UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DE ESTÁGIO

ANÁLISE COMPARATIVA DO USO DE TECNOLOGIAS DE GRADES COMPUTACIONAIS DE DESKTOP

RICARDO ARAÚJO SANTOS

ESTAGIÁRIO

RAQUEL VIGOLVINO LOPES

ORIENTADORA ACADÊMICA

MARCUS WILLIAMS AQUINO DE CARVALHO

SUPERVISOR TÉCNICO

CAMPINA GRANDE, PARAÍBA, BRASIL

ANÁLISE COMPARATIVA DO USO DE TECNOLOGIAS DE GRADES COMPUTACIONAIS DE DESKTOP

Aprovado em _____

BANCA EXAMINADORA

Prof^a. Dr^a. Raquel Vigolvino Lopes

ORIENTADORA ACADÊMICA

Prof. Dr. Francisco Vilar Brasileiro

MEMBRO DA BANCA

Prof^a. Dr^a. Joseana Macêdo Fechine

MEMBRO DA BANCA

Agradecimentos

Ao Laboratório de Sistemas Distribuídos pela oportunidade de realizar o estágio. Aos supervisores pela orientação. Aos integrantes do LSD pelas dicas e à equipe de suporte pela paciência.

Apresentação

Como parte das exigências do curso de Ciência da Computação, da Universidade Federal de Campina Grande, para cumprimento da disciplina de estágio integrado, é apresentado o relatório de estágio seguinte.

O estágio foi realizado no LSD sob supervisão acadêmica da professora doutora Raquel Vigolvino Lopes e técnica de Marcus Williams Aquino de Carvalho, no período 2009.1.

O conteúdo do relatório está distribuído conforme descrição a seguir:

Seção 1 - Introdução

Seção 2 - Objetivos

Seção 2 - Ambiente de Estágio

Seção 3 - Fundamentação Teórica e Tecnologias Utilizadas

Seção 4 - Metodologia

Seção 4 - Resultados Obtidos

Seção 5 - Considerações Finais

Referências Bibliográficas

Resumo

O uso de grades computacionais de desktop no contexto da pesquisa científica é uma alternativa barata a aquisição de supercomputadores ou clusters dedicados e muito tem sido pesquisado nessa área. Algumas tecnologias de grades computacionais têm se tornado bem conhecidas e aplicam na prática muito do conhecimento produzido nas pesquisas. Este estágio tem por objetivo comparar e contrastar algumas dessas tecnologias identificando características predominantes nos pacotes de *middleware* mais em uso atualmente.

Sumário

Lista de Siglas e Abreviaturas

BoT Bag-of-Tasks

DAG Directed Acyclic Graph

GRAM Grid Resource Access and Management

HTTP Hypertext Transfer Protocol

I/O Input/Output

JDF Job Descriptor File

JVM Java Virtual Machine

LAL Laboratoire de L'Accélérateur Linéaire

LAN Local-area Network

LRI Laboratoire de Recherche en Informatique

LSD Laboratório de Sistemas Distribuídos

MPP Massively Parallel Processor

NoF Network of Favors

NOW Network of Workstations

PUC-Rio Pontifícia Universidade Católia do Rio de Janeiro

QoS Quality of Service

RNP Rede Nacional de Pesquisa

SSL Space Sciences Laboratory

UFG Universidade Federal de Goiás

UFMA Universidade Federal do Maranhão

UFMS Universidade Federal de Mato Grosso do Sul

URI Unified Resource Idintifier

USP Universidade de São Paulo

VME Virtual Machine Environment

XMPP Extensible Messaging and Presence Protocol

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A pesquisa científica em algumas áreas tem demandado cada vez mais poder computacional, seja para a realização de simulações como para a execução de experimentos. Uma saída natural é a aquisição de supercomputadores ou máquinas dedicadas (*clusters*) exclusivamente à computação paralela, o que nem sempre é possível dado seu elevado custo. As grades computacionais [**?**, **?**] surgiram com a idéia de resolver tais problemas computacionais de maneira eficiente e com baixo custo, através do compartilhamento de recursos.

Uma forma comum de compartilhamento de recursos faz uso de poder computacional ocioso, formando o que se chama no contexto desse trabalho de grades computacionais de *desktop*. Exemplos de grades desse tipo são: Condor [**?**], BOINC [**?**], XtremWeb [**?**] e OurGrid [**?**].

Muito tem sido publicado em conferências e jornais sobre essas ferramentas. No entanto, não são conhecidos estudos empíricos que visem uma caracterização do uso das ferramentas disponibilizadas, impossibilitando identificar o estado-da-prática desta área. Acredita-se que um estudo prático, em contrapartida aos estudos teóricos conhecidos, é de grande importância tanto para identificação de lacunas quanto para seleção do melhor serviço em diferentes contextos.

Realizar esse tipo de pesquisa num estágio é uma oportunidade de pôr em prática alguns conhecimentos adquiridos durante a vida acadêmica do estudante de graduação já que é feita a ponte entre as teorias aprendidas nas disciplinas e como elas são usadas na prática por um pesquisador.

O estágio do qual se trata esse relatório foi realizado como disciplina do currículo opta-

tivo do Curso de Bacharelado em Ciência da Computação no período 2009.1 e realizado no Laboratório de Sistemas Distribuídos na Universidade Federal de Campina Grande.

Capítulo 2

Objetivos

O principal objetivo desse trabalho é fazer uma comparação quanto ao uso de algumas tecnologias de grades de desktops não comerciais.

Como objetivos específicos esperava-se:

- Realizar um levantamento bibliográfico sobre as grades desktop mais usadas atualmente;
- Levantar métricas que possam ser usadas nesse estudo comparativo;
- Comparar as grades identificadas na etapa inicial do estágio com base nas métricas a serem definidas.

Capítulo 3

Ambiente de Estágio

O estágio foi desenvolvido no Laboratório de Sistemas Distribuídos (LSD) do Departamento de Sistemas e Computação (DSC) da Universidade Federal de Campina Grande.

O laboratório foi criado em 1996, como forma de aglutinar pesquisadores e alunos do DSC e de outros departamentos em torno de projetos na área de Sistemas Distribuídos. Atualmente o LSD é coordenado pelo professor Francisco Brasileiro. As pesquisas do LSD estão concentradas em Grades Computacionais, Sistemas *Peer-to-Peer*, *Cloud Computing*, Tolerância a Falhas, Desenvolvimento de Software Distribuído e Aplicações Industriais.

3.1 Estrutura Física

O LSD está instalado num prédio com $550m^2$ de área e conta com a colaboração de dezenas de alunos desenvolvendo trabalhos de doutorado, mestrado e iniciação científica, além de vários pesquisadores. Os diferentes projetos em execução estão distribuídos em 8 (oito) salas climatizadas, cada uma contanto com quadro branco, pincéis e um acervo bibliográfico de diversos temas em computação e engenharia. Cada pessoa possui um posto de trabalho individual, com máquinas conectadas à Internet via POP-PB da RNP.

Endereço: Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso, 882 - Bloco CO

Bodocongó, CEP 58109-970 Campina Grande - PB, Brasil

Fone: +55 83 3310 1365

Fax: +55 83 3310 1498

3.2 Rucursos Utilizados

3.2.1 Hardware

Foram disponibilizados quatro computadores de configurações distintas.

3.2.2 Software

Os recursos de software utilizados foram os seguintes:

- Eclipse: Como ambiente de desenvolvimento da aplicação Java usada para testar as grades computacionais. Além disso, o Eclipse em conjunto com o plugin Texlipse foi usado como editor de texto LaTeX;
- VServer: Como ambiente de máquinas virtuais.

Capítulo 4

Fundamentação Teórica

Esta seção tem por finalidade introduzir alguns dos conceitos abordados no estágio. Primeiro é dada uma contextualização com o histórico das grades computacionais. Depois, são definidos alguns conceitos importantes e dados exemplos de algumas aplicações executadas em grades computacionais.

4.1 Grades Computacionais

4.1.1 Histórico

Usar computadores ociosos conectados em rede para execução de tarefas não é uma idéia nova. Em 1982, o *Worm* da Xerox executava nos *desktops* da empresa [**?**] aplicações que variavam muito de objetivo. Os *worms* se espalhavam na rede detectando máquinas ociosas e se replicando, e realizavam tarefas complexas como diagnóstico de problemas na Ethernet e até a criação de animações em tempo real.

No início dos anos 90, com a difusão dos *applets* Java [**?**], surgiram sistemas como Javelin [**?**] e Bayanihan [**?**]. Tais aplicações se propagavam através dos navegadores, o que era uma vantagem dado que o tráfego era realizado via *Hypertext Transfer Protocol* (HTTP). Os sistemas eram flexíveis e facilmente instaláveis. No entanto, os navegadores executavam os *applets* em *sandboxes* o que tornava impossível o acesso a alguns serviços como monitores de carga de CPU e memória principal. As aplicações executadas eram restritas também já que não podiam realizar leitura ou escrita no sistema de arquivos local. A escalabilidade não

foi provada dado que esses sistemas não foram implantados em mais que algumas dezenas de recursos. Além disso, não existiam ferramentas de gerência para tais aplicativos.

Ainda nos anos 90, Anderson, Culler e Patterson [**?**] argumentavam que as *Networks of Workstations* (NOWs) podiam ter um desempenho igual ou maior que máquinas com Massively Parallel Processor (MPP). Este argumento era suportado por três particularidades: (i) a capacidade das redes locais (Local-area Networks - LANs) permitia a escala das aplicações com protocolos assíncronos de transferência de dados; (ii) as máquinas *desktop* estavam melhorando significativamente em poder de processamento; e (iii) operações de I/O continuavam a ser um gargalo para supercomputadores e que podia ser contornado com a paralelização nas NOWs.

Foi também nos anos 90, que um sistema chamado Condor [**?**] foi ganhando espaço. O Condor é um *middleware* para execução distribuída de aplicações em *batch*. O Condor originalmente foi concebido para uso em ambientes LAN, mas atualmente permite a comunicação em escala global, chegando a ter cerca de 274498 máquinas em 1897 *pools* em Março de 2009[**?**].

A difusão da Internet no fim dos anos 90 trouxe a possibilidade de conectar milhões de *desktops* ao redor do mundo e com isso projetos de computação voluntária como, por exemplo, o SETI@home [**?**], ganharam popularidade. A computação voluntária fazia uso dos *desktops* de pessoas comuns ao redor do mundo para a execução de aplicações altamente paralelizáveis em troca de prestígio social. O SETI@home, por exemplo, divulga os maiores contribuintes na sua página inicial além de fornecer proteções de tela com imagens em três dimensões relacionadas à pesquisa.

Após isso, vários projetos em universidade começaram a desenvolver sistemas para aplicações em grades de *desktop*, entre eles o XtremWeb [**?**] e BOINC [**?**]. Algumas empresas também decidiram lançar seus produtos destinados a comunidade corporativa como o Entropia [**?**], United Devices (apud [**?**]). E mais recentemente, em 2004, InteGrade [**?**] e OurGrid [**?**] surgem como iniciativas nacionais no uso de poder computacional de *desktops* ociosos.

4.1.2 Conceitos

Grades computacionais são sistemas distribuídos caracterizados pelo compartilhamento de recursos heterogêneos localizados em diferentes domínios administrativos (federados) [**?**]. O *middleware* de grade computacional é a camada de software que possibilita a comunicação entre os recursos da grade e as aplicações de forma consistente e homogênea. A arquitetura do middleware define o propósito, as funções e interações entre os componentes para gerenciar recursos tais como processamento e armazenamento [**?**].

Não existe um consenso na literatura sobre a classificação das grades computacionais. Porém alguns conceitos se destacam e são necessários para o entendimento desse trabalho.

As grades computacionais são chamadas oportunistas quando os recursos que compõem a grade são usados quando estão ociosos. Estudos mostram que estações de trabalho ficam ociosas entre 60% e 80% do seu tempo de vida[**?**]. Dessa forma, grades oportunistas oferecem uma forma barata de se prover computação de alta vazão usando estações *desktop* ociosas.

Nesse contexto existem também as grades de *desktops*, que podem ser definidas como um conjunto de *desktops* conectados em rede com a intenção de se prover computação distribuída de alta vazão. Os recursos de processamento e armazenamento são utilizados de forma oportunista, logo são, naturalmente, grades oportunistas, ou seja, as aplicações do dono da máquina têm prioridade sobre as aplicações executadas na grade e estas são executadas somente quando não há demanda do dono da máquina. Além disso, a disponibilidade do recurso depende de fatores, como falha de rede, *hardware* ou *software*, e não existe nenhuma garantia sobre os recursos doados aos clientes da grade. Além da volatilidade dos recursos, outra característica que se destaca é a heterogeneidade dos recursos que compõem a grade no tocante a processadores, sistema operacional, freqüência de *clock*, espaço em disco e velocidade de acesso à rede, por exemplo.

Quando usuários domésticos passam a doar recursos ociosos para a grade sem, em troca, consumir recursos da grade, tem-se o que se chama de computação voluntária. O domínio sobre os recursos da grade estende as fronteiras dos laboratórios envolvidos e a comunidade passa a ter certo envolvimento com a pesquisa. Isso acontece, por exemplo, com o projeto SETI@Home onde qualquer pessoa pode doar recursos para a grade e em troca os doadores mais ativos são listados na página principal do projeto.

Em contrapartida, existem ainda as grades computacionais de serviço. Nesse caso, organizações virtuais tipicamente mantém uma grade formada por recursos dedicados e oferecem garantias de Qualidade de Serviço (*Quality of service* - QoS) a seus clientes. Além disso, os recursos são fechados por políticas de acesso restrito firmados por essas organizações virtuais. Este trabalho mantém o foco nas grades computacionais de *desktop* classificadas como oportunistas.

Além dos termos apresentados, as próximas seções usam alguns termos informalmente convencionados pela comunidade científica. Cliente é aquele que deseja executar aplicações ou *jobs*, que são conjuntos de tarefas ou *tasks*. O cliente submete aplicações para a grade de modo que o escalonador do servidor envie tarefas para *workers* que estejam disponíveis. Os *workers*, por sua vez, são *daemons* que executam nas máquinas (recursos) doadas à grade.

Uma vez definidos os termos usados, pode-se pensar em características de um modelo ideal de grade de *desktops*. Algumas são:

1. Escalabilidade: A vazão do sistema deve aumentar com a adição de máquinas no sistema;

2. Tolerância a faltas: O sistema deve tolerar falhas tanto do servidor de aplicações como dos *workers*. Nesse trabalho, incluem-se no termo falhas não só as falhas de *hardware* e *software* como também, a interrupção da execução do *worker* por atividade do teclado ou mouse, ou ainda, por desligamento da máquina, por exemplo;

3. Proteção: A máquina deve ser protegida de aplicações maliciosas que sejam submetidas à grade;

4. Gerenciabilidade: Com o grande número de recursos sendo doados na grade em diversos sites diferentes, torna-se necessário que atividades gerenciais como instalação, atualização e monitoramento possam ser efetivadas automaticamente;

5. Não-intrusividade: O usuário dono da máquina deve ter prioridade sobre as aplicações que queiram executar na grade. A aplicação pode ser finalizada e recomeçada em outra máquina ou pode ser pausada e recomeçada assim que o dono da máquina não a estiver utilizando.

6. Usabilidade: O usuário não deve encontrar obstáculos para executar uma aplicação na grade, de modo que não seja necessário modificar aplicações. Isso impediria a execução de aplicações legadas ou de código proprietário ao qual o usuário não tem acesso para fazer as modificações necessárias para executar na grade.

No contexto deste trabalho, dois desses aspectos foram analisados: gerenciabilidade e usabilidade.

4.1.3 Arquitetura

Cada middleware de grades de desktops define uma arquitetura formada por componentes que desempenham diferentes papéis na grade e cada um estabeleceu sua forma própria de nomear tais componentes, mas em geral eles desempenham serviços bem semelhantes em três níveis distintos definidos em [**?**] (ver Figura 4.1):

1. O cliente: o cliente possui o código executável da aplicação, que será submetida para execução na grade. Assim, o módulo cliente é usado para controlar a execução das tarefas, servindo de *front-end* entre a grade e seus usuários;

2. A gerência de aplicação e recursos: armazena informações sobre as aplicações até que recursos estejam disponíveis para executá-la e realiza o casamento entre aplicações e recursos disponíveis;

3. Os *workers*: são os recursos da grade, nos quais as tarefas do cliente executam de fato.

4.1.4 Aplicações

Vários tipos de aplicações podem tirar proveito da execução em grades de *desktops*. Entre as áreas mais comuns se encontram a biologia computacional [**?**, **?**], os modelos climáticos [**?**] e aplicações de física [**?**]. Com o advento da computação em grades de *desktop*, foi possível levar as simulações e experimentos científicos a um novo patamar, dada a possibilidade de explorar grandes faixas de parâmetros nos simuladores ou mesmo realizar simulações e experimentos mais complexos.

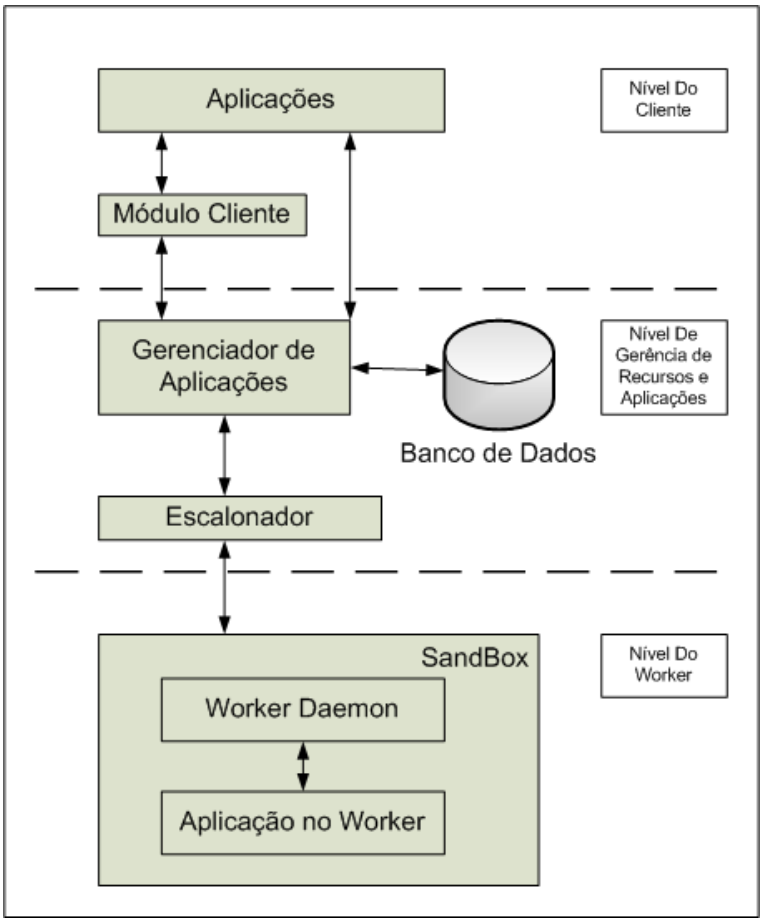

Figura 4.1: Modelo Geral de Grades Computacionais de Desktop

Qualquer aplicação distribuída pode ser executada numa grade computacional. Aplicações paralelas especialmente são as que mais podem tirar proveito desse tipo de arquitetura distribuída. Uma classe de aplicações paralelas notável é a das aplicações *Bag-of-Tasks* (BoT) que são aquelas aplicações paralelas cujas tarefas são independentes entre si, ou seja, não há comunicação entre as tarefas. Essas aplicações são bastante comuns na prática, presentes em mineração de dados, varredura de parâmetros, simulações e cálculo de fractais, por exemplo [**?**].

Capítulo 5

Metodologia

A idéia deste trabalho é realizar uma análise comparativa a respeito do uso das grades computacionais de desktop mais populares da atualidade. Optou-se por uma abordagem qualitativa para avaliar o uso de cada ferramenta. Inicialmente foi construída uma base de informações sobre as ferramentas estudadas, tais como artigos e documentação das ferramentas. Em um segundo momento, foram escolhidas algumas métricas a serem usadas como base para a comparação entre estas ferramentas. À luz de tais métricas, as ferramentas foram utilizadas e foi realizada uma análise comparativa das ferramentas que refletem o estado da prática nesta área. As métricas levaram em conta as diversas fases do uso das tecnologias como a instalação e o uso da grade em si, como também custos de manutenção e preparação de aplicações para execução na grade, por exemplo.

Dessa forma, pode se dizer que a metodologia deste trabalho se divide nas seguintes etapas:

- Formulação do problema e dos procedimentos da pesquisa:
  - Definição do problema;
  - Identificação das ferramentas a serem analisadas;
- Planejamento das métricas avaliadas:
  - Determinar o que deve ser estudado nas ferramentas;
  - Identificar aspectos comuns nas ferramentas;

- Determinação os cenários de experimentação:
  - Determinar a seleção de amostras;
  - Verificar e validar a seleção de amostras no tocante a representar a população;
- Realização os experimentos e coleta de dados;
- Desenvolvimento de um plano de análise da amostra:
  - Filtrar dados relevantes;
- Processar e analisar os dados obtidos gerando uma tabela comparativa das ferramentas usadas.

5.1 Atividades

As seguintes atividades foram realizadas:

Levantamento bibliográfico: Apesar da natureza prática da avaliação qualitativa, se faz necessário uma investigação bibliográfica a fim de se identificar as mais importantes grades computacionais de desktop usadas atualmente;

Elaboração das métricas de interesse para a avaliação: foram identificados os pontos nos quais as tecnologias serão comparadas e contrastadas;

Projeto de experimentos: Planejar os experimentos foi necessário para se cobrir os detalhes de interesse das métricas identificadas;

Preparação do ambiente experimental: A avaliação prática requisitou instalar cada uma das ferramentas identificadas;

Realização dos experimentos: Com a finalidade de coletar observar as métricas e coletar dados sobre elas, quando possível;

Escrita do relatório de estágio: Documentar a pesquisa realizada;

Preparação da apresentação de defesa do estágio: Preparar apresentação da pesquisa e conclusões obtidas.

Capítulo 6

Resultados

6.1 Definição das Métricas

Nesse trabalho procurou-se dar um cunho mais prático aos resultados e isso guiou a escolha dos aspectos a serem levados em conta ao avaliar os *middleware* de grades. As métricas observadas nesse trabalho foram divididas em três categorias: (i) instalação e configuração, (ii) gerência e (iii) uso da grade computacional.

6.1.1 Instalação e Configuração

Existem duas visões que podem ser investigadas nesse caso: (i) juntar-se a uma grade existente, mantida por outra instituição e (ii) começar uma nova comunidade. Com isso algumas perguntas foram identificadas para guiar o processo:

- Quantas entidades precisam ser instaladas?
- A quais plataformas e sistemas operacionais é dado suporte?
- Alguma entidade precisa ser instalada em máquina dedicada?
- É necessário permissão de super-usuário para instalar algum componente?
- Necessita de negociação para entrar numa grade existente?
- Quantas portas precisam ser abertas para permitir a comunicação entre as entidades?

6.1.2 Gerência

Nesse tópico foram identificadas questões relativas a administração dos recursos, em termos de ferramentas e tecnologias oferecidas pelos pacotes de middleware de grades computacionais aos gerentes e equipe de suporte. Dada a natureza distribuída dos componentes, pode ser trabalhoso obter um estado atual do funcionamento da grade sem a ajuda desse tipo de ferramenta gerencial. Dessa forma, foram identificadas as seguintes perguntas:

- Existem ferramentas que dão apoio à gerencia?
- É dado suporte a algum mecanismo de virtualização?
- Existem mecanismos de proteção que protejam os recursos das aplicações e viceversa?

6.1.3 Uso da Grade Computacional

Implantar e gerenciar uma grade computacional não faria sentido se não existisse a necessidade de usá-la. Nessa etapa da análise as perguntas identificadas visam esclarecer quão fácil é usar a grade implantada. Um *trade-off* importante é a transparência e a configurabilidade. Transparência é esconder do usuário o funcionamento ou mesmo a existência de mecanismos como replicação, realocação, concorrência e migração ou mesmo onde as aplicações estão sendo executadas e se os componentes falham em algum momento. As perguntas identificadas nessa etapa foram:

- A aplicação necessita estar escrita numa linguagem determinada?
- Que tipos de aplicações são suportados?
- É preciso escrever algo mais para executar uma aplicação?
- Existem mecanismos de detecção de ociosidade?
- Existem mecanismos de incentivo para doar recursos à grade?
- O mecanismo de *checkpoint* é suportado?

6.2 Preparação do Ambiente

Uma vez identificadas as métricas, foi necessário projetar um ambiente onde se pudesse testar funcionalidades de cada middleware de grade computacional. A tecnologia de máquinas virtuais foi usada pois permite a emulação isolada de ambientes de computação distribuído de forma barata.

Nesse estágio, o ambiente de máquinas virtuais (*Virtual Machine Environment* - VME) usado foi o Linux-VServer [**?**]. O VServer, a partir de uma modificação no kernel da máquina hospedeira, aloca dinamicamente os recursos tais como memória, espaço em disco e *tick* de CPU [**?**]. Ele foi escolhido dado o fato de já estar instalado nas máquinas usadas no ambiente do estágio e sua facilidade de uso.

6.3 Middleware de Grades Computacionais

Durante a fase de revisão bibliográfica, foram identificados cinco pacotes de *middleware* de grades computacionais: Condor, XtremWeb, BOINC, OurGrid e InteGrade.

6.3.1 Condor

Condor foi criado na década de 80 na Universidade de Winconsin inicialmente como um sistema de processamento em *batch* utilizando os computadores da universidade. O Condor é um projeto bem maduro com relação aos outros pacotes de *middleware* identificados e tem evoluído bastante desde seu surgimento. Uma das características que acompanha o projeto desde seus estágios iniciais é a flexibilidade, ou seja, a decisão final é sempre do usuário. Isso permite ao Condor se adaptar aos mais variados ambientes embora exija mais trabalho de configuração.

6.3.2 XWHEP

O XWHEP (*XtremWeb for High Energy Physics*), apesar do que o nome possa indicar, é uma plataforma para usar computação voluntária sobre a internet para aplicações de propósito geral.

O XWHEP nasceu do XtremWeb [**?**] que foi um *middleware* desenvolvido no Laboratoire de Recherche en Informatique (LRI) [**?**]. Num esforço em conjunto com o Laboratoire de L'Accélérateur Linéaire (LAL) [**?**], um laboratório naturalmente de física de altas energias, foi criado o XWHEP com a finalidade de estudar sistemas distribuídos em larga escala.

6.3.3 OurGrid

A comunidade OurGrid é formada por todos os usuários e desenvolvedores do *middleware* OurGrid. Tal middleware possibilita a criação de grades computacionais peer-to-peer cujo principal objetivo é reduzir o tempo de execução de aplicações BoT (*bag-of-tasks*), que são aplicações paralelas com tarefas independentes, ou seja, que não necessitam de comunicação entre si. São exemplos de tais aplicações a varredura de parâmetros e o processamento de imagens [**?**]. O OurGrid está em produção desde dezembro de 2004 e existe uma comunidade de laboratórios (também chamados de *sites*), liderados pelo Laboratório de Sistemas Distribuídos (LSD), formando uma grade aberta, *free-to-join* e cooperativa na qual tais laboratórios doam seus recursos ociosos em troca de acessar recursos ociosos de outros laboratórios quando precisarem [**?**].

6.3.4 BOINC

BOINC, acrônimo para *Berkeley Open Infrastructure for Network Computing*, é uma plataforma para computação em recursos públicos (*public computing*) desenvolvido pelo mesmo time responsável pelo SETI@home no *Space Sciences Laboratory* (SSL)[**?**] na Universidade da Califórnia, Berkeley. O BOINC tem como objetivos principais: (i) promover a criação de mais projetos usando computação em recursos públicos e incentivar uma grande porcentagem dos usuários domésticos a participar de um projeto desses. Existem dois componentes no BOINC: (i) Server, responsável por armazenar o projeto e as aplicações que o compõe; e (ii) Client, instalado nas máquinas voluntárias que desejam doar seus recursos. Tanto o código do Server quando o do Client necessitam de modificações para executar um novo projeto e por isso não foi incluído neste trabalho.

6.3.5 InteGrade

O projeto InteGrade é encabeçado pela Universidade de São Paulo (USP), Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Universidade Federal de Goiás (UFG), Universidade Federal do Maranhão (UFMA) e Universidade Federal de Mato Grosso do Sul (UFMS). O *middleware* InteGrade viabiliza a execução de aplicações paralelas usando ciclos ociosos das estações de trabalho *desktop* de um laboratório, por exemplo, embora recursos dedicados também possam ser acoplados à grade.

O InteGrade é uma das poucas iniciativas brasileiras no desenvolvimento de grades computacionais e ainda está em estágio inicial visto que o software não tem uma versão oficial disponível. O Integrade teve seu quinto release candidate publicado em meados de Junho e não foi possível incluí-lo na avaliação. No entanto, algumas características preliminares puderam ser extraídas. O *middleware* é orientado a aplicações paralelas com um número significativo de comunicação entre os nós que a executam e atualmente dá suporte a aplicações seqüenciais, BoT e paralelas acopladas (em MPI e BSP). A instalação é automatizada com o *IGDeployer* (*feature* do novo *release*) e requer algumas bibliotecas e comunicação por Secure Shell (SSH) entre as máquinas usando pares de chave.

6.4 Avaliação

6.4.1 Instalação e Configuração

O processo de instalação e configuração é diferente em todos os *middleware* de grade avaliados. O *trade-off* aqui é configurabilidade em detrimento da simplicidade. Como essa categoria exige um maior nível de detalhes que as outras, cada *middleware* de grade será explicado separadamente e, por fim, alguns aspectos são comparados entre eles.

Condor

Máquinas numa grade Condor podem assumir três papéis diferentes como mostrado na Figura 6.1: (i) a *Submit Machine*, é qualquer máquina na grade que interage submetendo tarefas; (ii) as *Execution Machine*, são as máquinas que têm seus recursos doados; e (iii) o *Central Manager*, que faz o casamento entre tarefas submetidas e recursos disponíveis bus-

cando melhor adaptar os requisitos das tarefas. Existe a possibilidade de usar um quarto componente, o *Checkpoint Server*, usado para armazenar os dados de *checkpoint* das tarefas submetidas mas que não faz parte da distribuição padrão do Condor e, portanto, não foi usado. O Condor constrói *pools* de máquinas em LANs para a submissão de tarefas em *batch*. Com a evolução do *middleware*, foram experimentadas várias formas de se conectar *pools* entre si e atualmente é usado o mecanismo chamado de *direct flocking* no qual uma *Submit Machine* submete as tarefas para outro *Central Manager* quando tais tarefas não podem ser executadas no *pool* local. Outro mecanismo que também pode ser usado é o *Grid Resource Access and Management* (GRAM), mas que é provido em outro pacote chamado Condor-G e que, por ser direcionado ao gerenciamento de tarefas em grades computacionais já existentes (e não criar uma nova grade) usando Globus Toolkit, ficou fora do escopo deste trabalho.

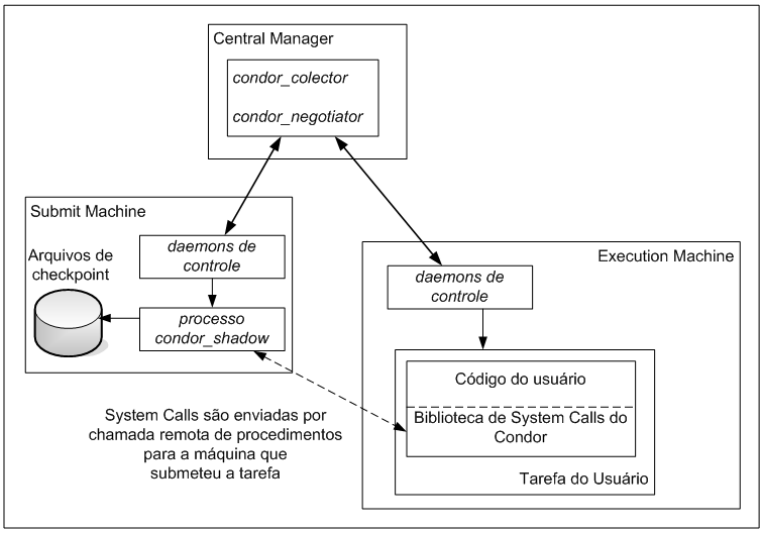

Figura 6.1: Arquitetura do Condor

Ainda na Figura 6.1, é mostrada a composição interna de cada componente. O Central Manager provê dois serviços básicos: (i) o *condor_collector*, que coleta informações dos componentes mantendo um histórico do estado da grade; e (ii) o *condor_negotiator*, responsável pelo *matching* entre recursos disponíveis e aplicações submetidas. As *Submit Machines* possuem vários daemons responsável pelo controle das tarefas em execução entre eles o *condor_shadow*, que atua como um gerente dos recursos para a aplicação executando na *Execution Machine*, isto é, ela recebe as *system calls* da tarefa do cliente, executando no recurso doado (usando o ambiente de execução *Standard*, explicado mais a frente) e redire-

cionadas pela API do Condor.

Num *pool* Condor só existe um único *Central Manager*. Ele é responsável por coletar informações dos outros componentes e ainda por realizar o casamento entre recursos e tarefas. Caso essa máquina falhe não será possível mais executar nenhuma tarefa, apesar de que as tarefas já em execução continuam a executar. Dada essa importância, recomenda-se instalar o Central Manager numa máquina com maior disponibilidade ou que possa ser reiniciada o mais rápido possível no caso de falha. Também se deve considerar recursos de disco e capacidade de tráfego de rede para essa máquina.

O *middleware* está disponível para as plataformas Intel x86 e 64, PowerPC, SPARC e HP/PA, sendo possível instalar em diferentes sistemas operacionais como MacOS, Linux Debian, Windows NT, Solaris, FreeBSD, AIX, HP/UX, RHEL, entre outros.

Existem *pools* Condor espalhados por todo o mundo. Formar comunidades ou entrar numa comunidade formada depende das intenções das entidade que mantêm cada uma delas. O administrador de um laboratório pode criar um pool com os recursos existentes e permitir a execução de tarefas de qualquer outro pool, porém precisa de autorização do administrador de um pool para usar seus recursos. Existem configurações específicas referentes à permissão de execução de aplicações de outros pools e que precisam ser manualmente escolhidas.

XWHEP

O XWHEP também possui três componentes fundamentais (Figura 6.2): (i) o *Server*, entidade centralizada encarregada de gerenciar a plataforma; (ii) o *Worker*, software distribuído nas máquinas que terão seus recursos doados; e (iii) o *Client*, software que interage com a plataforma submetendo tarefas.

Os componentes dependem fundamentalmente do *Server* e não suportam falhas dele. O banco de dados central também é um ponto único de falha.

O middleware é escrito na linguagem Java e portanto requer a JVM para compilar os pacotes. Porém, a instalação do server só pode ser feita em máquinas com Linux ou Mac OS X, enquanto Worker e Client podem ser instalados em em Linux, Windows (executando no ambiente Cygwin) e MacOS.

O XWHEP é usado pelo LAL [**?**] para execução de aplicações envolvendo física de altas energias e usuários interessados em contribuir podem voluntariamente doar os ciclos ociosos

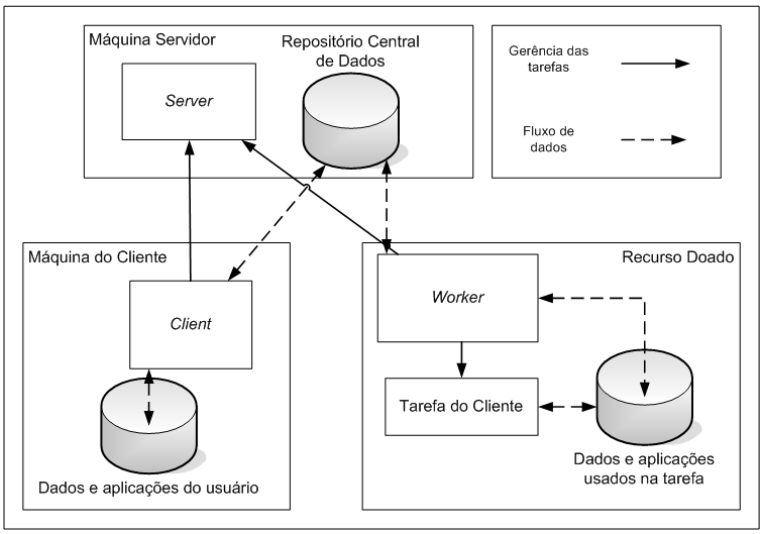

Figura 6.2: Arquitetura do XWHEP

de suas máquinas, instalando o Worker, para a execução de aplicações do LAL. No entanto não existe uma comunidade aberta onde o usuário possa também consumir recursos da grade.

OurGrid

O OurGrid possui quatro componentes fundamentais: (i) o *Peer*, componente existente em cada site e responsável por fazer o casamento entre recursos e tarefas a serem executadas; (ii) o *Discovery Service*, entidade centralizada encarregada de manter um catálogo de *Peers*; (ii) o *Worker*, que executa nas máquinas com recursos a serem doados; e (iii) o *Broker*, que submete e acompanha a execução dos *jobs* na grade. O *Discovery Service* não acompanha a distribuição padrão oferecida na página do OurGrid [**?**]. Um administrador que queira implantar uma comunidade a parte deve procurar uma versão estável no repositório do código ou entrando em contato com o time de desenvolvimento.

Na Figura 6.3, é apresentada a distribuição desses componentes em dois *sites* distintos. Cada site representa um domínio administrativo diferente gerenciado por um *peer*. Por ser um componente centralizado, o *Discovery Service* é um ponto de falha. Entre suas atribuições estão a de manter a comunidade conectada, informando a cada *Peer* a lista dos *Peers* presentes na comunidade. Uma vez que este componente falhe, não será possível a entrada de novos *Peers* na comunidade. Atualmente, com a versão estável do *Discovery Service*, é possível instalar mais de um por comunidade, atuando como um servidor de repli-

cação.

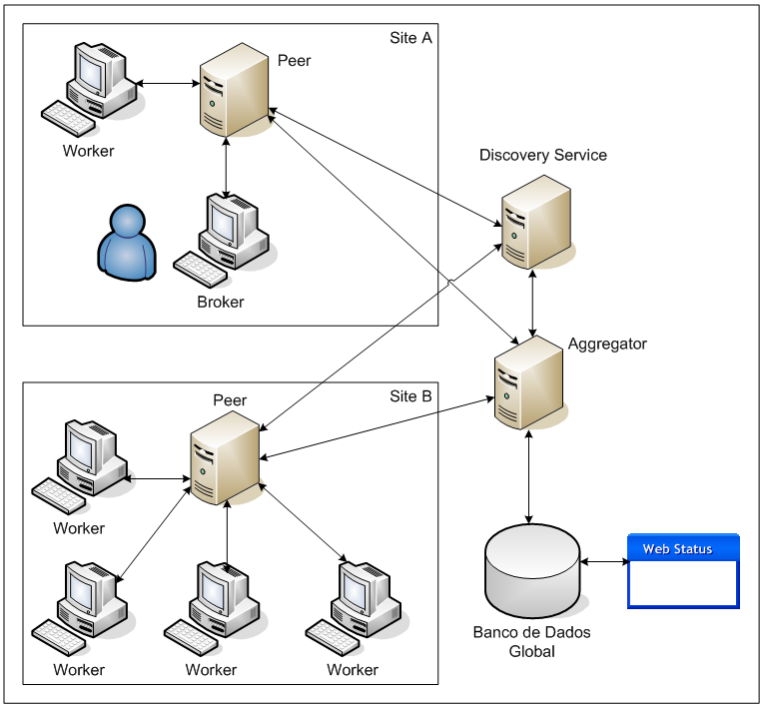

Figura 6.3: Arquitetura do OurGrid

Como o OurGrid é escrito na linguagem Java [**?**] e executa sobre a *Java Virtual Machine* (JVM) é independente de plataforma e sistema operacional dado que estes dêem suporte a tecnologia Java.

Atualmente, há uma comunidade aberta *free-to-join* do OurGrid liderada pelo LSD [**?**]. Qualquer interessado em usar o *middleware* OurGrid pode ser juntar a comunidade compartilhando seus recursos.

Configuração

O Condor segue o princípio de deixar todas as decisões a critério do usuário e isso o torna bastante difícil de configurar pois existem muitas decisões a serem tomadas antes de iniciar a instalação. As configurações do Condor são divididas em quatro categorias: (i) obrigatórias, sem elas a grade não funciona; (ii) obrigatórias com opções padrão; (iii) políticas de execução; e (iv) arquivos de log e opções mais específicas de cada tipo de aplicação suportada.

É recomendado que os componentes do *middleware* executem com privilégio de super usuário dado que de outro modo qualquer pessoa poderia fazer com que os daemons do

condor executassem programas maliciosos.

O Condor usa as portas 9618 e 9614 respectivamente para o coletor de dados e o negociador. Os outros serviços executam usando portas escolhidas randomicamente pelo sistema mas que podem ser configuradas para usar a faixa padrão de 9600 a 9700. O número de portas no *Central Manager* é determinado por informações como, por exemplo, o número de máquinas no pool, número de processadores nos recursos doados e número máximo de tarefas simultâneas nos recursos individualmente.

Um caso similar é o XWHEP. A instalação é bastante problemática dado que muitas das opções de configuração são obrigatórias e confusas. Como a arquitetura da grade requer que todos os componentes tenham acesso a um único banco de dados usado para depositar os arquivos das aplicações, é necessário que se proveja acesso ao banco a partir de todas as máquinas na grade abrindo a porta 3306 usada pelo MySQL, o que nem sempre é uma prática aceitável pelos administradores da rede. Além desta, outras portas precisam ser abertas para permitir comunicação com o Server (4321 a 4329) e com o Worker (4323).

De todos os pacotes de *middleware* analizados, o OurGrid é o que requer menos parâmetros de configuração a serem atribuídos. Como os componentes se comunicam através do protocolo Extensible Messaging and Presence Protocol (XMPP), é necessário que duas portas estejam abertas para comunicação entre todos os componentes (por padrão 5222 e 5223) e o servidor XMPP e uma porta no servidor aberta para comunicação com o mundo (por padrão, a porta 5269). A configuração dos componentes é simples e pode ser feita usando a interface gráfica ou por meio de comandos.

6.4.2 Gerência e Administração

O Condor oferece alguns comandos próprios para administradores do sistema como o *condor_status* para buscar informações momentâneas da grade, como o estado das máquinas que compõem a grade ou as máquinas disponíveis para executar tarefas; e o *condor_stats* que provê informações históricas da grade armazenada no CentralManager. Exemplos das saídas do comando *condor_status* podem ser encontradas na Figura 6.4 e 6.5

De modo similar o XWHEP oferece scripts para verificar o estado da grade. Também é possível configurar parâmetros dos Workers como, por exemplo, número máximo de tarefas simultâneas e regras de ativação do Worker, via uma interface web fornecida pelo

```
Name               JavaVendor Ver    State     Activity LoadAv Mem   ActvtyTime

slot1@ricardo0.lsd Sun Micros 1.6.0_ Owner     Idle     0.200   984  0+00:05:07
slot2@ricardo0.lsd Sun Micros 1.6.0_ Owner     Idle     0.000   984  0+00:05:08
slot1@ricardo1.lsd Sun Micros 1.6.0_ Owner     Idle     0.010   759  0+00:05:09
slot2@ricardo1.lsd Sun Micros 1.6.0_ Owner     Idle     0.000   759  0+00:05:10
slot1@ricardo2.lsd Sun Micros 1.6.0_ Unclaimed Idle     0.010   991  0+00:00:04
slot2@ricardo2.lsd Sun Micros 1.6.0_ Unclaimed Idle     0.000   991  0+00:05:06

                     Total Owner Claimed Unclaimed Matched Preempting Backfill

         INTEL/LINUX     6     4       0         2       0          0        0

               Total     6     4       0         2       0          0        0
```

Figura 6.4: Saída do comando *condor_status*

```
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@ricardo2.lsd LINUX      INTEL  Unclaimed Idle     0.010   991  0+00:00:04
slot2@ricardo2.lsd LINUX      INTEL  Unclaimed Idle     0.000   991  0+00:05:06

                     Total Owner Claimed Unclaimed Matched Preempting Backfill

         INTEL/LINUX     2     0       0         2       0          0        0

               Total     2     0       0         2       0          0        0
```

Figura 6.5: Saída do comando *condor_status -available*

XWHEP 6.6.

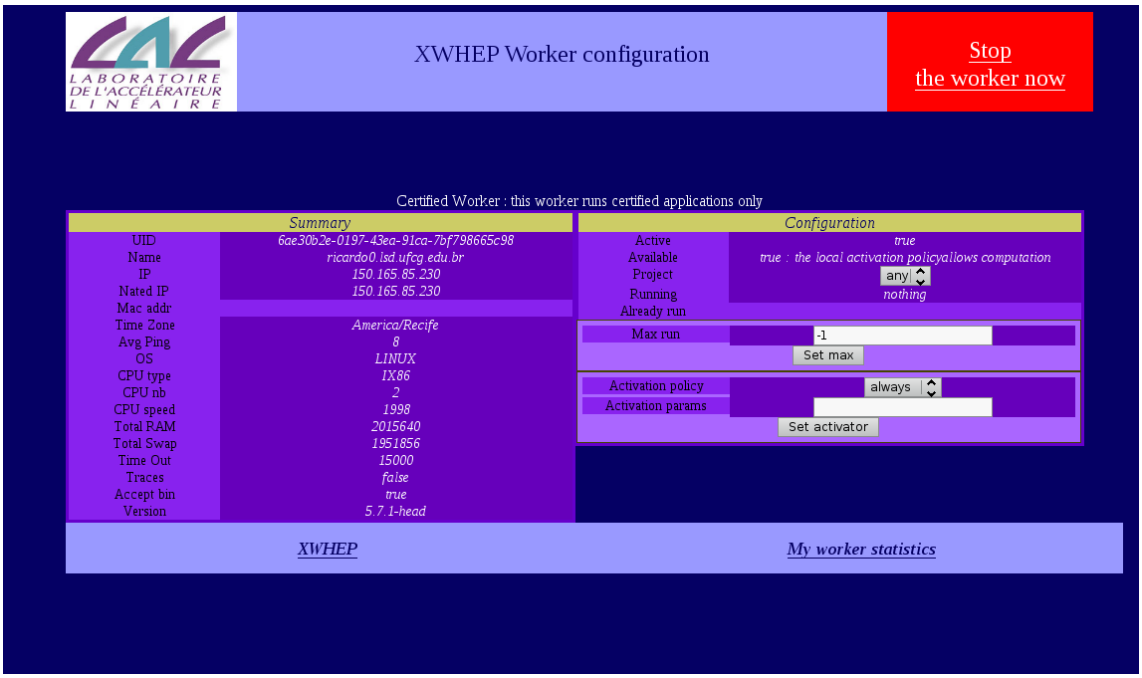

Figura 6.6: Configuração do *Worker* no XWHEP

Por sua vez, o OurGrid disponibiliza o portal WebStatus com informações recentes sobre a grade como número de Peers conectados, máquinas disponíveis e clientes executando tarefas no momento (ver Figura 6.7). Esses dados são coletados dos *Peers* e *Discovery Service* pelo *Aggregator* periodicamente.

A própria arquitetura do condor contribui para a proteção do recurso doado diante de aplicações maliciosas. O Condor permite a execução de VMs como se fossem aplicações. A aplicação é dada como finalizada quando a imagem é encerrada.

O XWHEP não suporta nenhum tipo de ambiente de execução virtualizado. Já o OurGrid Worker pode ser instalado numa máquina virtual Vserver tendo os benefícios deste tipo de ambiente de execução.

6.4.3 Aplicações

No Condor os ambientes de execução são chamados de universos. Atualmente são disponibilizados nove universos, são eles:

- *Standard*: provê *checkpointing* e *system calls* remotas, mas com restrições;

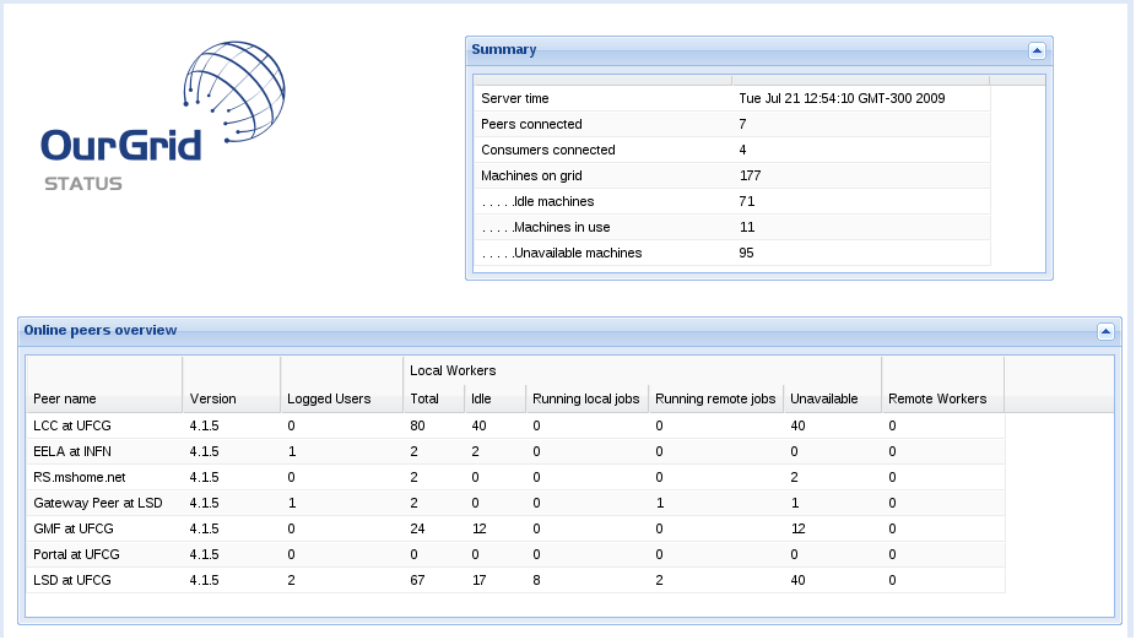

Figura 6.7: OurGrid WebStatus

- *Vanilla*: sem *checkpoint* e *system calls* remotas. Útil para programas que não podem ser "relinkados" e para scripts Shell. Os arquivos de entrada e saída podem estar num sistema de arquivos compartilhados ou pode ser usada o mecanismo de transferência de arquivos do Condor;
- *Grid*: permite ao usuário submeter tarefas para sistemas com interface similar ao Globus Toolkit usando o Condor;
- Java: Tarefas BoT escritas para a JVM;
- *Scheduler*: permite submeter tarefas leves para o daemon *condor_schedd* (futuramente substituído pelo universo Local);
- *Local*: Executa logo que submetido na máquina local sem esperar por *matching* com outra máquina (a tarefa iniciada não pode ser mais preemptada);
- *Parallel*: para programas que necessitam de várias máquinas por tarefa como, por exemplo, as escritas usando o padrão MPI;
- VM: usado quando a tarefa não é só uma aplicação, mas uma imagem de disco (facilita a execução de máquinas virtuais) VMware ou Xen.

```
universe = java
executable = sim.jar
jar_files = sim.jar
arguments = sim.core.PreProcess raw_data.dat
transfer_input_files = simulator/input/raw_data.dat
output = sim_output
error = sim_err
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
queue
```

Figura 6.8: Condor Submit Description File

Uma vez escolhido um universo adequando a aplicação que se deseja executar, é necessário escrever um *Submit Description File* (SDF) no qual será indicado o universo escolhido, os arquivos da aplicação, o modo de transferência entre outras informações. Um exemplo de SDF é mostrado em 6.8. Uma aplicação Java é posta na fila de execuções. O Java Archive (JAR) da aplicação é submetido e a classe `sim.core.Preprocess` é executada tendo como arquivo de entrada "raw_data.dat" localizado na máquina que submete a tarefa, a partir da pasta corrente, no caminho "simulator/input/", e cujos arquivos de saída sim_output e sim_error serão transferidos ao final da execução.

O Condor ainda permite que tarefas sejam executadas respeitando um Grafo Acíclico Direcionado (Directed Acyclic Graph ou DAG). Isso é útil quando uma tarefa depende dos resultados de outras.

O XWHEP executa aplicações Java e arquivos executáveis nos sistemas operacionais suportados (Linux, Mac OS X e Windows). Tanto o arquivo da aplicação como os dados de entrada são tratados como dados puros e para executar um comando é necessário deixá-los acessíveis via um Identificador de Recursos Uniforme (*Unified Resource Identifier* ou URI).

O OurGrid suporta, de maneira similar, aplicações Java e arquivos executáveis nos sistemas operacionais suportados. Porém os arquivos da aplicação ficam na máquina onde está o Broker. Uma vez que um novo Worker seja escolhido para executar a tarefa, os arquivos são transferidos e a tarefa inicia só após completar esta transferência. As aplicações são submetidas por meio de um arquivo de metadados chamado Job Descriptor File (JDF). O exemplo

```
job :
label   : Job1

task :
init    : put sim.jar sim.jar
store sim.dat sim.dat
remote  : java -cp sim.jar. StartSim 1000 864000 sim.dat output-$JOB.$TASK
final   : get output-$JOB.$TASK output-$JOB.$TASK
```

Figura 6.9: OurGrid Job Description File

de JDF na Figura 6.9 é referente à execução da classe "StartSim" contida no jar "sim.jar" e tendo como entrada alguns argumentos numéricos e o arquivo de entrada sim.dat.

6.4.4 Usando a grade

Detecção de Ociosidade

Nesse tipo de grade computacional aplicações do usuário do recurso devem ser priorizadas em relação às executadas na grade. para isso são necessários mecanismos que detectem e informem à grade sobre a disponibilidade do recurso. No Condor a ociosidade é regulada através de uma expressão no arquivo de configuração do recurso. Essa expressão, chamada *ClassAd* (*classified advertisement*) também é enviada ao *Central Manager* para ser usada na etapa de *matching* entre as aplicações submetidas a grade e os recursos disponíveis. Um exemplo de ClassAd é dado na Figura 6.10 em que um recuso estará disponível na grade assim que a média de carga for menor que 30% e o teclado esteja inativo a mais de quinze minutos.

A ativação do Worker no XWHEP também é determinada por um parâmetro no arquivo de propriedades. As opções disponíveis são:

***AlwaysActive*:** É a opção padrão, na qual o worker está apto a executar tarefas;

***DateActivator*:** A disponibilidade é monitorada por data e tempo com sintaxe similar a do *crontab*;

```
[
MyType="Machine"
TargetType="Job"
Machine="machine1.lsd.ufcg.edu.br"
Requirements=(LoadAvg <= 0.3) \&\& (KeyBoardIdle > (15 * 60))
]
```

Figura 6.10: ClassAd do Condor

***CpuActivator*:** Disponibilidade regulada pela carga da CPU e disponível somente para *Workers* em Linux;

***MouseKdbActivator*:** O Worker está disponíevl 30 segundos após o dono ter usado o mouse ou teclado pela úmtima vez. Não funciona para Mac OS X;

***WinSaverActivator*:** Assim que a proteção de tela for ativada o Worker entra em modo disponível.

OurGrid Worker é feita automaticamente, e de maneira similar ao *MouseKdbActivator* do XWHEP, pelo monitoramento da atividade do mouse e teclado.

Mecanismos de Incentivo

Para incentivar a doação de recursos, o OurGrid implementa a Rede de Favores (*Network of Favors* ou NoF). A NoF intentiva interações com Peers colaboradores e desincentiva *free-riders*, que são peers que só consomem recursos. Esse incentivo é realizado com base em interações passadas mantidas no histórico de cada *Peer*. No Condor é possível priorizar aplicações usando ass expressões ClassAds na configuração da máquina. Por exemplo, adicionando a linha `Rank = Owner == ``ricardo''` a máquina configurada com o ClassAd na Figura 6.10 prioriza aplicações submetidas por pelo cliente do usuário ricardo. Por fim, não existem mecanismos similares no XWHEP.

Checkpoint

Dada a volatilidade dos recursos, característica inerente às grades computacionais de *desktops*, as tarefas podem ser interrompidas e o trabalho computado perdido. O mecanismo de

chackpoint permite arquivar o estado atual de execução de uma tarefa preemptada para uma posterior execução no mesmo recurso ou em um recurso diferente através da migração dessas informações. O condor, diferentemente do XWHEP e do OurGrid, permite fazer checkpoint no universo *Standard*. No entando algumas restrições são impostas entre elas:

- É necessário "relinkar" a aplicação usando o *condor_compile*;
- Não é permitido a criação de múltiplos processos;
- Só é permitida uma *thread* no nível do *kernel*;
- Não é permitido mapear arquivos na memória (chamadas chamadas *mmap()* e *munmap()*);
- Só permite manipulação de arquivos menores que 2 GB.

6.5 Resultados Obtidos

Como resultado da avaliação foi construída a Tabela 6.1.

Métrica Observada	**Condor**	**XWHEP**	**OurGrid**
Componentes Principais	3	3	3
Sistemas Operacionais suportados	Linux Debian, Windows NT, MacOS	Linux, Windows (Cygwin) e MacOS	Qualquer sistema operacional com suporte a Java
Configuração	Muitas propriedades precisam ser especificadas para pôr a grade em funcionamento	Poucas propriedades são necessárias, mas requer um banco de dados central configurado	Poucas propriedades são obrigatórias
Executa em modo "superusuário"	Sim	Sim	Não
Portas abertas no Firewall	2 no *Central Manager* + faixa dinâmica (por padrão 100)	9	3

Continua na próxima página

Tabela 6.1 – continuação da página anterior

Métrica Observada	**Condor**	**XWHEP**	**OurGrid**
Necessita de negociação para entrar numa grade existente?	Administrador do pool deve configurar	Não existe uma grade aberta	Não
Existem ferramentas de gerencia?	Sim	Sim	Sim
Suporte ao uso de tecnologia de virtualização?	Sim	Não	Sim
É oferecido algum tipo de proteção da máquina doada?	Execução em *sand-box*	Não	Pode ser executado com Vserver
Tipos de aplicações suportados	BoT, MPI, Paralelas acopladas	BoT, MPI	BoT
É necesário modificar as aplicações?	Sim[1]/Não	Não	Não
Mecanismos de detecção de ociosidade	Sim	Sim	Sim
Mecanismos de incentivo	Não	Não	Sim
Mecanismo de checkpoint	Sim	Não	Não
Comunidade ativa	Usuário pode doar recursos	Usuário pode doar recursos	Usuário pode doar e usar recursos

Tabela 6.1: Resultados da avaliação

Os resultados apresentados nessa tabela são úteis para a tomada de decisão sobre qual pacote de *middleware* escolher para instalar num ambiente com computadores *desk-top* disponíveis e que haja demanda por executar aplicações que não sejam possíveis, por questões de tempo ou demanda, de ser executadas usando a infraestrutura local.

Diferentes situações podem ser consideradas ao tomar esse tipo de escolha. No caso em que a urgência em pôr a grade computacional em funcionamento é um fator determinante, pacotes de middleware cuja instalação e configuração exigem menos conhecimento acerca da grade são preferencialmente escolhidos. Esse é o caso do OurGrid.

[1]No caso do uso de checkpoint

Por outro lado, quando uma flexibilidade maior é requerida em termos de configurações e tipos de aplicações a serem executadas, caso em que o perfil dos usuários varie bastante, um pacote de *middleware* mais flexível como o Condor é mais recomendado.

Esses resultados apresentados necessitam de um maior rigor metodológico, no entanto, servem como guia para administradores e usuários que desejam melhor utilizar seus recursos

Capítulo 7

Consideração Finais

O estágio trouxe novos conhecimentos a formação do estagiário como bacharelando em Ciência da Computação assim como deu oportunidade de pôr em prática alguns tópicos abordados em disciplinas teóricas. Tópicos muito importantes foram os relacionados à Gerência e Configuração de Redes de Computadores, os quais são oferecidos em disciplinas optativas. Tópicos teóricos das disciplinas Sistemas Operacionais, do currículo obrigatório, e Sistemas Distribuídos, do currículo optativo, foram de fundamental importância desde a etapa de levantamento bibliográfico como também configuração do ambiente. Grande parte da teoria vista nas disciplinas contribuiu de certo modo para a montagem do trabalho final.

Foi sentida uma dificuldade em se organizar e por em prática esse tipo de pesquisa com objetivos voltados a obtenção de resultados qualitativos. A disciplina de Metodologia Científica poderia ser adaptada para explorar esse tipo de pesquisa que apesar de pouco comum na área de ciências exatas pode fundamentar um trabalho mais qualitativo com os seus resultados.

Quanto ao trabalho realizado, alguns pontos negativos podem ser citados como: a opção por usar aplicações com alta demanda de CPU e pouca de dados teve como objetivo simplificar a análise realizada. Avaliar aplicações *data intensive* requer uma metodologia mais quantitativa para avaliar a eficácia dos mecanismos empregados. No entanto, a pesquisa feita permitiu construir a classificação dos middlewares mesmo sem usar tais tipos de aplicações dado que as características avaliadas puderam ser extraídas mesmo sem o uso de uma metodologia mais apurada quantitativamente.

O trabalho realizado pode ser estendido para incluir aplicações paralelas fortemente

acopladas (em que as tarefas se comunicam, a exemplo de MPI) visto que algumas das grades dão suporte à execução desse tipo de aplicação. Um trabalho similar pode ser realizado considerando grades de serviço tais como Globus ou até comparando o estado da prática entre as grades de serviço com as grades de Desktop oportunistas.

Observou-se que o OurGrid é o único que possui uma grade existente que é *free-to-join* e *free-to-execute*, isto é, aberta para a doação de recursos e uso de recursos para aplicações do usuário. No entanto, o OurGrid dá suporte apenas à execução de aplicações BoT, o que nem sempre satisfaz todos os usuários.

O Condor mostra-se o *middleware* mais estável e que detém a maior comunidade (quando se considera a união de vários pools). Por ter como objetivo a flexibilidade, aceita diferentes tipos de aplicações com diferentes requisitos, porém às custas de uma configuração cara.

Por fim, o XWHEP possui uma organização diferente em que os dados são disponibilizados por URIs e existe controle de acesso. Dessa forma as aplicações maliciosas não podem apagar dados do servidor. No entanto, a existência de um banco de dados central do qual todos os componentes dependem é uma preocupação para administradores da grade computacional.

Apêndice A

Plano de Estágio