este fim.

3.1 Desafios de um Sistema de Monitoramento Distribuído

A implementação de sistemas que realizem o monitoramento de ambientes distribuídos não é uma tarefa fácil, devido a complexidade dos ambientes computacionais. Existem algumas características que devem ser levadas em consideração, como escalabilidade, flexibilidade, portabilidade, robustez e segurança, que serão brevemente descritas.

3.1.1 Escalabilidade

Esta é uma das funcionalidades mais importantes em um sistema de monitoramento distribuído, o qual deve estar apto a coletar informações de um número crescente de usuários e recursos em uso. Portanto, ao desenvolver o sistema deve-se ter em mente que ele terá de ser escalável e que utilizará um modelo hierárquico que pode ser expandido. Além disso, os sensores utilizados na coleta dos dados não devem introduzir uma quantidade excessiva de carga no sistema para que o desempenho da Grade não seja afetado [25].

3.1.2 Flexibilidade

A ferramenta tem que ser suficientemente flexível para acomodar diferentes tipos de dados e também deve estar preparada para receber requisições dinâmicas e estáticas. Até mesmo políticas específicas relacionadas a aplicação de medidas podem ser adicionadas. Deve-se também ser flexível para atender a diferentes tipos de eventos, protocolos e normas.

3.1.3 Portabilidade

Uma das principais características de uma estrutura de Grade é a heterogeneidade, tanto de hardware como de software. Portanto, o sistema de monitoramento distribuído deve ser capaz de trabalhar entre diferentes tipos de sistemas e plataformas.

3.1.4 Robustez

Quanto maior o número de nós integrantes da Grade, maior será o tráfego na rede e o número de atividades de entrada e saída de dados, que refletirá em um aumento significativo na probabilidade de ocorrerem falhas no ambiente computacional. Dado isso, a robustez refere-se à forma com que o sistema de monitoramento vai se recuperar ao detectar uma destas falhas.

3.1.5 Segurança

Dependendo da natureza dos dados, alguns cuidados devem ser tomados para garantir o armazenamento, o processamento e a transmissão dos dados. Princípios de segurança como autenticação, confidencialidade e integridade devem ser mantidos em todas as operações que envolvem dados.

3.2 Ferramentas de Monitoramento

Nesta seção serão apresentados alguns dentre os principais sistemas que se propõem a realizar o monitoramento de recursos em Grades computacionais, como o Hawkeye [23], Monalisa [10], [11], Ganglia [6] e o R-GMA [24], [9].

3.2.1 Hawkeye

A ferramenta Hawkeye foi desenvolvida pela equipe criadora da Grade Condor, com o objetivo de detectar possíveis problemas na utilização dos recursos da Grade, como tráfego intenso na rede, alta carga de uso e falhas nos recursos computacionais.

O sistema provê mecanismos capazes de coletar informações, armazená-las e usufruir delas quando necessário. Devido ao fato de ser baseado na Grade Condor, a sua

configuração não é considerada difícil, facilitando a utilização para quem queira cadastrar sua máquina ou seu site no sistema [23].

Dentre os recursos monitorados pelo sistema os principais são:

- *Espaço em Disco:* monitora o espaço livre no disco das máquinas conectadas na Grade. Este monitoramento é baseado no programa Unix df, que mostra o espaço livre e ocupado de cada partição do disco.

- *Uso de Memória:* monitora o quanto de memória está sendo utilizado pelo sistema.

- *Erros na Rede:* controla o status da rede, verificando se a comunicação entre os nós está funcionando corretamente.

- *Abertura de Arquivos:* verifica quantas vezes cada arquivo foi acessado.

- *Monitoramento de Usuários:* verifica quais usuários estão logados no sistema. Esta verificação é feita pelo Unix w, um comando que provê um sumário no qual estão citados todos os usuários logados no sistema no momento.

3.2.2 Monalisa

O *MONitoring Agents in a Large Integrated Services Architecture* (Monalisa) é um sistema para monitoramento de recursos distribuídos que provê suporte na coleta de informações referentes aos recursos disponíveis nos domínios remotos e incorpora protocolos para agregar essas informações em um repositório *Web* centralizado. Essas informações são apresentadas aos usuários por meio de interfaces gráficas.

A arquitetura do Monalisa é formada por um conjunto de agentes, os quais são registrados como serviços dinâmicos e são capazes de contribuir para o monitoramento de diferentes recursos. Esses agentes coletam dados de domínios gerenciadores de redes, roteadores, ou diretamente de ferramentas de monitoramento distribuído, o que facilita a integração do projeto Monalisa com aplicações como Ganglia, Condor e Hawkeye.

Um problema que pode vir a surgir na utilização do Monalisa é o acúmulo de informações devido ao armazenamento por longos períodos de tempo. Para isso, o sistema utiliza mecanismos que reduzem a agregação de dados. Assim, evita-se que todas as informações sejam centralizadas, o que não é viável sob o ponto de vista de armazenamento e da rede.

Na consulta, as informações mais recentes podem ser acessadas diretamente de serviços de cada domínio. Porém, as informações armazenadas por períodos de tempo mais longos devem ser acessadas em repositórios centralizados.

3.2.3 R-GMA

A ferramenta R-GMA foi projetada para operar em ambientes distribuídos, sendo capaz de informar a um cliente quais nós estão disponíveis para executar tarefas na rede, incluindo sua carga de processamento atual e qual o software disponível no nó. Informações como a capacidade de armazenamento de dados e o número máximo de arquivos que podem ser armazenados na máquina também podem ser obtidas. É possível ainda acompanhar o progresso das uma tarefa em execução [24].

O sistema R-GMA é uma implementação da *Grid Monitoring Architecture* – GMA. Esta arquitetura define os principais componentes para o monitoramento de recursos distribuídos em uma Grade Computacional, como mostra a Figura 3.1.

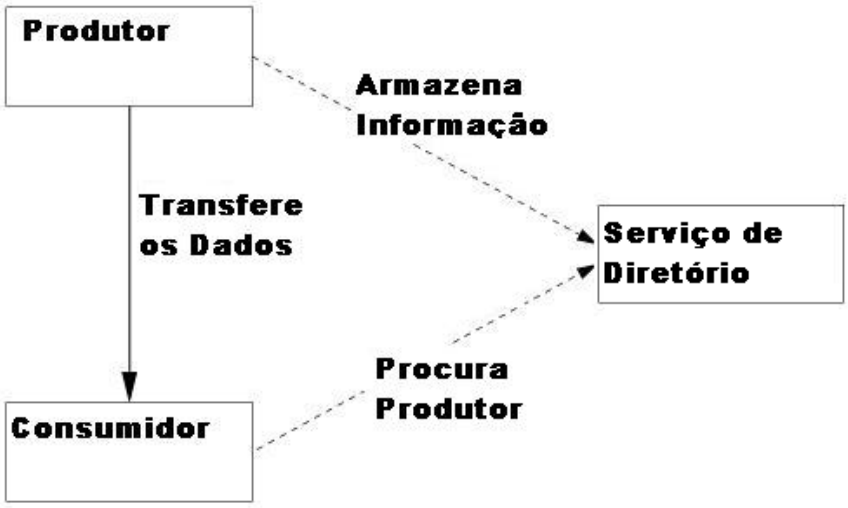

Figura 3.1: Modelo arquitetural do GMA [24].

- *Produtores:* um Produtor é um componente que disponibiliza seus recursos. Ele transmite ao Serviço de Diretório a sua localização e quais recursos estão disponíveis para serem usados pelos Consumidores.

- *Serviço de Diretório:* o Serviço de Diretório é um banco de dados que armazena as informações enviadas pelos Produtores e os disponibiliza aos Consumidores quando solicitadas.

- *Consumidores:* o Consumidor é o componente que deseja obter informações referentes aos recursos da Grade. Ele consulta o Serviço de Diretório para ver qual Produtor disponibiliza os recursos que ele necessita. Em seguida, ele consulta a localização que o Produtor armazenou no Serviço de Diretório e passa a se comunicar diretamente com o Produtor. A comunicação entre o Consumidor e o Produtor pode ser realizada por meio de consulta e resposta ou por um protocolo de assinatura. Consumidores podem também se registrar no Serviço de Diretório para receber informações quando novos Produtores são cadastrados no sistema.

O GMA não define o modelo de dados ou a linguagem de consulta que deve ser utilizada, tampouco os tipos de informações que devem ser armazenadas no Serviço de Diretório. Nesse sentido, as funcionalidades do GMA são complementadas pelo R-GMA, através de uma implementação, que utiliza o Sistema Gerenciador de Banco de Dados – SGBD - MySQL e Java *Servlet*.

No R-GMA as informações são armazenadas em um banco de dados relacional e podem ser consultada utilizando a linguagem SQL (Structured Query Language).

3.2.4 Ganglia

O Ganglia é uma ferramenta de monitoramento distribuído e escalável desenvolvida para trabalhar em ambientes computacionais de alto desempenho, como Clusters e Grades computacionais. O sistema fornece informações referentes aos diversos recursos disponíveis nos nós da Grade.

Trata-se de um software gratuito, que começou a ser desenvolvido na Universidade da Califórnia, *Campus* de Berkeley. É baseado em um protocolo que utiliza técnicas de difusão *multicast* para informar o *status* dos recursos e uma estrutura de árvore que representa as conexões ponto-a-ponto da rede de alto desempenho.

O Ganglia utiliza a técnologia XML para representar os dados, a XDR (*eXternal Data Representation*) para compactar os dados, viabilizando o envio dos mesmos pela rede

e o *Round Robin Database* (RRDtool), um *framework* que serve para armazenar os dados e gerar gráficos, facilitando a visualização dos mesmos [6].

A arquitetura do Ganglia é baseada em uma federação hierárquica de *clusters* [6]. Os dados referentes aos recursos dos computadores monitorados são coletados pelo *daemon gmond* e encaminhados aos demais computadores por meio de difusão *multicast* utilizando a tecnologia XDR. Esse envio dos dados diretamente a todos os computadores conectados na Grade garante que estes estejam sempre atualizados em relação ao estado global do ambiente. Caso um computador tenha problemas, qualquer outro está apto a responder às requisições de consulta. O armazenamento local é feito na memória principal de cada computador, sendo que os dados nunca são armazenados em disco.

A Figura 3.2 representa a arquitetura do Ganglia. O *ganglia monitoring daemon – gmond* – deve ser executado em todos os computadores da Grade. É responsável por monitorar os recursos locais do nó e responder às consultas submetidas pelos clientes, enviando informações em formato XML através do TCP (*Transmission Control Protocol*) ao *daemon gmetad* (*ganglia meta daemon*).

O *gmetad* possibilita o agrupamento de *clusters* em ambientes de Grade. O componente é responsável por centralizar os dados enviados pelos *gmonds*, sendo também possível agrupar dados enviados por outros gmetads, gerenciando tanto Grades quanto Clusters.

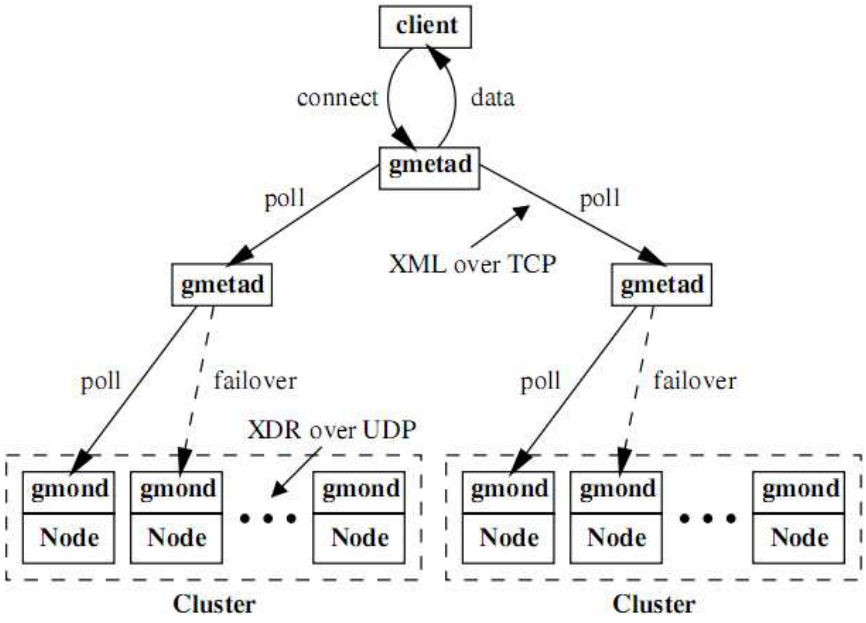

Figura 3.2: Modelo arquitetural do Ganglia [6].

O Ganglia possui por padrão algumas métricas utilizadas para levantar dados comumente desejados em sistemas de monitoramento, como número de processadores do nó, velocidade dos processadores, memória total disponível e memória virtual utilizada. Porém, pode surgir a necessidade de obter outras informações. Para isso, é necessário utilizar o programa de linha de comando – *gmetric* (*ganglia metric client*), que possibilita que o administrador do ambiente adicione métricas próprias, obtendo o monitoramento desejado.

Os dados são apresentados aos usuários por meio de uma interface web. O sistema de banco de dados RRDtoll, que armazena os dados do gmetad, gera gráficos que representam os recursos de cada máquina e a utilização destes recursos. A interface web apresenta os gráficos gerados pelo RRDtoll aos usuários do sistema.

Feita a descrição das principais ferramentas de monitoramento de recursos úteis para ambientes de Grade, foi realizada a instalação de uma delas, visando verificar seu funcionamento e dados coletados. Além disso, foi feita uma proposta de utilização desses recursos como forma de melhorar o escalonamento do OurGrid. Todas estas informações estão no Capítulo 4.

Capítulo 4

Estudo de Caso

4.1 Implantação do ambiente distribuído

O sistema OurGrid foi instalado e configurado, como mostra o Apêndice A, em 7 computadores, interconectados com tecnologias de *hardware* e s*oftware* distintas, gerando assim um ambiente distribuído heterogêneo. Este ambiente com diferentes tecnologias enriquece os dados obtidos pelos sistemas de monitoramento e aproxima mais o trabalho com o que acontece na prática, onde, a diversidade de *hardware* e s*oftware* é cada vez mais freqüente.

Um componente *Peer* e um componente *Broker* foram instalados em uma máquina escolhida para ser o servidor da Grade, como mostram as Figuras 4.1 e 4.2. Nas outras 6 máquinas foram instalados *Workers*, responsáveis por realizar a execução das aplicações.

A Tabela 4.1 ilustra as características que compõem cada um dos 7 nós da Grade, onde, a sigla SO refere-se ao sistema operacional e a sigla HD ao disco rígido de cada máquina. Todos os equipamentos possuem a arquitetura PCx86 de 32 bits.

Tabela 4.1: Tecnologias das máquinas da Grade.

Nós	Processador	Memória	SO	Disco	
				Total	Livre
Peer, Broker	2,08 GHz	0,74 GB	Linux, Debian	19 GB	9 GB
Worker 1	1,30 GHz	1,00 GB	Windows	25 GB	11 GB
Worker 2	2,08 GHz	0,74 GB	Linux, Debian	19 GB	7 GB
Worker 3	2,60 GHz	1,00 GB	Linux, Fedora	18 GB	3,5 KB
Worker 4	2,80 GHz	1,00 GB	Windows	19 GB	9 GB
Worker 5	2,60 GHz	1,00 GB	Linux, Fedora	21 GB	12 GB
Worker 6	2,60 GHz	1,00 GB	Linux, Fedora	18 GB	6 GB

A Figura 4.1 ilustra o *Peer* ligado e os 6 *Workers* disponíveis para receber aplicações. Os círculos verdes ao lado dos *Workers* mostram que estes estão ociosos (*IDLE)*, como mostra a legenda no rodapé do *Peer*. Quando um *Worker* está executando alguma tarefa, o mesmo fica marcado como amarelo ou em uso (*INUSE*).

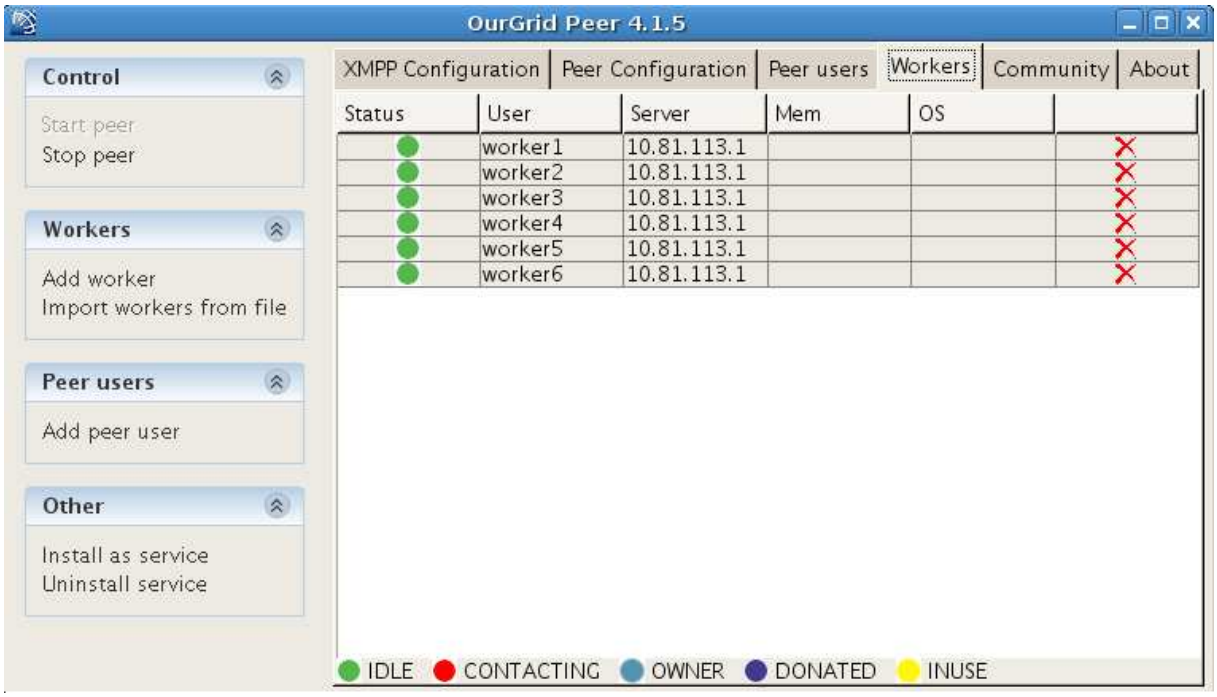

Figura 4.1: Componente *Peer* em execução.

A Figura 4.2 apresenta o componente *Broker* submetendo uma aplicação aos *Workers*. O programa está dividido em 26 tarefas, sendo que 25 delas já foram finalizadas e uma ainda está em execução. O quadro a direita da figura mostra a quantidade de tarefas, os dados da execução, os resultados obtidos e qual foi o tempo de duração.

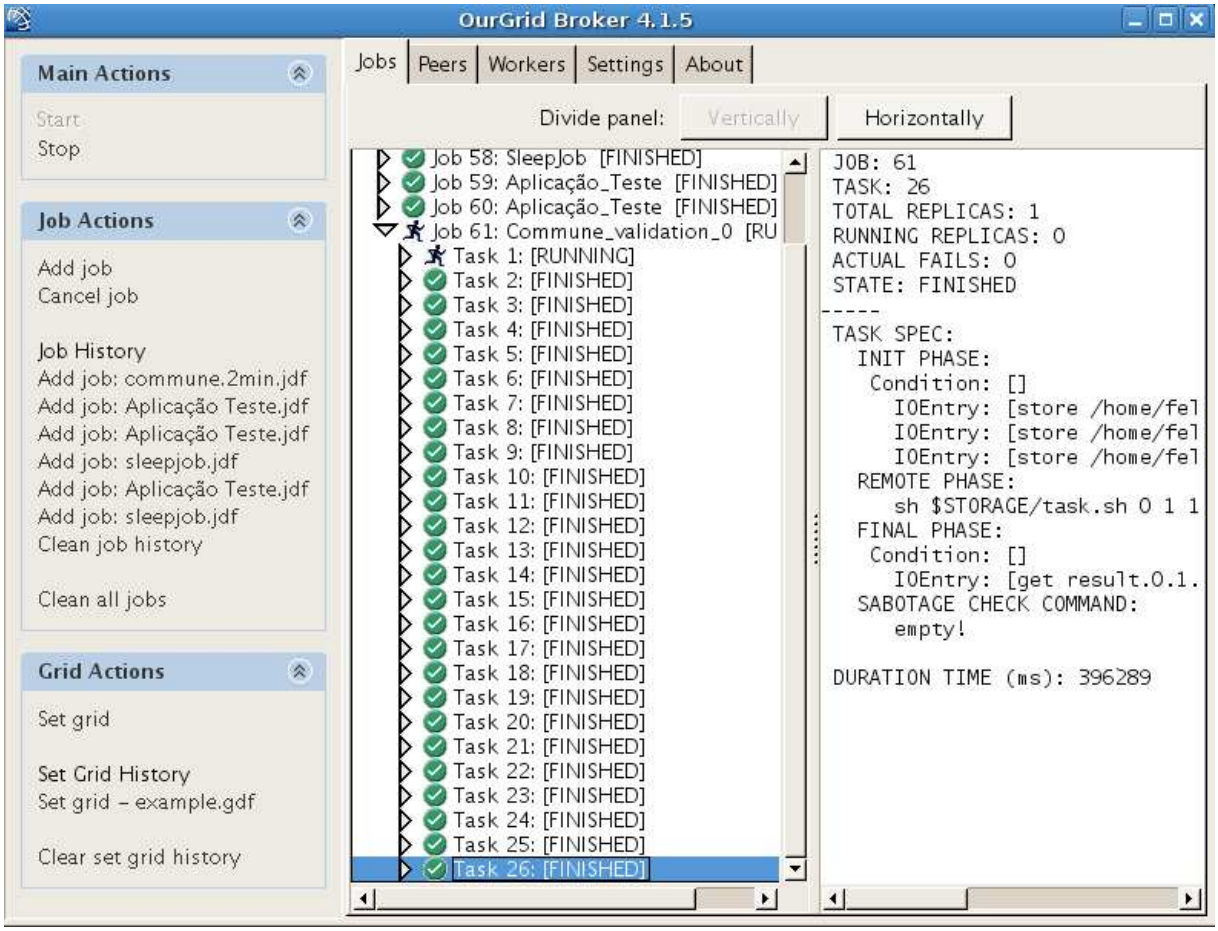

Figura 4.2: Componente *Broker* Submetendo tarefas à Grade.

A aplicação executada na Grade, apresentada na Figura 4.2, verifica se o protocolo de conexão do Projeto Commune, desenvolvido na UFCG, possui *deadlocks*, *livelocks* e se há perdas e estados inatingíveis na conexão entre dois nós [27].

4.2 Monitorando o sistema OurGrid com a ferramenta Ganglia

A ferramenta Ganglia foi instalada e configurada, como mostra o Apêndice B, para monitorar os recursos disponíveis nos nós da Grade. Um componente *gmond* foi instalado em cada *Worker* com o objetivo de coletar as informações de cada máquina e enviar ao nó central, onde estes dados são agrupados e apresentados ao usuário.

No nó central, onde foram inplantados os componentes *Peer* e *Broker* do OurGrid, foi instalado o *daemon gmetad* responsável por agrupar os dados enviados pelos *gmonds*. A

interface web também foi instalada nesta máquina, na qual os dados de cada *Worker* passaram a ser visualizados como mostra a Figura 4.3.

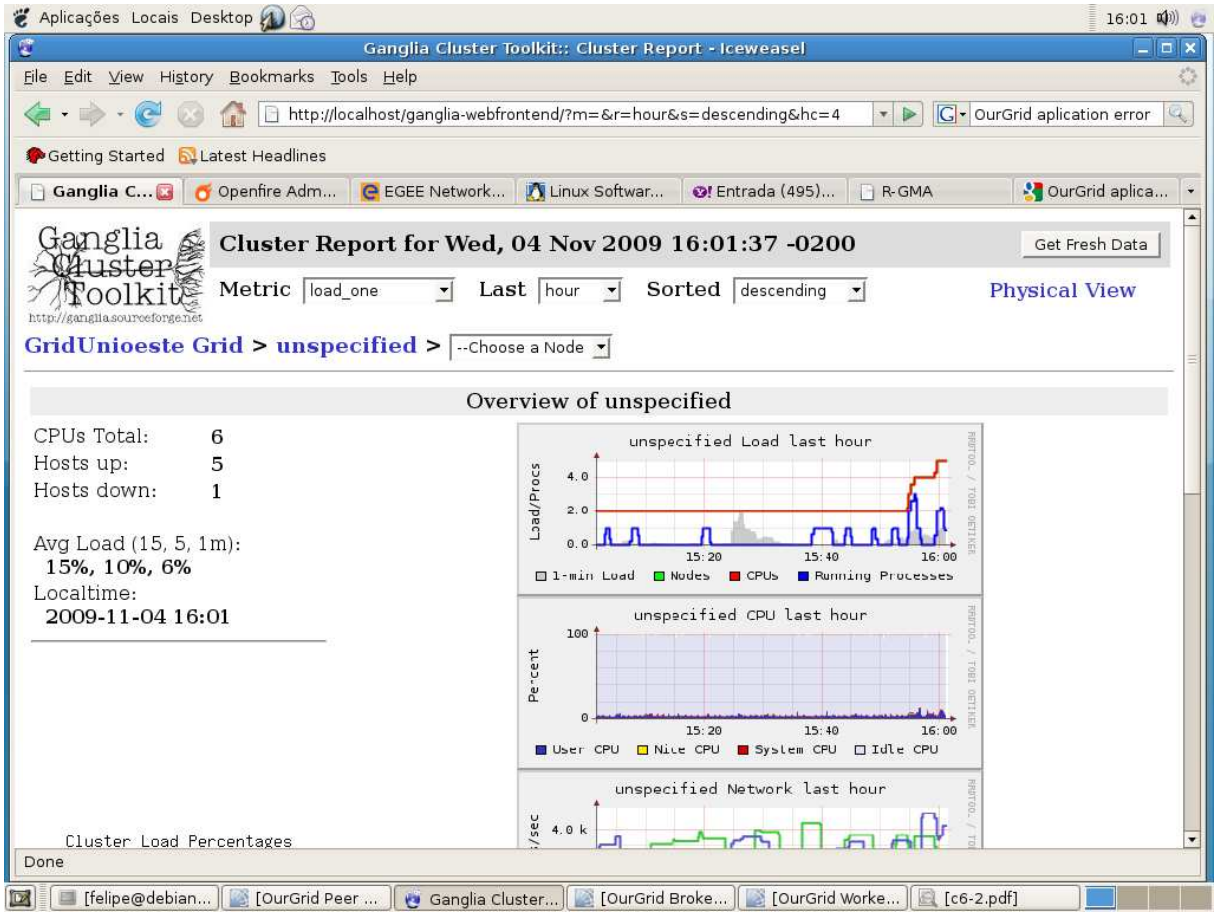

Figura 4.3: Gráficos gerados pela interface web.

O primeiro gráfico na Figura 4.3 contém uma linha azul que representa os processos em execução nas máquinas da Grade e uma linha vermelha que mostra o número de CPUs disponíveis no momento. A medida que as máquinas foram ficando *online*, a linha foi incrementada até alcançar o valor 6, referente aos 6 *Workers* da grade implementada. O segundo gráfico mostra a utilização das máquinas da Grade. A esquerda dos gráficos é informado o número total de nós e quais estão disponíveis. Quando um nó está disponível, este aparece como *Host Up*, caso ocorra alguma falha na comunicação, ou o mesmo seja desligado, ele passa a ser citado como *Host Down*.

Na máquina central foi instalado junto à interface web o *framework Round Robin Database – RRDTool*, ferramenta que armazena os dados agrupados pelo *gmetad* e gera gráficos a fim de facilitar o entendimento das informações coletadas.

4.2.1 Coleta de Dados

O *daemon gmond*, baseado em algumas métricas que estão configuradas por padrão em seu arquivo de configuração *gmond.conf*, efetua o monitoramento dos recursos em um nó. Caso outros dados sejam relevantes, existe um programa, *gmetric*, que possibilita a personalização de novas métricas.

Após serem estabelecidas todas as métricas que serão monitoradas, o *gmond* passa a coletar as informações. Os dados são periodicamente transmitidos para todos os nós da Grade utilizando uma técnica de difusão *multicast*, porém, estes dados são agrupados apenas na máquina que contém o componente *gmetad* instalado.

A lista das métricas coletadas por padrão pelo *gmond* é a seguinte:

- Número da CPU;
- Velocidade da CPU;
- Carga de utilização da CPU;
- Espaço total em disco;
- Espaço disponível em disco;
- Memória total;
- Memória Compartilhada;
- Memória *swap*;
- Memória disponível;
- Tempo que foi inicializado o monitoramento;
- Tipo da máquina;
- Sistema Operacional;
- Localização (IP);
- Processos executados;
- Entrada de *Bytes*;
- Saída de *Bytes*;

- Interrupções ocorridas.

Utilizando o Ganglia para monitorar o ambiente descrito na seção anterior, além dos dados supracitados, foi possível obter informações quanto aos recursos aglomerados obtidos com as 6 máquinas que compunham o ambiente de Grade, ilustrado na Figura 4.4.

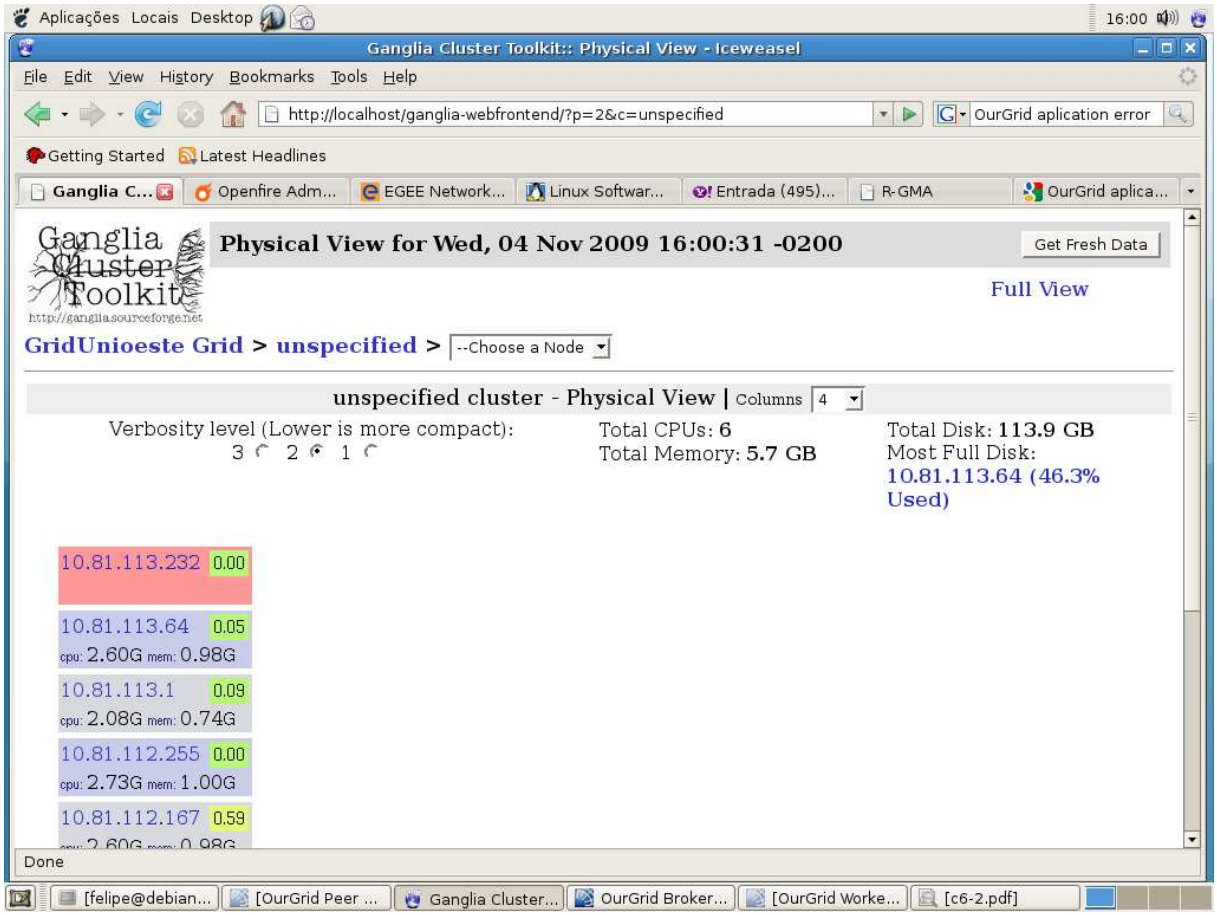

Figura 4.4: Recursos aglomerados obtidos pela ferramenta Ganglia

A esquerda da Figura 4.4 são apresentados o endereço IP e os recursos de cada máquina individualmente. O retângulo vermelho ao redor de um dos nós significa que o mesmo está inativo no momento, ou ele está desligado ou com problemas de conexão. A direita da figura é ilustrado o total de memória obtido, 5.7 GB e o total de espaço em disco, 113.9 GB.

Por serem máquinas com poder computacional limitado, a capacidade máxima obtida não foi muito expressiva, porém com os dados coletados neste ambiente de Grade, pode-se ter uma idéia do poder computacional em um sistema com mais de mil máquinas

trabalhando em conjunto, como ocorre no National Grid Service na universidade de Oxford no Reino Unido [26].

4.3 Proposta para otimização do escalonador da Grade utilizando um sistema de monitoramento de recursos

O escalonador é um componente utilizado em sistemas distribuídos para possibilitar o uso de recursos em ambientes geograficamente dispersos. Um escalonador deve receber as aplicações submetidas pelos clientes e alocar aos nós da Grade para que estas sejam executadas.

No OurGrid o componente responsável por escalonar as tarefas é o *Peer*. Este recebe as tarefas do *Broker*, aloca nos *Workers* para que sejam executadas, coleta os resultados e reenvia ao *Broker* para serem apresentadas ao usuário.

As aplicações submetidas pelo *Broker*, antes de serem alocadas nos nós finais da Grade para execução, são divididas em diversas tarefas. Estas são encaminhadas para uma fila de espera e alocadas aos *Workers* de forma aleatória, utilizando a heurística de escalonamento *Work Queue with Replication – WQR* [22].

Este escalonador trabalha em um ambiente muito dinâmico como é a Grade, utilizando um método de replicação de tarefas. Estas são envidas uma para cada *Worker* disponível para serem executadas. Quando não há mais tarefas para serem enviadas as ainda em execução são replicadas entre os *Workers* disponíveis. Quando um *Worker* terminar a execução ele envia a resposta e as outras réplicas são abortadas.

Os resultados obtidos por esta heurística de escalonamento são satisfatórios pelo fato do mesmo executar todas as tarefas mesmo que uma réplica falhe, porém, poderiam ser otimizados caso fossem baseados em dados coletados por um sistema de monitoramento de recursos diminuindo o número de falhas e o disperdício de ciclos de processamento. Algumas tarefas que deveriam ser executadas em um sistema operacional (SO) específico são alocadas em máquinas com outro SO, gerando erro e como consequência perda de tempo de execução como ilustra a Figura 4.5.

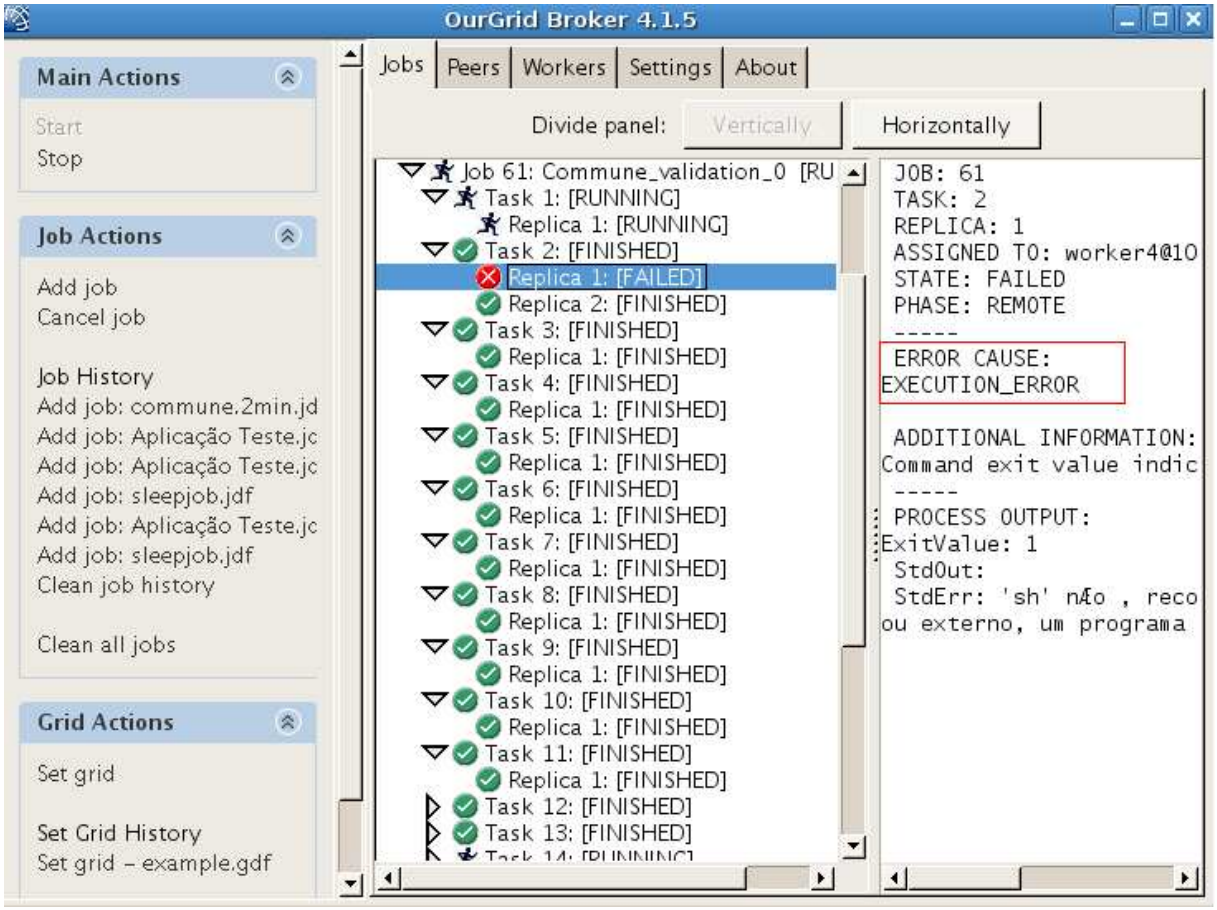

Figura 4.5: Erro na execução de uma tarefa.

As tarefas da aplicação ilustrada na Figura 4.5 deveriam executar apenas em máquinas com SO Linux. Como o sistema OurGrid não tem conhecimento quanto ao SO onde seus *Workers* estão instalados, a tarefa número 2 foi alocada à uma máquina com SO *Windows*, o que acarretou um erro de execução. Se o escalonador utilizasse dados coletados por uma ferramenta de monitoramento, este reconheceria quais máquinas tem SO Linux e o erro poderia ser facilmente evitado, ganhando ciclos de processamento e tempo de execução.

Com o objetivo de solucionar o problema supracitado os desenvolvedores do OurGrid criaram uma forma de dizer ao *Peer* qual o SO e a memória de cada *Worker*. Porém, isto é feito manualmente no momento em que o *Worker* é adicionado ao *Peer*. Esta solução funciona satisfatoriamente na criação de uma Grade local onde se tem conhecimento da arquitetura das máquinas, mas, em um sistema de Grade que deve ser escalável, podendo ter nós geograficamente dispersos com arquiteturas heterogêneas e desconhecidas, esta técnica se torna pouco eficiente.

A ferramenta Ganglia trabalhando em conjunto com a Grade resolveria este problema coletando os dados dos nós e informando ao *Peer*. A Figura 4.6 mostra os seguintes dados coletados pelo Ganglia:

- Arquitetura da Máquina: x86;
- Sistema Operacional: Linux;
- Velocidade da CPU: 2134 MHz;
- Memória: 776616 KB.

O primeiro gráfico ilustrado na Figura 4.6 mostra o momento em que uma aplicação foi submetida ao *Worker2*. A linha azul, que representa os processos em execução, ocila enquanto a tarefa está sendo executada. Já o terceiro gráfico ilustra o carga de utilização da CPU.

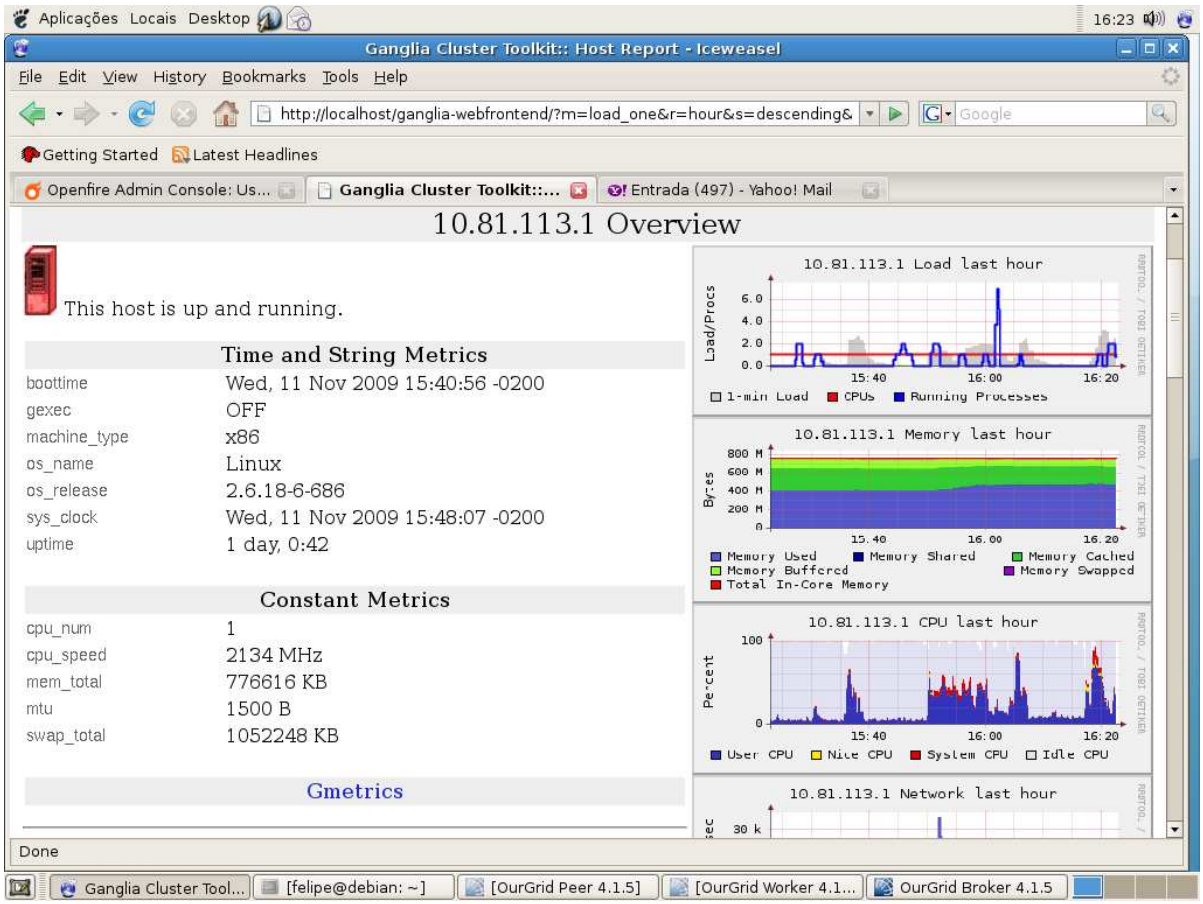

Figura 4.6: Dados coletados pela ferramenta Ganglia.

Outro erro que pode ser previnido utilizando uma ferramenta de monitoramento é evitar a execução de tarefas ou armazenamento de dados em máquinas que estão com seu disco rígido cheio ou com pouco espaço, como mostra a Figura 4.7.

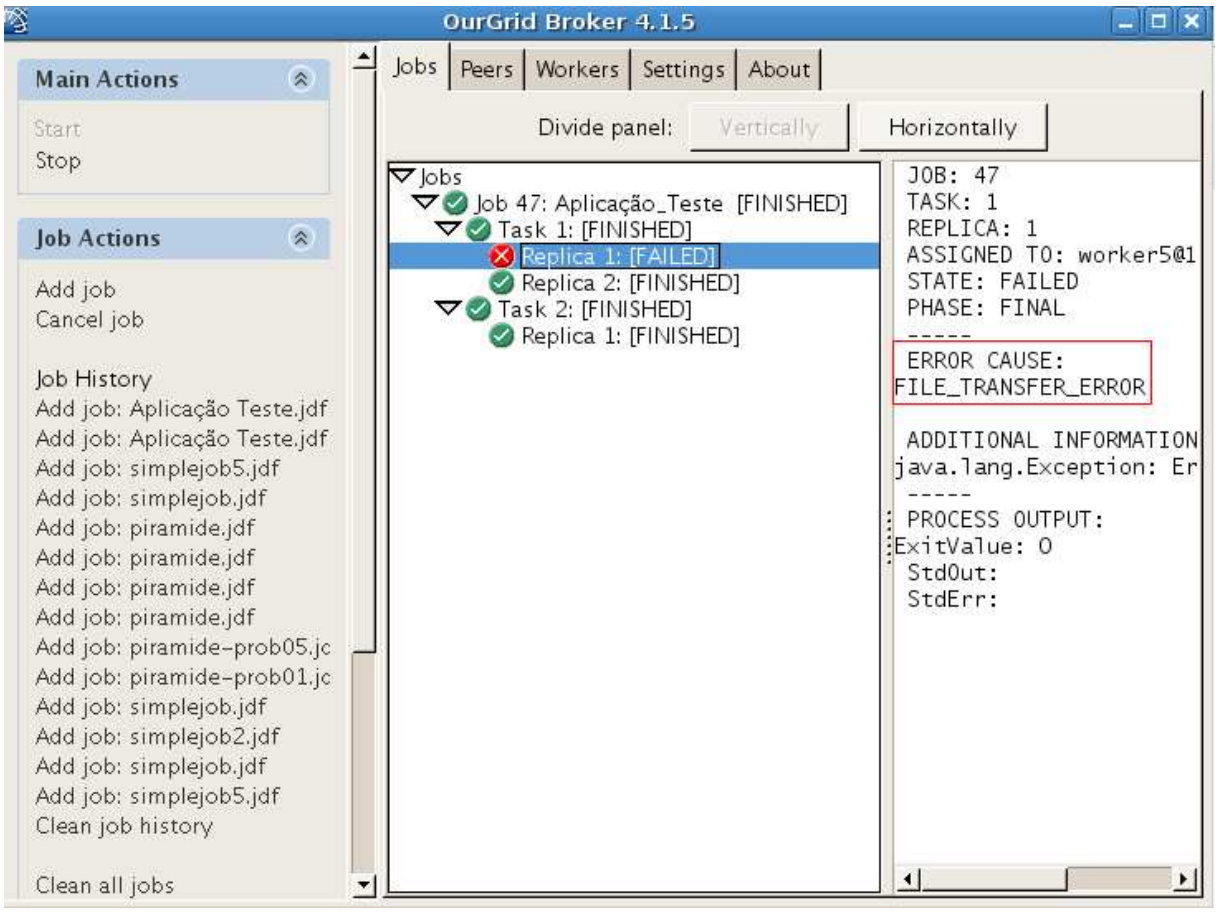

Figura 4.7: Erro no armazenamento de dados de uma tarefa.

A aplicação ilustrada na Figura 4.7 executa uma multiplicação de matrizes e armazena o resultado em um arquivo de texto. Este arquivo ocupa aproximadamente 5400 bytes de espaço em disco. A fim de testar o escalonador do OurGrid o disco do *Worker5* foi preenchido até ficar com 3500 bytes livres. Quando a tarefa foi enviada para este *Worker* houve um erro na transferência do arquivo devido a falta de espaço em disco. Esta ação atrasou a execução da aplicação, pois a tarefa 2 teve que ser escalonada para outro *Worker*.

O erro causado pela falta de espaço em disco pode ser evitado utilizando informações geradas por um sistema de monitoramento distribuído, que pode informar ao *Peer* quanto há de espaço disponível em cada *Worker*.

A ferramenta Ganglia coleta estas informações como ilustra a Figura 4.8. O primeiro gráfico representa a capacidade máxima do disco rígido de uma máquina, 19 GB, o segundo representa quanto desta capacidade não está sendo utilizada, 9 GB.

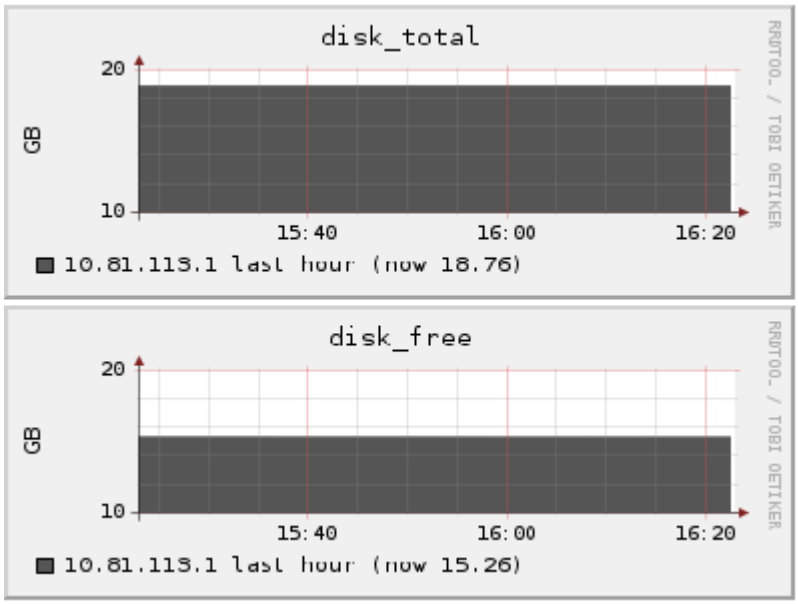

Figura 4.8: Espaço em disco gerado pelo Ganglia.

Além de informações referentes ao SO e ao espaço em disco citadas acima, outras informações coletadas por um sistema de monitoramento podem ser úteis na otimização do escalonador da Grade. Uma fila de prioridade dos nós pode ser implementada utilizando informações como memória e CPU. As máquinas que possuem estes recursos em maior quantidade devem ser utilizadas primeireiramente na execução de tarefas, pois diminuem o tempo de processamento. A localização das máquinas também deve ser levada em consideração, pois o tempo de transmissão das tarefas é acrescido no tempo final de execução.

Dentre os recursos monitorados, os de maior relevância para o escalonador são os seguintes:

- Sistema Operacional;
- Espaço disponível em disco;
- Velocidade da CPU;
- Carga de utilização da CPU;
- Memória;

- Localização (IP).

No parágrafo seguinte é descrito, como deve trablhar o escalonador da Grade com base nosa dados de um sistema de monitoramento.

Utilizando os 6 *Workers* apresentados na Tabela 4.1, deseja-se executar uma aplicação que necessita de máquinas com SO Linux e 5 Gb de espaço livre em disco. Inicialmente os *Workers* 1 e 4 seriam eliminados pelo escalonador, por não possuírem SO Linux. Após verificar o espaço livre no disco das máquinas, o *Worker* 3 também seria eliminado, restando apenas os *Workers* 2, 5 e 6. Caso a aplicação fosse composta por apenas 2 tarefas, estas deveriam ser escalonadas aos *Workers* 5 e 6 devido ao fato dos mesmos possuirem mais capacidade de CPU e memória, o que agiliza o processo de execução.

Com o escalonador trabalhando baseado em informações referentes aos recursos das máquinas, como mostra o exemplo acima, as aplicações seriam finalizadas mais rapidamente quando comparado ao escalonador *WQR*. O qual não leva em consideração as restrições apresentadas pelas aplicações.

Capítulo 5

Considerações Finais

Grades computacionais ampliam a capacidade de execução de aplicações paralelas utilizando recursos geograficamente distribuídos em diferentes domínios administrativos. Nos últimos anos, grandes projetos como Globus, Condor e OurGrid foram iniciados a fim de expandir os estudos de Grades computacionais. Devido ao grande potencial da área e ao interesse que estas vêm dispertando nas grandes empresas de tecnologias como SUN, Oracle e IBM, justifica-se a elaboração deste trabalho.

As organizações que utilizam ou desenvolvem esta tecnologia desejam aumentar a sua capacidade de processamento de dados. Contudo, devido a heterogeneidade de um ambiente de Grade, ferramentas de monitoramento e mecanismos de escalonamento precisam ser construídos de maneira que melhorem a utilização dos recursos, aumentando assim o desempenho na execução de aplicações paralelas.

Uma limitação encontrada neste trabalho está relacionada ao escalonamento das tarefas na Grade. O escalonador atual do OurGrid não faz nenhuma verificação quanto às restrições apresentadas por cada tarefa, submetendo-as para máquinas que não atendem os requisitos mínimos, resultando em erros e tendo que submete-las novamente para outros nós, perdendo tempo de processamento e desperdiçando ciclos de clock.

Neste sentido, foi proposto neste trabalho implantar um ambiente de Grade computacional utilizando a plataforma OurGrid, instalar e configurar ferramentas capazes de monitorar os recursos disponíveis na Grade e mostrar como estes recursos podem ser úteis na otimização de um escalonador de aplicações paralelas.

Através dos resultados obtidos foi possível concluir que os dados coletados por uma ferramenta de monitoramento automatizada são essenciais para oferecer informações que possam otimizar o escalonamento de aplicações em um ambiente de Grade.

Além disso, verificou-se que a ferramenta Ganglia apresenta-se como um mecanismo interessante para este tipo de coleta, pois pode ser executada em ambientes heterogêneos, coleta uma grande variedade de informações e armazena os dados de forma padronizada com o uso de XML.

Com este trabalho abrem-se perspectivas para realização de novas pesquisas e como sugestão para trabalhos futuros têm-se:

- Implementar de fato um escalonador baseado nos recursos capturados por uma ferramenta de monitoramento;
- Realizar testes usando maior quantidade de recursos de diferentes domínios para identificar novos comportamentos e possíveis problemas na alocação dos recursos;
- Testar as outras plataformas de Grade Computacional disponíveis atualmente como Globus Toolkit e Condor;
- Adaptar e testar a execução de aplicações sequenciais na Grade Computacional;
- Desenvolver em conjunto com a equipe do OurGrid um *Worker* apenas para armazenamento de dados.

Apêndice A

Instalação da Grade Computacional

A.1 Instalação do Java

Todos os componentes do OurGrid rodam sobre Plataformas Windows ou Linux e requerem que o Java Runtime Environment (JRE) esteja instalado em todas as máquinas.

Certifique-se de que o Java Runtime Environment (JRE) versão 1.6.0 ou superior se encontra instalado. Isto pode ser feito através do comando `java –version`. Caso contrário faça o download da instalação do Java Runtime Environment (JRE) tanto para Windows como Linux do site `http://java.sun.com/javase/downloads/ index.jsp`.

Para Instalar o Java Runtime Environment (JRE):

1) Digite chmod +x jre~~ para obter permissão.
2) Digite ./jre~~ para instalar.

A.2 Instalação do Servidor XMPP

Todos os componentes do OurGrid requerem um servidor XMPP para funcionarem adequadamente e são certificados para funcionar com o servidor OpenFire, embora outros servidores também possam funcionar. Abaixo você encontra um guia detalhado sobre como obter, instalar e criar um servidor OpenFire para trabalhar com OurGrid.

Este documento lista os passos requeridos para configurar um servidor XMPP Openfire 3.6.0 ou superior.

A.3.1 Requerimentos do Sistema

1) Plataformas: Windows, Linux, Unix or Mac OS
2) Java Runtime Environment (JRE) versão 1.6.0 ou superior.

A.3.2 Instalando o Openfire

1) Faça o download da versão desejada do site *http://www.igniterealtime.org/downloads/index.jsp.*

2) `Chmod +x openfire para obter permissão`

3) Extraia o *tarball* para a pasta "*usr/local"* se estiver utilizando plataforma Linux ou execute o arquivo *openfire_3_6_0a.exe* caso esteja utilizando plataforma Windows.

Utilizando plataforma Linux vá para o diretório recém criado de nome "*openfire*" e inicie o servidor através do comando "`bin/openfire.sh`". Utilizando plataforma Windows dê duplo clique no arquivo "*Openfire Server*" recém criado e clique no botão "*Start*".

4) Em poucos segundos o servidor estará rodando e será possível configurá-lo através do console de administração via *web*. Para acessá-lo aponte seu browser para o endereço "`http://host:9090`", onde "*host*" é o nome da máquina onde o servidor acabou de ser instalado. Se estiver na mesma máquina onde está instalado o Openfire, a seguinte URL irá funcionar normalmente: "`http://127.0.0.1:9090`". Então você poderá configurar a primeira tela de configuração do servidor.

A.3.3 Configurando o Servidor

1) Em "*Language Settings*" selecione "*English"*.

2) Em "*Server settings*" preencha o campo domínio com o nome ou o endereço IP do servidor.

3) Em "*Database settings*" selecione "*Embedded database*". Isto fará com que o servidor utilize o banco de dados localmente. Um banco de dados externo pode ser usado, mas esta opção ainda não foi testada no OurGrid.

4) Em "*Profile settings*" selecione "*Default*". Isto significa que os usuários e grupos serão armazenados no banco de dados do servidor.

5) Em "*Administrator account*", preencha o campo e-mail e escolha uma senha de administrador.

Agora a configuração inicial está pronta e já é possível conectar-se no console de administração utilizando o username "`admin`" e a senha que acabou de escolher.

A maioria das propriedades padrão do Openfire são adequadas para um funcionamento normal do OurGrid. Entretanto, algumas configurações precisam ser modificadas. Lembre-se

de clicar em "*Save settings*" toda vez que modificar as configurações de uma seção para garantir que as mesmas sejam salvas.

6) Na guia "*Server manager*", seção "*System properties*" adicione uma nova propriedade:

- **Nome da propriedade**: `xmmp.server.certificate.verify`
- **Valor da propriedade**: `false`

Esta propriedade é obrigatória, pois ela define onde é feita a checagem da certificação da chave de autenticação. Como no OurGrid essa autenticação é feita no *Peer*, é necessário setar esta propriedade como "*false*" para que o servidor não efetue a checagem nele mesmo e sim no componente *Peer*.

7) Na guia "*Server settings*", seção "*Server to server*" certifique-se que a comunicação servidor-servidor esteja marcada como "*Enabled*". Isto fará com que servidores remotos possam trocar pacotes com este servidor pela porta (5269). Também é recomendado que não modifique o valor padrão desta porta.

8) Na guia "*Server settings*", seção "*Registration & login*" é recomendado que seja desabilitada a opção para a criação de contas automáticas por clientes em "*Inband Account Registration*". Entretanto, para tornar a instalação dos *Workers* mais fácil e rápida, pode-se optar por deixar essa opção habilitada temporariamente fazendo com que todas as contas de usuários para cada *Worker* sejam criadas automaticamente. Em "*Change password*" selecione "*Disabled*" e em "*Anonymous login*" selecione "*Disabled*". Isto faz com que nenhum usuário possa alterar a sua senha sem o consentimento do adminstrador e que somente usuários que tenham contas possam conectar-se ao servidor.

9) Na guia "*Server settings*", seção "*Resource policy*" escolha a opção "*Never kick*". Como o servidor XMPP permite que múltiplos *logins* para uma mesma conta de usuário sejam possíveis, se uma conexão solicitar um recurso que já esteja em uso, o servidor deve decidir como tratar este conflito. Através da opção "*Never kick*" se um conflito ocorrer, o servidor não permitirá que a conexão que gerou o conflito acesse o recurso.

10) Na guia "*Server settings*", seção "*Offline messages*" escolha a opção "*Drop*". Isto faz com que mensagens que forem enviadas para usuários que não estejam conectados sejam descartadas.

11) Na guia "*Server settings*", seção "*Security settings*" selecione a opção "*Custom*" para "*Client Connection Security*" e então marque "*Not available*" para "*Old SSL*

method" e para "*TLS method*". E em "*Server connection security*" marque a opção "*Optional*".

Como a parte de autenticação no OurGrid é feito pelo *framework Commune* do OurGrid, é necessário desabilitar estas opções.

A.3.4 Habilitando Usuários

É importante lembrar que qualquer máquina que se conecte a Grade, seja ela um *Worker*, *Peer* ou *Broker*, precisa necessariamente ter um *login* de usuário cadastrado no banco de dados do Servidor. Para isto o administrador deve acessar a guia "*Users/Groups*" e na seção "*Users*" selecionar a opção "*Create new user*" e criar um nome de usuário e sua respectiva senha.

Agora seu servidor XMPP já está pronto para rodar. A interface de navegação do Openfire é bem amigável e de fácil entendimento, não apresentando nenhuma dificuldade ao usuário de verificar *logs*, checar sessões abertas, verificar usuários conectados, etc.

A.3 Instalação do Componente *Peer*

O *Peer* é o componente do OurGrid responsável por gerenciar as máquinas (*Workers*) que pertencem ao seu domínio administrativo e para obter acesso aos *Workers* de outros domínios administrativos que poderão ser utilizados por usuários de sua instituição. Em outras palavras, *Peer* é o serviço que dinamicamente provê *Workers* para a execução de tarefas.

A.3.1 Requerimentos do Sistema

1) Espaço em disco: Aproximadamente 21 MB para a instalação, mais espaço suficiente para armazenar os arquivos de *logs*.
2) Plataformas: Windows ou Linux.
3) Conectividade: O *Peer* deve ser capaz de se conectar a um servidor XMPP.
4) Java Runtime Environment (JRE) versão 1.6.0 ou superior.

A.3.2 Instalando o *Peer*

Para instalar o componente *Peer* do OurGrid basta apenas descompactar o pacote obtido do site `http://www.ourgrid.org/?p=downloadAndDocumentation`. O resultado da árvore de diretórios quando descompactada pode ser visto no Quadro A.1.

```
      |-- example.sdf
      |-- log4j.cfg.xml
      |-- peer
      |-- peer.bat
      |--
peer.properties
      |-- testjvm
      |--| certification
      |--| lib
      |--| logs
      |--| resources
```

Quadro A.1 – Árvore de diretórios do componente *Peer*

A.3.3 Acesso via modo gráfico (GUI)

Para acessar o modo gráfico do componente *Peer* dê um duplo clique no arquivo *"peer.bat"* ou execute a linha de comando `peer gui`. A interface gráfica do *Peer* é ilustrada na Figura A.1.

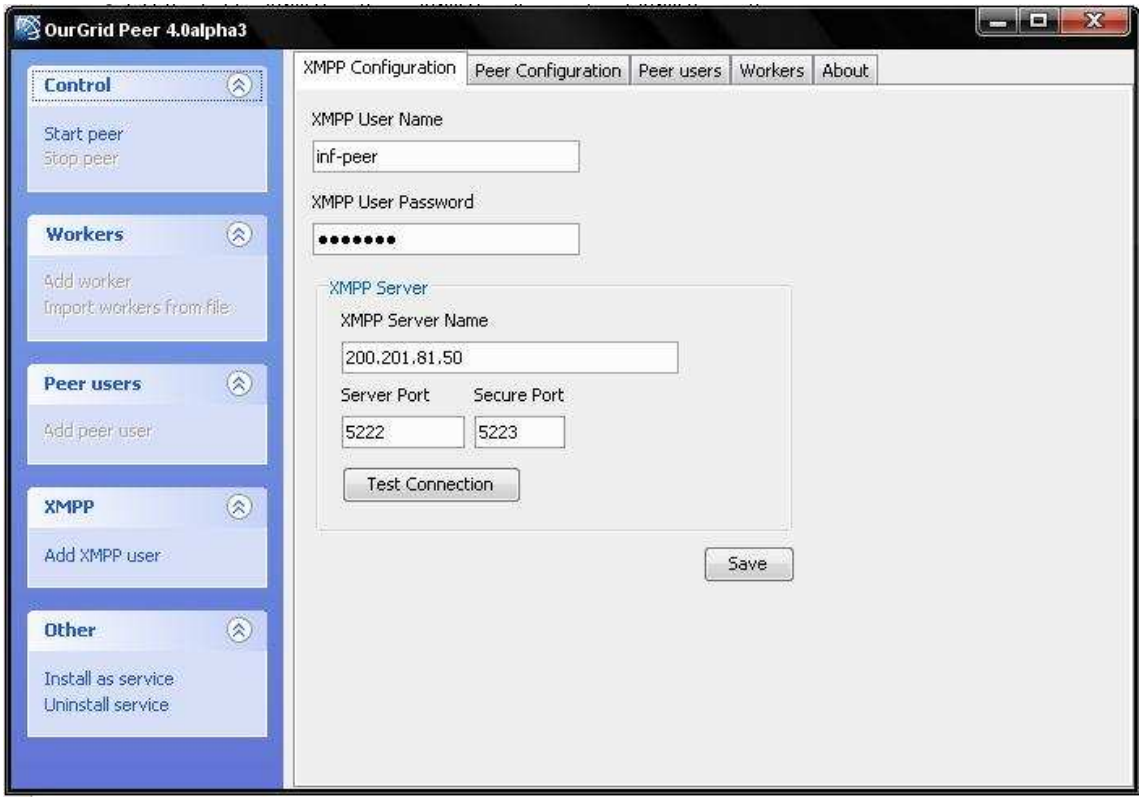

Figura A.1 – Janela inicial do componente *Peer*

Edite os campos *"XMMP User Name"*, *"XMPP User Password"* e *"XMPP Server Name"*. Não se esqueça de salvar as alterações e clique no Botão *"Start peer"* para inicializar o *Peer*. A Figura A.2 ilustra esse procedimento.

Figura A.2 – Procedimento de inicialização do *Peer*

A.3.4 Acesso via modo console

Primeiramente é necessário editar o arquivo *"peer.properties"* e atualizar os campos *"commune.xmpp.username"*, *"commune.xmpp.password"* e *"commune.xmpp. servername"*. Os valores padrões para as outras propriedades devem estar corretos para a inicialização. Após salvar o arquivo, digitar a linha de comando `peer start` para inicializar o *Peer*.

A.3.5 Adicionando Usuários ao Peer

Todo usuário que for acessar a Grade, além de ter seu nome de usuário cadastrado no servidor XMPP também necessita que seu *login* seja adicionado ao *Peer*. Para tal tarefa, clique em *"Add peer user"* e preencha todos os campos necessários. É necessário prestar atenção para que o nome do usuário e a senha coincidam com os dados que foram cadastrados no servidor XMPP.

A Figura A.3 ilustra o procedimento de adicionar usuários ao *Peer*.

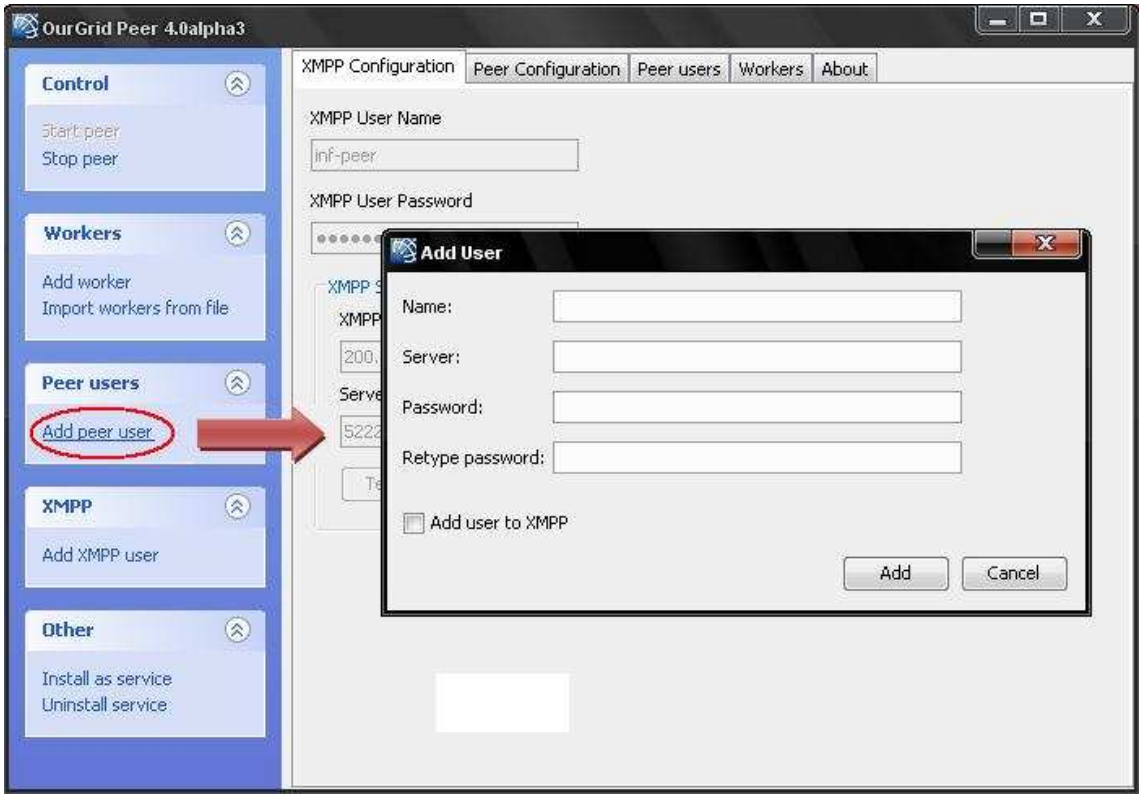

Figura A.3 – Procedimento para adicionar usuários ao *Peer*

A.3.6 Configurando os *Workers*

O próximo passo é configurar as máquinas que o *Peer* irá gerenciar. Para isto é necessário informar ao *Peer* quais máquinas estão disponíveis usando um arquivo de descrição de site (*Site Description File - SDF).*

Utilize um editor de texto de sua preferência para criar um arquivo "*mysite.sdf*" que conterá a descrição de todos os *Workers* de sua instituição. Um arquivo SDF contendo a descrição de dois *Workers* ficaria como ilustrado no Quadro A.2 onde: *workerdefaults* são as configurações padrão; *OS* é o sistema operacional da máquina onde o *Worker* está rodando; e *servername* é o nome do servidor XMPP de seu site e *username* é o nome da máquina onde o *Worker* está rodando.

```
workerdefaults :
copyTo : scp $localfile $machine:$remotefile
remExec : ssh -x $machine $command
copyFrom : scp $machine:$remotefile $localfile

worker :
     OS : linux
     servername : xmpp.ourgrid.org
     username : user1

worker :
     OS : linux
     servername : xmpp.ourgrid.org
     username : user2
```

QuadroA.2 – Exemplo de arquivo SDF contendo dois *Workers*

A.3.7 Adicionando *Workers* ao *Peer*

Utilizando o arquivo "*mysite.sdf*" que acabou de criar, execute a linha de comando `peer setworkers mysite.sdf`.

Para checar se o *Peer* carregou corretamente as configurações dos *Workers* execute a linha de comando `peer status`.

Também é possível adicionar *Workers* pelo modo gráfico (GUI). Para isto clique em *"Import Workers from file"* e selecione o arquivo "*mysite.sdf*". A Figura A.4 ilustra esse procedimento.

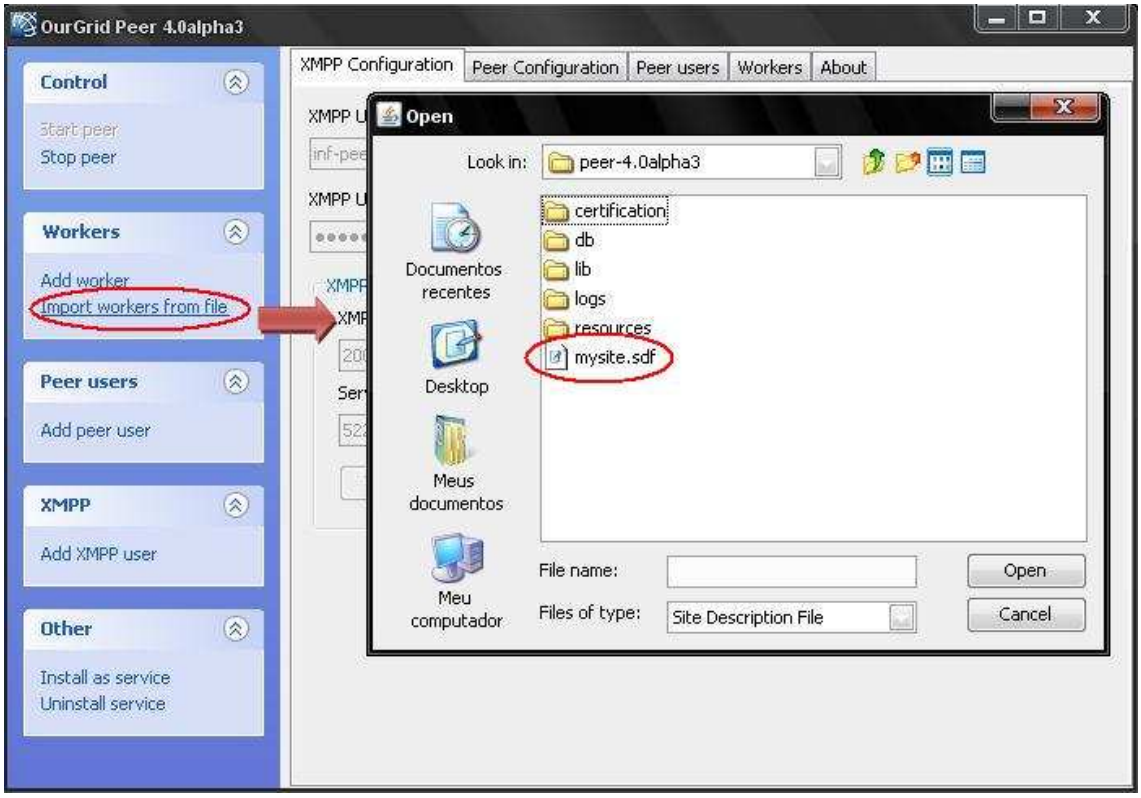

Figura A.4 – Procedimento para adicionar *Workers* ao *Peer*

Até este ponto o *Peer* de sua instituição já estará configurado, rodando e sabendo de todas as máquinas que ele irá gerenciar. O próximo passo é instalar e configurar o componente *Worker* nestas máquinas.

A.4 Instalação do Componente *Worker*

O *Worker* é o componente do OurGrid que efetivamente roda as tarefas submetidas por usuários da Grade. Quando um *Worker* recebe uma tarefa para processar, ele necessita criar uma máquina virtual para rodar a tarefa recebida. Como resultado disto, todos os processos da Grade (instâncias de uma tarefa) rodam numa máquina virtual, assim as máquinas onde os *Workers* executam estarão protegidas de uso malicioso evitando o acesso à áreas não permitidas do sistema e limitando as possibilidades de ocorrer danos ao sistema.

Para implementar isto, os *Workers* devem ser capazes de criar máquinas virtuais de acordo com a necessidade. Informações atualizadas sobre as máquinas virtuais que rodam no OurGrid podem ser obtidas pelo site `http://www.ourgrid.org`.

A.4.1 Requerimentos do Sistema

1) Espaço em disco: É requerido aproximadamente 2 GB de espaço livre para a completa instalação dos *Workers*. É importante salientar que esta grande quantidade de espaço necessário para instalar o *Worker* não é devido somente ao *Worker*, e sim devido ao ambiente virtual que ele usa para as questões de segurança. Além disto, a quantidade de espaço em disco disponível para a conta de usuário rodar o *Worker* irá determinar a quantidade de espaço que estará disponível para as tarefas que irão rodar na máquina.

2) Plataformas: Windows ou Linux.

3) Conectividade: O *Worker* deve ser capaz de se conectar a um servidor XMPP.

4) Java Runtime Environment (JRE) versão 1.6.0 ou superior.

A.4.2 Instalando o *Worker*

A.4.2.1 Instalação no Linux via Console

Em cada máquina da Grade executar os seguintes passos:

1) Descompactar o pacote obtido do site `http://www.ourgrid.org/?p=downloadAndDocumentation`.

2) Editar o arquivo "*worker.properties*" e atualizar os campos "*commune.xmpp.username"*, *"commune.xmpp.password"* e *"commune.xmpp.servername"* para conectar-se com o servidor XMPP.

3) Para que a comunicação entre *Peer* e *Worker* ocorra com sucesso é necessário copiar a chave pública do *Peer* e setá-la no *Worker*. Para isto, copie o valor do campo "*commune.publickey"* do arquivo "*peer.properties*" e cole no campo "*worker.peer.publickey*" do arquivo "*worker.properties*". Para o restante das propriedades os valores padrões devem estar corretos para a inicialização.

4) Executar o comando `worker start`.

A partir deste ponto o componente *Worker* já estará ligado e funcionando. Para checar o status do *Worker* digite o comando `worker status`.

A.4.2.2 Instalação no Windows via modo gráfico (GUI)

Em cada máquina descompactar o pacote obtido do site `http://www.ourgrid.org/?p=downloadAndDocumentation` e clicar duas vezes

no arquivo *"worker.bat"* ou executar a linha de comando `worker gui`. A interface gráfica do *Worker* é ilustrada na Figura A.5.

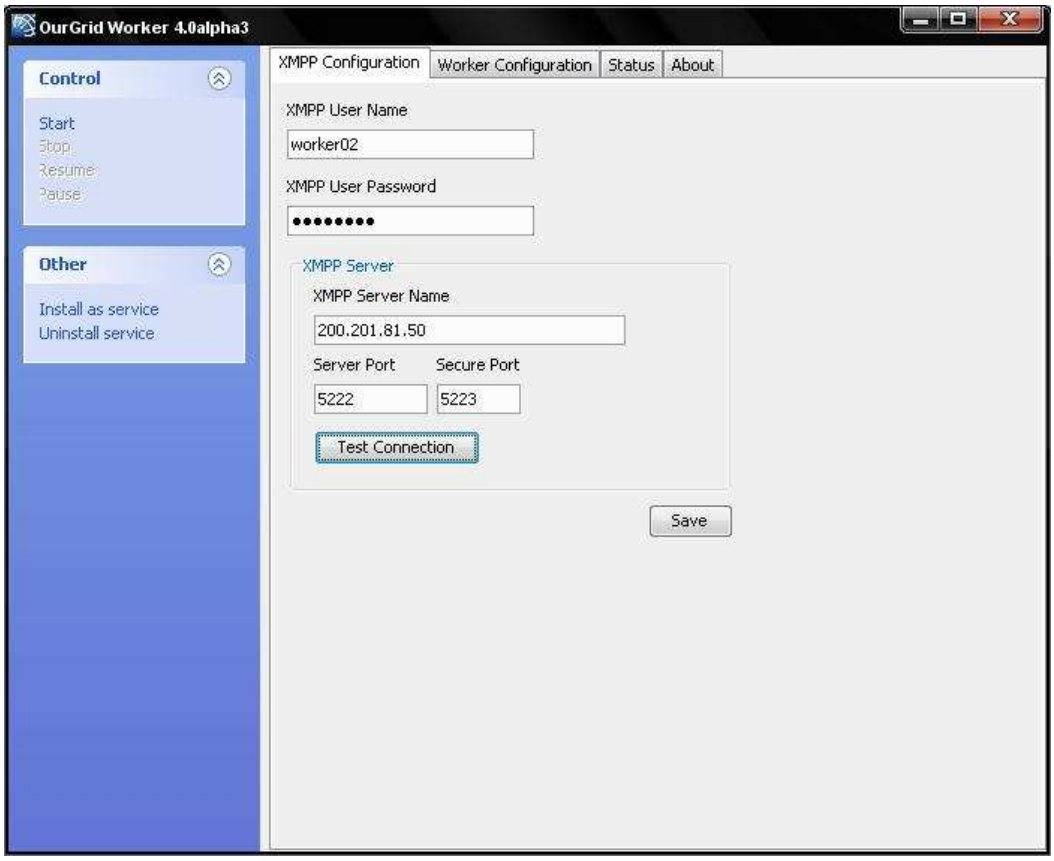

Figura A.5 – Janela inicial do componente *Worker*

Edite os campos *"XMMP User Name"*, *"XMPP User Password"* e *"XMPP Server Name"* com as informações pertinentes à Grade de sua instituição.

Para que a comunicação entre *Peer* e *Worker* ocorra com sucesso é necessário copiar a chave pública do *Peer* e setá-la no *Worker*. Para isto, copie o valor do campo "*commune.publickey*" do arquivo "*peer.properties*" e cole no campo "*Peer Public Key*" na guia "*Worker Configuration*" da GUI do *Worker*. A Figura A.6 ilustra esse procedimento.

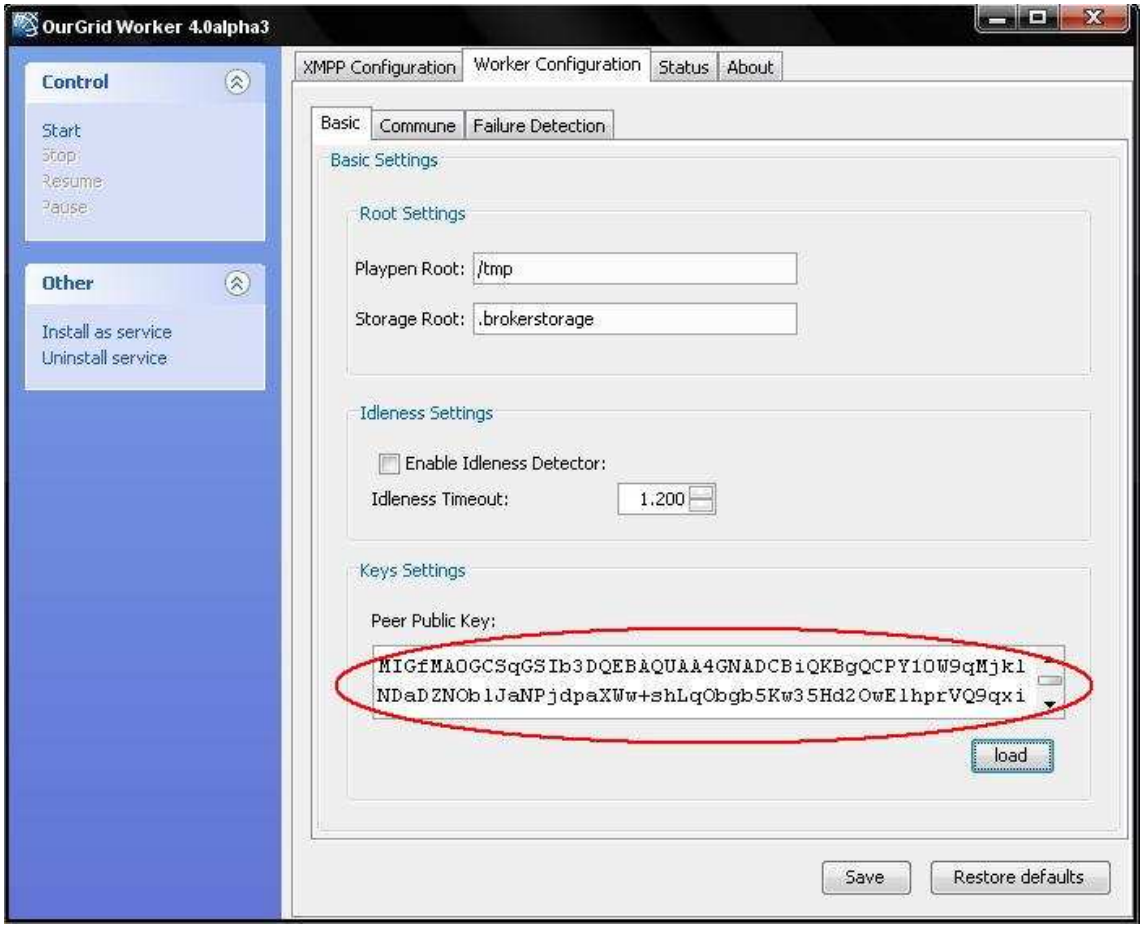

Figura A.6 – Procedimento para setar a chave pública do *Peer* no *Worker*

Não se esqueça de salvar as alterações e clique no Botão "*Start*" para inicializar o *Worker*. A Figura A.7 ilustra esse procedimento.

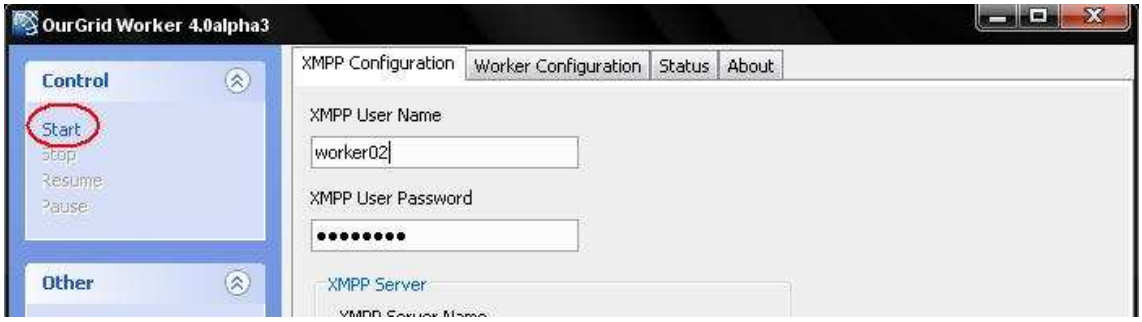

Figura A.7 – Procedimento de inicialização do *Worker*

Obs: Dentro do diretório do Worker tem um arquivo XML denominado worker.properties. Neste arquivo você encontra a configuração de disponibilidade do worker que por default está `worker.idlenessdetector=Yes,` isto significa que o worker vai poder ser usado apenas quando a máquina estiver ociosa. Para obter disponibilidade total do worker mude a configuração de Yes para No.

A.5 Instalação do Componente *Broker*

O *Broker* é o componente do OurGrid responsável pela submissão de tarefas (*jobs*) à Grade. Todo usuário que deseje rodar uma aplicação na Grade necessita instalar o componente *Broker* na sua máquina e configurá-lo para conectar-se ao *Peer* de diversas instituições.

A.5.1 Requerimentos do Sistema

1) Espaço em disco: Aproximadamente 9 MB para a instalação, mais espaço suficiente para armazenar os arquivos de *logs*.

2) Plataformas: Windows ou Linux.

3) Conectividade: O *Broker* deve ser capaz de se conectar à máquina onde o componente *Peer* está instalado e receber conexões de entrada via porta TCP.

4) Java Runtime Environment (JRE) versão 1.6.0 ou superior.

A.5.2 Instalando o *Broker*

Para instalar o componente *Broker* do OurGrid basta apenas descompactar o pacote obtido do site `http://www.ourgrid.org/?p=downloadAndDocumentation`. O resultado da árvore de diretórios descompactada pode ser visto no Quadro A.3.

```
|-- broker
|-- broker.bat
|-- log4j.cfg.xml
|-- testjvm
|--| certification
|--| examples
|--| lib
|--| logs
|--| resources
```

Figura A.3 – Árvore de diretórios do componente *Broker*

A.5.3 Acesso via modo gráfico (GUI)

Para acessar o modo gráfico do componente *Broker* dê um duplo clique no arquivo "*broker.bat*" ou execute a linha de comando `broker gui`. Este comando irá carregar a interface gráfica do *Broker* que é ilustrada na Figura A.8.

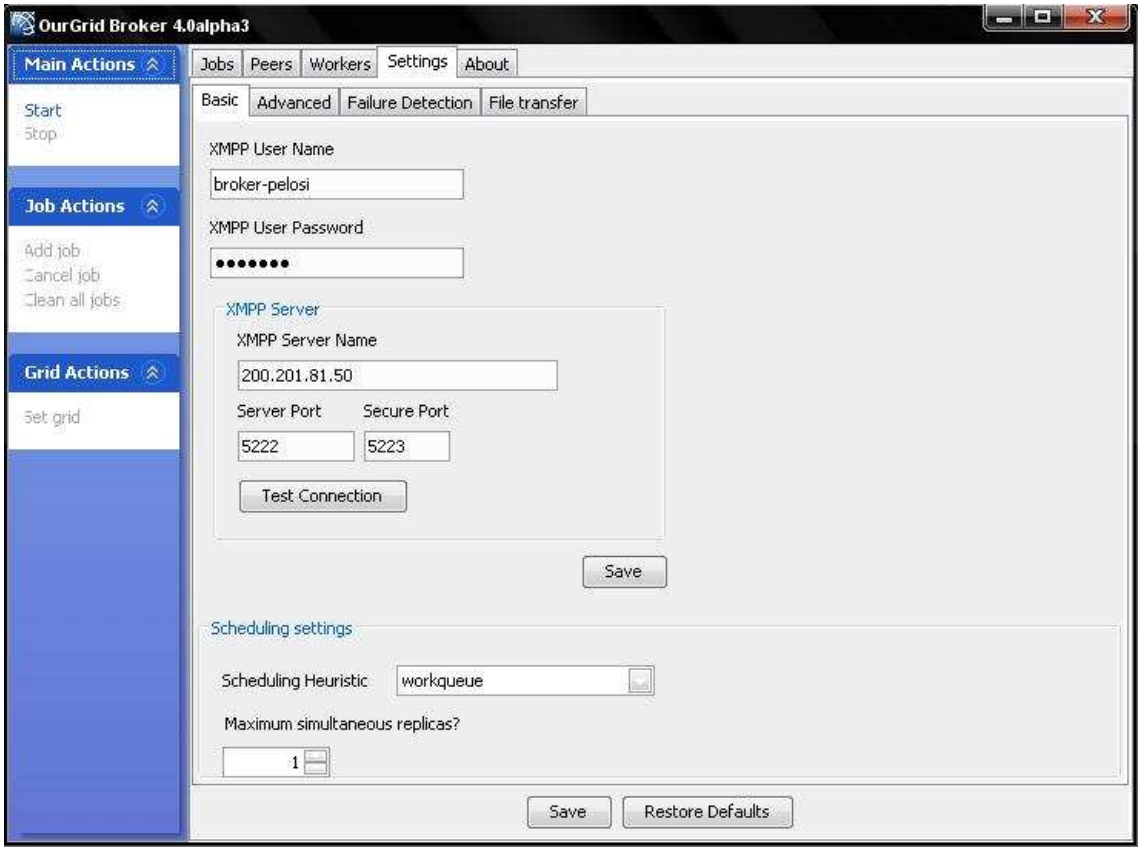

Figura A.8 – Janela inicial do componente *Broker*

Edite os campos "*XMMP User Name*", "*XMPP User Password*" e "*XMPP Server Name*". É provável que seja necessário contatar o administrador da Grade de sua instituição para conseguir um *login* de usuário para acessar o servidor XMPP.

Não se esqueça de salvar as alterações e clique no botão "*Start*" para inicializar o *Broker*. A Figura A.9 ilustra esse procedimento.

Figura A.9 – Procedimento de inicialização do *Broker*

A.5.4 Acesso via modo console

É necessário apenas digitar a linha de comando `broker start` para inicializar o *Broker*.

A.5.5 Criando um arquivo de descrição de Grade (*Grid File Description – GFD*)

O próximo passo é configurar os *Peers* cujo *Broker* irá conectar-se para enviar tarefas aos *Workers*. Para isto é necessário dizer ao *Broker* quais *Peers* estão disponíveis utilizando um arquivo de descrição de Grade (GDF).

Utilize um editor de texto de sua preferência para criar um arquivo "*mypeer.gdf*" que conterá a descrição de todos os *Peers* que deseja acessar. Tipicamente, utiliza-se somente um *Peer*, o qual estará rodando no seu *site*. E este *Peer* se encarregará de comunicar-se com o resto da comunidade para obter os recursos que necessita.

O Quadro A.4 ilustra um arquivo GDF contendo somente um *Peer* e o Quadro A.5 ilustra um arquivo GDF contendo dois *Peers* de *sites* distintos.

```
# Grid Description File

peer:
     label: Inf Peer
     username : inf-peer
     servername : 200.201.81.50
```

Quadro A.4 – Exemplo de arquivo GDF contendo apenas um *Peer*

```
# Grid Description File

peer:
     label: Inf Peer
     username : inf-peer
     servername : 200.201.81.50
peer:
     label: LSD Peer
     username : lsd-peer
     servername : xmpp.ourgrid.org
```

QuadroA.5 – Exemplo de arquivo GDF contendo dois *Peers* de *sites* distintos

A.5.6 Adicionando *Peers* ao *Broker*

Agora é necessário carregar o arquivo "*mypeer.gdf*" recém criado para que o *Broker* possa se comunicar com os *Peers* inseridos neste arquivo. A interface gráfica do *Broker* possui a opção "*Set grid*", como indicado na Figura A.10 onde é possível carregar o arquivo de descrição de Grade.

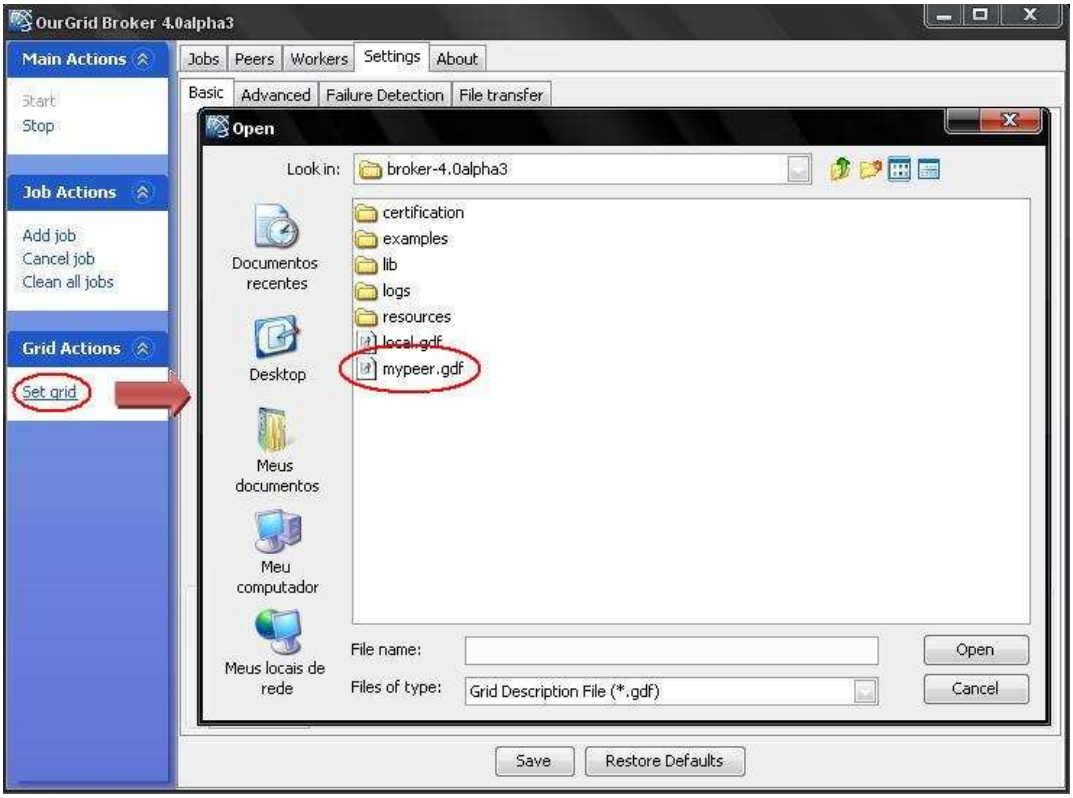

Figura A.10 – Procedimento para carregar arquivo GDF

Outra forma de carregar o arquivo GDF é executar a linha de comando `broker setgrid mypeer.gdf`.

Para checar se o *Broker* carregou corretamente as configurações dos *Peers* execute a linha de comando `broker status`. Após isto já é possível submeter tarefas à Grade.

A.5.7 Submetendo tarefas

Para submeter uma tarefa clique no botão "*Add job*" e carregue o arquivo de descrição de tarefas (*Job Description File – JDF*) que se deseja submeter à Grade. A Figura A.11 ilustra este procedimento.

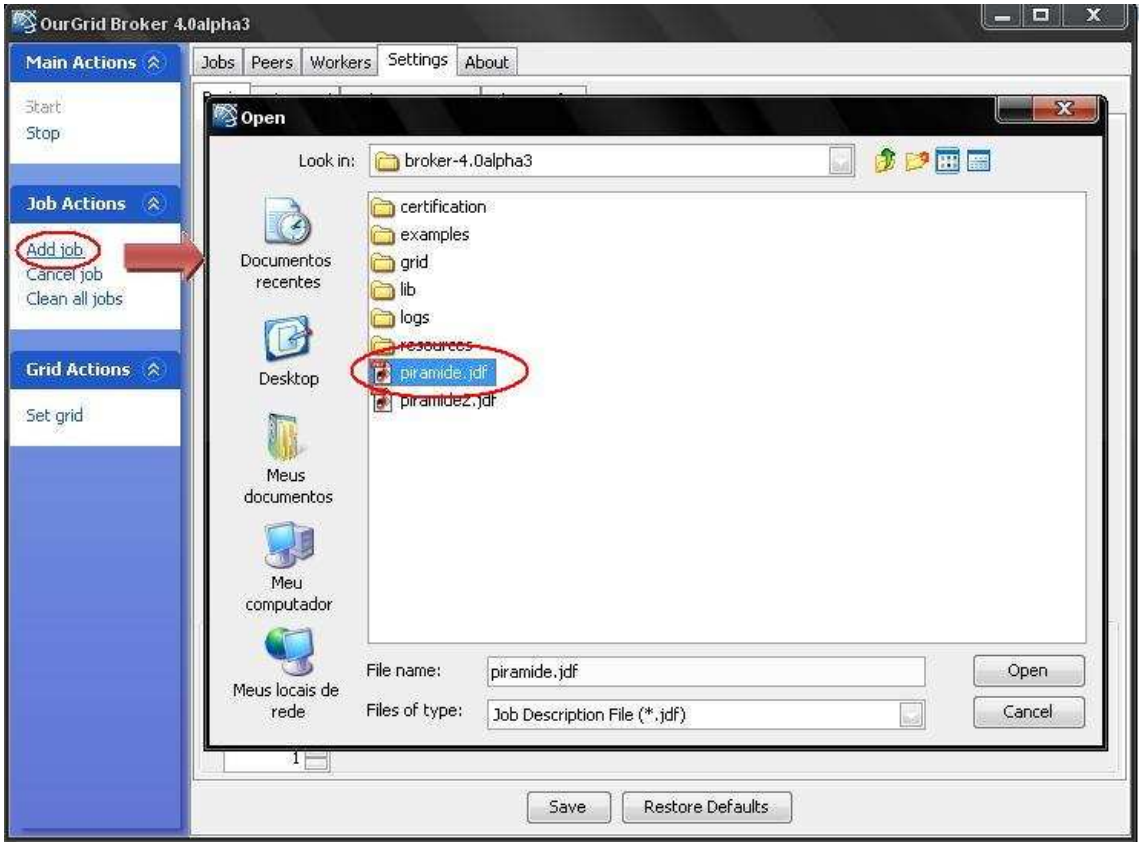

Figura A.11 – Procedimento para carregar arquivo JDF

Se preferir execute a linha de comando `broker addjob examples/addJob /[arquivo].jdf]`.

O Quadro A.6 ilustra um arquivo simples JDF contendo um *Job* chamado *gethostname* que contém três tarefas que irão retornar o nome do host das máquinas onde irão executar.

```
job:
     label: gethostname
task:
     remote: hostname > output-1
     final : get output-1 output-1
task:
     remote: hostname > output-2
     final : get output-2 output-2
task:
     remote: hostname > output-3
     final : get output-3 output-3
```

Quadro A.6 – Exemplo de arquivo JDF contendo três tarefas

Apêndice B

Instalação do Ganglia

O Ganglia é uma ferramenta de monitoramento distribuído desenvolvida para trabalhar em ambientes computacionais de alto desempenho, como Clusters e Grades computacionais. O sistema fornece informações referentes aos diversos recursos disponíveis nos nós da Grade.

A ferramenta é composta por dois daemons principais:

O *ganglia monitoring daemon – gmond* – deve ser executado em todos os computadores da Grade. É responsável por monitorar os recursos locais do nó e enviar as informações ao *gmetad* local.

O *ganglia meta daemon – gmetad* – agrupa os dados enviados pelos *gmonds*. Os *gmetads* podem trocar informações entre si permitindo que diferentes *clusters* sejam monitorados e agrupados de forma hierárquica.

As informações são apresentadas aos usuários por uma interface web capaz de reproduzir gráficos referentes aos dados do *gmetad*.

B.1 Instalação do gmond

O Ganglia-Monitor (*gmond*) deve ser instalado nos nós da Grade que serão monitorados. No caso do OurGrid, nas máquinas trabalhadoras (*Workers*). O sistema coletará as informações referentes aos recursos e repassará ao gmetad.

Abaixo são descritos os passos para instalação e configuração do componente *gmond*.

1) Utilizando o Sistema Operacional Linux, se a distribuição for Debian, o ganglia-monitor encontra-se no repositório. Basta abrir o terminal e executar a seguinte linha de comando:

```
apt-get install ganglia-monitor
```

2) Utilizando o Sistema Operacional Windows você terá que baixar o arquivo do link abaixo e dar um duplo click no arquivo executável.

http://sourceforge.net/projects/ganglia/files/ganglia%20monitoring%20core/3.0.0%20%28Kittyhawk%29/ganglia-3.0.0-setup.exe/download

3) Para iniciar o serviço basta executar no terminal a linha de comando.

```
/etc/init.d/ganglia-monitor start
```

4) O *gmond* gera os dados em formato XML e estes podem ser visualizados apontando seu web-browser para o endereço http://host:8649/ , onde "*host*" é o nome da máquina na qual o sistema foi instalado. No caso de estar na máquina local use http://localhost:8649/, como mostra a Figura B.1.

5) Para configurar o componente *gmond* será necessário abrir o arquivo *gmond.conf* que se encontra na pasta /etc/gmond.conf, onde será necessário definir qual a interface que realizará a divulgação dos dados através de multicast adicionando a linha mcast_if eth1. Também neste arquivo é definido o nome da Grade que está sendo monitorado.

6) Após as configurações serem alteradas o gmond deve ser reiniciado.

```
/etc/init.d/ganglia-monitor restart
```

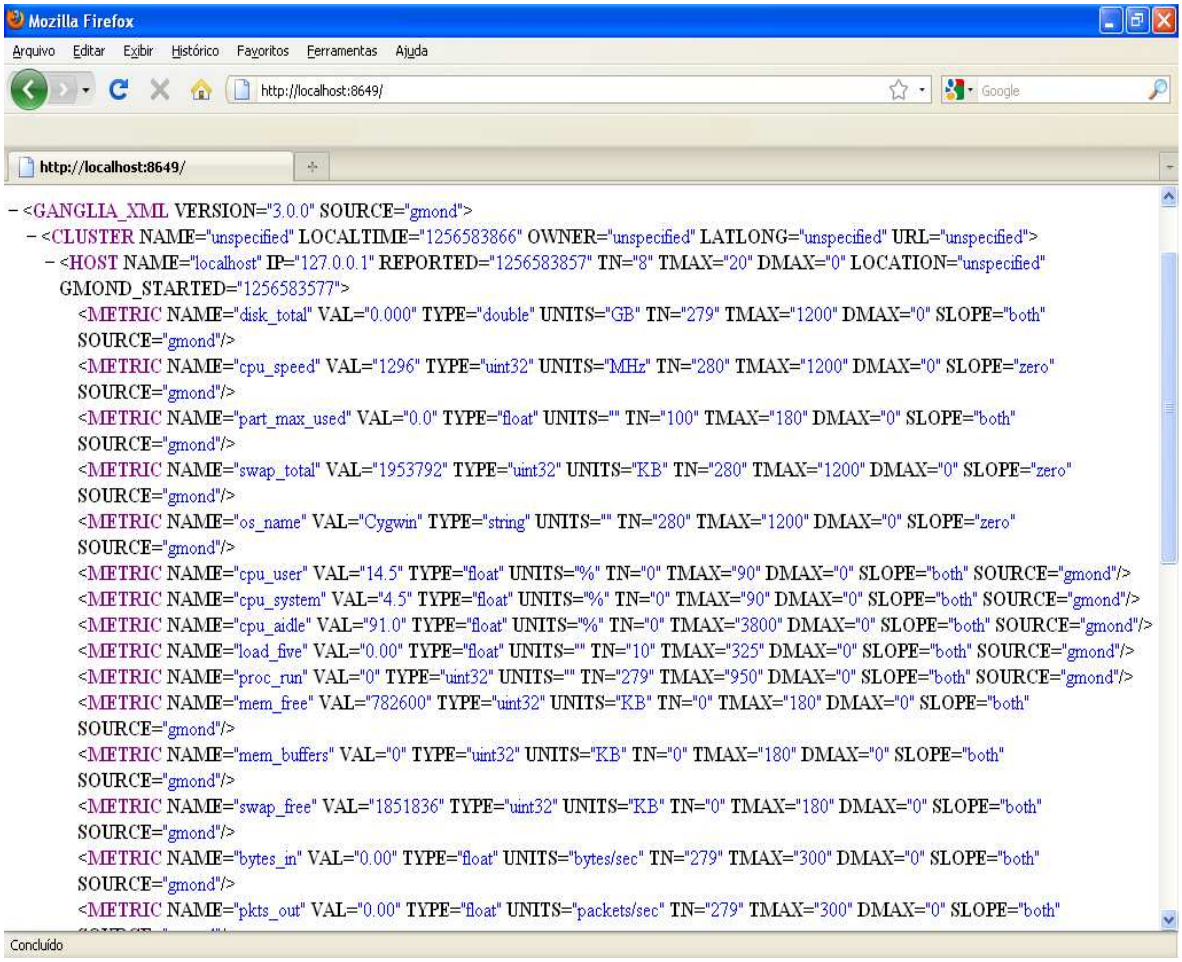

Figura B1 – Dados gerados pelo gmond.

A Figura B1 ilustra os dados capturados pelo *gmond* em um dos nós da Grade, representados em formato XML.

B.2 Instalação do gmetad

O Ganglia-Meta-Daemon (*gmetad*) é o componente responsável por centralizar os dados enviados pelos *gmonds*, sendo também possível agrupar dados enviados por outros *gmetads*, gerenciando tanto Grades quanto Clusters.

O componente *gmetad*, por centralizar os dados dos nós, deve ser instalado no servidor junto com o Peer no caso do OurGrid.

1) Para instalar o *gmetad* no Debian, pelo fato do mesmo já estar no repositório, basta abrir o terminal e executar a seguinte linha de código:

```
apt-get install gmetad
```

2) Para iniciar o serviço basta executar no terminal a linha de comando:

```
/etc/init.d/gmetad start
```

3) O *gmetad* automaticamente recebe os dados enviados pelos *gmonds* instalados na rede. Esses dados são agrupados e apresentados em formato XML e estes podem ser visualizados apontando seu web-browser para o endereço *http://host:8651*, onde "host" é o nome da máquina na qual o *gmetad* foi instalado. Se o servidor estiver na máquina local, será necessário usar *http://localhost:8651/*, como ilustra a Figura B.2.

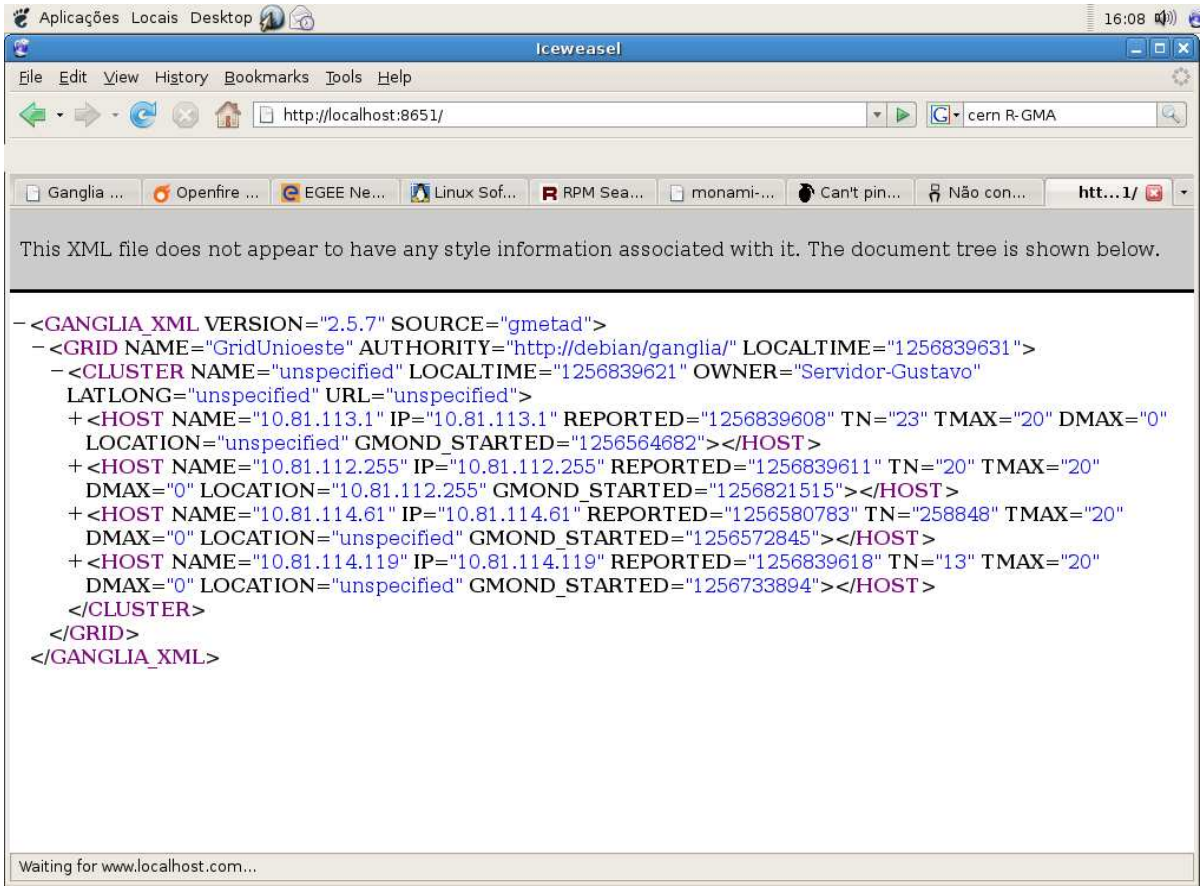

Figura B2 – Dados agrupados pelo *gmetad*.

A Figura B2 representa os dados agrupados pelo *gmetad*, onde cada HOST representa um nó da grade que vem sendo monitorado por um componente *gmond*.

B.3 Instalação da Interface web

A interface *web* representa as informações agrupadas pelo *gmetad* em forma de gráficos para que estas sejam interpretadas mais facilmente pelos usuários.

Este componente deve ser instalado na mesma máquina onde o *gmetad* foi instalado.

1) Antes de instalar a interface *web*, deve ser instalada a biblioteca Libconfuse e a ferramenta RRDTOLL, ambos encontram-se no repositório do Debian. Para isso, basta abrir o terminal e executar a seguinte linha de código:

```
apt-get install libconfuse-dev
apt-get install rrdtool
```

O RRDTOOL é o banco de dados responsável pela geração dos gráficos.

2) É necessária a instalação do PHP para que os gráficos possam ser vizualizados.

2.1) Abra o seguinte arquivo:

```
/etc/apt/sources.list
```

2.2) Adicionar os seguinte servidores na lista:

```
deb http://security.debian.org stable/updates main
deb http://ftp.br.debian.org/debian/ stable main
deb-src http://ftp.br.debian.org/debian/ stable main
deb http://ftp.us.debian.org/debian stable main
deb-src http://ftp.us.debian.org/debian/ stable main
```

2.3) No console digite para atualizar o repositório:

```
apt-get update
```

2.4) Instalando php5:

```
apt-get install php5-common libapache2-mod-php5 php5-
cli
```

2.5) Reiniciando o servidor:

```
/etc/init.d/apache2 restart
```

3) Baixe a interface web no link abaixo:

http://www.rpmfind.net/linux/rpm2html/search.php?query=ganglia-webfrontend

4) Extraia o tarball para a pasta:

`/var/www/`

5) Os gráficos podem ser visualizados, como mostra a Figura B3, direcionando o navegador web para o link:

`http://localhost/ganglia-webfrontend`

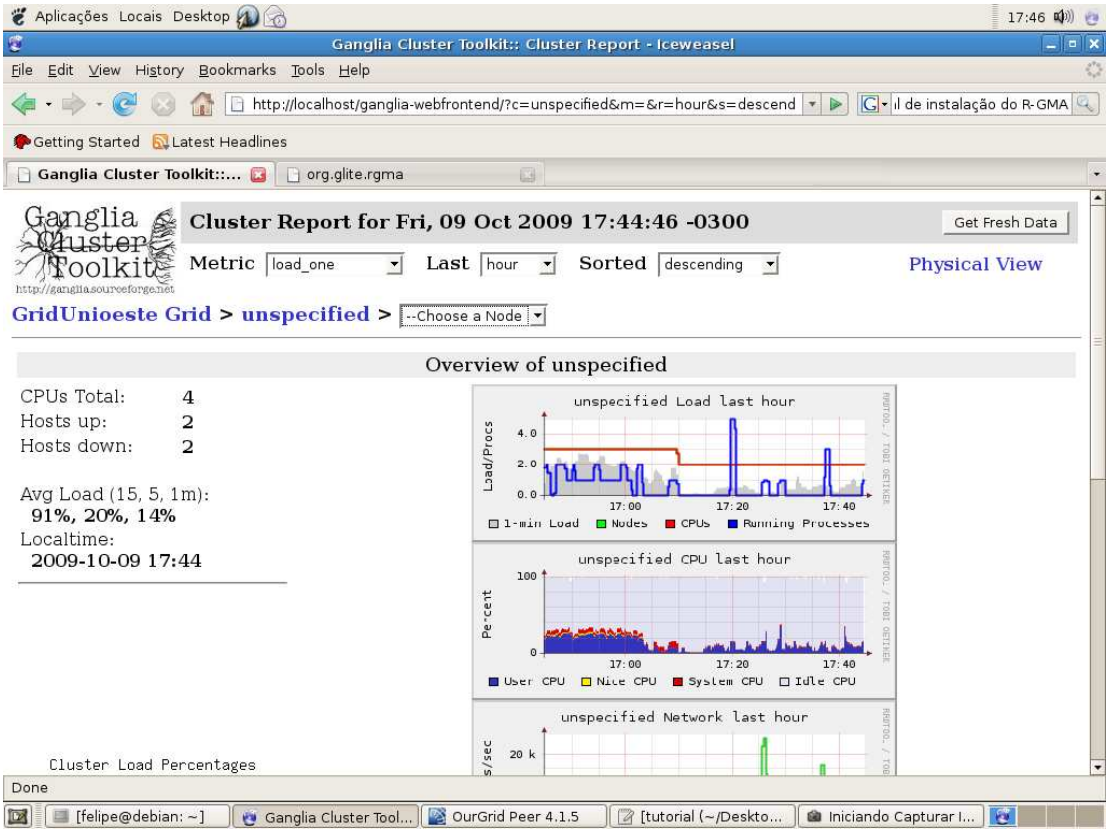

Figura B3 – Gráficos gerados pela interface web.

A Figura B3 ilustra os gráficos gerados pela interface *web*, a partir dos dados agrupados pelo *gmetad*.

A linha azul, no primeiro gráfico, representa os processos em execução nas máquinas da Grade e a linha vermelha as CPUs disponíveis. À esquerda dos gráficos é informado o número total de CPUs e quais estão disponíveis.

Referências Bibliográficas

[1] COLVERO, A. T.; DANTAS, M.; CUNHA, P. D. **Ambientes de *Clusters e Grids* Computacionais: Características, Facilidades e Desafios.** Florianópolis, Santa Catarina: Universidade do Extremo Sul Catarinense. 2004.

[2] ZEM, J. L.; BRITO, S. H. B. **Monitoramento Distribuído de *Clusters* e *Grids* Computacionais Utilizando o Ganglia.** Piracicaba, São Paulo: Universidade Metodista de Piracicaba. 2007.

[3] CIRNE, W.; SANTOS-NETO, E. ***Grids* Computacionais: da Computação de Alto Desempenho a Serviços Sob Demanda.** Campina Grande, Paraiba: Universidade Federal de Campina Grande, Laboratório de Sistemas Distribuídos. 2004.

[4] MINETTO, E. L. **Portais de *Grids* Computacionais.** Florianópolis, Santa Catarina: Universidade Federal de Santa Catarina. Fevereiro, 2005.

[5] DOMINGUES, H. H. **NCC – *Node Control Center.*** São Paulo, Universidade de São Paulo, Departamento de Ciências da Computação. 2006.

[6] MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. **The Ganglia Distributed Monitoring System: Design, Implementation, and Experience.** Berkeley, Estados Unidos. Junho, 2004.

[7] SIMON, Z., G. **Sistema de Coleta de Recursos em Grids**. Florianópolis, Universidade Federal de Santa Catarina. Outubro, 2005.

[8] NASCIMENTO, A. P. **Conhecendo As Grades Computacionais.** Niterói, Rio de Janeiro, Brasil. 2005.

[9] EGEE, Enabling Grids for E-sciencE. **R-GMA System Specification.** Março, 2008.

[10] CIRSTOIU, C.; COSTIN, C.; LATCHEZAR, L., B.; COSTAN, A. **Monitoring, Accounting and Automated Decision Support for the ALICE Experiment Based on the MonALISA Framework**. In Proceedings of the 2007 Workshop on Grid Monitoring, p.39-44, 2007.

[11] LEGRAND, I. C. et al. **Monalisa**: An Agent Based, Dynamic Service System to Monitor, Control and Optimize Grid Based Application. In Chep, Interlaken, 2004.

[15] FOSTER, I., KESSELMAN, C. e TUECKE, S. **The Anatomy of the Grid: Enabling Scalable Virtual Organizations.** Disponível em:

Http://www.globus.org/research/papers/anatomy.pdf. Acesso em: Março de 2009.

[16] CHERVENAK, A.; FOSTER, I.; KESSELMAN, C.; SALISBURY, C.; TUECKE, S. **The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets.** *J. Network and Computer Applications*, 2001.

[17] CZAJKOWSKI, K., FITZGERALD, S., FOSTERr, I. and KESSELMAN, C. **Grid Information Services for Distributed Resource Sharing**, 2001.

[18] HOSCHEK, W., JAEN-MARTINEZ, J., SAMAR, A., STOCKINGER, H. and STOCKINGER, K., **Data Management in an International Data Grid Project**. In *Proc. 1st IEEE/ACM International Workshop on Grid Computing*, 2000, Springer Verlag Press.

[19] PEIXOTO, D., M. et al. **Instalação e Configuração do Globus Toolkit para Computação em Grid.** CBPF-NT. Dezembro de 2003.

[20] IBM. **Introduction to Grid Computing With Globus.** Nova York, Estados Unidos da America. Setembro, 2003.

[21] PELOSI, O., L., J. **Adaptação do PIRAMEDE – Programa Interpretador e Resolvedor de Autômatos e MáquInas De Estados finitos.** Monografia. Cascavel, Paraná. Universidade Estadual do Oeste do Paraná. 2008.

[22] OURGRID. Site oficial do projeto OurGrid. Disponível em: *http://www.ourgrid.org*. Acesso em: 30 de Novembro de 2009.

[23] HAWKEYE. Disponível em: *http://www.cs.wisc.edu/condor/hawkeye/*. Acesso em: 23 de Junho de 2009.

[24] R-GMA. Site oficial do projeto R-GMA. Disponível em: *http://www.r-gma.org/*. Acesso em: 23 de Junho de 2009.

[25] CHAKRABARTI, A. **Grid Computing Security.** Disponível em: *http://books.google.com.br/books?id=1U3u8Ch4iwkC&pg=PA258&lpg=PA258&dq=hawkeye+GRID&source=bl&ots=bZMFniSV74&sig=n4b9J3UC_FX3eMHK5Q-7bY07eHo&hl=pt-BR&ei=WR1CSpeSH-OwtgfL6_SmCQ&sa=X&oi=book_result&ct=result***.** Acesso em: 13 de Junho de 2009.

[26] NGS. Site oficial do projeto National Grid Service. Disponível em: *http://www.grid-support.ac.uk/component/option,com_frontpage/Itemid,165/*. Acesso em 10 de Novembro de 2009.

[27] MIRANDA, R., BRASILEIRO, F. **Integrando uma Semântica de Falhas Consistente na Comunicação Assíncrona de Objetos Distribuídos.** Campina Grande, Paraíba. Universidade Federal de Campina Grande. 2009.

[28] BERMAN, F., WOLSKI, R. **The AppLeS Project:** A Status Repost. In Proceedings of 8th NEC research Symposium, 1997.

[29] LITZKOW, M. J., LIVNY, M., MUTKA, M. W. **Condor: A Hunter for Idle Workstation**. In Proceedings of the 8th International Conference on Distributed Computing Systems, p.104-111, 1988.

[30] WEISSMAN, J. B., GRIMSHAW, A. **A Federated Model for Sheduling in Wide-Area Systems.** In Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing, p.542-550, 1996.

[31] CONDOR. Site Oficial do Projeto CONDOR. Disponível em: *http://www.cs.wisc.edu/condor/*. Acesso em: 23 de Junho de 2009.