Análise de padrões de uso em grades computacionais

Danilo Matheus Rubio Conde

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Marcelo Finger

São Paulo, janeiro de 2008

Análise de padrões de uso em grades computacionais

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Danilo Matheus Rubio Conde
e aprovada pela Comissão Julgadora.

Banca Examinadora:

- Prof. Dr. Marcelo Finger (orientador) - IME-USP.
- Prof. Dr. Fabio Kon - IME-USP.
- Prof. Dr. Arlindo Flavio da Conceição - UNIFESP.

Agradecimentos

Agradeço principalmente à minha família pelo apoio incondicional e irrestrito para a realização deste trabalho. Em especial aos meus pais, cujos esforços para proporcionar aos seus filhos a melhor educação possível não podem ser medidos. Esta etapa de minha formação é mais um fruto das privações e sacrifícios que realizaram em prol da educação dos filhos.

Ao professor Marcelo Finger, por ter confiado em mim desde o início deste trabalho e pela paciência e apoio fundamentais para sua conclusão.

À Bruna, minha namorada, cuja compreensão e apoio, mesmo que nem sempre conseguisse demonstrá-los, além do afeto e carinho, foram muito importantes e inspiradores.

Aos amigos, por me ajudarem a manter o equilíbrio necessário entre trabalho, estudos e vida social, e pela compreensão nos momentos em que estive ausente. Em particular, aos que fazem parte dessa jornada desde a graduação no IME-USP, como o Emilio Francesquini, companheiro fiel desde os primeiros exercícios-programa, e ao Alexandre E. P. A. Gabriel pela ajuda na revisão deste texto.

À Objective Solutions, empresa onde trabalho desde a graduação, por ter proporcionado plenas condições para que eu pudesse trabalhar sem comprometer meus estudos e também por permitir o uso de seus equipamentos e facilidades para fins acadêmicos.

Resumo

As grades computacionais oportunistas utilizam recursos computacionais ociosos para executar aplicações que necessitem de alto poder computacional ou processem volumes muito grandes de dados. Os recursos computacionais compartilhados com tais grades, como memória e CPU, não são dedicados. Dessa forma, não é interesse tanto do proprietário dos recursos quanto do usuário que solicita a execução de uma aplicação na grade que sejam utilizados recursos quando eles não estiverem ociosos. Um mecanismo para previsão de ociosidade de recursos compartilhados de modo que a grade possa enviar aplicações para serem executadas em nós com previsão de ociosidade para as próximas horas tem um importante papel nesse contexto, tanto para aumentar o desempenho das aplicações executadas na grade, quanto para não comprometer o trabalho dos proprietários de recursos compartilhados.

Este trabalho envolveu a implementação de um mecanismo de análise de padrões de uso de recursos computacionais, baseado na análise de agrupamentos, que permite a predição de utilização futura de recursos. A eficácia dessa implementação foi analisada através de simulações de execuções de aplicações distribuídas em grades, onde demonstramos que podemos melhorar seus desempenhos se levarmos em conta a previsão de utilização dos recursos no escalonamento.

Palavras-chave: grades computacionais, aprendizado computacional, análise de agrupamentos.

Abstract

Opportunistic computational grids use idle resources to run applications that require high computational power or process large amounts of data. In this way, shared computer resources, such as CPU or memory, are non dedicated. Therefore, it's not desirable for the resource's owner nor the user who submits an application, to use resources that are not idle at the moment of submission. A mechanism that makes possible the prediction of future idleness of shared resources, enabling the grid to choose idle machines to send applications to, would be of much value.

This work involved the implementation of a mechanism that analyzes patterns of resource usage, based on cluster analysis, allowing the prediction of future usage. The eficiency and accuracy of this implementation was put into practice by simulating the execution of distributed applications in the grid, where we showed that if the scheduler takes into account the prediction of future resource usage, then the performance of the applications can be improved.

Keywords: grid computing, machine learning, clustering.

Sumário

1 Introdução **1**
1.1 Estrutura do texto 2

2 Análise de Padrões de Uso no InteGrade **5**
2.1 Introdução à arquitetura do InteGrade 5
2.2 Ciclo de vida de uma aplicação executada no InteGrade . . . 7
2.3 Análise de padrões no InteGrade: LUPA 9

3 Análise de Agrupamentos **13**
3.1 Análise de agrupamentos 15
3.1.1 Algoritmos de agrupamento 18
3.1.2 Técnicas hierárquicas 18
3.1.3 Método das k-Médias 20
3.2 Eficácia da Análise de Agrupamentos Aplicada ao Reconhecimento de Padrões de Uso 20

4 Implementação **23**
4.1 Arquitetura 23
4.2 Coleta 25
4.2.1 Informações sobre utilização de recursos no Linux . . . 28
4.3 Cálculo dos padrões 28
4.3.1 Processo de análise de agrupamentos 29
4.4 Mecanismo de predição 33
4.5 Recálculo dos padrões 38
4.6 Escalonamento 38

5 Experimentos **41**
5.1 Escalonamento 41
5.1.1 Descrição dos dados 41
5.1.2 Ensaios 42

5.1.3 Análise dos resultados 47
5.2 Sobrecarga do LUPA 51
5.2.1 Tempo de processamento 51
5.2.2 Consumo de memória 52
5.2.3 Conclusões 53

6 Conclusão 55
6.1 Limitações do LUPA e Trabalhos Futuros 56

A Resultado dos Ensaios de testes 59

Capítulo 1

Introdução

Em nossa sociedade é notável que a necessidade de maior poder computacional é crescente. A indústria e os institutos de pesquisa possuem problemas cada vez mais complexos e com maior quantidade de dados a serem processados.

Tradicionalmente, a resposta a essa necessidade é a construção de computadores cada vez mais poderosos. Esses equipamentos, entretanto, são muito caros e em poucos anos tornam-se obsoletos. Foi nesse contexto que surgiram as **grades computacionais** [7, 12].

As grades computacionais possibilitam que recursos de vários computadores de baixo custo sejam disponibilizados para execução de tarefas que requerem muita capacidade computacional.

Uma grade é dita **oportunista** se utiliza recursos ociosos de computadores não dedicados, como estações de trabalho de um laboratório de pesquisa que durante a noite ficam ligados porém sem nenhum usuário conectado, ou máquinas de professores enquanto os mesmos estão dando aulas.

Tanto para o usuário que submete uma aplicação para ser executada nas máquinas da grade, quanto para o proprietário de um recurso compartilhado, não é interessante que os recursos sejam utilizados pela grade enquanto a máquina não está ociosa. Do ponto de vista do primeiro, a tarefa provavelmente será executada em mais tempo do que se tivesse sido enviada para uma máquina ociosa e, do ponto de vista do outro, seus recursos estão sendo consumidos enquanto ele está precisando deles, comprometendo seu trabalho que, assume-se, é mais importante do que os trabalhos da grade.

Escolher quais máquinas serão utilizadas para a execução de uma determinada aplicação é tarefa do escalonador da grade. Vários pesquisadores [17, 10, 23] têm estudado métodos para obter melhores tempos de execução

de aplicações distribuídas em grades computacionais. Além do escalonamento para escolher quais recursos compartilhados serão utilizados, alguns projetos de grades computacionais, como o BOINC [6], ainda realizam escalonamento local [24] das aplicações da grade que estão executando em uma determinada máquina. O Condor [28, 26], quando detecta que o desempenho de uma aplicação em uma determinada máquina está muito abaixo do esperado, tem a capacidade de migrá-la para outra máquina.

Yang, Schopf e Foster [29] sugerem uma estratégia de escalonamento que leva em consideração a previsão da utilização e da variância de cada recurso na divisão das tarefas entre as máquinas que executarão uma determinada aplicação. O objetivo é obter tempos de execução menores e menos variáveis. O método sugerido é comparado a várias outras abordagens de escalonamento, inclusive algoritmos que consideram a previsão fornecida pelo *Network Weather Service* [27, 3], um sistema de monitoramento e previsão de recursos disponíveis em uma rede.

O InteGrade [14, 2, 11] é um projeto que desenvolveu um *middleware* para implantação de grades computacionais. A especificação do InteGrade inclui um mecanismo de análise de padrões de uso que auxilia a grade a saber quais máquinas estão ociosas. Cada computador que compartilha seus recursos possui um módulo de coleta e análise de informações sobre a utilização chamado LUPA (*Local Usage Pattern Analyzer*). A partir do histórico de utilização dos recursos, são estabelecidos padrões de uso, que permitem a previsão de utilização futura. Neste trabalho, descrevemos a implementação deste módulo e os experimentos realizados a fim de demonstrar que o escalomanento baseado na previsão da utilização futura de recuros pode aumentar o desempenho de aplicações distribuídas no InteGrade.

1.1 Estrutura do texto

Este trabalho está dividido em duas partes. A primeira parte — Capítulos 2 e 3 —, é mais teórica e serve para dar o contexto necessário para o entendimento dos capítulos seguintes. Na segunda parte — Capítulos 4, 5 e 6 — apresentamos a implementação realizada e os resultados obtidos.

O Capítulo 2 descreve a arquitetura do InteGrade e como o LUPA interage com resto do sistema. No Capítulo 3, fazemos uma breve introdução ao aprendizado computacional e à análise de agrupamentos, que é a técnica que será utilizada para o cálculo de padrões de uso. A arquitetura da implentação do módulo LUPA está descrita no Capítulo 4 e os experimentos realizados com essa implementação, bem como seus resultados, estão no

Capítulo 5. No Capítulo 6 apresentamos as conclusões e trabalhos futuros.

Capítulo 2

Análise de Padrões de Uso no InteGrade

O InteGrade é um sistema que permite a implantação de grades computacionais oportunistas [15, 26]. Uma grade é dita oportunista se utiliza recursos ociosos de máquinas não dedicadas. O projeto é desenvolvido por grupos de pesquisa de diversas universidades brasileiras, entre elas USP, UFG, UFMS, PUC-Rio e UFMA, e possui diversas linhas de pesquisa, tais como tolerância a falhas, execução de aplicações paralelas, segurança e agentes móveis, entre outras.

2.1 Introdução à arquitetura do InteGrade

O sistema está dividido em vários módulos e o projeto de cada um deles foi orientado a objetos. Além das vantagens mais conhecidas da orientação a objetos [13], essa decisão de projeto permitiu que a comunicação entre os módulos fosse facilmente feita através de chamadas remotas de métodos. A arquitetura para acesso aos objetos remotos usada foi o CORBA. Trata-se de um padrão eficiente, bem estabelecido, amplamente utilizado na indústria de software e independente da linguagem de programação, o que permitiu que módulos do InteGrade fossem escritos em linguagens diferentes (C++, Lua e Java).

A Figura 2.1 ilustra os tipos de nós diferentes que participam do InteGrade e a distribuição dos módulos entre eles. Os nós da grade que compartilham recursos são chamados **nós compartilhados** (*Resource Provider Node*). Além deles, em cada aglomerado (*cluster*) participante da grade, há um **nó gerenciador do aglomerado** (*Cluster Manager*), responsável

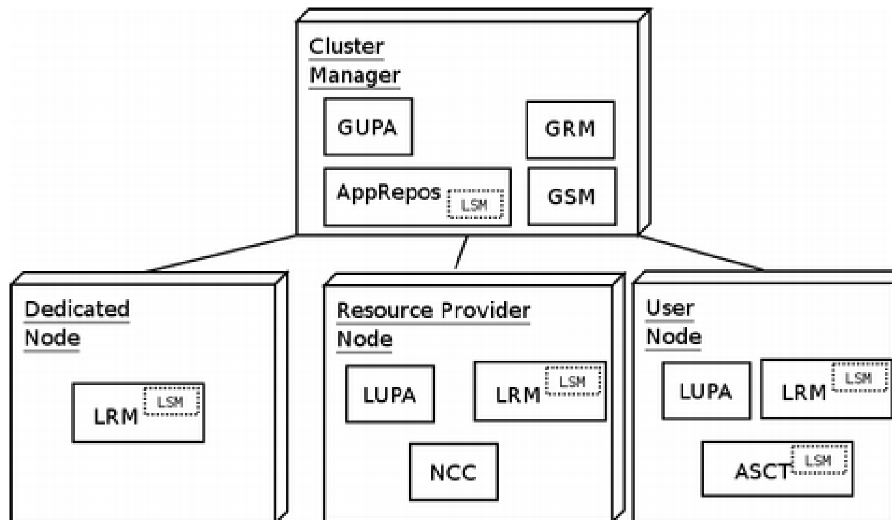

Figura 2.1: Arquitetura do InteGrade

por conhecer todos os recursos disponíveis em seu aglomerado e distribuir tarefas a eles.

Tipicamente, os seguintes módulos são executados em um nó gerenciador de aglomerado:

- *GRM (Global Resource Manager)*

  Atua como gerenciador dos recursos de toda a grade. É responsável por manter um cadastro dos recursos compartilhados por cada LRM (descrito mais adiante) e também por escalonar requisições de execução de aplicações na grade.

- *AR (Application Repository)*

  O repositório de aplicações armazena as aplicações que podem ser executadas na grade. Elas precisam ser submetidas pelos usuários, através do ASCT (descrito mais adiante), para que os LRMs possam executá-las.

- *GUPA (Global Usage Pattern Analyzer)*

  O objetivo deste módulo é auxiliar o GRM no escalonamento das aplicações através da análise dos padrões de uso das máquinas da grade como um todo, coletando informações dos diversos LUPAs presentes na grade.

Nos nós compartilhados, os seguintes módulos estão presentes:

- *LRM (Local Resource Manager)*

  É responsável por gerenciar e publicar os recursos compartilhados por um nó, permitindo a execução de aplicações da grade.

- *NCC (Node Control Center)*

  Permite que o proprietário de um nó compartilhado configure a quantidade de recursos que serão disponibilizados como, por exemplo, a quantidade máxima de memória ou o percentual máximo de utilização do processador que a grade poderá consumir.

- *LUPA (Local Usage Pattern Analyzer)*

  Este módulo analisa as informações sobre a utilização dos recursos compartilhados por uma máquina, identificando os vários padrões de uso que ela assume. Esses padrões serão utilizados para prever a disponibilidade de recursos, baseado no padrão de uso que a máquina se encontra, e esta informação será utilizada no escalonamento das aplicações na grade.

Ainda há mais dois outros tipos de nós: o **nó dedicado** (*Dedicated Node*), que é um nó onde é executado somente o LRM, e o **nó de usuário** (*User Node*), onde o ASCT é iniciado. O ASCT (*Application Submission and Control Tool*) é o aplicativo através do qual os usuários enviam aplicações para o AR e solicitam a execução de aplicações na grade.

2.2 Ciclo de vida de uma aplicação executada no InteGrade

A principal motivação para o surgimento das grades computacionais foi a necessidade crescente de poder computacional. As grades fornecem uma resposta a essa questão oferecendo vários processadores ao usuário que deseja executar aplicações computacionalmente caras. Nesta seção veremos os passos da execução de uma aplicação no InteGrade.

Para executar uma aplicação, precisam existir um nó gerenciador e pelo menos um nó compartilhado ou dedicado. A aplicação a ser executada precisa ser enviada para o AR, que armazena os arquivos executáveis (binários) que serão executados pelos LRMs e permite o cadastro de executáveis para plataformas de sistema diferentes (Linux 32 ou 64 bits, Windows, Mac OS X, por exemplo.) O aplicativo ASCT fornece uma interface gráfica (Figura

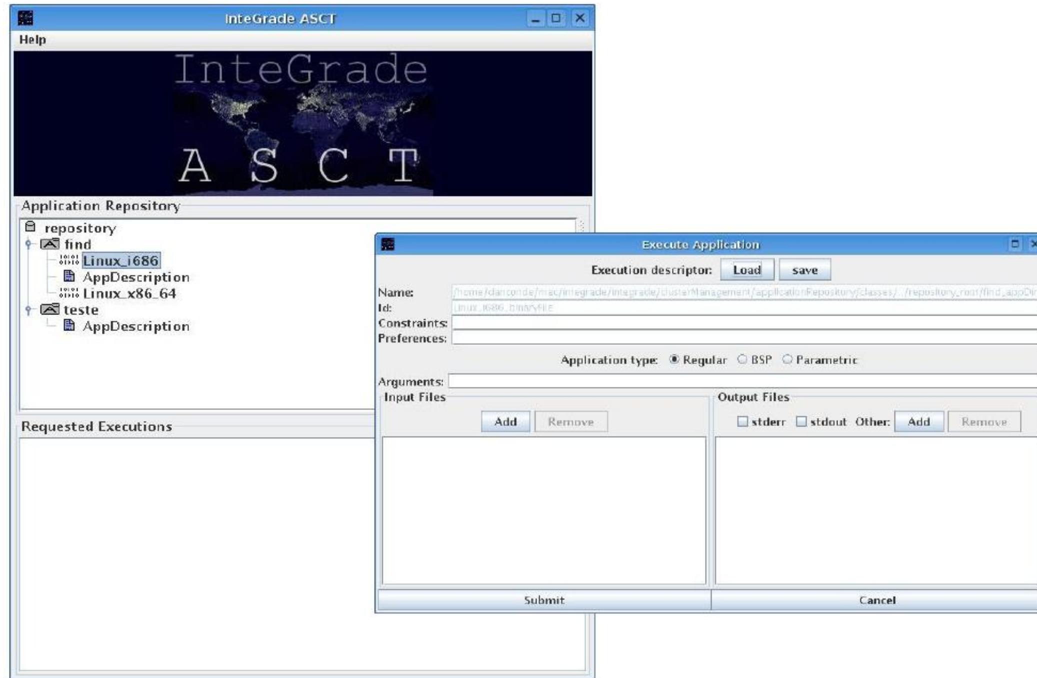

Figura 2.2: Solicitação de execução de aplicações no ASCT

2.2) para envio e consulta das aplicações existentes no repositório e também para requisições de execução de aplicações na grade.

A Figura 2.3 mostra os passos do protocolo de execução de aplicações. A solicitação de execução (1) parte do ASCT e é enviada ao GRM. Essa requisição pode conter algumas informações adicionais sobre a quantidade mínima de recursos exigidos ou a plataforma em que o usuário deseja que a aplicação seja executada.

O GRM sempre mantém uma visão razoavelmente atualizada dos LRMs e suas disponibilidades de recursos. Quando recebe uma solicitação de execução, seleciona (2) nesta lista os nós candidatos a executarem-na, ou seja, os nós que atendem os requisitos mínimos exigidos pela aplicação. Ao encontrar um nó candidato, o GRM confirma a oferta (3) de recursos junto ao LRM do nó. Nessa confirmação, caso o nó não possua os recursos exigidos, o GRM é informado e volta ao passo (2). Caso contrário, a aplicação será executada neste nó. Seu LRM solicita ao AR (4) o arquivo executável adequado e os arquivos de entrada (5) para o ASCT solicitante. A execução é iniciada (6) e o ASCT é informado que a execução foi aceita (7).

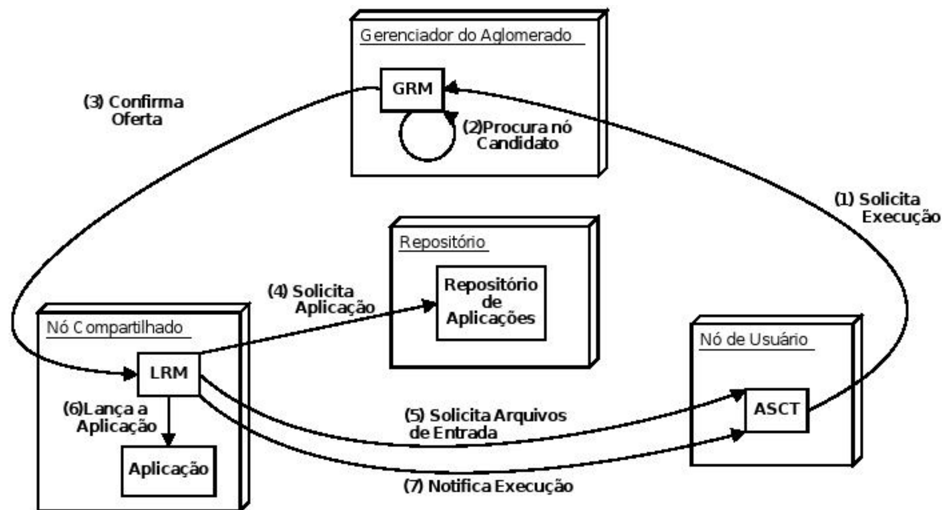

Figura 2.3: Execução de uma aplicação

2.3 Análise de padrões no InteGrade: LUPA

O InteGrade é uma grade oportunista, ou seja, que utiliza recursos ociosos não-dedicados. Um bom exemplo de conjunto de máquinas que podem ser ligados à grade é um laboratório de pesquisa. Em geral, essas estações de trabalho são bastante utilizadas durante o dia, porém, à noite e nos finais de semana, continuam ligadas mas sem nenhum usuário. Nesse contexto, observamos dois padrões de uso:

1. Dias úteis: uso intenso durante o dia, ociosidade durante a noite e madrugada;

2. Finais de semana e feriados: ociosidade o tempo todo.

Para essas máquinas, seria interessante aceitar qualquer solicitação de execução durante os finais de semana ou de noite. Durante os dias de semana, pode depender da quantidade de recursos necessários para a aplicação.

Um outro exemplo são os computadores dos professores (Figura 2.4). Há vários compromissos que se repetem toda semana, como aulas, seminários ou reuniões de grupos de pesquisa, durante os quais sua estação de trabalho fica ociosa, originando alguns padrões como: uso intenso apenas durante o período da manhã, apenas no período da tarde, apenas após o almoço e no final da tarde etc.

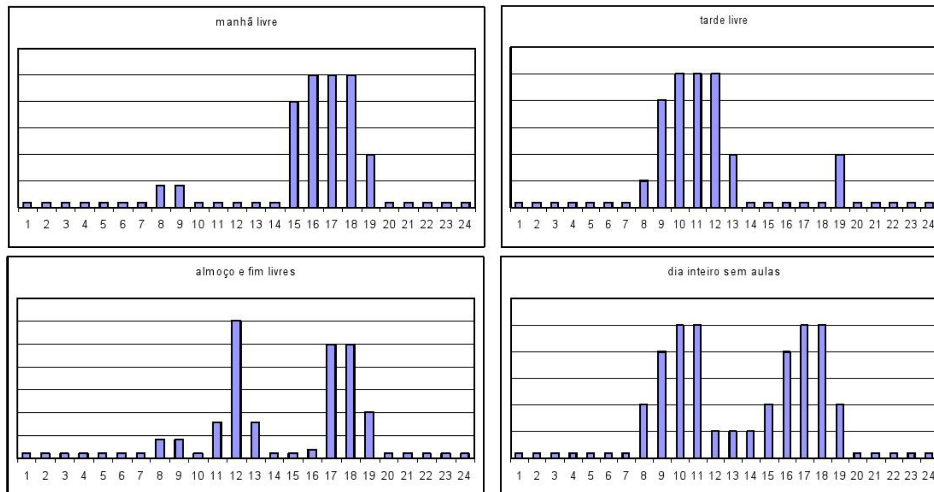

Figura 2.4: Exemplos de padrões de uso

Conforme visto anteriormente, o LUPA — *Local Usage Pattern Analyser* — é o modulo do InteGrade, executado em nós compartilhados, que é responsável por analisar o uso dos recursos das máquinas, de modo a estabelecer padrões de uso. Uma vez determinadas essas categorias de uso, podemos identificar em qual delas o nó se encontra e prever qual será o consumo de recursos nas próximas horas.

Quando um nó compartilhado é iniciado, um coletor de informações sobre o uso dos recursos é iniciado e, periodicamente (a cada 5 minutos, por exemplo), adiciona registros de utilização dos recursos em arquivos, que, posteriormente, serão analisados pelo LUPA. Os dados mais relevantes para análise dos padrões de uso são:

- quantidade de memória utilizada;

- percentual de CPU utilizada.

Essas informações são armazenadas localmente em cada estação de trabalho e a grade não fornece nenhuma forma de acesso a esses dados, pois eles podem comprometer a privacidade do usuário.

A identificação do padrão em que o nó se encontra e a predição do uso futuro são feitas no momento da confirmação da oferta de recursos feita pelo GRM, quando é solicitada a execução de uma aplicação (passo (3) da Figura 2.3).

O padrão atual é encontrado comparando-se o uso dos recursos nas últimas horas com os padrões já estabelecidos. Baseado nele, a disponibilidade de recursos a partir do ponto onde o nó se encontra no padrão é calculada e, caso satisfaça as exigências especificadas na solicitação de execução da aplicação, o LUPA confirma a oferta de recursos para o GRM. Esta confirmação significa que esta máquina está aceitando a aplicação da grade em questão. Porém, caso a previsão seja de que não haverá recursos suficientes, o GRM é informado e procura por outros nós candidatos a executarem a aplicação (passo (2) da Figura 2.3).

O processo de identificação dos padrões de uso é feito através da análise de agrupamentos, num processo de aprendizado não-supervisionado. A próxima seção fará uma breve introdução ao aprendizado computacional e alguns algoritmos de agrupamento. Mais adiante, no Capítulo 4, será apresentada a implementação do LUPA.

Capítulo 3

Análise de Agrupamentos

O estudo do aprendizado computacional, ou aprendizado de máquina [20, 21], é de fundamental importância no contexto da inteligência artificial [22, 18]. Qualquer sistema ou agente que se diga inteligente deve ser capaz de aprender, ou seja, modificar seu comportamento ao longo do tempo, de modo a exercer melhor suas funções.

Podemos dizer que um sistema aprende quando altera sua estrutura, seus algoritmos ou seus dados, de maneira a melhorar seu desempenho. Algumas dessas modificações, como a inserção de registros em um banco de dados, acabam caindo em território de outras disciplinas e não são necessariamente consideradas aprendizado. Mas, por exemplo, quando um mecanismo de classificação de *spams* (mensagens eletrônicas indesejadas) melhora seu desempenho conforme aumenta a quantidade de mensagens que seu usuário classificou como indesejáveis, podemos dizer com segurança que este sistema aprendeu.

O aprendizado foi definido por Herbert Simon como: *"Qualquer mudança num sistema que melhore o seu desempenho na segunda vez que ele repetir a mesma tarefa, ou uma tarefa da mesma população"* (Simon, 1983).

Conforme sugerido na definição acima, o desempenho deve melhorar não somente para uma segunda execução da mesma tarefa, mas para tarefas semelhantes às quais ele já executou uma vez. Para tanto, a representação das informações aprendidas é bastante relevante e influencia significativamente no algoritmo de aprendizagem. As representações mais utilizadas são sentenças da lógica proposicional e descrições probabilísticas, como redes bayesianas [22].

Geralmente, adicionam-se componentes de aprendizado de máquina em sistemas que resolvem problemas que dependem do ambiente onde ele é

executado. Algumas razões para isso acontecer são:

- Há casos onde as tarefas que o sistema deve realizar são definidas por exemplos. É desejável que o sistema se ajuste internamente a esses exemplos de modo a se comportar apropriadamente no ambiente onde se encontra. Um exemplo dessa situação é o classificador de *spams*. O mesmo sistema tem que ser capaz de se adaptar a vários usuários e vários idiomas.

- Mineração de dados (*data mining*): trata-se da busca de relações e propriedades comuns a vários objetos em grandes volumes de dados.

- É muito comum que, durante o projeto de um agente, não se tenha conhecimento completo do ambiente onde ele atuará, ou mesmo que se possua informações imprecisas. Além disso, pode ser que o ambiente se modifique com o tempo. Por isso, é interessante que o agente consiga adaptar-se ao ambiente conforme o explora.

Os problemas de aprendizagem costumam ser divididos em três tipos [22], de acordo com a forma de realimentação dos dados: supervisionado, não-supervisionado e por reforço.

No **aprendizado supervisionado**, o objetivo é aprender uma função a partir de exemplos de entrada e saída. Neste caso, temos a presença de um tutor, que fornece esses exemplos. Podemos considerar que existe uma função f e que o objetivo do sistema aprendiz é determinar uma função h que se aproxime de f a partir dos exemplos dados, que são, na verdade, exemplos de valores de saída da função f para determinadas entradas. Para os valores de f que não possuem exemplos, h terá que "adivinhar". Uma das maneiras de fazer isso é através de analogias. Por exemplo, se o sistema já sabe que uma moto e um carro são movidos por motores a combustão e que o combustível do carro é a gasolina, ele pode, por analogia, supor que gasolina também é o combustível de uma moto.

No **aprendizado não-supervisionado** não há a presença de um tutor e tampouco de exemplos. Em geral, a natureza do problema consiste em aprender propriedades ou padrões sobre um grupo de objetos de forma a conseguir agrupá-los de acordo com suas características. Um sistema de reconhecimento de faces, por exemplo, pode analisar as faces de sua base de dados e separá-las de acordo com algumas características marcantes, como cor da pele, presença de óculos ou tamanho do cabelo.

O agente do **aprendizado por reforço** deve aprender sobre o ambiente e seus objetivos a partir do reforço. Ele não é ajudado ou guiado por um

instrutor, mas recebe recompensas ou punições como conseqüência de suas ações. Um exemplo clássico é um robô que está em um labirinto que ele não conhece. O robô escolhe aleatoriamente uma de suas possíveis ações e, após executá-la, é recompensado. Seu objetivo é maximizar suas recompensas. Encontrar a saída do labirinto, nesse caso, lhe dará uma boa recompensa e, a cada vez que se mover sem encontrar a saída, será punido.

O problema da análise dos padrões de uso dos recursos computacionais é um problema de aprendizado não-supervisionado. A dissertação de mestrado [8] de Germano C. Bezerra teve como principal objetivo mostrar que podem-se utilizar técnicas de aprendizado não-supervisionado para identificar os padrões de uso e que, a partir deles, a utilização de recursos pode ser prevista. Os resultados obtidos por este trabalho serão vistos na Seção 3.2. A próxima seção é sobre análise de agrupamentos, a técnica que foi utilizada para a identificação dos padrões de uso.

3.1 Análise de agrupamentos

A análise de agrupamentos [16, 9, 5] – *clustering, cluster analysis* ou análise de conglomerados – é uma técnica utilizada para a separação de um conjunto em subconjuntos de objetos semelhantes. A categorização é baseada em algumas observações sobre os elementos que permitem determinar a semelhança entre eles, porém pouco se sabe sobre a estrutura das categorias, apenas que seus membros devem ser parecidos entre si e diferentes dos membros das outras categorias.

A separação de objetos por semelhança é uma tarefa natural e que faz parte de nosso cotidiano. Está presente em tarefas simples, como separar os feijões de aparência ruim antes de colocá-los na panela, e em tarefas mais complexas que necessitam de métodos mais formais e processamento por computadores, como a classificação de vários indivíduos em função de sua constituição corpórea.

Quando se refere à análise de agrupamento, não se está falando de uma única técnica bem especificada, mas de uma variedade de métodos com o mesmo objetivo que variam bastante nos detalhes de suas implementações. No entanto, há uma série de elementos que são comuns a todas suas variações, formando uma espécie de arcabouço para definição de processos de análise de agrupamentos. Anderberg [5] sugere que os seguintes elementos fazem parte desse arcabouço:

1. *Escolha dos dados a serem analisados*

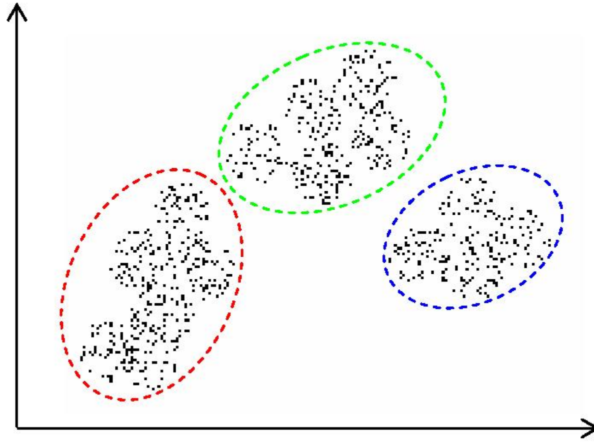

Figura 3.1: Exemplo de agrupamentos. Cada região delimitada pelas linhas tracejadas representa um agrupamento diferente.

Os objetos a serem analisados podem ser das mais variadas naturezas, como pessoas, células, planetas, cidades etc. e pode ser que não seja necessário utilizarmos o conjunto inteiro para a análise. Pode haver uma seleção de acordo com algum critério, como eliminar as informações mais antigas, ou uma escolha aleatória.

2. *Seleção das variáveis*

   Tradicionalmente, cada objeto da análise é representado por uma coleção de variáveis. Considerando que os objetos são pessoas, teremos variáveis como: nome, idade, sexo, altura, peso, nacionalidade, escolaridade etc. A escolha de quais variáveis serão usadas na análise é de fundamental importância pois a semelhança entre os objetos será calculada em função delas. Dependendo da informação que se deseja extrair dos dados, algumas variáveis serão mais relevantes que outras.

3. *O que agrupar*

   Apesar da maioria das análises representarem os objetos como uma lista de variáveis, existem outras abordagens que misturam objetos e variáveis.

4. *Homogeneização das variáveis*

   A diferença entre as unidades de medida e escalas das variáveis pode influenciar significativamente no cálculo da semelhança entre os obje-

tos, caso não sejam padronizados. Por exemplo, se estivermos classificando pessoas de acordo com sua constituição corpórea e levarmos em consideração apenas o peso e a altura, medidos em quilos e metros, respectivamente, se não for feita nenhuma padronização das variáveis, a variável peso terá mais influência no processo, pois é comum pessoas terem diferenças de 20 quilos de peso, enquanto que a altura costuma variar cerca de 0,20 m. Existem técnicas para reduzir as variáveis para uma forma padrão que são recomendadas em muitos casos, mas não obrigatórias.

5. *Medidas de semelhança*

   Um ponto crucial na análise de aglomerados é medir a semelhança entre dois objetos que, em geral, é considerada como uma distância. A escolha da medida certamente influencia no resultado final da análise. O método mais comum é a distância euclidiana, definida por $d(a, b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$, porém existem várias outras maneiras.

6. *Escolha dos algoritmos e implementação*

   Conforme já foi mencionado, a análise de agrupamentos não é uma técnica bem especificada e definida, há muitas variantes no processo. Uma delas é a escolha do algoritmo que irá processar os dados. Há várias opções que acabam produzindo resultados diferentes, e uns mais rápidos, outros mais lentos.

7. *Número de agrupamentos*

   A decisão do número de agrupamentos é uma tarefa difícil e de muita importância para que os resultados sejam satisfatórios, pois os algoritmos levam em conta esse número na distribuição dos elementos entre os grupos. A escolha, no entanto, pode ser pela intuição de alguém que conheça o domínio dos dados ou usando algum algoritmo para determinação desse valor.

8. *Interpretação dos resultados*

   A execução de um algoritmo de análise de agrupamentos tem como entrada um conjunto de objetos e como saída um conjunto de agrupamentos dos objetos da entrada. É necessário que a saída seja avaliada para decidir se o resultado é satisfatório ou se é necessário fazer ajustes no algoritmo, medida de semelhança, número de agrupamentos etc. Tendo um resultado final, cabe ao analista rotular e interpretar quais informações que os agrupamentos estão fornecendo.

3.1.1 Algoritmos de agrupamento

Os algoritmos de análise de agrupamentos podem ser classificados [9] de acordo com seu método de separação dos objetos e as duas principais categorias são:

1. Técnicas hierárquicas: os objetos vão sendo agrupados hierarquicamente, formando uma árvore de classificação.

2. Técnicas de partição: o resultado da execução é um conjunto de partições do conjunto de objetos.

Veremos adiante o funcionamento das técnicas hierárquicas e o algoritmo das k-Médias, que é de partição.

3.1.2 Técnicas hierárquicas

Nesta seção, serão consideradas as técnicas hierárquicas aglomerativas, onde todos os objetos começam como agrupamentos e a cada passo da execução um agrupamento é misturado a outro. Dessa forma, se o conjunto de objetos tiver n elementos, começaremos com n agrupamentos e, após $n-1$ passos, sobrará apenas um. Essas uniões de agrupamentos pode ser representadas graficamente através de um dendrograma (Figura 3.2), que indica quais agrupamentos foram unidos e o grau de semelhança entre eles. Essa representação facilita a decisão de quantos agrupamentos devem ser considerados como resposta final ou, em outras palavras, até que passo o algoritmo deve executar.

Seja $d(i,j)$ uma função que calcula a distância entre os objetos i e j. A partir dela, iremos construir uma matriz de similaridade entre os elementos do conjunto. Como $d(i,j)$ é simétrica, isto é, $d(i,j) = d(j,i)$, a matriz será triangular inferior. Obtida a matriz, os passos do algoritmo são:

Preparação: Construa n agrupamentos, um com cada elemento do conjunto de entrada, e os rotule de 1 a n.

Algoritmo:

Repita os seguintes passos até haver apenas um agrupamento:

1. Encontre o par de agrupamentos p e q, $p > q$, com menor distância entre si, ou seja, os dois elementos mais semelhantes. Eles serão a entrada na matriz com o menor valor.

2. Reduza o número de agrupamentos em um, transformando o agrupamento q na fusão de p e q. Atualize matriz de similaridade, removendo

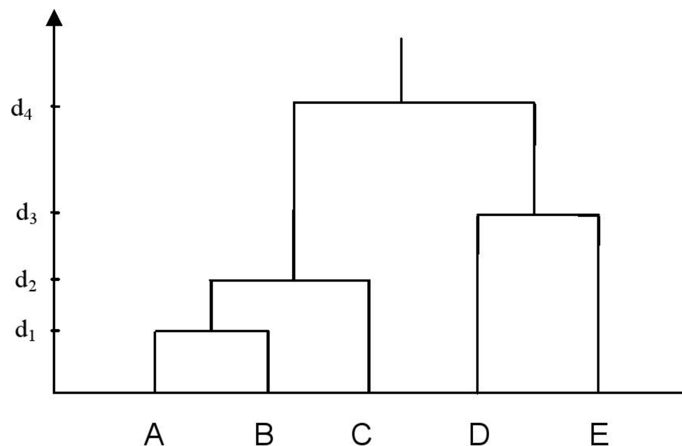

Figura 3.2: Dendrograma

a linha e a coluna referentes a p e calculando as novas distâncias para q.

3. Armazene que nesse passo os agrupamentos p e q foram fundidos e qual era a distância entre eles antes da operação.

Os algoritmos hierárquicos aglomerativos têm essa estrutura geral e costumam variar apenas na forma como calculam a distância entre agrupamentos (passo 1) e no resultado das fusões (passo 2). Há várias formas diferentes de fazer esses cálculos e eles influenciam bastante no resultado final. Alguns métodos existentes são os seguintes:

- Método da centróide: As distâncias entre agrupamentos são calculadas em função das distâncias dos centros de cada um deles.

- Método das médias das distâncias: Ao fundir dois agrupamentos, a distância do novo para os demais será a média das distâncias dos agrupamentos fundidos para os demais antes da fusão.

- Método da ligação simples (*Single Linkage*): A distância entre dois agrupamentos é a distância entre os elementos de um grupo e de outro que são mais próximos entre si. Ou seja, $d(A, B) = \min\{d(i, j) : i \in A, j \in B\}$.

- Método da ligação completa (*Complete Linkage*): Esse método é o contrário da ligação simples, a distância é dada pela distância dos

elementos mais distintos de um agrupamento e de outro, ou seja, $d(A, B) = \max\{d(i, j) : i \in A, j \in B\}$.

3.1.3 Método das k-Médias

O método das k-Médias (k-*Means*) é um dos algoritmos que usa a técnica de partição. Nesse contexto, é fornecido um conjunto de n objetos e deseja-se encontrar as k melhores partições do conjunto, satisfazendo a premissa de que os objetos de uma mesma partição devem ser semelhantes entre si e significativamente diferentes dos das demais. É um pressuposto desses algoritmos que o número k de partições esperadas é conhecido.

Os passos do algoritmo das k-Médias são os seguintes:

1. Escolha k objetos e crie uma partição com cada um deles.

2. Insira cada objeto que ainda não está em nenhuma partição à partição da qual ele está menos distante.

3. Calcule a distância de cada objeto a cada uma das partições. Se a partição mais próxima não for a que ele está, mova-o para ela.

4. Repita o passo acima até convergir, ou seja, não existir mais nenhum objeto que precise trocar de partição.

Essa é uma descrição de alto nível do algoritmo. As diversas versões dele possuem diferentes implementações para os três primeiros passos. A escolha dos k elementos que farão parte das primeiras partições pode ser feita aleatoriamente, usando os k primeiros objetos ou usando técnicas mais sofisticadas que podem ser encontradas na literatura. Para o passo 2 já é necessário uma noção da distância entre elemento e partição. Um dos critérios que pode ser utilizado é a distância entre o elemento e o centro da partição.

3.2 Eficácia da Análise de Agrupamentos Aplicada ao Reconhecimento de Padrões de Uso

Conforme mencionado anteriormente, a eficácia das técnicas propostas, assim como a previsibilidade da ociosidade de recursos computacionais, foi demonstrada na dissertação de mestrado [8] de Germano C. Bezerra.

O trabalho mostra que os algoritmos de análise de agrupamentos podem ser utilizados para identificar os padrões de uso dos recursos utilizando

os dados de monitoramento. Dessa forma, os objetos representando a utilização histórica serão separados em grupos de objetos com as mesmas características. Cada grupo desses possui os elementos de um modo de operação, ou comportamento protípico, sendo representado por um objeto representativo, que é calculado como a média entre todos os elementos do grupo.

Esse objeto representativo é importante para o algoritmo de predição de uso, pois representa o comportamento esperado de um determinado padrão de uso. Para prever a utilização futura, o algoritmo identifica o modo de operação atual, encontrando o objeto representativo com o intervalo referente às últimas horas mais semelhante aos dados de utilização recentemente coletados. Assim, a predição de uso para os próximos instantes é a parte do objeto representativo que representa as próximas horas.

Os objetos apresentados aos algoritmos de *clustering* são vetores de números, correspondentes a um determinado intervalo de tempo, onde cada valor é a medida de utilização de um recurso (CPU, memória etc.) em um instante desse intervalo. Os algoritmos são executados para cada tipo de recurso computacional, a fim de identificar os padrões de cada um deles. Apesar de parecer natural que cada objeto contenha informações de um intervalo de um dia, foram adotados intervalos de 48 horas para possibilitar a implementação de um algoritmo mais eficiente de predição (a Seção 4.3 detalha mais essa questão.)

Para avaliar a eficácia dos métodos propostos na predição do uso de recursos, algumas máquinas foram monitoradas para construir uma massa de dados que seria submetida ao processo de análise de agrupamentos. A partir desses dados foram realizados vários ensaios que consistiam, basicamente, em executar o processo de *clustering* e reconhecimento de padrões com diferentes parâmetros. Foram criadas algumas métricas para comparar o desempenho da solução proposta nas diversas combinações de parâmetros possíveis.

A massa de dados foi dividida em dois conjuntos distintos: corpo de treino e corpo de testes. O corpo de treino serve como entrada do algoritmo de *clustering* para o estabelecimento dos padrões de uso e o corpo de testes é usado para avaliar a qualidade do reconhecimento dos padrões. Os elementos do corpo de testes são enviados ao algoritmo de reconhecimento de padrões sem as informações a partir de um determinado instante, para que seja encontrado o padrão correspondente. A principal medida de qualidade dos ensaios é o índice de acertos. Considera-se que o algoritmo acertou a previsão quando a disponibilidade de recursos prevista condiz com a utilização do elemento do corpo de testes naquele mesmo intervalo de tempo, o qual não foi informado ao algoritmo de reconhecimento.

Os resultados dos ensaios foram bastante satisfatórios e serviram para concluir que a ociosidade de recursos computacionais é previsível e a análise de agrupamentos pode ser utilizada no reconhecimento dos padrões de uso. Além dessas conclusões, a implementação e os ensaios trouxeram informações relevantes em alguns tópicos relacionados ao reconhecimento de padrões de uso através da análise de agrupamentos:

- Utilização *versus* variação de utilização: o monitoramento dos recursos pode armazenar tanto a medida de utilização de um recurso como a variação da utilização em função do tempo. Os testes mostraram que a variação de utilização é mais eficiente na identificação dos modos de operação;

- Dados inexistentes: é muito comum haver intervalos de tempo onde não foi possível coletar os dados de utilização dos recursos. Há várias abordagens para preencher essas lacunas e a adotada, pela própria natureza do problema, foi repetir, por todo o intervalo onde não houve coleta, o valor da última medição realizada;

- Padronização das variáveis: nesse trabalho não julgou-se necessário a padronização das variáveis, utilizou-se a própria medida de utilização do recurso;

- Algoritmos de análise de agrupamentos: os diferentes algoritmos apresentaram bons índices de acerto e a diferença de desempenho entre eles é muito pequena;

- Dos comportamentos: observou-se que os índices de acerto são tão melhores quanto mais uniforme for a utilização da máquina.

Esses resultados são o ponto de partida para este trabalho, onde foi implementado o módulo LUPA do InteGrade, utilizando as técnicas de análise de agrupamento propostas na dissertação de Bezerra [8] para o estabelecimento de padrões de uso dos recursos computacionais compartilhados com a grade, bem como a previsão de utilização futura. No Capítulo 4 detalhamos a nossa implementação. Na Seção 4.3, em particular, boa parte das decisões de implementação foi baseada na dissertação de Bezerra [8] .

Capítulo 4

Implementação

A versão atual do InteGrade — 0.4 — possui uma implementação simples do módulo LUPA. Apesar de utilizar métodos pouco sofisticados, essa implementação é funcional e serviu para introduzir o LUPA na arquitetura do InteGrade. A previsão de utilização futura em um determinado instante é equivalente à média de utilização nesse mesmo horário nos três dias anteriores. Neste Capítulo, descrevemos a nova implementação do módulo LUPA, que utiliza análise de agrupamentos para o estabelecimento dos padrões de uso dos recursos computacionais compartilhados com a grade. Esta nova versão do módulo estará disponível em versões do InteGrade posteriores à 0.4.

4.1 Arquitetura

Assim como os outros módulos do InteGrade, a implementação do LUPA utiliza a técnica de programação orientada a objetos, permitindo que os principais conceitos sejam encapsulados em classes. Essas classes (Figura 4.1) são acessadas diretamente[1] pelo LRM, que utiliza as previsões de utilização futura fornecidas pelo LUPA e ainda as repassa a outros módulos do InteGrade que têm interesse nessas informações, como o GRM, que pode considerar a previsão de utilização dos recursos no escalonamento de aplicações executadas na grade, ou o GUPA, que pode usar alguma heurística para escolher qual nó compartilhado é mais apropriado para uma determinada aplicação. Na Figura 2.3, o LUPA encaixa-se na fase (3) do processo, a

[1] Com *diretamente* queremos dizer que essas classes são referenciadas explicitamente no código dos outros módulos do InteGrade que utilizam as informações do LUPA, não através de arcabouços de acesso remoto a objetos, como CORBA, por exemplo.

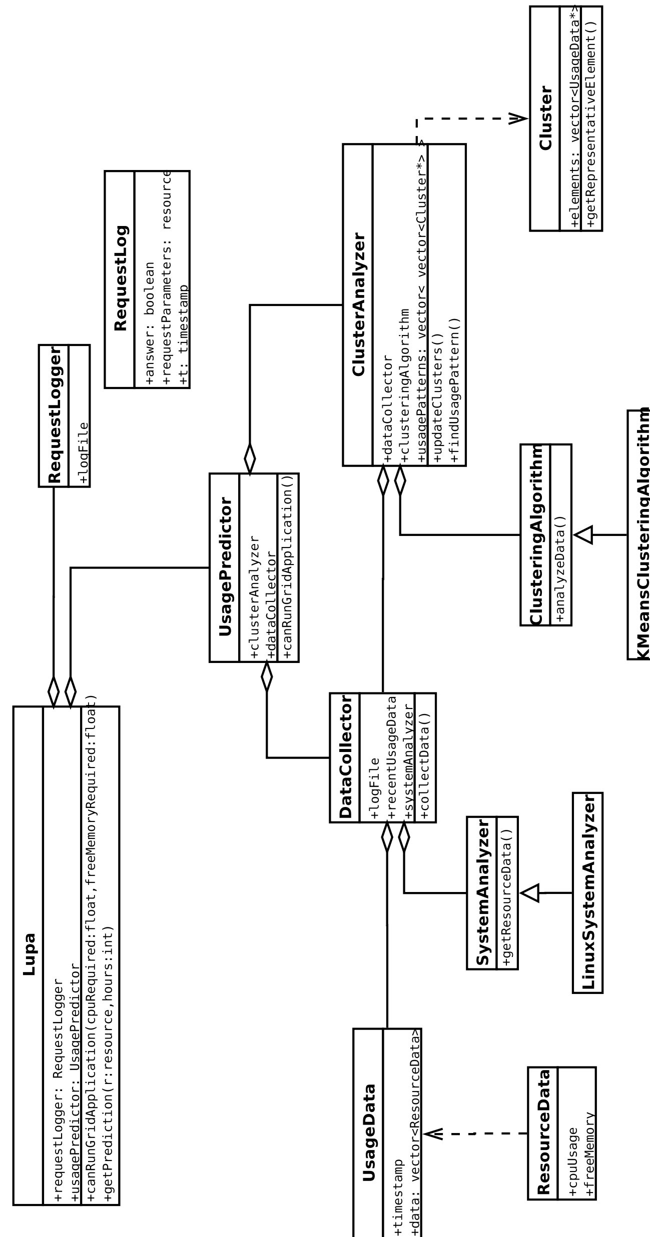

Figura 4.1: Diagrama simplificado da implementação

confirmação da disponibilidade de recursos. A linguagem de programação utilizada foi C++.

As classes do LUPA estão isoladas do resto do InteGrade, pois não possuem dependências de classes de outros módulos. Esse isolamento o deixa menos suscetível a qualquer impacto que possa ser ocasionado pela evolução do código fonte do InteGrade, que tem novas funcionalidades adicionadas constantemente.

O correto funcionamento do LUPA é verificado por um conjunto de testes automatizados que foram implementados com o auxílio da biblioteca CppUnit[1]. Esses testes garantem a corretude tanto de métodos mais básicos, como leitura e escrita no disco dos dados de utilização dos recursos, quanto de métodos mais sofisticados, como o cálculo dos padrões de uso e previsão de utilização.

Internamente, o LUPA pode ser dividido em três módulos: coleta, cálculo dos padrões de uso e predição (Figura 4.2). O módulo de coleta obtém periodicamente informações de utilização dos recursos computacionais da máquina onde está executando. Essas informações são processadas pelo módulo de cálculo dos padrões de uso para determinar quais são os modos de utilização deste nó da grade. A predição do uso é realizada quando deseja-se saber se haverá recursos disponíveis suficientes para a execução de uma aplicação da grade. Consiste em identificar, dentre os modos de uso já calculados, qual é o modo atual para conseguir prever a utilização futura. Nas seções seguintes (4.2, 4.3 e 4.4), cada um desses módulos será mais detalhado.

4.2 Coleta

O módulo de coleta é responsável por coletar e armazenar informações sobre a utilização de recursos computacionais. A coleta é realizada a cada cinco minutos e o histórico é gravado em um arquivo. O LUPA interage com o sistema operacional para obter esses dados e, dessa forma, sua implementação é dependente do sistema operacional utilizado no nó compartilhado. Na Seção 4.2.1 serão mostrados detalhes de como essas informações são obtidas no sistema operacional Linux.

Apesar do LUPA implementar atualmente a coleta de dados somente para o sistema operacional Linux, a inclusão de suporte a outros sistemas pode ser feita sem a necessidade de muitas alterações no código fonte existente, pois consiste, basicamente, em implementar uma subclasse de `SystemAnalyzer`. A função dessa classe (na verdade, de suas subclasses, pois

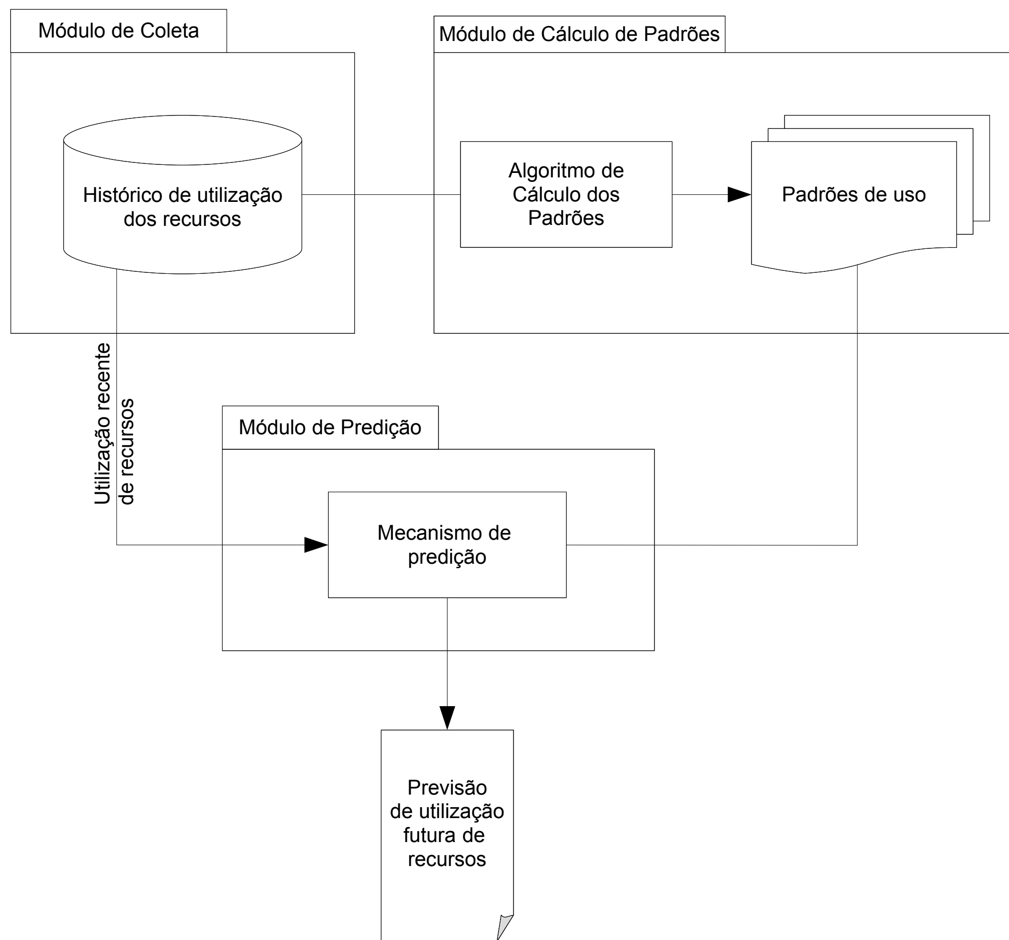

Figura 4.2: Arquitetura interna do LUPA

ela é abstrata) é fornecer os dados de utilização dos recursos para a classe `DataCollector`, que é responsável pela obtenção periódica desses dados e atualização do arquivo com o histórico de utilização, doravante denominado *arquivo de log*.

O formato do arquivo de log é bem simples. É um arquivo texto com algumas colunas separadas por tabulações. Cada linha representa as informações de utilização em um determinado instante. A primeira coluna é um número inteiro que corresponde ao instante em que os dados foram coletados e as demais colunas à utilização dos recursos. O LUPA obtém informações sobre a utilização dos seguintes recursos, que aparecem no arquivo de log nessa ordem:

- percentual de CPU utilizada (média de utilização desde a última coleta dos dados);
- Memória RAM disponível em kilobytes.

Os dados de utilização fornecidos pelo `SystemAnalyzer` são encapsulados em objetos da classe `ResourceData`. Essa classe basicamente possui um atributo para cada um dos recursos analisados pelo LUPA, que são definidos no tipo `resource`. Para incluir um novo recurso a ser analisado, temos que adicionar mais um valor ao tipo `resource` e mais um atributo na classe `ResourceData`.

Além da informação sobre os recursos, um `ResourceData` também possui um atributo booleano que indica se ele é uma *estimativa*. Dizemos que um `ResourceData` é uma estimativa se a informação contida nele sobre os recursos não foi preenchida com os dados obtidos do `SystemAnalyzer`, mas com uma estimativa de uso. Como o processo de identificação dos padrões é feito com base em uma sequência de informações sobre a utilização de recursos de vários dias, as estimativas servem para preencher lacunas em que não foi possível coletar dados (um período em que o computador ficou desligado, por exemplo). O critério utilizado é considerar que a estimativa da utilização num instante em que não houve coleta é igual à utilização da última coleta realizada. Mais adiante veremos que o algoritmo de cálculo dos padrões ignora dias em que o percentual de `ResourceData` estimados é maior que um determinado limiar.

O `DataParser` é um interpretador que lê o arquivo de log e cria uma estrutura de objetos correspondente ao conteúdo do arquivo. A saída do `DataParser` é utilizada como entrada do algoritmo de cálculo dos padrões de uso.

4.2.1 Informações sobre utilização de recursos no Linux

Conforme mencionado, a obtenção dos dados de utilização dos recursos depende de interação com o sistema operacional. Atualmente, a única implementação é para Linux. A classe `LinuxSystemAnalyzer` implementa as rotinas de acesso às informações no `/proc`.

O `/proc` é um pseudo-sistema de arquivos que serve de interface para as estruturas internas do *kernel* do Linux. Através dele, pode-se obter informações sobre processos em execução, sistemas de arquivos, configurações de rede, *hardware* instalado, estatísticas sobre o desempenho do sistema etc.

A utilização da CPU é consultada no `/proc/stat`, que contém estatísticas do sistema. O arquivo mostra a utilização de cada uma das CPUs do sistema, caso haja mais do que uma, e a utilização total de CPU, que é a média da utilização entre todas as CPUs. O LUPA considera esta última medida em sua coleta. Com isso, em uma máquina com duas CPUs, quando uma está ociosa e a outra 100% utilizada, o LUPA considera que a utilização de CPU é 50%.

A quantidade de memória disponível é obtida no `/proc/meminfo`. A memória disponível é a soma da memória livre (`MemFree`), *caches* (`Cached`) e *buffers* (`Buffers`).

4.3 Cálculo dos padrões

O módulo de cálculo dos padrões realiza a tarefa de identificar os padrões de uso dos recursos computacionais compartilhados com a grade. Para isso, os dados coletados pelo módulo de coleta são utilizados como entrada no algoritmo de análise de agrupamentos a fim de separá-los em grupos, cada um representando um padrão de uso.

A classe `ClusterAnalyzer` contém a lógica do cálculo dos padrões. Boa parte das decisões tomadas com relação ao processo de análise de agrupamentos está nela (Seção 4.3.1). Os dois principais elementos de um `ClusterAnalyzer` são um `DataCollector`, que fornece os dados de utilização dos recursos, e um `ClusteringAlgorithm`, que é uma classe abstrata que encapsula a implementação de um algoritmo de análise de agrupamentos. Na Seção 4.3.1 veremos que a implementação utilizada de `ClusteringAlgorithm` é a `KMeansClusteringAlgorithm`.

O algoritmo de análise de agrupamentos recebe como entrada uma lista de `UsageData` e o `resource` a ser analisado e calcula uma lista de `Cluster`. O `UsageData` é uma sequência de `ResourceData` em um período de 48 horas. Apesar de parecer natural que, tanto os objetos que são entrada para

o algoritmo de análise de agrupamentos quanto sua saída, representem a utilização no período de 24 horas, foi adotado o período de 48 horas, onde as primeiras 24 horas de um `UsageData` coincidem com as últimas 24 horas do `UsageData` anterior. Como os padrões calculados representam o comportamento esperado, se utilizássemos um intervalo de 24 horas, não seria possível prever a utilização durante a madrugada, entre o final de um dia e o início de outro, pois o padrão de uso identificado terminaria à meia-noite. Na Seção 4.4 serão apresentadas outras motivações dessa abordagem.

Os `UsageData` processados são separados em agrupamentos, representados por objetos da classe `Cluster`. Cada `Cluster` possui a lista dos `UsageData` nele contidos e um *objeto representativo*, que também é um `UsageData`. Esse objeto representativo caracteriza cada um dos agrupamentos. Um `UsageData` que está contido em um agrupamento é mais semelhante ao objeto representativo de seu agrupamento do que o de qualquer outro.

A lista dos `Cluster` representando os padrões calculados é mantida em memória e em momento algum é armazenada de forma persistente. Na Seção 5.2 veremos que essa estratégia não compromete o desempenho da máquina onde o LUPA está sendo executado.

Na próxima seção, passaremos pelas etapas do processo de análise de agrupamentos sugeridas na Seção 3.1.

4.3.1 Processo de análise de agrupamentos

1. *Escolha dos dados a serem analisados*

   Conforme mencionado anteriormente, a entrada do algoritmo é composta por objetos `UsageData`, que são obtidos a partir do arquivo de log. Apesar de todo o histórico de utilização dos recursos estar registrado no arquivo de log, é realizada uma seleção dos dados a serem considerados pelo algoritmo. Apenas os `UsageData` que satisfazem as seguintes condições são considerados:

   - o `UsageData` não tem mais do que 6 meses de idade;
   - menos de 10% dos `ResourceData` desse `UsageData` são estimativas

   A motivação da primeira condição é evitar que dados de utilização muito antigos tenham influência nos padrões de utilização, pois os padrões serão recalculados freqüentemente (Seção 4.5) e também pode acontecer dos modos de utilização terem mudado por uma miríade de motivos, como a mudança na agenda do usuário da máquina ou, até

mesmo, a troca do usuário por outro. A segunda condição visa evitar que dados poucos precisos, por possuirem muitas medidas estimadas, viciem o resultado.

O algoritmo possui uma restrição quanto à quantidade mínima de dados suficiente para o cálculo dos padrões. O cálculo não é possível caso não existam menos do que 20 `UsageData` válidos. Dessa forma, é necessário que o arquivo de log contenha dados de utilização de, no mínimo, 21 dias.

2. *Seleção das variáveis*

   Os `UsageData` possuem o histórico de utilização de um período de 48 horas de vários recursos computacionais. No entanto, para o algoritmo de análise de agrupamentos, será considerado apenas um recurso computacional por vez. Ou seja, os agrupamentos calculados para cada recurso serão diferentes. Assim, o aumento ou diminuição no uso de um determinado recurso não influencia nos padrões de utilização de um outro recurso, apesar de parecer intuitivo que haja uma correlação entre alguns recursos, como CPU e memória.

   O algoritmo não utiliza as medidas exatamente como foram obtidas pelo mecanismo de coleta. É realizado um pré-processamento para calcular a variação das medidas e esses valores é que são utilizados efetivamente pelo algoritmo de análise de agrupamentos. A variação das medidas é a diferença entre uma medida e a medida imediatamente anterior a esta. Considerando que é possível construir um gráfico da medida de um determinado recurso em função do tempo, o conceito dessa variação das medidas é semelhante ao de uma função derivada, no sentido de indicar o quanto as medidas estão aumentando ou diminuindo em função do tempo. A Figura 4.3 ilustra a variação das medidas num período de 24 horas, comparando-as com as próprias medidas observadas.

   A utilização das variações das medidas torna o algoritmo mais sensível a mudanças na atividade sendo exercida na máquina. No momento em que um usuário se conecta a uma máquina e abre os aplicativos que deseja usar, haverá o registro de um aumento na utilização de memória, por exemplo. Independente da quantidade de memória que estava sendo utilizada antes, o aumento no consumo será praticamente o mesmo se o usuário iniciar os mesmos programas outro dia. Foi demonstrado na dissertação de Bezerra [8] que são obtidos melhores resultados na predição da utilização de recursos computacionais quando

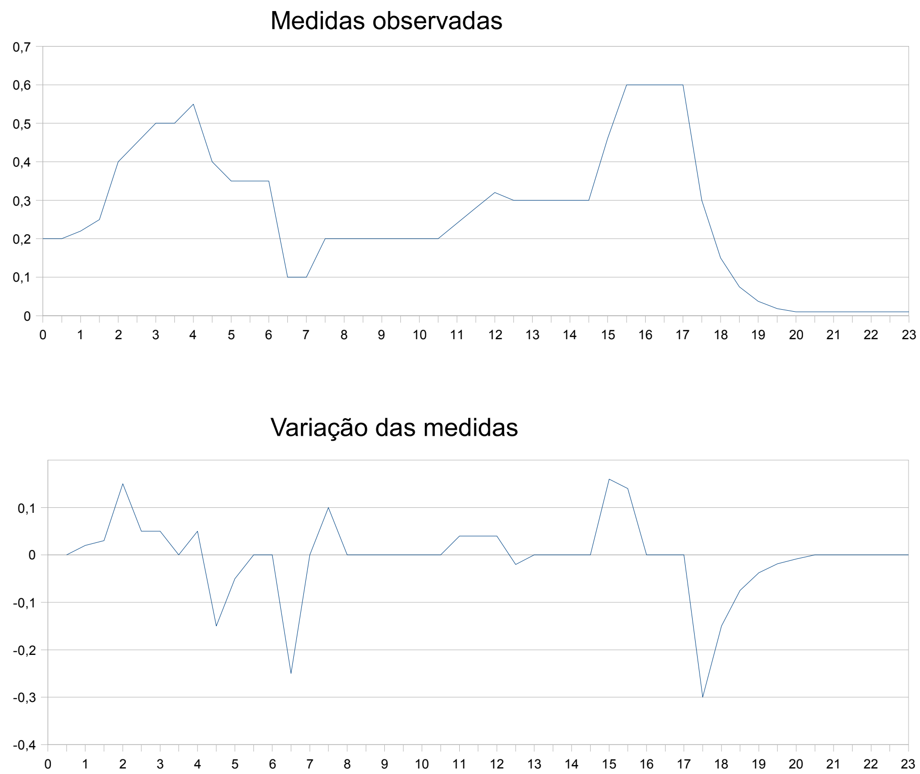

Figura 4.3: Medidas observadas *vs.* Variação das medidas

utilizam-se as variações das medidas ao invés das próprias medidas.

3. *O que agrupar*

   O que o algoritmo separa em grupos (objetos `Cluster`) são os `UsageData`. Mas, para representar cada um desses grupos, há um *objeto representativo*, que é a média dos `UsageData`. A média é calculada somando todos os `ResourceData` referentes ao um mesmo instante e dividindo-os pela quantidade de `UsageData` desse `Cluster`.

   O objetivo da separação dos objetos em grupos é a identificação de padrões de uso. Estabelecidos os grupos, ou *clusters*, o comportamento esperado de cada um dos padrões é o objeto representativo.

4. *Homogeneização das variáveis*

   Não foi utilizado nenhum procedimento para homogeneização das variáveis, mas a utilização das variações das medidas ao invés das próprias medidas, de certa forma, já faz esse papel.

5. *Medidas de semelhança*

   O sistema implementa dois tipos de medidas de semelhança (ou distâncias): entre dois `UsageData` e entre `UsageData` e `Cluster`. A distância entre `UsageData` e `Cluster` é a distância entre o `UsageData` e o objeto representativo do `Cluster`, que é o centro do agrupamento. Dessa forma, internamente, o cálculo efetivo desses dois tipos de distância sempre é feito sobre dois `UsageData`. A medida utilizada é a distância euclideana, definida por $d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$. Dado que na dissertação de Bezerra [8] observou-se que, para o problema em questão, a métrica da distância entre elementos influencia pouco no resultado do algoritmo, a distância euclideana foi escolhida por sua simplicidade de implementação.

6. *Escolha dos algoritmos e implementação*

   Assim como a medida de semelhança, também foi demonstrado em [8] que as diferentes implementações do algoritmo de análise de agrupamentos trazem resultados semelhantes, não sendo significativa a diferença entre eles. Foi utilizado o método das k-Médias (3.1.3) devido a, principalmente, ser o algoritmo mais rápido. Incluímos uma etapa de pré-processamento dos objetos de entrada, onde apenas alteramos a ordem deles aleatoriamente. Durante o desenvolvimento, notamos que quando os objetos de entrada eram apresentados ao algoritmo na ordem cronológica, havia uma tendência deles se concentrarem num

único agrupamento. A aleatorização dessa lista resultou numa melhor distribuições dos objetos entre os agrupamentos.

No primeiro passo do algoritmo, a escolha dos k elementos que farão parte dos agrupamentos iniciais, estão sendo usados os primeiros `UsageData` dessa lista passada como parâmetro para o algoritmo, cuja ordem foi aleatorizada. A busca pelo agrupamento adequado dos demais elementos é feita calculando a distância entre os elementos e os agrupamentos já estabelecidos. Essa distância é descrita no item anterior, e é a distância entre objetos e o centróide dos agrupamentos.

7. *Número de agrupamentos*

   O método das k-Médias exige que o número de agrupamentos seja pré-estabelecido. No trabalho de Bezerra[8] foram realizados testes com 5 e 10 agrupamentos, sendo que não houve diferença significativa nos resultados alternando esses parâmetros. Devido à própria natureza do problema, parece mais intuitivo utilizar 5 agrupamentos, que é o parâmetro adotado para esta implementação.

8. *Interpretação dos resultados*

   As técnicas aqui usadas foram analisadas no mestrado de Bezerra [8] e bons resultados foram obtidos. Neste trabalho, realizamos uma série de experimentos para analisar esta implementação específica e seu desempenho no escalonamento de aplicações distribuídas em grades computacionais. No Capítulo 5, descrevemos os experimentos realizados e os resultados obtidos.

4.4 Mecanismo de predição

A predição da utilização dos recursos computacionais é baseada no histórico de uso de uma determinada máquina (Seção 4.2), a partir do qual são estabelecidos os padrões de uso (Seção 4.3). Esses padrões, caracterizados por seus objetos representativos, são os modos de utilização estabelecidos para a máquina e o comportamento esperado em cada um desses modos é equivalente ao objeto representativo desse padrão.

O primeiro passo para prever a utilização é identificar o modo de utilização atual. Essa tarefa consiste em localizar, dentre os padrões calculados, qual deles tem o intervalo equivalente à coleta das últimas 24 horas de utilização mais semelhante com os dados efetivamente coletados nas últimas 24 horas, que denominaremos *amostra de utilização recente*. A partir dessa

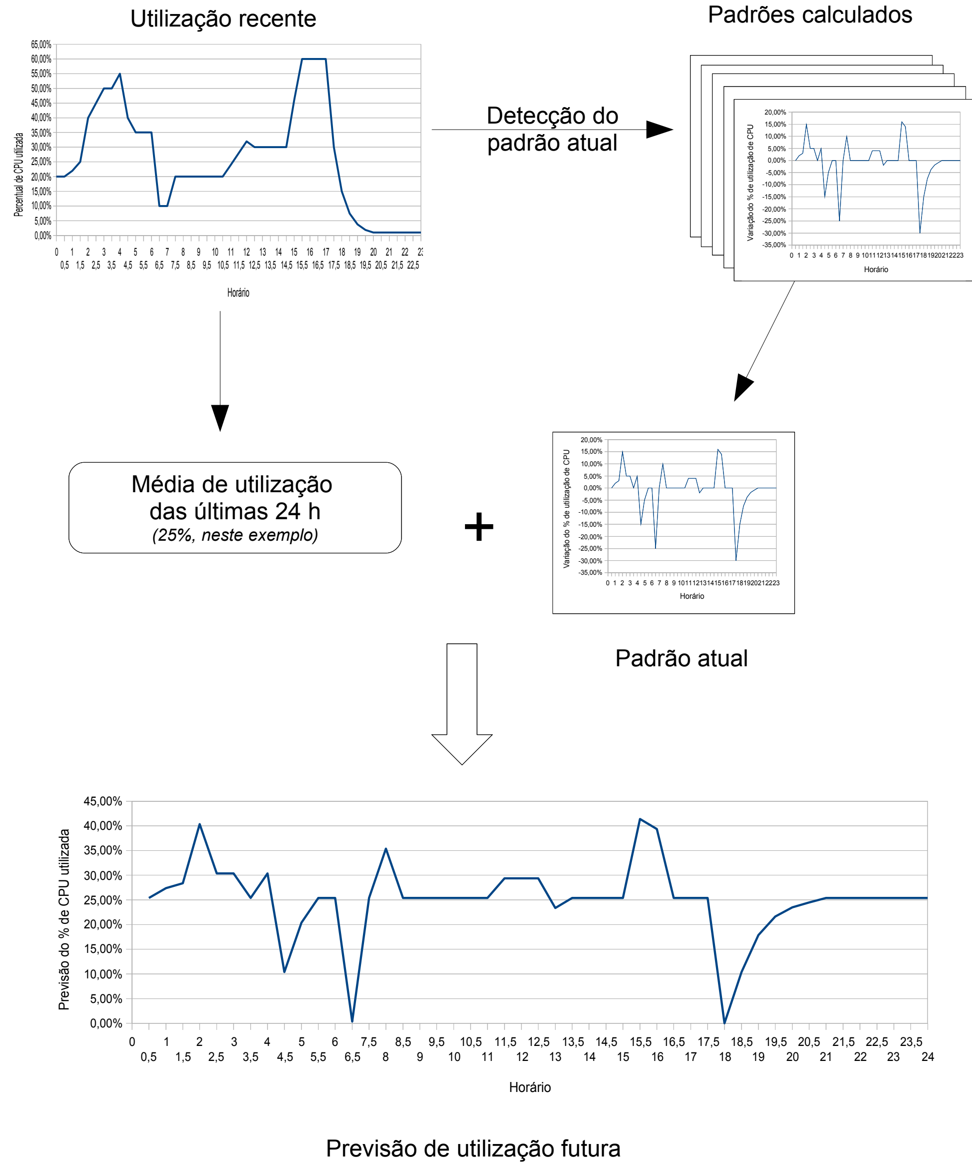

Figura 4.4: Visão geral do mecanismo de predição

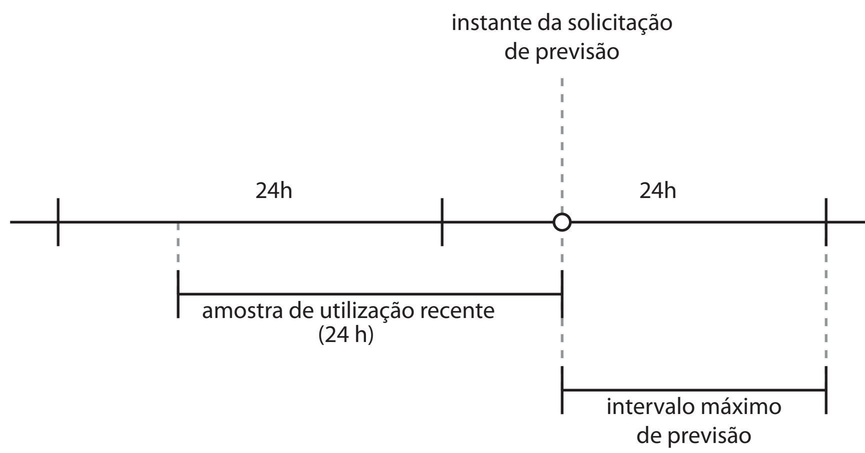

Figura 4.5: Intervalos de tempo da predição

amostra, é calculada a variação das medidas de utilização recente dos recursos, já que os padrões estabelecidos são baseados nessas variações. Esses dados são representados internamente por objetos `UsageData` que, no entanto, não possuem a lista inteira de `ResourceData` preenchida. Para encontrar o padrão mais semelhante a esse objeto, utilizamos a mesma métrica de distância de `UsageData` a `Cluster` utilizada pelo algoritmo de cálculo dos agrupamentos, porém aplicada somente ao intervalo de `ResourceData` que estão preenchidos.

Uma vez que já se sabe o padrão atual de utilização, já é possível prever a utilização futura. O intervalo máximo de tempo em que será possível prever a utilização (*intervalo máximo de previsão*) é delimitado pelo instante em que foi realizada a previsão e o final do `UsageData` que representa o padrão, conforme demonstrado na Figura 4.5. Dessa forma, notamos que a previsão da utilização futura é limitada até a meia-noite do dia atual. Nos casos em que a previsão é solicitada próxima a esse horário, o intervalo máximo de previsão poderá ser muito curto. Atualmente, não há nenhum tratamento especial para esses casos, mas uma possibilidade seria diminuir o tamanho da amostra de utilização recente de forma que o intervalo máximo de previsão sempre tivesse um tamanho mínimo, que garantisse a qualidade da previsão realizada.

O padrão encontrado, todavia, não fornece uma previsão absoluta da utilização dos recursos no futuro próximo, pois contém uma previsão da variação da utilização. Dado esse caráter relativo, é preciso um referencial para ser possível fornecer uma previsão absoluta, como "a previsão de utilização de CPU na próximo hora é de no máximo 40%", aos módulos do sistema que utilizam esses dados. Neste trabalho, o referencial adotado foi a média de utilização da amostra de utilização recente. Ou seja, se a média de utilização recente de CPU é de 10% e a previsão da variação seja um aumento de 25%, a previsão absoluta será de utilização será de 35%. A Figura 4.4 ilustra esse mecanismo.

O LUPA possui dois métodos para responder a um pedido da previsão de utilização:

1. *Fornecendo a previsão de utilização nas próximas horas*

   O solicitante acessa o método

   ```
   double[] Lupa::getPrediction(resource r, int hours),
   ```

   que devolve uma seqüência de valores numéricos que representa a previsão de utilização do recurso `r` nas próximas `hours` horas a intervalos de cinco minutos.

2. *Verificando se satisfaz à solicitação do usuário*

   O método

   ```
   bool Lupa::canRunGridApplication(double freeCpuRequired,
                   double freeMemoryRequired),
   ```

   devolve *true* caso a previsão seja de que haverá pelo menos `freeCpuRequired` de CPU e `freeMemoryRequired` de memória disponíveis pelas próximas seis horas[2]. Porém, nem sempre será possível realizar uma previsão de utilização dos recursos baseada no padrão de utilização atual. Temos, basicamente, dois casos onde isso pode ocorrer: (i) não há dados suficientes para o cálculo dos padrões ou (ii) não há dados suficientes para construir a amostra de utilização recente. O caso (i) deve acontecer, em geral, quando o InteGrade está rodando há pouco tempo em uma máquina e ainda não há dados suficientes[3] sobre a utilização dos recursos para calcular os padrões. Já o caso (ii) ocorre quando o InteGrade acaba de ser iniciado e não possui dados recentes de utilização para a identificação do padrão de uso atual. Em ambos os casos, o método `Lupa::getPrediction(..)` devolve uma seqüência vazia. O método `Lupa::canRunGridApplication(..)` considera que a previsão de utilização é equivalente à média de utilização das últimas quatro horas, exceto quando não há dados recentes nem dessas últimas quatro horas. Nesse caso, devolve *true*. Este método está descrito no Algoritmo 1.

[2] Esse valor de seis horas da previsão foi definido inicialmente, mas pode ser facilmente alterado para outros valores.

[3] O algoritmo exige que hajam dados válidos da coleta de recursos por no mínimo 21 dias, conforme Seção 4.3.1 - item 1

Algoritmo 1: canRunGridApplication(..)

Entrada: Parâmetros freeCpuRequired e freeMemoryRequired
Saída: *true*, caso a previsão seja de que haverá pelo menos `freeCpuRequired` de CPU e `freeMemoryRequired` de memória livres nas próximas 6 horas, ou *false*, caso contrário

se *Padrões estão calculados e há dados de utilização recente* **então**
  se *a previsão satisfaz os parâmetros* **então**
    retorna *true*
  senão
    retorna *false*
senão se *Há dados de utilização das últimas 4 horas* **então**
  se *Média de utilização das últimas 4h satisfaz os parâmetros* **então**
    retorna *true*
  senão
    retorna *false*
senão
  retorna *true*

4.5 Recálculo dos padrões

Os padrões de utilização podem mudar com o tempo. Uma máquina pode ter, por exemplo, seus usuários trocados, os usuários podem sofrer alterações súbitas em suas rotinas, como férias ou mudança no horário de trabalho, ou uma estação de trabalho pode tornar-se um servidor e vice-versa. Todas essas situações causam mudança nos padrões de uso. E, por isso, o processo de aprendizado dos padrões deve ser constante.

Esse tema, a evolução do aprendizado, já foi bastante estudado e várias abordagens podem ser encontradas na literatura. Neste trabalho, no entanto, não foram utilizadas técnicas elaboradas. Os padrões são recalculados uma vez por dia. Conforme veremos na Seção 5.2, a sobrecarga causada pelo recálculo dos padrões não compromete o desempenho da máquina onde o sistema está rodando.

4.6 Escalonamento

O algoritmo atual de escalonamento das aplicações distribuídas no InteGrade é o *round robin*, onde os nós disponíveis são colocados em uma lista circular e as aplicações são enviadas para os próximos nós da lista. No entanto, antes

da aplicação ser efetivamente enviada, é feita a seguinte chamada ao LUPA do nó em questão:

`Lupa::canRunGridApplication(..)`.

Caso a resposta seja *true*, a aplicação é enviada ao nó, caso contrário, é feita a mesma chamada ao próximo nó da lista, e assim por diante, até que se encontre um nó que responda *true*.

Na versão atual do InteGrade, o LUPA está desabilitado, o que é equivalente aos nós sempre devolverem *true* para a chamada acima. Mas, nas versões posteriores à atual (0.4), o LUPA será reabilitado, e a resposta da chamada acima será baseada nesta implementação nova, descrita na seção anterior. Ou seja, neste trabalho não houve alteração no algoritmo de escalonamento em si, mas na implementação dos métodos que ele já acessava. O algoritmo atual é equivalente ao `can run`, que será descrito nos experimentos do próximo capítulo, na Seção 5.1.

Capítulo 5

Experimentos

Neste capítulo serão descritos os experimentos realizados com o LUPA a fim de verificar se as técnicas estudadas e implementadas podem melhorar o desempenho de aplicações executadas em grades computacionais oportunistas através de um escalonamento mais eficaz. Também analisaremos a sobrecarga no sistema devido à utilização dessas técnicas.

5.1 Escalonamento

O objetivo desses experimentos foi mostrar como podemos usar a previsão de utilização de recursos do LUPA para escalonarmos aplicações em grades computacionais oportunistas de forma mais eficaz. Foram realizadas simulações do escalonamento e execução de aplicações baseados nos arquivos de log que foram coletados em algumas máquinas dos laboratórios Eclipse e LCPD do IME-USP e dois servidores da Objective Solutions[1].

5.1.1 Descrição dos dados

Em dezembro de 2006 foi colocado um coletor de dados para gerar o arquivo de log nas máquinas dos laboratórios Eclipse e LCPD. Desde então, os dados de utilização têm sido coletados nessas máquinas mas houve alguns períodos em que o coletor esteve parado. Em geral, esses períodos não passaram de duas semanas. A causa, na maioria das vezes, foram manutenções nas redes. Isso ocasionou algumas lacunas nos arquivos de log, onde não há informações sobre utilização. Na Objective Solutions, o coletor foi iniciado em outubro

[1] A Objective Solutions é a uma empresa de desenvolvimento de software que gentilmente permitiu o uso de sua infra-estrutura para a realização deste trabalho.

de 2007. Os dados utilizados neste trabalho possuem informações coletadas até janeiro de 2008. Na tabela abaixo temos a lista dos arquivos de log que foram considerados nesses experimentos e as quantidades de dias válidos em cada um deles. Dias válidos são dias em que temos pelo menos 90% da informação de utilização dos recursos. Nesses experimentos, ignoramos os dias que não são válidos e utilizamos apenas os arquivos de log com no mínimo 40 dias válidos.

Arquivo	**Dias válidos**
bauru.log	60
urano.log	51
jupiter.log	85
plutao.log	95
oracle08.sp.objective.log	55
hubble.log	58
venus.log	120
mercurio.log	199
europa.log	103
callisto.log	87
villa.log	53
saturno.log	116
giga.log	74
oracle09.log	41
motuca.log	112

Essas máquinas têm usos variados. Algumas delas são servidores com bastante variação no consumo de recursos, servidores com uso mais uniforme de recursos ou estações de trabalho dos laboratórios de pesquisa do IME-USP. As informações monitoradas foram de utilização de CPU e consumo de memória.

5.1.2 Ensaios

Os ensaios consistem em simular o escalonamento de uma aplicação na grade usando quatro algoritmos de escalonamento diferentes e comparar seus desempenhos. Os ensaios solicitam aos algoritmos que escolham n nós da grade para executar uma aplicação na próximas h horas. Os algoritmos utilizados foram:

- `round robin` — os nós da grade que estão compartilhando recursos são colocados em uma lista circular e os escolhidos são os n próximos elementos dessa lista, que é ciclada n posições após essa escolha;
- `can run` — é como o `round robin`, porém a lista circular contém apenas os nós que respondem *true* à chamada de `canRunGridApplication(..)` com os parâmetros de no mínimo 50% de CPU e 100MB de memória disponíveis (Seção 4.4);
- `get prediction` — são escolhidos os n nós com menor previsão de utilização nas próximas h horas. Essa previsão é fornecida pelo método `getPrediction(..)` (Seção 4.4);
- `last 4h` — a previsão de utilização de CPU é a média de utilização das últimas 4 horas e são escolhidos os nós com menor previsão de utilização.

O algoritmo `round robin` é a implementação atual do InteGrade, os algoritmos `can run` e `get prediction` utilizam as técnicas de análise de agrupamentos para predição de utilização futura de recursos computacionais estudadas neste trabalho e o `last 4h` utiliza uma técnica simples, mas que traz bons resultados. Como alternativa ao `last 4h`, foram analisados algoritmos análogos, utilizando, porém, a média das últimas 8, 12 e 24 horas. Consideramos os seus desempenhos equivalentes e escolhemos utilizar apenas as últimas 4 horas devido o método `canRunGridApplication(..)` do LUPA utilizar a média das últimas quatro horas de utilização como previsão no caso de falta de dados para o cálculo dos padrões, além de ser o mais eficiente.

Em cada ensaio, é realizada uma seqüência de testes. Inicialmente, são selecionados os dias válidos, dentre todos os encontrados em todos os arquivos de log, em que há no mínimo m máquinas com dados coletados para esses dias. Esse conjunto de dias são os dias elegíveis à execução de testes e o conjunto de máquinas disponíveis associada a esses dias, as máquinas elegíveis. Para cada um desses dias, são executados 24 testes, um em cada hora do dia. Chamaremos esses instantes de *instantes de teste*. Cada teste corresponde a executar o escalonamento com os quatro algoritmos acima num mesmo instante de teste.

Neste trabalho, consideramos que o desempenho de um algoritmo é tão melhor quanto mais CPU disponível houver nas máquinas que ele escolheu. A métrica de desempenho dos algoritmos adotada é a média de CPU disponível entre as máquinas escolhidas no período entre o instante de teste e as próximas h horas, definido por:

$$desempenho_i(t) = 1 - \frac{\sum^{m \in M_i(t)} utilizacao(m,t,h)}{n},$$

onde i é um algoritmo de escalonamento, $M_i(t)$ é o conjunto de n máquinas que o algoritmo i escolheu para rodar uma aplicação por h horas a partir do instante de teste t e $utilizacao(m,t,h)$ é uma função que devolve a média de utilização de CPU, entre zero e um, da máquina m no período de h horas após o instante t, baseada no arquivo de log dessa máquina. Vemos que essa medida é baseada nos arquivos de log pois, como o instante de teste está num dia válido para essas máquinas, sabemos qual foi a utilização de CPU que ocorreu entre o instante de teste e as próximas h horas.

Exemplificando: se o desempenho de um algoritmo for 0.75, significa a média de CPU ociosa entre as máquinas escolhidas era de 75%. O menor valor possível é zero, que significa que todas as máquinas estavam com as CPUs totalmente utilizadas, e o maior valor, um, significa que todas estavam completamente ociosas.

Os parâmetros para um ensaio de testes são os seguintes:

- n — quantidade de nós que serão necessários para executar a aplicação;
- h — tempo em horas de execução da aplicação;
- m — quantidade mínima de máquinas disponíveis em um mesmo dia para considerá-lo elegível à execução de testes;
- d — percentual máximo de disponibilidade de CPU média entre as máquinas elegíveis para execução de um teste.

O parâmetro d impõe uma restrição sobre os instantes de testes em que serão efetivamente realizados testes. Só serão executados testes em instantes de teste t que satisfaçam:

$$1 - \frac{\sum^{m \in E(t)} utilizacao(m,t,h)}{|E(t)|} < d,$$

onde $E(t)$ é o conjunto de máquinas elegíveis para execução de um teste no instante t. Quanto menor o valor de d, ou seja, quanto menos CPU livre houver nas máquinas que compartilham recursos no instante do teste, mais importante é o papel de um algoritmo de escalonamento eficaz para conseguir selecionar os nós que ofereçam melhores condições para a execução da aplicação. Podemos dizer também, intuitivamente, que quanto menor o valor de d, mais difícil é a tarefa do algoritmo de escalonamento.

Os algoritmos `can run` e `get prediction` utilizam métodos do LUPA, implementados neste trabalho. Para estes casos, os arquivos de log descritos na seção anterior foram utilizados para o cálculo dos padrões de uso. O cálculo é feito no início de cada ensaio de teste e não há recálculo durante um ensaio, ou seja, os padrões são os mesmos para todos os instantes de teste. Com isto, estamos supondo que, durante todo o período de coleta de dados de uma máquina, não houve mudança em seus padrões de uso. Quando é feita uma predição de uso futuro pelo LUPA, os arquivos de log também são utilizados para a identificação da amostra de utilização recente. Dessa forma, os arquivos de log são utilizados tanto para treinamento quanto para testes, sem separação. Isso não compromete os resultados, uma vez que supusemos que os padrões de uso se mantêm constantes durante o período de coleta.

A saída de cada teste contém os seguintes dados:

- instante de teste;
- quantidade de máquinas elegíveis para esse instante de teste;
- desempenho do `round robin` (média do desempenho de 200 simulações consecutivas desse algoritmo);
- desempenho do `can run` (média do desempenho de 200 simulações consecutivas desse algoritmo);
- desempenho do `get prediction` (não é necessário executar mais de uma simulação pois o algoritmo é determinístico para um mesmo conjunto de parâmetros);
- desempenho do `last 4h` (não é necessário executar mais de uma simulação pois o algoritmo é determinístico para um mesmo conjunto de parâmetros);
- desempenho do pior escalonamento;
- desempenho do melhor escalonamento.

Os desempenhos do melhor e pior escalonamentos são baseados nos arquivos de log e não representam uma estratégia de escalonamento, servem apenas como referência para o desempenho dos outros quatro algoritmos. Na tabela abaixo estão os resultados de alguns testes do ensaio com os parâmetros: $n = 2$, $h = 2$, $m = 4$ e $d = 0.7$.

Instante de teste	#	rrobin	can run	get pr.	last 4h	pior	melhor
2007-11-05 19:00	5	0.6322	0.8872	0.9997	0.9997	0.2514	0.9997
2007-11-08 01:00	5	0.6913	0.9920	0.9999	0.9955	0.2415	0.9999
2007-11-09 01:00	5	0.6600	0.9947	0.9998	0.9998	0.1587	0.9998

A saída dessa seqüência de testes provê uma visão analítica caso a caso, mas o resultado do ensaio como um todo é mostrado de forma sintética, comparando o desempenho dos algoritmos entre si. Essas comparações são apresentadas em uma tabela onde cada célula $[i, j]$ é uma comparação entre o algoritmo da i-ésima linha com o algoritmo da j-ésima coluna. A informação em cada célula é composta por três valores: *pior*-*equivalente*-*melhor*, sendo:

- *pior*: percentual de vezes em que o algoritmo i foi pior que o j (o algoritmo i é pior que o j em um teste se:

$$desempenho_i(t) - desempenho_j(t) < -0.02) \ ;$$

- *equivalente*: percentual de vezes em que o desempenho dos algoritmos foi equivalente (o desempenho dos algoritmos é equivalente em um teste se:

$$-0.02 < desempenho_i(t) - desempenho_j(t) < 0.02);$$

- *melhor*: percentual de vezes em que o algoritmo i foi melhor que o j (o algoritmo i é melhor que o j em um teste se:

$$desempenho_i(t) - desempenho_j(t) > 0.02).$$

Com isso, percebemos que estamos considerando diferenças de desempenho inferiores a 2% insignificantes. A motivação dessa decisão é eliminar casos em que há diferença no desempenho devido aos algoritmos terem escolhido máquinas diferentes, mas, no entanto, essa diferença não causa impacto significante no tempo de execução da aplicação.

Abaixo nós temos um exemplo de tabela para o ensaio com os seguintes parâmetros: $n = 2$, $h = 2$, $m = 4$ e $d = 0.7$, onde foram realizados 150 testes. Como exemplo, devemos ler a célula da primeira linha, segunda coluna (`round robin` *versus* `get prediction`) como: o `round robin` foi pior que o `get prediction` em 83,3% dos testes, equivalente em 16% e melhor em 0,7%.

	`can run`	`get prediction`	`last 4h`
`round robin`	83.3 - 16.7 - 0.0	83.3 - 16.0 - 0.7	71.3 - 16.0 - 12.7
`can run`		57.3 - 42.0 - 0.7	46.7 - 32.7 - 20.7
`get prediction`			4.0 - 72.7 - 23.3

Além desse comparativo de desempenho, ainda comparamos o desempenho dos algoritmos `can run`, `get prediction` e `last 4h` ao `round robin` para identificarmos o quanto cada um deles pode, de fato, melhorar o desempenho de uma aplicação executada numa grade oportunista. A comparação é feita com o `round robin` porque, além dele ser o algoritmo implementado na versão anterior do InteGrade, o desempenho dele tende a ser igual à média de CPU disponível entre todas máquinas elegíveis a um teste. Foram calculadas a *média de ganho* de CPU disponível quando os algoritmos são melhores que o `round robin` e a *média de perda* de CPU disponível quando eles são piores. Na tabela abaixo temos esses valores para o ensaio do exemplo acima ($n = 2$, $h = 2$, $m = 4$ e $d = 0.7$). Podemos ver, por exemplo, que o algoritmo `get prediction` consegue, em média, escolher máquinas para rodar a aplicação com 41% a mais de CPU disponível que o `round robin`. E, de acordo com a tabela acima, isso ocorre em 83,3% dos casos. Já nos 0,7% dos casos em que seu desempenho é pior, encontra-se 18% menos CPU disponível.

	`can run`	`get prediction`	`last 4h`
média de ganho	0.33	0.41	0.39
média de perda	nan	-0.18	-0.08

Essas duas últimas tabelas são a base para a análise dos resultados desse experimento, onde mostraremos que as técnicas estudadas e implementadas podem trazer bons resultados se usadas no escalonamento de aplicações distribuídas em grades oportunistas. Vale ressaltar que nesses ensaios estamos avaliando somente a previsão da utilização de CPU. Além disso, não estamos levando em conta que as CPUs das máquinas têm velocidades diferentes. Estamos supondo nesta simulação que quanto mais tempo de CPU disponível conseguirmos através do escalonamento, melhor será o tempo de execução da aplicação.

5.1.3 Análise dos resultados

Os resultados dos ensaios de testes nos mostram que as técnicas implementadas trazem bons resultados quando utilizadas no escalonamento de

aplicações distribuídas em grades oportunistas. Em geral, os algoritmos `can run`, `get prediction` e `last 4h` têm desempenhos melhores que o `round robin`. Percebemos que há um salto grande de melhora no desempenho do `can run` com relação ao `round robin`, e um salto menor entre o `can run` e o `get prediction` e o `last 4h`. Estes dois últimos não apresentam uma diferença muito expressiva entre seus desempenhos. Nos ensaios realizados, notamos uma ligeira vantagem para o `get prediction`, principalmente quando a relação m/n é aumentada.

Comparamos os resultados de ensaios variando a relação m/n, ou seja, entre a quantidade de nós disponíveis e o número de nós necessários para a aplicação, pois quanto maior esta relação, maior a quantidade possível de combinações de nós a serem escolhidas e, conseqüentemente, maior a dificuldade de se escolher a combinação de nós que maximize o tempo disponível de CPU ou, pelo menos, de aproximar-se disso. Nos ensaios abaixo podemos observar como o `can run` e o `get prediction` obtém melhores resultados conforme a relação m/n aumenta e, no entanto, essa tendência não é observada no `last 4h`. Vale ressaltar, no entanto, que em ambientes reais de grades, os valores de m devem ser de centenas e n, dezenas. Nestes experimentos, analisamos com valores bem menores do que esses.

- Ensaio: $n = 2$, $h = 4$, $m = 3$ e $d = 0.8$. Testes: 507.

	can run	get prediction	last 4h
round robin	78.3 - 17.8 - 3.9	88.8 - 9.5 - 1.8	80.7 - 9.5 - 9.9
can run		53.5 - 44.8 - 1.8	50.7 - 36.5 - 12.8
get prediction			7.7 - 77.1 - 15.2

- Ensaio: $n = 2$, $h = 4$, $m = 4$ e $d = 0.8$. Testes: 433.

	can run	get prediction	last 4h
round robin	88.0 - 9.0 - 3.0	93.5 - 5.8 - 0.7	83.1 - 5.5 - 11.3
can run		62.6 - 34.9 - 2.5	61.7 - 23.6 - 14.8
get prediction			9.2 - 73.4 - 17.3

- Ensaio: $n = 2$, $h = 4$, $m = 5$ e $d = 0.8$. Testes: 369.

	can run	get prediction	last 4h
round robin	89.4 - 10.6 - 0.0	92.7 - 6.8 - 0.5	80.2 - 6.5 - 13.3
can run		68.8 - 27.9 - 3.3	65.6 - 19.8 - 14.6
get prediction			10.6 - 69.1 - 20.3

- Ensaio: $n = 2$, $h = 4$, $m = 6$ e $d = 0.8$. Testes: 133.

	can run	get prediction	last 4h
round robin	92.5 - 7.5 - 0.0	97.7 - 0.0 - 2.3	81.2 - 0.0 - 18.8
can run		83.5 - 13.5 - 3.0	64.7 - 9.0 - 26.3
get prediction			9.0 - 54.1 - 36.8

Analisamos também o impacto do parâmetro d nos resultados. Quanto menor esse valor, mais importante o papel do escalonamento para encontrar os nós com maior disponbilidade de CPU. Não conseguimos estabelecer um padrão de como os algoritmos reagem às variações desse valor mas observamos que, mesmo com os valores mais baixos que conseguimos testar, ainda obtivemos bons resultados por parte dos algoritmos implementados. O que é possível observar é a melhoria do desempenho do `get prediction` com relação ao `last 4h`, que pode ser observado também na média de ganho de desempenho obtida. Nos exemplos abaixo podemos observar a variação nos resultados conseqüente da alteração de d.

- Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.9$. Testes: 797.

	can run	get prediction	last 4h
round robin	87.0 - 12.3 - 0.8	90.6 - 9.0 - 0.4	89.6 - 10.0 - 0.4
can run		51.7 - 45.9 - 2.4	55.3 - 40.2 - 4.5
get prediction			8.0 - 83.7 - 8.3

	can run	get prediction	last 4h
média de ganho	0.17	0.21	0.20
média de perda	-0.03	-0.10	-0.07

- Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.8$. Testes: 354.

	can run	get prediction	last 4h
round robin	82.5 - 17.5 - 0.0	86.2 - 13.8 - 0.0	84.2 - 15.5 - 0.3
can run		43.2 - 54.5 - 2.3	48.9 - 44.6 - 6.5
get prediction			7.6 - 83.1 - 9.3

	can run	get prediction	last 4h
média de ganho	0.24	0.28	0.27
média de perda	nan	nan	-0.06

- Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.7$. Testes: 140.

	can run	get prediction	last 4h
round robin	67.9 - 32.1 - 0.0	81.4 - 18.6 - 0.0	81.4 - 18.6 - 0.0
can run		7.9 - 90.7 - 1.4	11.4 - 76.4 - 12.1
get prediction			2.9 - 80.7 - 16.4

	`can run`	`get prediction`	`last 4h`
média de ganho	0.33	0.37	0.32
média de perda	nan	nan	nan

- Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.6$. Testes: 103.

	`can run`	`get prediction`	`last 4h`
`round robin`	73.8 - 26.2 - 0.0	87.4 - 12.6 - 0.0	87.4 - 12.6 - 0.0
`can run`		4.9 - 94.2 - 1.0	5.8 - 80.6 - 13.6
`get prediction`			1.0 - 83.5 - 15.5

	`can run`	`get prediction`	`last 4h`
média de ganho	0.36	0.38	0.33
média de perda	nan	nan	nan

Conforme podemos observar nos ensaios acima, obtivemos bons resultados com relação à média de ganho de CPU disponível. Nos ensaios acima, obtivemos de 17% a 36% de média de ganho no `can run`, 21% a 38% no `get prediction` e 20% a 33% no `last 4h`. Esses valores são expressivos e podem representar uma melhora significativa no tempo de execução de aplicações distribuídas em grades oportunistas. Quanto à média de perda, analisando o resultado dos ensaios, podem-se observar valores próximos à média de ganho. Não acreditamos que isso comprometa a eficácia dos algoritmos implementados, pois a média de perda é calculada sobre os testes em que o `round robin` tem desempenho melhor que os outros algoritmos e isso mostrou-se exceção em nossos experimentos. Assim como no comparativo de desempenho entre os algoritmos, notamos que há um salto na média de ganho entre o `can run` e o `round robin` e um salto menor entre o `can run` e o `get prediction` e o `last 4h`. O mesmo acontece para a média de perda, que, em geral é menor para o `can run`. Mais dados sobre as médias de ganho e perda podem ser encontrados no Apêndice A, onde há uma listagem dos resultados dos ensaios de testes mais relevantes para essas análises e conclusões.

Quanto ao parâmetro h, realizamos alguns ensaios com diferentes valores e observamos que esse parâmetro não influencia muito nas tabelas comparativas que estamos utilizando. Os resultados seguem as mesmas tendências já mencionadas independente do tempo de execução da aplicação.

A Tabela 5.1 resume a análise dos resultados dos experimentos realizados.

Desse modo, concluímos que a utilização das técnicas implementadas no escalonamento de aplicações distribuídas em grades oportunistas pode

	`can run`	`get prediction`	`last 4h`
desempenho	melhor que o `round robin`	melhor que o `can run`	semelhante ao `get prediction`
impacto do aumento de m/n	melhora no desempenho	melhora no desempenho	sem impacto
impacto do aumento de d	sem impacto	melhor desempenho com relação ao `last 4h`	pior desempenho com relação ao `get prediction`
impacto do aumento de h	sem impacto	sem impacto	sem impacto

Tabela 5.1: Resumo comparativo dos comportamentos dos algoritmos.

diminuir o tempo de execução dessas aplicações. De acordo com nossos experimentos, o algoritmo `get prediction` é o mais eficaz nessa tarefa.

5.2 Sobrecarga do LUPA

Na seção anterior, vimos que o LUPA pode aumentar a eficácia do escalonamento de aplicações distribuídas no InteGrade. Nesta seção, analisaremos a sobrecarga causada pelo uso das técnicas de análise de agrupamentos implementadas. Os testes foram executados em um *notebook* com processador AMD Turion 64 1.8 GHz, 1 GB de memória RAM e distribuição Kubuntu versão 7.10 (32 bits) do sistema operacional Linux.

5.2.1 Tempo de processamento

Nestes experimentos, medimos o tempo de processamento necessário para calcular os padrões de uso (CPU e memória) e o tempo de resposta dos métodos `canRunGridApplication(..)` e `getPrediction(..)` (Seção 4.4). Foram utilizados os dados dos arquivos de log com no mínino de 60 dias válidos. Na Tabela 5.2 está o resultado desses testes em segundos, separados por arquivo de log. O tempo do cálculo dos padrões é a média de 20 execuções da rotina de cálculo dos padrões. Os tempos de resposta dos métodos são médias de tempo de 24 chamadas pra cada dia válido do arquivo de log.

Arquivo	Dias válidos	Cálculo dos padrões	`canRunGrid Application(..)`	`get Prediction(..)`
jupiter.log	85	0.455333	0.000284314	0.000147059
plutao.log	95	0.394667	0.000280702	0.000144737
venus.log	120	0.447333	0.000295139	0.000145833
mercurio.log	199	0.870667	0.00239069	0.00147599
europa.log	103	0.457333	0.000299353	0.000141586
callisto.log	87	0.371333	0.000287356	0.000148467
saturno.log	116	0.451333	0.000301724	0.000150862
giga.log	74	0.362	0.000326577	0.000140766
motuca.log	112	0.485333	0.000282738	0.00014881

Tabela 5.2: Tempo de processamento (em segundos) de algumas tarefas do LUPA.

5.2.2 Consumo de memória

O objetivo desse experimento é calcular o consumo de memória das estruturas de dados utilizadas na predição de utilização futura de recursos. Foi criado um programa que analisa os arquivos de log, calcula os padrões e mantém as respectivas estruturas de dados em memória. Medimos o quanto a memória consumida por este programa aumenta após o cálculo dos padrões. A medição do consumo de memória do programa foi feita através do utilitário `top`, que obtém as informações do sistema operacional. Na Tabela 5.3 encontramos a média de memória consumida para dez repetições do cálculo dos padrões por arquivo de log.

Durante esse experimento, notamos que há vazamento de memória no cálculo de padrões pelo LUPA. O método

```
ClusterAnalyzer::updateClusters(vector<UsageData*>* uds),
```

responsável por calcular os padrões de uso, não faz a desalocação de memória das estruturas de dados que armazenam os padrões atuais quando os está recalculando. A conseqüência é que, a cada recálculo de padrões, temos um aumento no consumo da memória alocada pelo LUPA de acordo com a tabela acima. Os padrões são recalculados no máximo uma vez por dia. Dessa forma, um LUPA executando há alguns meses em uma máquina pode consumir um quantidade exageradamente grande de memória. Trata-se de um *bug* no sistema e sua correção deve ter prioridade alta.

Arquivo de log	Dias válidos	Consumo (KB)
jupiter.log	85	2394,4
plutao.log	95	2518,4
venus.log	120	3016,8
mercurio.log	199	4754,8
europa.log	103	2763,2
callisto.log	87	2256,4
saturno.log	116	2964,8
giga.log	74	1874,8
motuca.log	112	2728,4

Tabela 5.3: Consumo de memória das estruturas de dados do LUPA.

5.2.3 Conclusões

Baseado nos resultados dos experimentos que mediram a sobrecarga de CPU e memória conseqüentes das técnicas implementadas de análise de agrupamentos para a predição de utilização futura de recursos computacionais, concluímos que o módulo implementado não sobrecarrega o sistema de forma a comprometer seu desempenho, desde que o vazamento de memória descrito acima seja solucionado.

Capítulo 6

Conclusão

As grades computacionais oportunistas são uma tecnologia emergente e representam uma solução de baixo custo para aplicações que necessitem de alto poder computacional. Um mecanismo capaz de prever a utilização futura dos recursos compartilhados em seus nós, auxiliando o escalonamento de aplicações distribuídas na grade, aumenta o nível de qualidade do serviço oferecido.

Implementamos o módulo LUPA do InteGrade, cuja função é monitorar a utilização de CPU e memória dos nós compartilhados da grade a fim de estabelecer padrões de uso. Esses padrões são utilizados para prever a utilização futura de CPU e memória. As técnicas de cálculo de padrões utilizadas haviam sido propostas no mestrado de Bezerra [8], onde se demonstrou que é possível prever a utilização futura de recursos através do reconhecimento de padrões, calculados a partir da análise de agrupamentos. O foco de seus experimentos foi analisar a influência das variantes do processo de análise de agrupamentos (Seção 3.1) no reconhecimento de padrões para previsão de utilização futura. As medidas de qualidade do resultado foram tiradas máquina a máquina, não no contexto de uma grade computacional. Em nosso trabalho, aproveitamos os resultados obtidos em [8] para modelagem do processo de análise de agrupamentos para o LUPA. Realizamos experimentos para verificar que é possível aumentar o desempenho de aplicações na grade se o algoritmo de escalonamento levar em conta a previsão de utilização futura de cada nó. Além do ganho em tempo de execução, que beneficia quem solicita a execução de uma aplicação, também beneficia quem compartilha recursos com a grade, pois o escalonador passa a evitar máquinas cuja previsão não seja de ociosidade.

Isto posto, o InteGrade tem potencial para ser um projeto de grade com-

putacional pioneiro na utilização da análise de agrupamentos para previsão de utilização futura de recursos no escalonamento.

O código fonte desta implementação está integrado ao código do InteGrade e pode ser encontrado em seu repositório de versões, acessível através do Portal do InteGrade [2]. Os arquivos de log e programas utilizados nos experimentos também estão disponíveis no Portal.

6.1 Limitações do LUPA e Trabalhos Futuros

A principal limitação da implementação atual do LUPA é que a coleta de dados sobre a utilização de recursos está implementada somente para o sistema operacional Linux. Uma alternativa a implementar a coleta pra cada um dos outros sistemas operacionais suportados pelo InteGrade seria passar a utilizar o sistema Ganglia. O Ganglia [4, 19] é um sistema de monitoramento distribuído utilizado em grades computacionais e *clusters* pelo mundo. Sua implementação é bastante robusta e sobrecarrega muito pouco os nós onde é instalado, permitindo a obtenção de informações do sistema de forma eficiente. Além disso, o Ganglia já suporta uma grande variedade de sistemas operacionais, como Windows, Linux, HP-UX, Solaris, FreeBSD etc.

Outras limitações na coleta de dados são:

- são coletadas informações apenas sobre a utilização de CPU e memória. Seria interessante coletar também: memória *swap*, uso do disco rígido, banda de rede disponível, entre outras;

- o mecanismo de coleta não ignora a CPU e memória utilizadas por aplicações da grade. Isso pode distorcer os registros de monitoramento e causar impacto no cálculo dos padrões.

A falha descrita na Seção 5.2.2, que ocasiona o vazamento de memória no cálculo de padrões, compromete seriamente a confiabilidade desta implementação. Enquanto não for corrigida, o LUPA poderá manter alocada uma quantidade inconcebível de memória após algumas semanas executando.

Neste trabalho não foi implementado um algoritmo de escalonamento no InteGrade. No entanto, a implementação atual do InteGrade já utilizava o algoritmo `can run`, mas a implementação anterior do LUPA era muito simples e estava desabilitada, o que tornava o escalonamento equivalente ao `round robin`. Reabilitamos o LUPA, e agora, nas versões posteriores à 0.4, o escalonamento será feito com o algoritmo `can run`. Em nossos experimentos, no entanto, concluímos que os melhores resultados foram obtidos pelo `get prediction`. Dessa forma, recomendamos que este algoritmo seja utilizado

em um próximo trabalho de implementação, que envolva escalonamento no InteGrade.

Apêndice A

Resultado dos Ensaios de testes

Nesta seção, listamos uma série de resultados de ensaios de testes que foram relevantes para a análise dos experimentos realizados. Esses ensaios foram utilizados para a análise dos efeitos das alterações nos diferentes parâmetros dos ensaios de testes.

1. Ensaio: $n = 2$, $h = 4$, $m = 2$ e $d = 0.8$. Testes: 515.

	`can run`	`get prediction`	`last 4h`
`round robin`	75.1 - 21.0 - 3.9	86.2 - 10.9 - 2.9	79.6 - 10.7 - 9.7
`can run`		46.6 - 50.7 - 2.7	49.1 - 36.9 - 14.0
`get prediction`			10.1 - 75.0 - 15.0

	`can run`	`get prediction`	`last 4h`
média de ganho	0.25	0.29	0.29
média de perda	-0.10	-0.19	-0.21

2. Ensaio: $n = 2$, $h = 4$, $m = 3$ e $d = 0.8$. Testes: 507.

	`can run`	`get prediction`	`last 4h`
`round robin`	77.7 - 18.7 - 3.6	88.8 - 9.7 - 1.6	80.7 - 9.5 - 9.9
`can run`		51.9 - 46.5 - 1.6	50.5 - 36.7 - 12.8
`get prediction`			8.3 - 76.1 - 15.6

	can run	get prediction	last 4h
média de ganho	0.24	0.29	0.29
média de perda	-0.10	-0.15	-0.21

3. Ensaio: $n = 2$, $h = 4$, $m = 4$ e $d = 0.8$. Testes: 433.

	can run	get prediction	last 4h
round robin	88.0 - 9.2 - 2.8	93.1 - 5.8 - 1.2	83.1 - 5.5 - 11.3
can run		60.3 - 36.7 - 3.0	58.0 - 27.3 - 14.8
get prediction			10.2 - 71.8 - 18.0

	can run	get prediction	last 4h
média de ganho	0.24	0.29	0.29
média de perda	-0.08	-0.13	-0.22

4. Ensaio: $n = 2$, $h = 4$, $m = 5$ e $d = 0.8$. Testes: 369.

	can run	get prediction	last 4h
round robin	90.5 - 9.5 - 0.0	93.0 - 6.8 - 0.3	80.2 - 6.5 - 13.3
can run		71.8 - 26.3 - 1.9	68.0 - 15.7 - 16.3
get prediction			7.0 - 72.4 - 20.6

	can run	get prediction	last 4h
média de ganho	0.24	0.30	0.29
média de perda	nan	-0.07	-0.22

5. Ensaio: $n = 2$, $h = 4$, $m = 6$ e $d = 0.8$. Testes: 133.

	can run	get prediction	last 4h
round robin	91.7 - 8.3 - 0.0	98.5 - 0.0 - 1.5	81.2 - 0.0 - 18.8
can run		83.5 - 14.3 - 2.3	66.2 - 7.5 - 26.3
get prediction			11.3 - 51.9 - 36.8

	can run	get prediction	last 4h
média de ganho	0.21	0.29	0.25
média de perda	nan	-0.12	-0.09

6. Ensaio: $n = 2$, $h = 2$, $m = 4$ e $d = 0.7$. Testes: 150.

	can run	get prediction	last 4h
round robin	82.7 - 17.3 - 0.0	83.3 - 16.0 - 0.7	71.3 - 16.0 - 12.7
can run		62.7 - 36.0 - 1.3	55.3 - 24.0 - 20.7
get prediction			5.3 - 71.3 - 23.3

	can run	get prediction	last 4h
média de ganho	0.33	0.41	0.39
média de perda	nan	-0.32	-0.08

7. Ensaio: $n = 2$, $h = 4$, $m = 4$ e $d = 0.7$. Testes: 146.

	can run	get prediction	last 4h
round robin	80.8 - 19.2 - 0.0	82.2 - 16.4 - 1.4	69.9 - 16.4 - 13.7
can run		54.8 - 39.7 - 5.5	47.9 - 31.5 - 20.5
get prediction			8.2 - 67.8 - 24.0

	can run	get prediction	last 4h
média de ganho	0.34	0.40	0.39
média de perda	nan	-0.16	-0.08

8. Ensaio: $n = 2$, $h = 12$, $m = 4$ e $d = 0.7$. Testes: 131.

	can run	get prediction	last 4h
round robin	81.7 - 18.3 - 0.0	81.7 - 18.3 - 0.0	67.2 - 18.3 - 14.5
can run		64.9 - 33.6 - 1.5	55.7 - 24.4 - 19.8
get prediction			6.1 - 68.7 - 25.2

	can run	get prediction	last 4h
média de ganho	0.34	0.40	0.40
média de perda	nan	nan	-0.10

9. Ensaio: $n = 2$, $h = 24$, $m = 4$ e $d = 0.7$. Testes: 117.

	can run	get prediction	last 4h
round robin	79.5 - 20.5 - 0.0	78.6 - 20.5 - 0.9	69.2 - 20.5 - 10.3
can run		58.1 - 36.8 - 5.1	60.7 - 28.2 - 11.1
get prediction			13.7 - 70.9 - 15.4

	can run	get prediction	last 4h
média de ganho	0.34	0.40	0.41
média de perda	nan	-0.17	-0.10

10. Ensaio: $n = 2$, $h = 2$, $m = 4$ e $d = 0.8$. Testes: 403.

	can run	get prediction	last 4h
round robin	87.8 - 8.7 - 3.5	93.1 - 6.0 - 1.0	81.6 - 6.2 - 12.2
can run		59.3 - 39.0 - 1.7	56.3 - 27.0 - 16.6
get prediction			6.2 - 74.4 - 19.4

	can run	get prediction	last 4h
média de ganho	0.25	0.30	0.29
média de perda	-0.08	-0.16	-0.22

11. Ensaio: $n = 2$, $h = 4$, $m = 4$ e $d = 0.8$. Testes: 433.

	can run	get prediction	last 4h
round robin	88.5 - 9.0 - 2.5	93.3 - 5.8 - 0.9	83.1 - 5.5 - 11.3
can run		58.0 - 37.4 - 4.6	58.2 - 27.0 - 14.8
get prediction			11.3 - 70.9 - 17.8

	can run	get prediction	last 4h
média de ganho	0.24	0.29	0.29
média de perda	-0.08	-0.17	-0.22

12. Ensaio: $n = 2$, $h = 12$, $m = 4$ e $d = 0.8$. Testes: 392.

	can run	get prediction	last 4h
round robin	89.0 - 7.7 - 3.3	92.3 - 6.1 - 1.5	81.1 - 6.1 - 12.8
can run		65.3 - 30.9 - 3.8	64.8 - 20.7 - 14.5
get prediction			8.9 - 70.9 - 20.2

	can run	get prediction	last 4h
média de ganho	0.23	0.29	0.29
média de perda	-0.12	-0.18	-0.22

13. Ensaio: $n = 2$, $h = 24$, $m = 4$ e $d = 0.8$. Testes: 398.

	can run	get prediction	last 4h
round robin	87.9 - 8.8 - 3.3	92.2 - 6.5 - 1.3	81.4 - 6.8 - 11.8
can run		66.8 - 27.1 - 6.0	65.6 - 19.1 - 15.3
get prediction			5.0 - 71.6 - 23.4

	can run	get prediction	last 4h
média de ganho	0.22	0.28	0.27
média de perda	-0.08	-0.17	-0.22

14. Ensaio: $n = 2$, $h = 2$, $m = 4$ e $d = 0.9$. Testes: 905.

	can run	get prediction	last 4h
round robin	88.5 - 8.7 - 2.8	93.6 - 5.5 - 0.9	89.1 - 5.4 - 5.5
can run		60.2 - 38.1 - 1.7	58.3 - 33.1 - 8.5
get prediction			5.9 - 81.9 - 12.3

	can run	get prediction	last 4h
média de ganho	0.18	0.23	0.22
média de perda	-0.07	-0.14	-0.21

15. Ensaio: $n = 2$, $h = 4$, $m = 4$ e $d = 0.9$. Testes: 884.

	can run	get prediction	last 4h
round robin	88.6 - 9.2 - 2.3	93.1 - 5.8 - 1.1	88.3 - 5.5 - 6.1
can run		60.5 - 37.8 - 1.7	59.0 - 32.4 - 8.6
get prediction			8.0 - 79.1 - 12.9

	can run	get prediction	last 4h
média de ganho	0.18	0.23	0.22
média de perda	-0.11	-0.16	-0.21

16. Ensaio: $n = 2$, $h = 12$, $m = 4$ e $d = 0.9$. Testes: 1078.

	can run	get prediction	last 4h
round robin	82.3 - 14.8 - 2.9	88.1 - 4.7 - 7.1	88.5 - 4.5 - 7.0
can run		55.4 - 34.4 - 10.2	59.1 - 32.8 - 8.1
get prediction			15.1 - 70.0 - 14.8

	`can run`	`get prediction`	`last 4h`
média de ganho	0.17	0.21	0.20
média de perda	-0.06	-0.26	-0.22

17. Ensaio: $n = 2$, $h = 24$, $m = 4$ e $d = 0.9$. Testes: 1152.

	`can run`	`get prediction`	`last 4h`
`round robin`	76.6 - 20.0 - 3.5	88.8 - 5.2 - 6.0	88.8 - 4.9 - 6.2
`can run`		58.1 - 33.2 - 8.7	60.2 - 31.9 - 7.8
`get prediction`			10.3 - 78.1 - 11.5

	`can run`	`get prediction`	`last 4h`
média de ganho	0.16	0.19	0.19
média de perda	-0.07	-0.24	-0.18

18. Ensaio: $n = 4$, $h = 4$, $m = 4$ e $d = 0.7$. Testes: 146.

	`can run`	`get prediction`	`last 4h`
`round robin`	11.6 - 88.4 - 0.0	83.6 - 16.4 - 0.0	82.9 - 16.4 - 0.7
`can run`		4.8 - 95.2 - 0.0	6.2 - 93.2 - 0.7
`get prediction`			15.1 - 81.5 - 3.4

	`can run`	`get prediction`	`last 4h`
média de ganho	0.12	0.15	0.18
média de perda	nan	nan	-0.03

19. Ensaio: $n = 4$, $h = 4$, $m = 4$ e $d = 0.8$. Testes: 416.

	`can run`	`get prediction`	`last 4h`
`round robin`	38.7 - 61.3 - 0.0	73.1 - 26.9 - 0.0	72.8 - 26.9 - 0.2
`can run`		10.8 - 89.2 - 0.0	14.9 - 84.9 - 0.2
`get prediction`			9.9 - 85.8 - 4.3

	`can run`	`get prediction`	`last 4h`
média de ganho	0.17	0.18	0.19
média de perda	nan	nan	-0.03

20. Ensaio: $n = 4$, $h = 4$, $m = 4$ e $d = 0.9$. Testes: 884.

	can run	get prediction	last 4h
round robin	64.0 - 35.4 - 0.6	83.9 - 16.1 - 0.0	83.5 - 16.1 - 0.5
can run		33.8 - 65.7 - 0.5	37.6 - 61.4 - 1.0
get prediction			8.7 - 86.4 - 4.9

	can run	get prediction	last 4h
média de ganho	0.14	0.16	0.17
média de perda	-0.05	nan	-0.04

21. Ensaio: $n = 4$, $h = 4$, $m = 6$ e $d = 0.7$. Testes: 48.

	can run	get prediction	last 4h
round robin	22.9 - 77.1 - 0.0	100.0 - 0.0 - 0.0	97.9 - 0.0 - 2.1
can run		12.5 - 87.5 - 0.0	16.7 - 81.2 - 2.1
get prediction			45.8 - 41.7 - 12.5

	can run	get prediction	last 4h
média de ganho	0.10	0.16	0.23
média de perda	nan	nan	-0.03

22. Ensaio: $n = 4$, $h = 4$, $m = 6$ e $d = 0.8$. Testes: 116.

	can run	get prediction	last 4h
round robin	47.4 - 52.6 - 0.0	100.0 - 0.0 - 0.0	99.1 - 0.0 - 0.9
can run		38.8 - 61.2 - 0.0	53.4 - 45.7 - 0.9
get prediction			36.2 - 47.4 - 16.4

	can run	get prediction	last 4h
média de ganho	0.14	0.18	0.21
média de perda	nan	nan	-0.03

23. Ensaio: $n = 4$, $h = 4$, $m = 6$ e $d = 0.9$. Testes: 555.

	can run	get prediction	last 4h
round robin	79.6 - 19.5 - 0.9	95.7 - 4.3 - 0.0	95.0 - 4.3 - 0.7
can run		53.0 - 45.9 - 1.1	58.7 - 39.8 - 1.4
get prediction			14.2 - 78.2 - 7.6

	can run	get prediction	last 4h
média de ganho	0.12	0.16	0.17
média de perda	-0.08	nan	-0.04

24. Ensaio: $n = 6$, $h = 4$, $m = 6$ e $d = 0.8$. Testes: 116.

	can run	get prediction	last 4h
round robin	12.9 - 87.1 - 0.0	16.4 - 82.8 - 0.9	20.7 - 79.3 - 0.0
can run		12.9 - 86.2 - 0.9	18.1 - 81.9 - 0.0
get prediction			8.6 - 88.8 - 2.6

	can run	get prediction	last 4h
média de ganho	0.13	0.20	0.19
média de perda	nan	-0.03	nan

25. Ensaio: $n = 6$, $h = 4$, $m = 7$ e $d = 0.8$. Testes: 48.

	can run	get prediction	last 4h
round robin	31.2 - 68.8 - 0.0	41.7 - 58.3 - 0.0	50.0 - 50.0 - 0.0
can run		37.5 - 62.5 - 0.0	41.7 - 58.3 - 0.0
get prediction			12.5 - 79.2 - 8.3

	can run	get prediction	last 4h
média de ganho	0.14	0.21	0.19
média de perda	nan	nan	nan

26. Ensaio: $n = 6$, $h = 4$, $m = 8$ e $d = 0.8$. Testes: 19.

	can run	get prediction	last 4h
round robin	84.2 - 15.8 - 0.0	100.0 - 0.0 - 0.0	100.0 - 0.0 - 0.0
can run		89.5 - 10.5 - 0.0	89.5 - 10.5 - 0.0
get prediction			5.3 - 73.7 - 21.1

	can run	get prediction	last 4h
média de ganho	0.12	0.22	0.20
média de perda	nan	nan	nan

27. Ensaio: $n = 6$, $h = 4$, $m = 9$ e $d = 0.8$. Testes: 19.

	`can run`	`get prediction`	`last 4h`
`round robin`	84.2 - 15.8 - 0.0	100.0 - 0.0 - 0.0	100.0 - 0.0 - 0.0
`can run`		84.2 - 15.8 - 0.0	89.5 - 10.5 - 0.0
`get prediction`			21.1 - 57.9 - 21.1

	`can run`	`get prediction`	`last 4h`
média de ganho	0.13	0.21	0.20
média de perda	nan	nan	nan

28. Ensaio: $n = 3$, $h = 6$, $m = 3$ e $d = 0.6$. Testes: 125.

	`can run`	`get prediction`	`last 4h`
`round robin`	56.8 - 43.2 - 0.0	72.0 - 28.0 - 0.0	72.0 - 28.0 - 0.0
`can run`		4.0 - 96.0 - 0.0	4.8 - 84.8 - 10.4
`get prediction`			0.8 - 86.4 - 12.8

	`can run`	`get prediction`	`last 4h`
média de ganho	0.36	0.38	0.33
média de perda	nan	nan	nan

29. Ensaio: $n = 3$, $h = 6$, $m = 4$ e $d = 0.6$. Testes: 103.

	`can run`	`get prediction`	`last 4h`
`round robin`	83.5 - 16.5 - 0.0	87.4 - 12.6 - 0.0	87.4 - 12.6 - 0.0
`can run`		4.9 - 95.1 - 0.0	5.8 - 82.5 - 11.7
`get prediction`			1.0 - 83.5 - 15.5

	`can run`	`get prediction`	`last 4h`
média de ganho	0.36	0.38	0.33
média de perda	nan	nan	nan

30. Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.6$. Testes: 103.

	`can run`	`get prediction`	`last 4h`
`round robin`	72.8 - 27.2 - 0.0	87.4 - 12.6 - 0.0	87.4 - 12.6 - 0.0
`can run`		4.9 - 94.2 - 1.0	5.8 - 79.6 - 14.6
`get prediction`			1.0 - 83.5 - 15.5

	can run	get prediction	last 4h
média de ganho	0.36	0.38	0.33
média de perda	nan	nan	nan

31. Ensaio: $n = 3$, $h = 6$, $m = 6$ e $d = 0.6$. Testes: 18.

	can run	get prediction	last 4h
round robin	88.9 - 11.1 - 0.0	100.0 - 0.0 - 0.0	100.0 - 0.0 - 0.0
can run		0.0 - 100.0 - 0.0	0.0 - 22.2 - 77.8
get prediction			0.0 - 11.1 - 88.9

	can run	get prediction	last 4h
média de ganho	0.40	0.40	0.15
média de perda	nan	nan	nan

32. Ensaio: $n = 3$, $h = 6$, $m = 7$ e $d = 0.6$. Testes: 14.

	can run	get prediction	last 4h
round robin	85.7 - 14.3 - 0.0	100.0 - 0.0 - 0.0	100.0 - 0.0 - 0.0
can run		0.0 - 100.0 - 0.0	0.0 - 14.3 - 85.7
get prediction			0.0 - 0.0 - 100.0

	can run	get prediction	last 4h
média de ganho	0.41	0.42	0.10
média de perda	nan	nan	nan

33. Ensaio: $n = 3$, $h = 6$, $m = 3$ e $d = 0.7$. Testes: 219.

	can run	get prediction	last 4h
round robin	50.2 - 49.8 - 0.0	53.4 - 46.6 - 0.0	52.1 - 47.9 - 0.0
can run		5.9 - 94.1 - 0.0	7.3 - 83.1 - 9.6
get prediction			0.9 - 88.1 - 11.0

	can run	get prediction	last 4h
média de ganho	0.34	0.37	0.32
média de perda	nan	nan	nan

34. Ensaio: $n = 3$, $h = 6$, $m = 4$ e $d = 0.7$. Testes: 141.

	`can run`	`get prediction`	`last 4h`
`round robin`	61.7 - 38.3 - 0.0	83.0 - 17.0 - 0.0	80.9 - 19.1 - 0.0
`can run`		9.2 - 90.8 - 0.0	11.3 - 74.5 - 14.2
`get prediction`			1.4 - 81.6 - 17.0

	`can run`	`get prediction`	`last 4h`
média de ganho	0.33	0.37	0.32
média de perda	nan	nan	nan

35. Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.7$. Testes: 140.

	`can run`	`get prediction`	`last 4h`
`round robin`	67.9 - 32.1 - 0.0	82.9 - 17.1 - 0.0	81.4 - 18.6 - 0.0
`can run`		8.6 - 90.0 - 1.4	10.7 - 75.7 - 13.6
`get prediction`			1.4 - 82.9 - 15.7

	`can run`	`get prediction`	`last 4h`
média de ganho	0.33	0.37	0.32
média de perda	nan	nan	nan

36. Ensaio: $n = 3$, $h = 6$, $m = 6$ e $d = 0.7$. Testes: 43.

	`can run`	`get prediction`	`last 4h`
`round robin`	90.7 - 9.3 - 0.0	97.7 - 2.3 - 0.0	93.0 - 7.0 - 0.0
`can run`		14.0 - 83.7 - 2.3	18.6 - 34.9 - 46.5
`get prediction`			4.7 - 39.5 - 55.8

	`can run`	`get prediction`	`last 4h`
média de ganho	0.31	0.36	0.22
média de perda	nan	nan	nan

37. Ensaio: $n = 3$, $h = 6$, $m = 7$ e $d = 0.7$. Testes: 19.

	`can run`	`get prediction`	`last 4h`
`round robin`	84.2 - 15.8 - 0.0	100.0 - 0.0 - 0.0	84.2 - 15.8 - 0.0
`can run`		0.0 - 100.0 - 0.0	0.0 - 15.8 - 84.2
`get prediction`			0.0 - 0.0 - 100.0

	can run	get prediction	last 4h
média de ganho	0.37	0.38	0.09
média de perda	nan	nan	nan

38. Ensaio: $n = 3$, $h = 6$, $m = 3$ e $d = 0.8$. Testes: 499.

	can run	get prediction	last 4h
round robin	66.3 - 31.3 - 2.4	72.5 - 27.1 - 0.4	71.5 - 28.3 - 0.2
can run		32.9 - 65.1 - 2.0	37.7 - 56.5 - 5.8
get prediction			6.0 - 87.4 - 6.6

	can run	get prediction	last 4h
média de ganho	0.24	0.27	0.26
média de perda	-0.08	-0.08	-0.06

39. Ensaio: $n = 3$, $h = 6$, $m = 4$ e $d = 0.8$. Testes: 413.

	can run	get prediction	last 4h
round robin	78.7 - 18.6 - 2.7	87.2 - 11.6 - 1.2	86.4 - 13.3 - 0.2
can run		38.0 - 59.6 - 2.4	44.6 - 49.9 - 5.6
get prediction			8.0 - 84.3 - 7.7

	can run	get prediction	last 4h
média de ganho	0.24	0.27	0.26
média de perda	-0.08	-0.08	-0.06

40. Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.8$. Testes: 354.

	can run	get prediction	last 4h
round robin	84.2 - 15.8 - 0.0	86.4 - 13.6 - 0.0	84.2 - 15.5 - 0.3
can run		42.7 - 55.4 - 2.0	48.9 - 43.2 - 7.9
get prediction			7.6 - 82.2 - 10.2

	can run	get prediction	last 4h
média de ganho	0.24	0.28	0.27
média de perda	nan	nan	-0.06

41. Ensaio: $n = 3$, $h = 6$, $m = 6$ e $d = 0.8$. Testes: 124.

	can run	get prediction	last 4h
round robin	87.1 - 12.9 - 0.0	100.0 - 0.0 - 0.0	93.5 - 5.6 - 0.8
can run		56.5 - 43.5 - 0.0	66.9 - 13.7 - 19.4
get prediction			14.5 - 58.9 - 26.6

	can run	get prediction	last 4h
média de ganho	0.20	0.26	0.22
média de perda	nan	nan	-0.06

42. Ensaio: $n = 3$, $h = 6$, $m = 7$ e $d = 0.8$. Testes: 41.

	can run	get prediction	last 4h
round robin	85.4 - 14.6 - 0.0	100.0 - 0.0 - 0.0	80.5 - 17.1 - 2.4
can run		41.5 - 58.5 - 0.0	41.5 - 2.4 - 56.1
get prediction			2.4 - 31.7 - 65.9

	can run	get prediction	last 4h
média de ganho	0.27	0.30	0.15
média de perda	nan	nan	-0.06

43. Ensaio: $n = 3$, $h = 6$, $m = 3$ e $d = 0.9$. Testes: 978.

	can run	get prediction	last 4h
round robin	72.4 - 25.9 - 1.7	81.3 - 17.3 - 1.4	81.6 - 17.7 - 0.7
can run		44.9 - 52.7 - 2.5	47.6 - 48.8 - 3.6
get prediction			6.9 - 85.2 - 8.0

	can run	get prediction	last 4h
média de ganho	0.17	0.21	0.21
média de perda	-0.06	-0.07	-0.06

44. Ensaio: $n = 3$, $h = 6$, $m = 4$ e $d = 0.9$. Testes: 885.

	can run	get prediction	last 4h
round robin	81.6 - 17.3 - 1.1	90.5 - 8.9 - 0.6	90.1 - 9.2 - 0.8
can run		47.8 - 49.7 - 2.5	51.2 - 44.0 - 4.9
get prediction			7.0 - 84.7 - 8.2

	can run	get prediction	last 4h
média de ganho	0.17	0.21	0.21
média de perda	-0.05	-0.07	-0.06

45. Ensaio: $n = 3$, $h = 6$, $m = 5$ e $d = 0.9$. Testes: 797.

	can run	get prediction	last 4h
round robin	84.6 - 14.8 - 0.6	90.8 - 9.2 - 0.0	89.6 - 10.0 - 0.4
can run		53.8 - 44.4 - 1.8	57.3 - 38.5 - 4.1
get prediction			7.0 - 83.9 - 9.0

	can run	get prediction	last 4h
média de ganho	0.16	0.21	0.20
média de perda	-0.05	nan	-0.07

46. Ensaio: $n = 3$, $h = 6$, $m = 6$ e $d = 0.9$. Testes: 543.

	can run	get prediction	last 4h
round robin	89.5 - 9.9 - 0.6	94.5 - 5.5 - 0.0	93.6 - 5.9 - 0.6
can run		64.3 - 34.3 - 1.5	67.4 - 27.1 - 5.5
get prediction			8.8 - 78.8 - 12.3

	can run	get prediction	last 4h
média de ganho	0.14	0.18	0.17
média de perda	-0.02	nan	-0.07

47. Ensaio: $n = 3$, $h = 6$, $m = 7$ e $d = 0.9$. Testes: 372.

	can run	get prediction	last 4h
round robin	93.3 - 5.6 - 1.1	99.2 - 0.8 - 0.0	97.0 - 2.2 - 0.8
can run		78.2 - 19.9 - 1.9	78.5 - 13.2 - 8.3
get prediction			6.5 - 77.4 - 16.1

	can run	get prediction	last 4h
média de ganho	0.12	0.17	0.16
média de perda	-0.05	nan	-0.07

48. Ensaio: $n = 3$, $h = 6$, $m = 8$ e $d = 0.9$. Testes: 270.

	can run	get prediction	last 4h
round robin	96.7 - 3.3 - 0.0	98.5 - 1.5 - 0.0	99.3 - 0.0 - 0.7
can run		83.0 - 12.6 - 4.4	86.7 - 12.2 - 1.1
get prediction			9.6 - 80.4 - 10.0

	can run	get prediction	last 4h
média de ganho	0.11	0.16	0.16
média de perda	nan	nan	-0.08

49. Ensaio: $n = 3$, $h = 6$, $m = 9$ e $d = 0.9$. Testes: 265.

	can run	get prediction	last 4h
round robin	97.0 - 3.0 - 0.0	98.9 - 1.1 - 0.0	99.2 - 0.0 - 0.8
can run		84.5 - 12.8 - 2.6	87.9 - 10.9 - 1.1
get prediction			9.8 - 80.0 - 10.2

	can run	get prediction	last 4h
média de ganho	0.11	0.16	0.16
média de perda	nan	nan	-0.08

50. Ensaio: $n = 2$, $h = 4$, $m = 2$ e $d = 0.98$. Testes: 1928.

	can run	get prediction	last 4h
round robin	53.6 - 42.9 - 3.5	78.3 - 17.8 - 3.9	77.6 - 17.2 - 5.1
can run		51.5 - 45.3 - 3.3	53.7 - 39.1 - 7.2
get prediction			10.5 - 78.5 - 11.0

	can run	get prediction	last 4h
média de ganho	0.15	0.16	0.16
média de perda	-0.06	-0.08	-0.14

51. Ensaio: $n = 3$, $h = 4$, $m = 3$ e $d = 0.98$. Testes: 1795.

	can run	get prediction	last 4h
round robin	51.5 - 46.6 - 1.9	76.0 - 22.0 - 1.9	75.6 - 22.6 - 1.8
can run		47.6 - 50.3 - 2.2	49.0 - 46.6 - 4.5
get prediction			8.6 - 81.2 - 10.2

	can run	get prediction	last 4h
média de ganho	0.15	0.15	0.15
média de perda	-0.05	-0.05	-0.05

52. Ensaio: $n = 4$, $h = 4$, $m = 4$ e $d = 0.98$. Testes: 1670.

	can run	get prediction	last 4h
round robin	43.1 - 56.2 - 0.7	69.5 - 29.7 - 0.8	68.9 - 29.8 - 1.3
can run		36.0 - 63.2 - 0.8	38.3 - 59.4 - 2.3
get prediction			8.0 - 85.2 - 6.8

	can run	get prediction	last 4h
média de ganho	0.12	0.13	0.13
média de perda	-0.04	-0.04	-0.04

53. Ensaio: $n = 5$, $h = 4$, $m = 5$ e $d = 0.98$. Testes: 1451.

	can run	get prediction	last 4h
round robin	34.3 - 65.3 - 0.4	54.2 - 42.4 - 3.4	58.2 - 41.4 - 0.4
can run		33.4 - 66.0 - 0.6	34.2 - 64.2 - 1.6
get prediction			8.0 - 87.0 - 5.0

	can run	get prediction	last 4h
média de ganho	0.10	0.12	0.12
média de perda	-0.04	-0.07	-0.03

54. Ensaio: $n = 5$, $h = 4$, $m = 5$ e $d = 0.75$. Testes: 178.

	can run	get prediction	last 4h
round robin	5.1 - 94.9 - 0.0	22.5 - 69.7 - 7.9	33.1 - 66.9 - 0.0
can run		3.9 - 96.1 - 0.0	3.9 - 94.4 - 1.7
get prediction			19.7 - 78.7 - 1.7

	can run	get prediction	last 4h
média de ganho	0.10	0.11	0.16
média de perda	nan	-0.04	nan

Referências Bibliográficas

[1] Cppunit wiki. http://cppunit.sourceforge.net/cppunit-wiki. Acessado em Janeiro/2008.

[2] Integrade. http://www.integrade.org.br. Acessado em Janeiro/2008.

[3] Network weather service. http://nws.cs.ucsb.edu/ewiki/. Acessado em Janeiro/2008.

[4] Projeto ganglia. http://ganglia.sourceforge.net/. Acessado em Janeiro/2008.

[5] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press Inc., 1973.

[6] David P. Anderson. Boinc: A system for public-resource computing and storage.

[7] Fran Berman, Geofrey Fox, Anthony J. G. Hey, and Tony Hey. *Grid Computing: Making the Global Infrastructure a Reality.* John Wiley & Sons, Inc., 2003.

[8] Germano Capistrano Bezerra. Análise de conglomerados aplicada ao reconhecimento de padrões de uso de recursos computacionais. Master's thesis, IME-USP, 2006.

[9] W. O. Bussab. Introdução à análise de agrupamentos. 1990. 9o. Simpósio Nacional de Probabilidade e Estatística.

[10] Mark Silberstein Dan. Scheduling mixed workloads in multi-grids: The grid execution hierarchy.

[11] Rafael Y. de Camargo. *Armazenamento distribuído de dados e checkpointing de aplicações paralelas em grades oportunistas.* PhD thesis, IME-USP, 2007.

[12] Raphael Y. de Camargo, Andrei Goldchleger, Marcio Carneiro, and Fabio Kon. Grid: An Architectural Pattern. In *The 11th Conference on Pattern Languages of Programs (PLoP'2004)*, Monticello, Illinois, September 2004.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., 1995.

[14] A. Goldchleger. Integrade: Um sistema de middleware para computação em grade oportunista. Master's thesis, IME-USP, 2004.

[15] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459, March 2004.

[16] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[17] Nguyen The Loc, Said Elnaffar, Takuya Katayama, and Ho Tu Bao. Grid scheduling using 2-phase prediction (2pp) of cpu power. *Innovations in Information Technology*, 2006.

[18] G. F. Luger. *Artificial Intelligence: structures and strategies for complex problem solving*. Pearson Education Limited, 2002.

[19] Mattew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(5):817–840, 2004.

[20] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[21] N. J. Nilsson. *Introduction to Machine Learning*. 1996. An early draft of a proposed textbook.

[22] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[23] Jennifer M. Schopf. A general architecture for scheduling on the grid.

[24] D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini, and G.R. Nudd. Local grid scheduling techniques using performance prediction. *Computers and Digital Techniques, IEE Proceedings*, 2003.

[25] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[26] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geofrey Fox, Anthony J. G. Hey, and Tony Hey, editors, *Grid Computing — Making the Global Infrastructure a Reality*. John Wiley & Sons, Ltd, 2003.

[27] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[28] D. Wright. Cheap cycles from the desktop to the dedicated cluster: Combining opportunistic and dedicated scheduling with condor, 2001.

[29] Lingyum Yang, Jennifer M. Schopf, and Ian Foster. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. *SuperComputing 2003*, Novembro 2003.