

PREDIÇÃO DE TEMPO DE EXECUÇÃO DE
TAREFAS EM GRADES COMPUTACIONAIS
PARA RECURSOS NÃO DEDICADOS

FLÁVIO VINÍCIUS DE ANDRADE

PREDIÇÃO DE TEMPO DE EXECUÇÃO DE
TAREFAS EM GRADES COMPUTACIONAIS
PARA RECURSOS NÃO DEDICADOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: JOSÉ MARCOS SILVA NOGUEIRA.
CO-ORIENTADORA: LILIAN NORONHA NASSIF.

Belo Horizonte

19 de novembro de 2008

© 2008, Flávio Vinícius de Andrade.
Todos os direitos reservados.

A553p Andrade, Flávio Vinícius de
Predição de tempo de execução de tarefas em grades
computacionais para recursos não dedicados / Flávio
Vinícius de Andrade. — Belo Horizonte, 2008
xx, 87 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de
Minas Gerais

Orientador: José Marcos Silva Nogueira.

Co-orientadora: Lilian Noronha Nassif.

1. Computação em grade. 2. Predição.
3. Computação - Teses. I. Título.

CDU 519.6*24



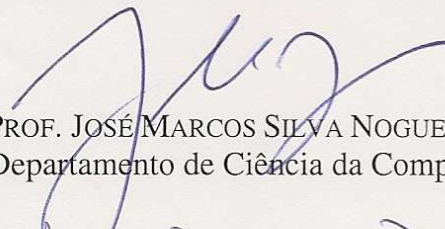
UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

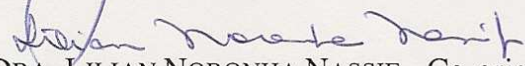
FOLHA DE APROVAÇÃO

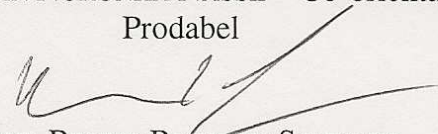
Predição de Tempo de Execução de Tarefas em
Grades Computacionais para Recursos Não Dedicados

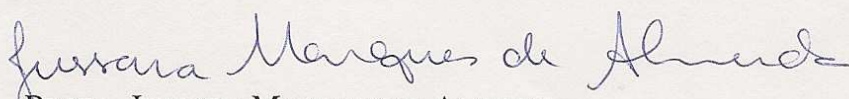
FLAVIO VINICIUS DE ANDRADE

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROF. JOSÉ MARCOS SILVA NOGUEIRA - Orientador
Departamento de Ciência da Computação - UFMG


DRA. LILIAN NORONHA NASSIF - Co-orientadora
Prodabel


PROF. BRUNO RICHARD SCHULZE
Laboratório Nacional de Computação Científica - LNCC


PROFA. JUSSARA MARQUES DE ALMEIDA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 19 de novembro de 2008.

Dedico esta dissertação aos meus pais Ágida e Maurício e à minha noiva Renata.

Agradecimentos

Agradeço ao professor José Marcos Nogueira e a pesquisadora Lilian Noronha Nassif pela suas orientações para o desenvolvimento deste trabalho, e também durante a minha Iniciação Científica. Sou grato também pelo projeto, trabalhos e conferência que participei desde a época da graduação. Agradeço pela correção dos trabalhos, os ensinamentos e a atenção recebida.

Agradeço aos colegas do laboratório de redes ATM, à turma do mestrado, aos demais colegas do DCC, pelas dicas, incentivo e trocas de experiências. Agradeço também aos bolsistas João, Danilo e Carlos pela participação no projeto Marenegri que teve como um dos resultados esta dissertação. Agradeço ao DCC/UFMG, seus professores e funcionários, pela contribuição para a minha formação, e pela oportunidade de fazer o mestrado. Agradeço aos organizadores do CCGRID 2007 pelo incentivo e contribuição financeira para participação da conferência que foi de grande importância para direcionar este trabalho.

Agradeço a empresa Syst, em especial à minha gerente Janaína Dias, pelos vários dias que saí mais cedo e fui dispensado para o desenvolvimento deste trabalho, e aos demais colegas da empresa. E agradeço a FAPEMIG pelo apoio financeiro que possibilitou a realização deste trabalho.

Agradeço à minha noiva Renata, que sempre me apoiou e não me deixou desanimar, e também ao apoio da minha cunhada Patrícia. E também agradeço meus pais Águida e Maurício pelo incentivo e por sempre me apoiarem incondicionalmente.

Agradeço a Deus, por ter me dado saúde, determinação, persistência e por ter conhecido todas essas pessoas que foram fundamentais para o desenvolvimento deste trabalho.

Resumo

A predição do tempo de execução de *jobs* para recursos de grades computacionais (*grid*) é um dado importante para escalonadores e *brokers*. As grades computacionais, por configurarem ambientes compostos por diversos e diferentes recursos e usuários, com diferentes tipos de *jobs*, não seguindo um padrão para sua submissão, tornam a predição do tempo de execução de *jobs* um grande desafio. Neste trabalho abordamos o problema de predição do tempo de execução de *jobs* para recursos não dedicados. Para tratar deste problema baseamo-nos em uma metodologia proposta no trabalho de tese de doutorado de Lílian Noronha Nassif, chamada PredCase, que se mostrou eficiente para predição do tempo para recursos dedicados. Em nosso método, chamado NdrPredCase, utilizamos casos passados de execução de *jobs* para calcular a predição do tempo de execução para um novo *job*, como é feito no PredCase. A partir do paradigma de Raciocínio Baseado em Casos (RBC) desenvolvemos as fases do NdrPredCase: recuperação dos casos similares ao novo *job*, reuso dos casos recuperados para gerar uma solução inicial, adaptação da solução para a carga de trabalho prevista e armazenamento das informações de descrição do *job*, do cálculo da predição e de sua execução. A maior contribuição deste trabalho é a adaptação da solução inicial obtida a partir dos casos passados, utilizando a relação entre a carga de trabalho passada com a carga de trabalho prevista para a execução do *job*. Os resultados dos experimentos realizados mostraram que o NdrPredCase tem uma boa acurácia para o cálculo de predição de *jobs* para recursos não dedicados quando temos um número de casos passados suficientemente grande, e é eficiente quanto ao desempenho para calcular a predição.

Palavras-chave: grade computacional, predição do tempo de execução, raciocínio baseado em casos, recursos não dedicados.

Abstract

Runtime prediction of jobs to grid resource is an important data to schedulers and brokers. Grids configure environment composed by distinct and several resources and users, with different kind of jobs, do not have a pattern to submission of job, and hence runtime prediction of job is a large challenge. This work explore the problem of runtime prediction of jobs to non-dedicated resources. We treat this problem based in a methodology proposal in Ph.D. thesis of Lilian Noronha Nassif, named PredCase, it showed a efficient methodology for dedicated resources. In our method, named NdrPredCase, we utilize past cases of runtime jobs to calculate the runtime prediction to a new job, how done in PredCase. We utilized the paradigm Case-Based Reasoning to develop the NdrPredCase phases: retrieval cases alike to new job, reuse this cases to calculate the initial solution, adaptation of solution to a forecast workload and retain information about job description, solution description and running data. The main contribution of this work is the adaptation of a initial solution, it calculated with past cases, using a relation of previous workload and forecast workload to running job. Results of experiments demonstrated that NdrPredCase has a good accuracy to calculate runtime prediction to non-dedicated resources if the amount of past cases is sufficient, and that NdrPredCase has a good performance to calculate this prediction.

Keywords: grid computing, runtime prediction, case-based reasoning, non-dedicated resources.

Lista de Figuras

2.1	No método <i>Nearest-neighbor</i> é escolhido os valores mais próximos do query point . Fonte: [Chris Atkeson 1997].	11
2.2	Algoritmo <i>Weighted average</i> . Fonte: [Chris Atkeson 1997].	12
2.3	Algoritmo <i>Locally weighted regression</i> . Fonte: [Chris Atkeson 1997].	12
2.4	Ciclo do RBC. Fonte: [Aamodt and Plaza 1994].	14
3.1	Classificação das técnicas de predição. Fonte: [Menascé et al. 1994].	22
3.2	Histogramas dos erros relativos para predição do tempo de execução. Fonte: [Li and Wolters 2006].	29
3.3	Histogramas dos erros relativos para predição do tempo de espera na fila. Fonte: [Li and Wolters 2006].	29
3.4	Estrutura de dados do PredCase. Fonte: [Nassif 2006].	31
3.5	Desempenho dos algoritmos <i>refined nearest-neighbor</i> e <i>nearest-neighbor</i> para diferentes tamanho de base de casos. Fonte: [Nassif 2006].	34
3.6	Percentual de erro médio da predição <i>versus</i> o grau de similaridade entre os casos. Fonte: [Nassif 2006].	35
4.1	Relação entre a carga de trabalho média e o tempo de execução. Fonte: [Dinda and O'Hallaron 2000].	48
5.1	Diagrama de fluxo de comunicação entre as entidades do GridSim. Fonte: [Buyya and Murshed 2002].	53
5.2	Modelo usado pelo GridSim para simular sistemas <i>time-shared</i> . Fonte: [Buyya and Murshed 2002].	54
5.3	Modelo usado pelo GridSim para simular sistemas <i>space-shared</i> . Fonte: [Buyya and Murshed 2002].	55
5.4	Arquitetura do MASK. Fonte: [Nassif 2006].	56
5.5	Exemplo do <i>log</i> utilizado para validação. Cada linha contém informações de um <i>job</i> que foi submetido.	60

5.6	Distribuição do tempo de execução dos <i>jobs</i> analisados para o cenário 10.	64
5.7	Distribuição da diferença entre o tempo previsto pelo PredCase e o tempo real da execução dos <i>jobs</i> analisados, sem <i>outliers</i> .	65
5.8	Distribuição da diferença entre o tempo previsto pelo NdrPredCase e o tempo real da execução dos <i>jobs</i> analisados, sem <i>outliers</i> .	65
5.9	Erro relativo do PredCase e do NdrPredCase.	66
5.10	Distribuição do tempo de execução dos <i>jobs</i> analisados para o cenário 100.	67
5.11	Distribuição da diferença entre o tempo previsto pelo PredCase e o tempo real da execução dos <i>jobs</i> analisados.	68
5.12	Distribuição da diferença entre o tempo previsto pelo NdrPredCase e o tempo real da execução dos <i>jobs</i> analisados.	68
5.13	Erro relativo do PredCase e do NdrPredCase.	69
5.14	Comportamento do erro absoluto normalizado (α) do PredCase.	70
5.15	Comportamento do erro absoluto normalizado (α) do NdrPredCase.	70
5.16	Distribuição da carga de trabalho média durante a execução dos <i>jobs</i> .	71
A.1	Utilização do DAS-2 em função do horário do dia, observada durante o ano de 2003. " <i>Average</i> " é a média da utilização em todos os dias no ano. " <i>Average*</i> " é a média da utilização para todos os dias ativos no ano, dos quais são excluídos o intervalo de tempo que o sistema ficou ocioso e os dias sem chegada de <i>jobs</i> . Fonte: [Li et al. 2004].	80
A.2	Ciclo diário da chegada de <i>jobs</i> durante os dias úteis da semana no DAS-2. Fonte: [Li et al. 2004].	80

Lista de Tabelas

3.1	Erro médio absoluto e normalizado do tempo de execução e do tempo de espera. Fonte: [Li and Wolters 2006].	29
5.1	Dados para criação dos <i>jobs</i> para a validação do NdrPredCase. Cada linha representa um comando para executar um <i>job</i> . O tempo de duração foi obtido executando os comandos na máquina <i>Ixion</i> . Fonte: [Nassif 2006]. . .	58
5.2	Clusters do DAS-2. Fonte: [DAS 2008].	59
5.3	Configurações de máquinas para simulação.	61
5.4	Predições calculadas pelos métodos PredCase e NdrPredCase. Para os <i>jobs</i> que executaram com uma média de carga de trabalho de até dez <i>jobs</i> concorrentes. $\bar{x}_{ e }$ representa a média dos erros absolutos com seu respectivo desvio padrão σ e α representa o erro absoluto normalizado.	66
5.5	Resultados do cálculo da predição pelos métodos PredCase e o NdrPredCase.	69

Sumário

Agradecimentos	ix
Resumo	xi
Abstract	xiii
Lista de Figuras	xv
Lista de Tabelas	xvii
1 Introdução	1
1.1 Motivação	3
1.2 Objetivos	6
1.3 Organização do documento	7
2 Técnicas de aprendizagem baseada em casos	9
2.1 Aprendizado baseado em instâncias	10
2.2 Raciocínio baseado em casos	13
2.2.1 Organização da base de casos	15
2.2.2 Recuperação de casos	17
2.2.3 Reúso da solução (adaptação)	18
2.2.4 Revisão e armazenamento (aprendizado)	18
2.3 Conclusão	19
3 Predição do tempo de execução de <i>jobs</i>	21
3.1 Técnicas para cálculo de predição de desempenho	21
3.2 Trabalhos de predição do tempo de execução de <i>jobs</i>	24
3.2.1 Predição baseada em histórico e categorias de casos	24
3.2.2 Predição baseada em IBL e algoritmo genético	26
3.2.3 Predição baseada em RBC: PredCase	30

3.3	Conclusão	35
4	Um método para predição de tempo de execução para recursos não dedicados	37
4.1	Organização dos casos	38
4.2	Recuperação de casos	40
4.3	Reúso da solução	44
4.3.1	<i>Raw prediction</i>	44
4.3.2	<i>Forecaster</i>	45
4.3.3	Predição adaptada	46
4.3.4	Predição ajustada	48
4.4	Armazenamento	49
4.5	Conclusão	50
5	Validação do NdrPredCase	51
5.1	Simulação	51
5.1.1	GridSim	51
5.1.2	Simulando o MASK no GridSim	55
5.1.3	Criação dos <i>jobs</i>	57
5.1.4	Cenário e parâmetros de simulação	60
5.1.5	Métricas para validação do NdrPredCase	62
5.2	Resultados e análise da predição adaptada	63
5.2.1	Análise do cenário com no máximo dez <i>jobs</i> concorrentes (cenário 10)	63
5.2.2	Análise do cenário com no máximo cem <i>jobs</i> concorrentes (cenário 100)	67
5.3	Conclusão	71
6	Conclusão	75
6.1	Trabalhos futuros	77
A	Um estudo de caso do <i>grid</i> DAS-2	79
	Referências Bibliográficas	83

Capítulo 1

Introdução

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

Ian Foster, 2004

Computação em grade, *grid computing* ou simplesmente *grid*, é uma tecnologia emergente que envolve o uso colaborativo de computadores, composta por ambientes geograficamente distribuídos e integrados, voltados para o fornecimento em larga escala de recursos, com o objetivo de fornecer alto desempenho na execução de aplicações [Foster et al. 2001]. O *grid* é de uso colaborativo porque depende da cooperação das várias entidades envolvidas que fornecem seus próprios recursos para poder usufruir dos recursos do *grid*. As entidades podem estar próximas ou muito distantes geograficamente. A participação de uma entidade em um *grid* vem da necessidade de uma capacidade computacional que ultrapassa aqueles instalados em tal entidade. O principal objetivo de um *grid* é prover um melhor desempenho e/ou uma melhor capacidade de armazenamento, o que dependerá principalmente da colaboração e dos objetivos comuns das entidades participantes. Dessa forma, o enfoque do *grid* é a resolução destes problemas de compartilhamento de recursos em organizações multi-institucionais, visando atender aos seus objetivos e ao compartilhamento de poder computacional e de armazenamento [Nassif 2006]. O resultado desta integração submetido a políticas e regras individuais e gerais é um grande sistema distribuído formando uma

organização virtual¹ [Irving et al. 2004].

De acordo com Foster [2002], um sistema *grid* é aquele que:

- coordena recursos que não estão sujeitos a um controle centralizado. Um *grid* integra e coordena recursos e usuários que participam de diferentes domínios de controle, por exemplo: usuários de universidades diferentes, companhias diversas, até mesmo de países diferentes, bem como diferentes assuntos, como de segurança, política, pagamento, entre outros;
- usa protocolos e *interfaces* padronizados, abertos e de propósito geral. Isto permite que um grande número de usuários compartilhem vários tipos de recursos computacionais como, por exemplo, armazenamento de dados, programas e poder de processamento, sendo este compartilhamento de recursos cuidadosamente controlado [Irving et al. 2004];
- fornece qualidade de serviços² não triviais. Um *grid* permite que seus recursos constituintes sejam usados de forma coordenada para fornecer diferentes qualidades de serviços relacionadas, por exemplo, com tempo de resposta, vazão de dados, utilidade e segurança.

Distinguem-se na literatura, de forma geral, duas principais áreas de atuação dentro de *grids*, o *data grid* e o *grid computing*. O *data grid* tem como objetivo principal o armazenamento e a partilha de grandes quantidades de dados. O *grid computing* tem como objetivo principal a computação distribuída de tarefas que demandam grande capacidade de processamento. Normalmente o termo *grid* é usado para referir-se ao *grid computing*, uma vez que esta é a área que tem estado mais ativa e na qual foram feitos os maiores avanços.

Segundo Irving et al. [2004], as principais características de um *grid* são:

- O *grid* é criado através da instalação de softwares de serviços, ou *middleware*, por exemplo o *Globus Toolkit* [Globus 2008], sobre uma infra-estrutura de conjuntos de redes de computadores. O *middleware* fornece recursos de *hardware* e *software*

¹Uma organização virtual (*Virtual Organization* - VO) é um conjunto de indivíduos e/ou instituições que compartilham recursos sob regras bem definidas [Nassif 2006].

²Qualidade de serviço, ou *quality of service* (QoS) representa um conjunto de características qualitativas e quantitativas necessárias para um sistema alcançar as funcionalidades requeridas para uma aplicação. O processamento de QoS num sistema distribuído começa com o estabelecimento dos parâmetros exigidos pelo usuário. Esses parâmetros são mapeados e negociados entre os componentes do sistema, assegurando que todos podem atingir um nível de QoS aceitável. Recursos são então alocados e monitorados, havendo possibilidade de renegociação caso as condições do sistema se alterem [Vogel et al. 1995].

para alocação de recursos, autenticação de usuários, escalonamento distribuído de recursos e tarefas, entre outros;

- Um *grid* é projetado para ser escalonável, ou seja, ele deve ser capaz de executar perfeitamente bem, tanto com poucos, quanto com muitos participantes;
- Um *grid* é freqüentemente observado como uma forma de se ter um supercomputador de preço acessível.

Como argumentamos, um *grid* possui natureza bastante complexa. Nele podemos ter diferentes configurações de máquinas integrando o sistema. Máquinas que podem ser desconectadas ou conectadas ao sistema a qualquer momento, que podem fazer parte de diferentes domínios sujeitas a diferentes políticas e que podem estar distantes geograficamente. Um dos fatores de menor controle é aquele relacionado com a utilização destes recursos pelos diferentes usuários, uma vez que é difícil traçar um perfil de utilização do ambiente por um período grande de tempo, tanto quanto ao número de *jobs* (tarefas), quanto aos tipos de aplicações submetidas. A partir de suas características podemos concluir que qualquer estudo de um ambiente *grid* é algo não trivial.

1.1 Motivação

Sendo os ambientes *grid* heterogêneos, dinâmicos, e com diversos recursos distribuídos geograficamente, a escolha de recursos e o escalonamento em tais ambientes são tarefas desafiadoras. As informações dinâmicas dos diversos recursos são cruciais para a fase de escalonamento [Li 2007]. Segundo Nassif et al. [2007b] e Li and Wolters [2006], a melhor escolha para executar um *job* num determinado recurso de um *grid* depende de diversas informações sobre tal recurso. Essas informações podem ser estáticas, como o tipo da máquina, o número de processadores, a velocidade de leitura e escrita no disco para computação de I/O intensivo, o tamanho da memória *cache* e a capacidade de armazenamento, que podem ser obtidas por monitorações e serviços de informações. Porém estas informações podem não ser suficientes para uma melhor decisão, por exemplo, de escalonamento de tarefas (*scheduling*). Caso se baseie somente nestas informações, a máquina de melhor poder computacional poderá ter sempre a preferência para a execução de *jobs*, o que não garante a melhor escolha, pois executando muitos *jobs* de forma concorrente, seu desempenho será menor do que o esperado. Conseqüentemente, a melhor opção em um dado momento poderia ser a de submeter um *job* a uma máquina com menor poder computacional, mas que esteja

ociosa. Logo, além das informações estáticas dos recursos, há também as informações dinâmicas, tais como o tempo de execução de um *job*, o tempo de espera na fila para iniciar sua execução e a carga de trabalho³ de um determinado recurso em um dado momento. Esse tipo de informação é muito importante para atingir a melhor escolha quando do escalonamento de tarefas, mas pode não ser obtida apenas pela monitoração do recurso. Em muitas situações pode ser necessário gerar previsões baseadas no histórico das execuções a partir dos dados armazenados nos recursos [Nabrzyski et al. 2003] *apud* [Li and Wolters 2006]. Por exemplo, a escolha do melhor recurso pode ser realizada utilizando essa previsão do tempo de execução do *job*, pela qual a menor previsão para um *job* em um dado recurso fará com que este seja escolhido.

O trabalho de Nassif [2006] foi desenvolvido com o objetivo de prover uma solução para o problema de seleção de recursos em *grids*. Seu principal interesse é possibilitar uma seleção direcionada ao atendimento das necessidades do usuário com relação a desempenho, política de acesso, garantia de serviço e custo. O trabalho foi desenvolvido utilizando agentes de software que têm como tarefa a seleção de um recurso com base nos acordos de nível de serviço SLA (*Service Level Agreements*)⁴ estabelecidos entre agentes que representam usuários e agentes que representam recursos. Esse trabalho resultou no desenvolvimento do *middleware* MASK (*Multi-Agent System broKer*).

O MASK é dividido em três módulos principais que correspondem às fases de seleção do melhor recurso. O primeiro módulo é responsável por fazer a previsão do tempo de execução de *jobs* de forma distribuída e utilizando a técnica de raciocínio baseado em casos (RBC). Esse módulo é chamado de PredCase, neste o cálculo da previsão é feito a partir da média de execuções passadas de *jobs*. Essa previsão média é então adaptada utilizando uma função que relaciona a carga de trabalho média durante a execução dos *jobs* passados e a carga de trabalho média recente detectada pelo módulo de monitoração, obtendo a previsão final. Dessa forma, o aprendizado do processo de cálculo da previsão depende da inclusão de novos casos e soluções na base de casos. O cálculo da previsão permite que o MASK dê garantia para o provimento do serviço e escalonamento de *jobs*.

O segundo módulo do MASK, chamado POLAR (*Policy-based Access Restriction*

³Foi considerada carga de trabalho a quantidade de *jobs* que estão executando de forma concorrente.

⁴Um SLA (*Service Level Agreements*) é um acordo formal negociado entre duas partes. É um contrato que existe entre consumidores e seus provedores de serviços. É registrado o comum entendimento entre as partes sobre os serviços, as prioridades, responsabilidades, garantias, ou seja, coletivamente o nível do serviço. Um SLA pode especificar por exemplo os níveis de disponibilidade, performance, utilidade, operação, ou outros atributos como critério de cobrança e até as penalidades caso haja uma violação do SLA. Um SLA deve conter parâmetros mensuráveis, os quais o provedor de serviços se compromete a atender [Lewis 1999].

Model), é responsável pela verificação de acesso para um usuário. Segundo Nassif et al. [2009], a restrição de acesso deve ser verificada de forma dinâmica e integrada com as políticas declaradas pelo proprietário do recurso, pois caso a verificação de acesso não seja feita, os *jobs* podem ser submetidos a recursos que não aceitem sua execução. Isto irá causar re-submissões do mesmo *job* para outros recursos que também poderão impedir a execução por uma ou outra política. Para atender a esta abordagem, o POLAR é definido como um modelo de verificação de restrições de acesso baseado em políticas de granularidade fina de forma dinâmica e distribuída, sendo acoplado ao processo de negociação do serviço para evitar re-submissões. Ele permite negociações de métricas de níveis de serviço flexíveis, como horário de submissão, número máximo de *jobs* executando de forma concorrente, preço e garantia da execução dos *jobs*, entre outros. Portanto este módulo permite que o administrador do recurso possa controlar os *jobs* que irão executar nele.

O terceiro módulo do MASK é responsável pela tomada de decisão de seleção do melhor recurso num *grid*. Para isso foram utilizadas técnicas da teoria da decisão que associam probabilidades às preferências envolvidas. "O modelo tem como entrada de dados, a predição de execução obtida no modelo de predição e os dados da negociação do provimento do serviço obtidos através do modelo de verificação de restrições de acesso. A preferência do usuário que direciona a seleção é modelada em uma função multi-atributo para preço e desempenho. O recurso selecionado é aquele que maximiza a função utilidade do usuário"[Nassif 2006].

O módulo PredCase do MASK foi validado somente para recursos dedicados e seu desempenho para recuperação de casos pode ser melhorado. O PredCase gera resultados satisfatórios quando os recursos são dedicados, ou seja, quando cada recurso executa um *job* de cada vez, ficando o erro de predição em torno de 9%. Porém, apesar de possuir algoritmos que procuram adaptar a carga de trabalho passada com a carga atual, não há estudo quando utilizado em recursos não dedicados. Quanto ao desempenho para o cálculo da predição, a fase de recuperação de casos leva em média um segundo para uma base com um total de 300 casos, quando executado em uma máquina com 1024MB de memória principal, processador Intel Pentium 4 de 3GHz. Apesar que, esse tempo pode ser longo ou curto dependendo do tempo de execução do *job*, ele pode ser melhorado a partir das modificações na estrutura da base de casos, e de sua localização, carregando-a na memória principal no momento em que os casos são recuperados. A fim de resolver estas demandas do PredCase, propomos neste trabalho melhorar o cálculo de predição de *jobs* para recursos não dedicados, inserindo novas técnicas baseadas em algoritmos de aprendizagem e de carga de trabalho, além de modificar a estrutura de sua base de casos visando um melhor desempenho.

Segundo Kapadia et al. [1999], em um ambiente *grid* existem dois desafios para o cálculo da predição da utilização do recurso para um dado *job*. O primeiro desafio é devido à natureza do resultado que precisa ser calculado em tempo real, o qual deve ser feito depois que o usuário submete um *job*, porém antes que este comece a executar para poder ser escalonado. Isto impõe um limite superior à quantidade de informação que pode ser pesquisada e utilizada por algoritmos de aprendizagem, pois um algoritmo mais simples, com poucos dados e conseqüentemente mais rápido, pode gerar predições com erros grandes. Algoritmos mais complexos e que utilizam maior volume de informação tendem atingir resultados melhores, mas com tempos maiores de execução. O segundo desafio é devido às próprias características dos ambientes *grid*, que por estarem baseados geralmente sobre grandes redes de computadores, podem levar a variações na disponibilidade e desempenho dos recursos. Estas variações podem ser causadas por sobrecarga de roteadores e mudança de configuração de equipamentos, que podem ocorrer a qualquer momento.

1.2 Objetivos

O objetivo deste trabalho é calcular a predição do tempo de execução de *jobs* em *grids* com recursos computacionais não dedicados. Particularmente, estaremos interessados em:

- melhorar o desempenho do cálculo da predição, sobretudo da fase de recuperação do PredCase;
- estudar o comportamento do PredCase para recursos não dedicados;
- aplicar novas técnicas de predição baseadas em casos passados e na previsão da carga de trabalho futura ao módulo do PredCase.

Para calcular a predição do tempo de execução, aplicaremos técnicas que se baseiam em casos passados para gerar predições e também desenvolveremos um método para prever a carga futura de um dado recurso. Usaremos o paradigma de Raciocínio baseado em casos (RBC) utilizado no PredCase, combinado com métodos de Aprendizado baseado em instâncias IBL (do inglês *Instance based learning*), usado nos trabalhos de Li et al. [2005], Li and Wolters [2006] e Li [2007]. O principal diferencial do nosso modelo está na combinação da predição gerada a partir dos casos passados de um dado recurso com a previsão de sua carga de trabalho futura. A previsão da carga de trabalho, informação que não é utilizada em outros modelos de predição, será

obtida a partir da monitoração da carga de trabalho passada do recurso e do tempo de execução dos casos similares ao novo *job* que será executado.

Quanto ao desempenho do cálculo da predição, pretendemos diminuir o tempo para seu cálculo modificando a estrutura de dados da base de casos e sua organização. A estrutura atual da base de casos do PredCase torna a recuperação de dados lenta, pois há necessidade de pelo menos dois acessos na base de casos. Nosso objetivo é modificar esta estrutura para que seja feito somente um acesso à base de casos quando uma pesquisa for realizada. Outro fator que prejudica o desempenho do PredCase é a organização da base de casos em arquivos com formato XML. Verificamos, em alguns testes, que este é um fator crítico durante a fase de recuperação de dados no PredCase. Nossa proposta também inclui a diminuição do número de informações armazenadas, resumando os dados a serem armazenados na base de casos. Isto permitirá que a base de casos possa estar disponível já em um formato adequado na memória principal quando a predição for calculada, o que diminuirá bastante o tempo de recuperação dos casos.

1.3 Organização do documento

Os capítulos da dissertação estão organizados da seguinte forma:

Capítulo 2: Apresenta duas técnicas de aprendizagem que utilizam casos passados para gerar predições, *Instance based learning* e *Case-based reasoning*.

Capítulo 3: Faz uma revisão bibliográfica dos trabalhos mais importantes de predição do tempo de execução de *jobs*, sobretudo aqueles que utilizam casos passados. Foi principalmente a partir da combinação das técnicas aplicadas nesses trabalhos que chegamos ao nosso algoritmo.

Capítulo 4: Descreve o desenvolvimento do algoritmo, mostra como as técnicas do Capítulo 3 são utilizadas e combinadas, além de apresentar o desenvolvimento de outras técnicas para predição do tempo de execução de *jobs* para recursos não dedicados.

Capítulo 5: Faz uma análise do algoritmo desenvolvido neste trabalho para diferentes configurações de *grid*, a fim de estudar seu comportamento.

Capítulo 6: Faz um resumo dos objetivos definidos com os resultados alcançados, além de projetar futuras abordagens para a evolução deste trabalho.

Capítulo 2

Técnicas de aprendizagem baseada em casos

Neste capítulo abordaremos a técnica de Aprendizado Baseado em Instâncias (IBL do inglês *Instance Based Learning*) e o paradigma de Raciocínio Baseado em Casos (RBC, ou CBR, do inglês *Case-based Reasoning*). Essas técnicas são utilizadas em alguns trabalhos sobre predição de *jobs*, bem como em nosso trabalho. Elas se baseiam na geração de soluções para novos problemas considerando um histórico de casos e/ou soluções passadas. Para isto, serão desenvolvidas fases e estruturas de dados para armazenamento e recuperação de casos relevantes, adaptação da solução encontrada para um novo problema e armazenamento na base desses novos casos. Nosso trabalho, como será visto no capítulo 4, utiliza o paradigma RBC e uma técnica de IBL.

Na maioria dos métodos de aprendizagem, um único modelo global é usado para ajustar todo o *training data*¹ [Chris Atkeson 1997]. No modelo global há um esforço para estabelecer uma única função que mapeia uma entrada a uma saída, $y = f(x, \theta)$, onde θ é um vetor de tamanho finito, utilizando todo o *training data*. Enquanto os métodos desse modelo podem teoricamente se aproximar de alguma função contínua, eles podem não ser apropriados para todo tipo de aplicação [Kapadia et al. 1999]. Um exemplo de aplicação que não é atendida pelo modelo global, citado por Kapadia et al. [1999]: as ferramentas de simulação de dispositivos semi-condutores permitem tipicamente que o usuário simule em uma, duas ou três dimensões. Em geral, diferentes técnicas são utilizadas para cada um desses casos, implicando que o mapeamento da entrada-saída (*input-output*) para tais ferramentas consistirá de três distintos conceitos, não sendo possível estabelecer uma única função que atenda os três conceitos. Na tentativa de resolver problemas como este, o modelo local tenta superar alguns problemas

¹ *Training data* é o conjunto de instâncias utilizadas pelo método de aprendizagem.

do modelo global pela divisão do espaço de entrada em muitas partições [Moore et al. 1997]. Nos métodos do modelo local cada partição i é aproximada por função independente, $y_i = f_i(x, \theta_i)$, as funções f_i são mantidas o tão simples quanto possível [Kapadia et al. 1999]. Um problema que surge é a seleção de partições apropriadas para o sistema de aprendizado, no qual há um esforço em utilizar o *training data* somente de uma região ao redor da localização próxima ao *query point* [Kapadia et al. 1999]. O *query point* em nossa aplicação, representa o novo *job* num espaço de várias instâncias de casos passados.

2.1 Aprendizado baseado em instâncias

Os métodos de Aprendizado Baseados em Instâncias (IBL) operam sobre uma base de instâncias. Esses métodos armazenam *training data*, apesar de não requererem uma fase explícita para isto, e calculam predições para uma particular consulta aplicando um modelo de indução sobre as instâncias próximas dos dados de entrada desta consulta (*query point*) [Li et al. 2005] [Kapadia et al. 1999]. Dessa forma, esses métodos permitem que partições sejam criadas, e que a predição seja calculada apenas sobre a partição mais próxima ao novo problema, característica desejada para os métodos de aprendizagem local.

Os métodos IBL processam os pontos próximos de um *query point* com o objetivo de estimar uma saída apropriada que represente as características de tal conjunto [Chris Atkeson 1997], sendo necessário para esses métodos o desenvolvimento de uma função de distância (do inglês *distance function*). A função de distância calcula a distância entre cada instância presente no (*training data*) e o *query point*, sendo preciso definir uma métrica para medi-la [Li et al. 2005]. Três importantes métodos de IBL, também chamados de modelos de indução², são o *Nearest Neighbor* (NN), o *Weighted Average* (WA) e o *Locally Weighted Regression* (LWR), descritos brevemente a seguir [Kapadia et al. 1999]:

- O método *k-nearest neighbor* (k-NN) gera previsões para um dado *query point* usando uma média aritmética dos valores de saída (*output*) das k instâncias mais próximas dele. Sua implementação requer a especificação dos seguintes parâmetros: o número de instâncias (k) da vizinhança que participarão do cálculo da previsão e uma métrica apropriada para o cálculo da distância;

²Alguns trabalhos fazem referência ao NN, WA e o LWR como modelos de indução, sendo componentes da técnica IBL [Li et al. 2005], outros como métodos de aprendizado baseado em instância [Kapadia et al. 1999].

- o método *k-weighted average* (k-WA ou n-WA) gera suas previsões utilizando uma média ponderada dos valores de saída das k instâncias mais próximas do *query point*. O peso atribuído a uma instância é dado por uma função inversa da sua distância do *query point*, chamada de *kernel function*. Dessa forma, os seguintes parâmetros são necessários para implementação do método: o número de k pontos da vizinhança que serão incluídos no cálculo da previsão, uma métrica apropriada para o cálculo da distância e uma *kernel function*;
- o método *Locally weighted regression* (LWR) ajusta os pontos próximos do *query point* em uma superfície, geralmente usando um modelo linear ou quadrático. Para aplicar o método LWR é necessário especificar, além dos parâmetros necessários ao algoritmo n-WA, a ordem do polinômio que será gerada a superfície. Geralmente são utilizados modelos lineares ou quadráticos, pois polinômios de ordem maior estão associados a um maior custo computacional.

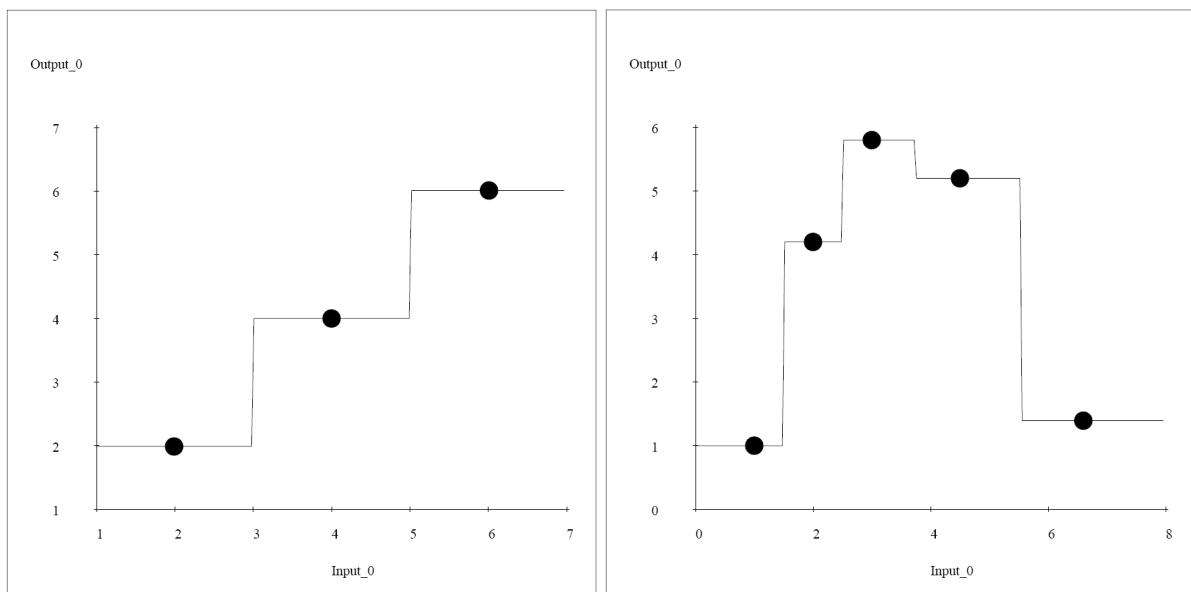


Figura 2.1. No método *Nearest-neighbor* é escolhido os valores mais próximos do *query point*. Fonte: [Chris Atkeson 1997].

As Figuras 2.1, 2.2 e 2.3 são exemplos dos três métodos de indução, *Nearest neighbor*, *Weighted average* e *Locally weighted regression*, respectivamente. Nestas figuras, o *Input_0* representa o *query point*, o *Output_0* representa a saída do método de indução em questão e dos pontos, que representam os casos da base. Pelo gráfico podemos observar que os resultados tendem a aqueles pontos mais próximos ao *query*

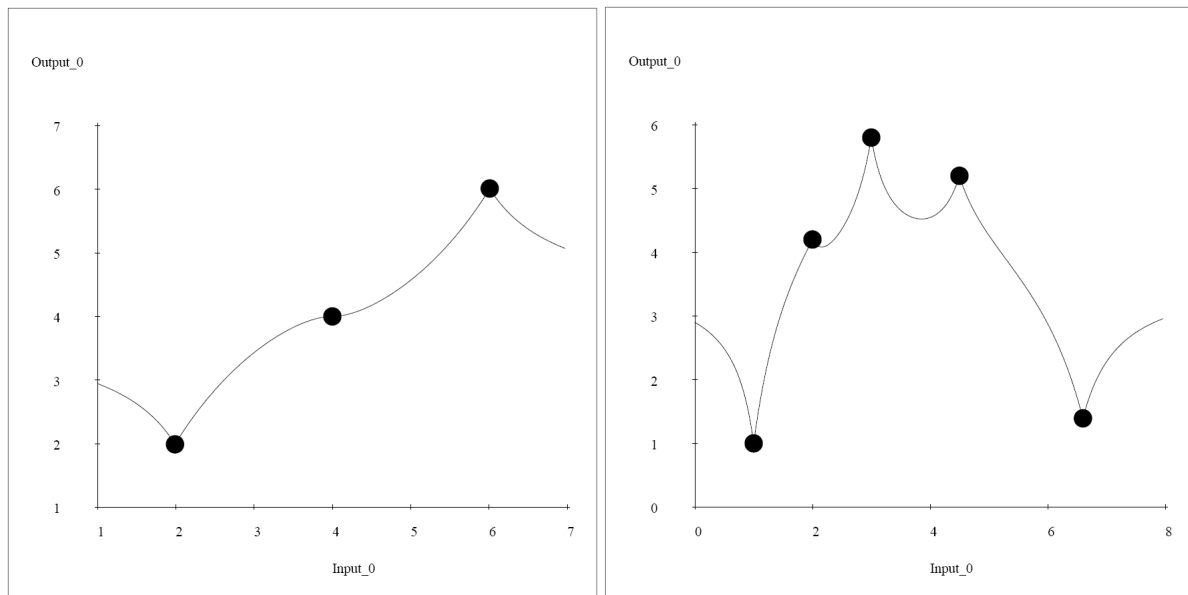


Figura 2.2. Algoritmo *Weighted average*. Fonte: [Chris Atkeson 1997].

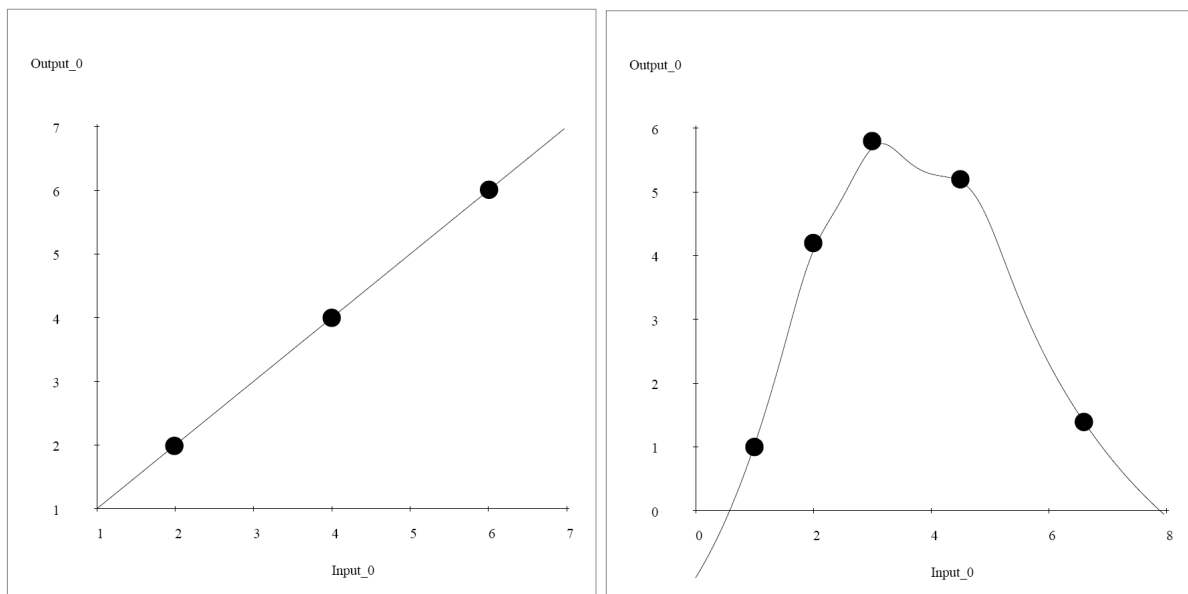


Figura 2.3. Algoritmo *Locally weighted regression*. Fonte: [Chris Atkeson 1997].

point, variando de acordo com o algoritmo que é utilizado. Para escolher qual o melhor dentre os métodos de indução para uma aplicação, é preciso uma análise sobre a distribuição dos casos da base para definir qual método é o mais indicado.

2.2 Raciocínio baseado em casos

O paradigma de Raciocínio Baseado em Casos (RBC) pode ser usado para resolver novos problemas, adaptando soluções que foram utilizadas para resolver problemas anteriores [Riesbeck and Schank 1989] *apud* [Abel and Castilho 1996]. Os casos passados são usados para explicar, criticar e interpretar novas situações para um novo problema [Nassif 2006]. Na terminologia do RBC, um caso normalmente denota um problema, uma situação, uma prévia experiência de uma situação que foi capturada e aprendida de uma dada maneira. Essa experiência pode ser usada na solução de futuros problemas, e é referenciada como caso passado (*past case*), *previous case*, *stored case*, ou *retained case*. De modo correspondente, um novo caso ou um caso não resolvido é a descrição de um novo problema a ser resolvido [Aamodt and Plaza 1994].

O RBC é capaz de utilizar um conhecimento específico de uma experiência prévia para resolver novos problemas [Aamodt and Plaza 1994]. Ele resolve um novo problema adaptando casos bem sucedidos do passado para problemas similares [Watson and Marir 1994]. O RBC é uma abordagem incremental e que mantém seu aprendizado, visto que uma nova experiência é obtida quando um novo problema é resolvido, ficando esta experiência imediatamente disponível para futuros problemas [Aamodt and Plaza 1994]. É um paradigma que adapta soluções passadas para novas demandas. E possui um processo cíclico e integrado de solução de problemas capaz de aprender a partir de uma experiência e solucionar novos problemas [Aamodt and Plaza 1994].

Uma importante característica do RBC é sua ligação com o aprendizado. A força propulsora por trás dos métodos baseados em casos vem em grande extensão das comunidades de *machine learning*, sendo o RBC considerado como um sub-campo de *machine learning*. Dessa forma, o RBC não denota somente um particular método de raciocínio, mas também denota uma técnica de *machine learning* capaz de permitir a manutenção do aprendizado por um longo período, para isso atualizando sua base de casos depois que um problema foi resolvido. Quando um problema é resolvido com sucesso, a experiência é guardada para resolver problemas similares no futuro. Quando o esforço para resolver um problema falha, a razão de tal falha é identificada e lembrada para evitar a mesma falha no futuro [Aamodt and Plaza 1994].

O RBC abrange diferentes métodos para organizar, recuperar, utilizar e indexar o conhecimento guardado dos casos passados. Os casos podem manter uma experiência concreta e um conjunto de casos similares podem formar um caso genérico. Eles podem ser armazenados em unidades separadas de conhecimento, ou podem ser separados em sub-unidades e distribuídos dentro da estrutura de conhecimento. Eles ainda podem

ser indexados por um vocabulário aberto ou prefixado, dentro de uma estrutura plana ou uma estrutura indexada e hierarquizada. A solução de um caso prévio pode ser diretamente aplicada para o problema presente, ou modificada de acordo com as diferenças entre os casos. O RBC pode usar uma base de casos largamente distribuída, ou uma base com casos mais típicos. Podem ser autocontidos e automatizadas ou integrar fortemente com os usuários e guiar suas escolhas. Os casos passados podem ser recuperados e avaliados seqüencialmente ou em paralelo [Aamodt and Plaza 1994].

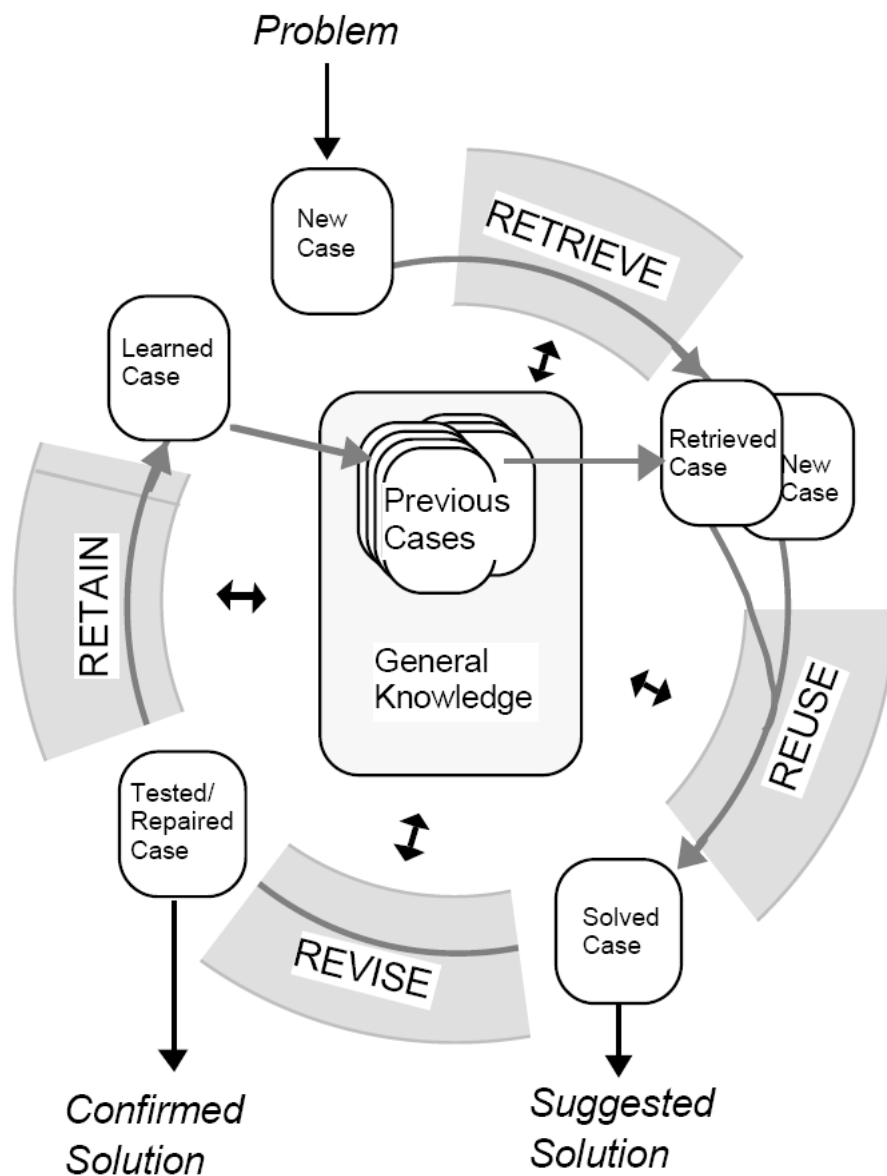


Figura 2.4. Ciclo do RBC. Fonte: [Aamodt and Plaza 1994].

O ciclo do RBC pode ser dividido, de maneira geral, em quatro fases:

1. Recuperação dos casos mais similares;
2. Reúso da informação e do conhecimento nos casos recuperados para resolver o problema;
3. Revisão da solução passada ou do conjunto delas;
4. Retenção de partes da experiência com o objetivo de disponibilizá-las para uso na resolução de problemas futuros.

A Figura 2.4 representa as fases do RBC descrita por Melchior [1999]: a descrição de um novo problema gera um novo caso, que é usado para recuperar da base de casos (*RETRIEVE*) um ou mais casos similares ao novo problema. A solução apresentada pelos casos recuperados é então combinada com o novo caso através da reutilização, gerando uma solução proposta para o problema inicial (*REUSE*). Com a fase de revisão (*REVISE*), esta solução é testada (sendo aplicada ao ambiente no mundo real ou avaliada por um especialista) e revisada se necessário³, produzindo um novo caso. Esse será armazenado como um novo caso aprendido (*RETAIN*). O uso do conhecimento geral sobre o domínio geralmente faz parte do ciclo, sendo suportado pelas fases RBC. Este suporte varia nos diversos métodos utilizados pelos sistemas RBC existentes, sendo algumas vezes muito fraco ou mesmo nulo e em outras fornecendo um suporte muito forte. Cada fase será descrita com mais detalhes nas próximas subseções.

2.2.1 Organização da base de casos

Segundo Kolodner [1993] *apud* [Melchior 1999], “um caso é uma porção de conhecimento contextualizado representando uma experiência que ensina uma lição fundamental para atingir os objetivos do raciocinador”.

Sendo o caso uma porção de conhecimento representando uma experiência, é de grande importância definir como e quais informações farão parte de cada caso. O conteúdo de um caso é formado por três partes principais: **descrição do problema** ou **situação**, **solução** e **resultado**, os quais chamaremos de componentes do caso [Melchior 1999][Abel and Castilho 1996].

O componente descrição do problema ou situação representa os dados que serão utilizados, sobretudo para o cálculo da similaridade entre os casos, o qual indica para um novo problema quais casos são considerados similares para serem recuperados.

³A solução é revisada caso seja verificado que ela não resolveu o problema com sucesso.

Os dados que descrevem as características de cada caso devem ser suficientemente detalhados, para que o sistema seja capaz de julgar a aplicabilidade de cada caso quando surgir uma nova situação ou problema. O componente solução descreve como o problema foi resolvido, variando conforme o tipo de problema que o sistema propõe-se a resolver. Além da solução de um caso recuperado ser usada para calcular uma solução para o novo caso, ela também pode possuir dados que acrescentem informações para a fase de adaptação e revisão da solução proposta. O componente resultado é um *feedback* da solução sugerida pelo sistema, ele informa o desempenho e eficiência da solução proposta. Para uma nova situação o sistema pode utilizar este componente para antecipar problemas em potencial adaptando a solução sugerida [Watson and Marir 1994].

Métodos baseados em RBC são altamente dependentes da estrutura e do conteúdo de suas coleções de casos. Como a solução para um novo problema é resolvida a partir de uma experiência anterior, armazenada em uma base de casos, os processos de busca e casamento precisam ser eficientes. Além disso, para cada nova solução obtida são necessários mecanismos que integrem esta nova experiência à base de casos. Decidir o que será armazenado na base de casos é o primeiro desafio a ser resolvido na representação de casos do RBC. Esta decisão envolve achar uma estrutura apropriada para descrever o conteúdo dos casos, decidir como a memória deve ser organizada e indexada para uma eficiente recuperação e reúso [Aamodt and Plaza 1994]. Os procedimentos de busca, o armazenamento e a base, que serve como repositório de casos, são usualmente referenciados como modelo de memória de casos [Watson and Marir 1994][Kolodner 1993]. Juntos, a base e os procedimentos disponibilizam os casos para a consulta e permitem a incorporação de novos casos, mantendo a acessibilidade dos dados presentes na memória [Melchioris 1999].

Quanto à organização da base de casos, duas principais classes de memórias têm sido propostas, as desestruturadas, ou (*flats*), e as estruturadas [Nassif 2006]. Na memória *flat* todos os casos estão no mesmo nível e a fase de recuperação avalia todos os casos da base, o que torna essa fase mais lenta, porém o melhor caso é sempre encontrado [Nassif 2006]. As memórias estruturadas são organizadas por generalizações ou abstrações, por exemplo, árvores B, *M-tree* [Nassif 2006]. Essa forma de organização facilita a avaliação de uma nova situação, permite o controle por indexação e a busca por casos similares é mais eficiente, principalmente em bases de casos muito grandes [Bichindaritz 2006]. Porém, nas memórias estruturadas nem sempre o melhor caso é encontrado [Nassif 2006].

2.2.2 Recuperação de casos

A fase de recuperação de casos se inicia com um problema de descrição e termina quando os casos mais similares são encontrados [Aamodt and Plaza 1994]. O objetivo da fase de recuperação é encontrar os casos que sejam os mais similares possíveis a um novo problema ou uma nova situação, a partir de uma base de casos, formando um pequeno conjunto de casos que serão úteis para propor uma solução. Calcular o quão similar são dois casos depende de escolher quais características de descrição são relevantes para o que se deseja medir. Vários atributos podem ser utilizados para descrever um problema ou um caso, mas nem todos atributos são relevantes e têm igual importância. É necessário definir os atributos que são importantes para um determinado contexto. Características que são de grande relevância em determinadas situações podem ter menor importância em um outro contexto. Dar pesos diferentes para cada uma dessas características é uma solução para diferenciar a relevância dos atributos, sendo um trabalho complexo e que vai resultar no sucesso ou insucesso do cálculo da similaridade. Algumas técnicas distribuem os pesos dinamicamente para um conjunto de atributos, e para isso utilizam algoritmos complexos, como computação genética, na qual testam continuamente qual a melhor distribuição destes pesos.

A recuperação de casos similares envolve duas questões com diferentes pontos a serem analisados: a identificação das características relevantes e o método de recuperação de casos. O primeiro é responsável por identificar quais são as características do caso corrente, ou novo caso, que devem ser utilizadas para julgar a similaridade com aqueles armazenados. Essas características serão utilizadas pelos procedimentos de classificação para identificar quais dos casos armazenados têm potencial de serem mais úteis. A outra questão é quanto ao algoritmo de busca, que visa encontrar na base aqueles casos que potencialmente são úteis para sugerir uma boa solução para o novo caso. Esses algoritmos de busca estão relacionados diretamente às estruturas utilizadas na organização da base de casos. Devido a esta relação, podemos identificar duas principais técnicas utilizadas para a recuperação, a busca seqüencial e a busca indutiva.

A técnica de busca seqüencial é utilizada quando os casos estão armazenados seqüencialmente em uma lista simples. É calculada a similaridade de todos os casos, sendo retornados aqueles com maior similaridade com o novo caso. Este tipo de busca é de fácil implementação e sempre os melhores casos são recuperados, porém sua maior desvantagem é de se tornar lento quando o número de casos cresce. O algoritmo *refined nearest-neighbor* [Nassif 2006] é um exemplo de algoritmo de recuperação de casos para este tipo de estrutura.

Na técnica de busca indutiva constroem-se árvores de decisão, que geram agrupamentos nos quais os casos que possuem muitas características em comum são agrupados juntos. Numa estrutura deste tipo cada nó interno mantém as características semelhantes com os casos abaixo dele, sendo os casos mantidos pelo nó folha. A recuperação numa árvore de decisão é mais eficiente do que em uma busca seqüencial. Entretanto bons casos podem ser perdidos se a hierarquia dos nós não corresponderem às características mais relevantes do novo caso. Além disso, a inserção é uma operação complexa, e com o crescimento da base de casos é difícil manter a árvore numa estrutura ótima, tornando a busca ineficiente [Melchioris 1999] [Dalfovo et al. 2002].

2.2.3 Reúso da solução (adaptação)

Na fase de reúso da solução, também conhecida como fase de adaptação, uma proposta é sugerida a partir da lista de casos resultados da fase de recuperação. A solução pode ser adaptada ou não para as necessidades do novo problema. Esta fase envolve a identificação de diferenças entre o estado do ambiente em que o novo problema será submetido e aquele em que os casos passados recuperados foram executados. Se as diferenças são irrelevantes, a solução proposta não sofre adaptações, caso contrário uma adaptação é requerida. Porém esta adaptação é necessária na maioria dos sistemas [Lewis 1995].

Em geral existem quatro decisões principais que precisam ser executadas na fase de adaptação [Nassif 2006]:

- Identificar o que precisa ser alterado;
- Identificar quais partes da solução devem ser mudadas;
- Identificar os métodos de adaptação aplicáveis;
- Selecionar uma estratégia de adaptação e utilizá-la.

2.2.4 Revisão e armazenamento (aprendizado)

A revisão é a fase na qual a solução sugerida é avaliada. Se a solução proposta na fase de adaptação produzir resultados satisfatórios, a experiência obtida deve ser aprendida, sendo armazenada na memória de casos para uso futuro. Porém, quando a solução para uma nova situação ou problema não está correta, então apareceu uma oportunidade de aprender com a falha [Watson and Marir 1994]. Dessa forma, a solução deve ser reparada gerando nova solução para então ser armazenada na memória de casos. A

reparação da solução envolve a detecção dos seus possíveis erros e a explicação para a ocorrência destes erros na tentativa de evitá-los.

O armazenamento é a fase de incorporação da nova experiência obtida na resolução de um novo caso. Nessa fase o que é útil para o aprendizado é retido na memória de casos. O aprendizado gerado pela nova solução proposta vem da avaliação desta solução, podendo ser um caso de sucesso ou de falha, sendo esta última com possível reparo. A fase de armazenamento envolve a seleção das informações relevantes do caso e a forma que essas informações são retidas e indexadas para uma posterior recuperação, integrando o novo caso à memória [Aamodt and Plaza 1994].

A fase de armazenamento também é responsável em montar uma estrutura de índices que facilite o uso da memória de casos. O RBC deve integrar o conhecimento aprendido à memória de casos ajustando a indexação com as características que têm maior relevância para a fase de recuperação, na qual os casos mais similares a um novo problema são recuperados [Melchioris 1999]. Os índices e sua importância na indexação dos casos podem ser definidos previamente (solução mais simples) ou podem ser definidos de forma dinâmica (solução mais complexa), geralmente isso é feito usando pesos de acordo com a maior ou menor relevância do índice. Quando a importância dos índices é definida previamente, o contexto da aplicação deve estar bem representado para garantir que os casos mais relevantes sejam recuperados. E quando é feito de forma dinâmica, além de uma definição prévia dos pesos dos índices, estes serão ajustados continuamente pelo sucesso ou falha da utilização do conjunto de casos recuperados para a solução do novo problema. Neste caso, as características que foram julgadas relevantes na recuperação do caso com sucesso têm então sua importância aumentada, ao passo que as características que levaram aos casos não similares serem recuperados serão enfraquecidas [Melchioris 1999] na indexação do caso da base. A escolha na forma da estrutura do índice pode ser mais ou menos complexa dependendo do contexto, mais geral ou mais especializada, respectivamente. Porém uma solução mais complexa para atribuir importância aos índices não garante um resultado muito melhor.

2.3 Conclusão

Neste capítulo apresentamos alguns métodos de Aprendizado Baseado em Instâncias (IBL) e o paradigma de Raciocínio Baseado em Casos (RBC). Os métodos de IBL se baseiam em um conjunto de instâncias, conhecida na literatura como *training data*, para gerar soluções a partir da recuperação e do processamento das instâncias mais próximas do novo problema, gerando uma nova solução. Os métodos que implementam o RBC,

também se baseiam em um conjunto de casos passados, porém possui quatro fases bem definidas para gerar a nova solução: a recuperação de casos, o reuso da solução desses casos recuperados, a revisão da solução e seu armazenamento. O desenvolvimento do nosso trabalho envolveu o paradigma RBC e a utilização de técnicas de IBL, e o objetivo foi aproveitar as características destas técnicas já usadas por diferentes métodos de predição.

Capítulo 3

Predição do tempo de execução de *jobs*

*The heart of a capacity
planning process is its ability to
predict adequately the
performance of a particular
computer system configuration
executing a given workload.*

Menascé et al. [1994]

Neste capítulo discutiremos sobre algumas técnicas de predição de desempenho e sobre trabalhos relacionados à predição do tempo de execução de *jobs*. Será apresentada uma visão geral sobre as técnicas para predição de desempenho, justificando a escolha para nossa solução. Serão também apresentados os principais trabalhos e suas respectivas técnicas para cálculo da predição do tempo de execução de *jobs*, nos quais baseamos para a construção do nosso método, além de uma revisão de outras técnicas utilizadas para predição.

3.1 Técnicas para cálculo de predição de desempenho

De acordo com Menascé et al. [1994], as técnicas de predição de desempenho mais usadas são: regras práticas, análise de tendências, modelos de desempenho (analítico e simulação) e *benchmarks*. Estas técnicas diferem entre si em três aspectos básicos:

complexidade, acurácia e custo. Na Figura 3.1 mostramos as técnicas de predição mais usadas classificadas em ordem crescente de custo e complexidade. Quanto mais complexa e com maior custo for sua classificação, melhor é a acurácia. Dessa forma, as regras práticas são mais simples de serem usadas, porém não dão garantia de uma boa acurácia. No outro extremo da classificação, temos as técnicas que utilizam *benchmarks*, que são mais precisas, mas têm maiores custos e são mais complexas. As técnicas baseadas em modelos de desempenho têm sido amplamente utilizadas para predição de desempenho, devido a se situarem entre as regras práticas e os *benchmarks*, ou seja, têm custo e complexidade não muito altos, aplicabilidade mais geral e flexibilidade para seu uso [Menascé et al. 1994].

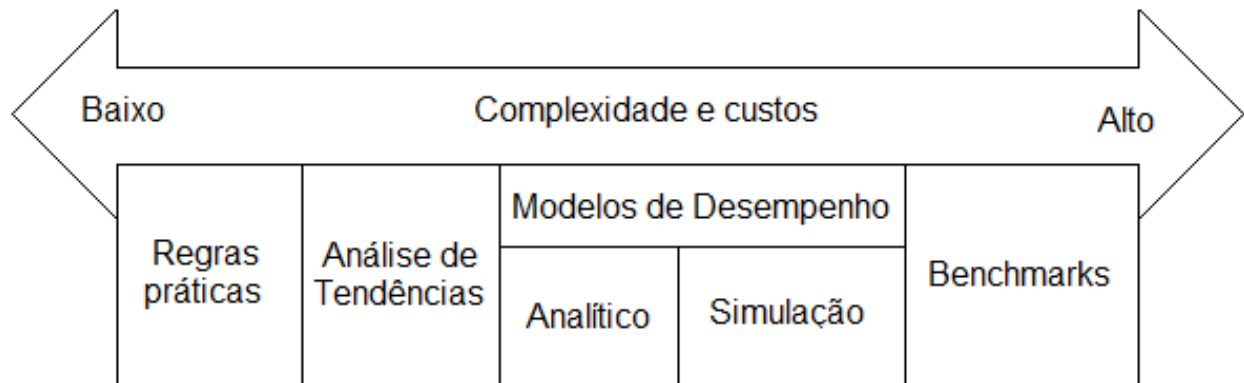


Figura 3.1. Classificação das técnicas de predição. Fonte: [Menascé et al. 1994].

As regras práticas são usadas para determinar toda a capacidade de um sistema em função da utilização de um componente individual. Focando em um recurso chave, há um esforço em prevenir uma carga excessiva sobre um componente de forma a não degradar o desempenho de todo o sistema. Em geral, as regras práticas devem ser usadas como um ponto inicial para abordagem do problema. Estas técnicas surgem de experiências pessoais, de *benchmarks* e de produção. O maior problema ao utilizar as regras práticas é quanto à sua limitação de não levar em conta a interação entre os componentes de um sistema. Além do mais, é uma técnica limitada para se usar em sistemas modernos, devido à dificuldade de se encontrar um recurso chave que represente o desempenho de todo sistema. Dessa forma, esta técnica não é recomendada para predição de desempenho [Menascé et al. 1994]. Exemplos de regras práticas são:

- a utilização de CPU em 80% é o limite para um bom serviço [Menascé et al. 1994];

- lei de Amdahl para IO: programas fazem um IO para cada 50.000 instruções [Gray and Shenoy 2000];
- lei de Amdahl para memória: em um sistema balanceado, a relação MB/MIPS é igual a um [Gray and Shenoy 2000];
- lei de Amdahl para paralelismo: se uma computação tem uma parte serial S e uma parte paralela P , então o *speedup*¹ máximo é $\frac{S}{S+P}$ [Gray and Shenoy 2000];
- uma regra prática para troca de mensagens do sistema é, enquanto uma mensagem de rede custa 10.000 instruções e mais 10 instruções por *byte* transmitido, o custo para um IO do disco custa 5.000 instruções e 0.1 instrução por *byte* [Gray and Shenoy 2000].

A análise de tendências pode ser vista como uma técnica que prevê o que irá acontecer em um sistema baseado em um histórico relacionado ao seu comportamento no passado. O cálculo de predições consiste em extrapolar a tendência para um nível esperado da carga de trabalho. Para fazer esta extrapolação geralmente é utilizado regressão linear entre a carga de trabalho passada e sua predição. Dessa forma, a partir de uma intensidade da carga de trabalho esperada, obtém-se o ponto correspondente da predição de desempenho. Assumir uma relação linear entre o desempenho e a carga de trabalho pode ser viável quando a contenção de recursos do sistema é mínima. Entretanto, a relação linear pode não gerar boas predições quando os componentes estão congestionados. Neste caso a análise de tendências não seria uma boa técnica de predição [Menascé et al. 1994].

Quanto à técnica de modelos de desempenho, existem duas abordagens, por simulação e analítica:

Os modelos de simulação são baseados em programas que simulam diferentes aspectos dinâmicos do sistema. Esses modelos são muito detalhados e caros para serem desenvolvidos, executados e validados. Porém, eles fornecem informações detalhadas sobre o fenômeno a ser investigado. Os modelos analíticos são compostos de um conjunto de fórmulas e algoritmos que fornecem valores de medida de desempenho. Para que esses modelos sejam matematicamente tratáveis, eles são menos detalhados que os modelos de simulação. Eles são menos precisos, mas rodam eficientemente [Nassif 2006].

Os *benchmarks* usam uma carga de trabalho artificial que se aproxima da carga de trabalho prevista para uma dada configuração do sistema. A utilização de *benchmarks*

¹Em computação paralela *speedup* é quanto um algoritmo paralelo é mais rápido do que um algoritmo seqüencial correspondente, ou seja, é a medida do ganho de tempo proporcionado pela computação paralela [Amdahl 2000].

tem sido muito útil para gerar previsões da capacidade de sistemas modernos. Para previsão de desempenho a principal questão é como uma particular configuração de um sistema irá se comportar quando processar uma carga de trabalho. Para verificar isto é necessário processar em tal sistema, um *benchmark* que se aproxime da carga de trabalho que se queira medir e assim medir os resultados do desempenho. Infelizmente, na prática, torna-se impossível analisar o desempenho para diferentes configurações do sistema devido ao alto custo envolvido na montagem das configurações e a existência de vários *benchmarks*. Além de ser impossível avaliar o impacto de novas características ainda não completamente desenvolvidas [Menascé et al. 1994].

Para o desenvolvimento do nosso método, utilizamos as técnicas de desempenho analítico. Nossa escolha está relacionada às técnicas aplicadas ao PredCase e aos trabalhos na literatura com bons resultados que utilizam essa técnica.

3.2 Trabalhos de previsão do tempo de execução de *jobs*

Nesta seção discutiremos alguns trabalhos de previsão de *jobs* em *grids* e suas respectivas técnicas e resultados.

3.2.1 Previsão baseada em histórico e categorias de casos

O trabalho de Smith et al. [1998] calcula previsões do tempo de execução de aplicações paralelas. Nesse trabalho, o cálculo da previsão é feito a partir da definição de categorias que organizam os casos similares em um mesmo conjunto. Esse trabalho foi um dos primeiros a apresentar uma forma de calcular a previsão tempo de execução de aplicações paralelas a partir de um histórico de casos. Ele parte da observação de que os tempos de execução de aplicações similares são mais próximos entre si do que daquelas aplicações que não têm nada em comum. Nesse trabalho, cada recurso (máquina) participante do *cluster* organiza os casos em diferentes categorias a partir da similaridade entre os *jobs* que já executaram naquele recurso. Uma vez agrupados os casos, a previsão para um novo problema é calculada pela categoria em que o novo caso é mais similar. A partir dessa categoria, a previsão é calculada, seja por um método mais simples, como a média do tempo de execução dos casos, ou por um método mais complexo, como o que atribui pesos diferentes de acordo com o grau de similaridade. Segundo Smith [Smith et al. 1998], quanto mais sofisticadas forem as definições de similaridade entre os casos, melhor será a acurácia da previsão.

Nesse trabalho, para definir as categorias dos casos similares, foram utilizados *templates*. *Templates* são formados por tuplas compostas pelas características das aplicações, por exemplo, tipo do *job* (exemplo BLAST), usuário, argumentos, tempo de submissão, entre outros, também chamamos essas características de atributos do *job* ou do caso. Cada tupla possui um ou mais atributos, e cada tupla representa uma categoria. Como a combinação de todos os atributos gera um número muito grande de tuplas, nesse trabalho de Smith et al. [1998], foram desenvolvidos e avaliados dois algoritmos, um guloso e outro genético, que combinam os atributos com a finalidade de pesquisar por tuplas que geram boas predições. Uma vez definido o conjunto de *templates*, cada caso é classificado naquelas categorias em que melhor se assemelha com suas características (atributos), ou seja, que tenha maior similaridade. Um mesmo caso pode ocorrer em mais de uma categoria atendendo a diferentes tuplas. Dessa forma, aqueles casos que estão numa mesma categoria são considerados similares. Finalmente, o algoritmo calcula a predição da categoria que um novo *job* é mais similar, a partir do tempo de execução dos casos passados de tal categoria.

A partir de simulações, realizadas com quatro *workloads* de *clusters*, os autores chegaram à conclusão que sua técnica gerou predições entre 14% e 60% melhores do que outras técnicas que existiam até então. Eles conseguiram fazer com que a média dos erros das predições ficasse entre 40% e 59% do tempo médio de execução das aplicações. Quanto aos algoritmos de pesquisa de *templates*, o erro médio ficou entre 42% e 70%, quando foi usado os melhores *templates* gerados pelo algoritmo guloso. E para os *templates* gerados pelo algoritmo genético os erros diminuíram entre 2% e 12% em relação ao algoritmo guloso.

Algumas observações importantes realizadas pelos autores desse artigo foram: as predições calculadas a partir de uma categoria apresentam melhor resultado com a média do tempo de execução dos casos do que com sua regressão linear; a incorporação de tuplas com as estimativas dos usuários para o tempo de execução dos *jobs* melhoraram a acurácia da predição em alguns *workloads*, entre 23% e 48%; uma das maiores dificuldades dos autores foi identificar qual combinação de *templates* iria gerar boas predições para cada tipo de *workload*; há grande correlação entre usuários, *scripts* para execução dos binários, e o próprio nome dos executáveis. Apesar dos erros das predições calculadas serem grandes, esse trabalho apresenta uma nova maneira de gerar predições a partir de casos passados, sendo referenciados em outros artigos mais atuais. Um ponto fraco deste trabalho foi a não caracterização do sistema que simularam, principalmente se o sistema possuía fila de espera ou se os *jobs* eram executados assim que eram submetidos (sem fila). Para esse tipo de aplicação, cálculo de predições do tempo de execução, a maior dificuldade está em prever quanto tempo os *jobs* permanecem na

fila de espera até serem executados, ou a quantidade de *jobs* concorrentes, quando não há fila de espera.

3.2.2 Predição baseada em IBL e algoritmo genético

Outro trabalho que utiliza casos passados para gerar predições do tempo de execução foi realizado por Hui Li junto a outros autores. O método foi inicialmente apresentado no artigo [Li et al. 2005], e nos anos seguintes foram realizadas evoluções e novos estudos do método, artigos [Li and Wolters 2006] e [Li 2007]. O principal objetivo desses trabalhos é calcular a predição do tempo de resposta em supercomputadores paralelos *space-shared*² organizados em *clusters*. O tempo de resposta de um *job*, segundo Li [2007], é definido como o intervalo de tempo entre a submissão do *job* e o término de sua execução, ele é composto principalmente: pelo tempo de execução do *job* e pelo tempo de sua espera na fila. Dessa forma, é necessário calcular duas medidas para a predição do tempo de resposta: o tempo de execução do *job* num recurso e o tempo que esse *job* espera na fila para executar. Nesse trabalho, a predição do tempo de resposta para um novo *job* é calculada a partir de casos passados similares, sendo o cálculo da similaridade entre o novo *job* e os casos passados seu maior desafio.

Para recuperação dos casos e cálculo da similaridade é utilizada a técnica IBL (*Instance Based Learning*), apresentada na seção 2.1. Nesse trabalho foram desenvolvidos dois principais componentes dessa técnica: a função de distância (do inglês *distance function*), que calcula o quanto dois casos são similares; e o modelo de indução³, que, a partir do grau de similaridade entre os casos e um novo *job*, calcula a predição do seu tempo de resposta. Nesse trabalho o grau de similaridade depende de dados dos *jobs*, também chamados de atributos, como o seu nome, o usuário que o submeteu e o seu tipo; e também depende do estado do recurso como, a quantidade de *jobs* na fila, a quantidade de CPUs livres e a quantidade de *jobs* em execução. Para a função de distância foi empregada a métrica *Heterogeneous Euclidean-Overlap Metric* (HEOM). A HEOM é utilizada para calcular a distância entre atributos nominais e numéricos, sendo a função *overlap*, Equação 3.1, para atributos nominais e a distância euclidiana normalizada para atributos numéricos. Maiores detalhes sobre essas funções estão no artigo [Wilson and Martinez 1997]. Quanto menor é a distância entre um caso e um *job*, maior é o grau de similaridade entre eles. Para o modelo de indução os autores

²Sistemas *space-shared* iniciam a execução de um *job* assim que ele chega no recurso, porém se todos os elementos de processamento estiverem ocupados, executando outro *job*, ele é colocado na fila de espera [Buyya and Murshed 2002]. Lembrando que um recurso (máquina) pode ter um ou mais desses elementos de processamento.

³Alguns trabalhos também referenciam os modelos de indução, ou do inglês *induction models*, do IBL como métodos ou algoritmos de aprendizagem.

utilizaram dois métodos o *1-Nearest-Neighbor* (1-NN) e o *n-Weighted-Average* (n-WA), apresentados na seção 2.1. O modelo de indução é utilizado para gerar previsões do tempo de resposta a partir do grau de similaridade entre o novo *job* e os casos armazenados, usando o tempo de resposta dos *jobs* passados (casos da base).

$$overlap(x, y) = \begin{cases} 0 & \text{se } x = y; \\ 1 & \text{caso contrário.} \end{cases} \quad (3.1)$$

A função de distância depende principalmente dos atributos escolhidos para calcular se um caso passado e um *job* são similares. Uma escolha mal feita desses atributos compromete o cálculo da previsão. Um *job* pode ter vários atributos, como nome do grupo, nome do usuário, nome do executável, entre outros. No trabalho, os autores selecionaram um conjunto de atributos que podem ser relevantes ou não durante o cálculo da previsão. Para definir o que é relevante, na tentativa de diminuir o erro da previsão, foi utilizada a pesquisa genética. Com esta técnica, o algoritmo seleciona quais atributos são mais interessantes, a partir de um *training data set* usando operações de algoritmos genéticos tais como seleção, mutação e cruzamento (*crossover*).

Para avaliar o algoritmo, os autores utilizaram duas métricas: o tempo para calcular a previsão e a acurácia da previsão. O tempo para calcular a previsão foi medido a partir da média das previsões calculadas em uma máquina Xeon com quatro processadores de 2.8GHz de frequência e 3 GB de memória compartilhada. Para medir a acurácia da previsão, foram calculados o erro médio absoluto e o erro relativo. O erro médio absoluto é a média da diferença entre o tempo previsto ou estimado (t_{est}) e o tempo real de execução (t_{real}), $\sum_{i=1}^N |t_{est} - t_{real}| / N$, medido em minutos. Este erro é então normalizado pela divisão da média do tempo de execução ou do tempo da fila de espera, de acordo com o que está sendo avaliado, Equação 3.2, sendo chamado de erro absoluto normalizado. Outra métrica utilizada foi a relação entre a diferença do tempo estimado e do tempo real gasto na execução do *job* e sua respectiva soma, dado pela fórmula $r_e = (t_{est} - t_{real}) / (t_{est} + t_{real})$, chamado pelos autores de erro relativo. A partir do erro relativo é possível construir um histograma no qual podemos visualizar a quantidade de erros das previsões calculadas estiveram mais próximos de zero. Nas análises, os autores diferenciaram suas avaliações para o tempo de espera e o tempo de execução.

$$\frac{\sum_{i=1}^N |t_{est} - t_{real}|}{N} * \frac{N}{\sum_{i=1}^N t_{real}} \quad (3.2)$$

O trabalho foi avaliado empiricamente utilizando *workloads traces* do *cluster* de produção do NIKHEF (*National Institute for Subatomic Physics in the Netherlands*)

[NIKHEF 2008] ⁴ e do *Blue Horizon* do SDSC (*San Diego Supercomputer Center*) [SDSC 2008] ⁵. A principal diferença entre os dois sistemas está relacionada com a política da fila de espera. O NIKHEF é baseado em grupos e o SDSC em filas de prioridades. Os principais resultados deste trabalho foram:

- A média dos erros absolutos variaram de 210,5 a 577,1 minutos; o erro absoluto relativo ficou entre 0,43 e 0,73; e o erro relativo ficou centralizado em zero com a maioria dos erros entre -0,5 e 0,5, Figura 3.2 e 3.3;
- 70% dos *jobs* com tempo de execução maior do que dez mil *segundos* tiveram o tempo de execução acima do estimado, o que contribuiu muito para os erros de predição;
- O algoritmo sofre de *lagging problem*, problema que aparece quando há grande submissão de *jobs* idênticos, segundo a definição de similaridade desenvolvida no trabalho, em um curto intervalo de tempo o que chamam de submissão em *bags*. Nesse tipo de submissão os mesmos casos da base são selecionados pela técnica IBL para todos os *jobs* na *bag*. Se a seleção de *jobs* não for boa, causará erros em todas as previsões geradas para os *jobs* da *bag*;
- Quanto ao desempenho para calcular uma predição, a média de tempo para esse cálculo está na ordem de milisegundos.

Os autores ainda fazem uma análise do comportamento do algoritmo quanto aos grupos de usuários:

- Os *short jobs* são em sua maioria do grupo de usuários classificado como cinco, que correspondem a 16,7% dos usuários. Seus *jobs* têm uma média de tempo de execução em minutos. Para esse grupo o algoritmo de predição funcionou bem, com um erro absoluto de 12,1 minutos para o tempo de resposta. Não comentam quais são os erros relativos;
- *Jobs* com tempo de execução mais longos, com centenas de minutos, apareceram em diversos grupos, com destaque para o grupo classificado como dois, correspondendo a 36,5% dos usuários. Para esse grupo, o algoritmo funcionou relativamente bem com um erro absoluto normalizado para o tempo de resposta de 0,45;

⁴O NIKHEF cluster possui um *mix* de processadores *Intel* e *AMD*, com 512MB à 1GB de memória, variando entre 244 a 288 nós, que foram adicionados e removidos durante o período do experimento.

⁵O SDSC é um sistema constituído de 144 nós *IBM SP*, com 8 processadores por nó.

- Para grupos com poucas instâncias similares na base de casos, classificados como grupo sete, o algoritmo não obteve boas predições;

A Tabela 3.1 e as Figuras 3.2 e 3.3 resumam os resultados da avaliação para esse trabalho.

Tabela 3.1. Erro médio absoluto e normalizado do tempo de execução e do tempo de espera. Fonte: [Li and Wolters 2006].

Nome	Tempo de execução		Tempo de espera	
	Erro absoluto	Erro normalizado	Erro absoluto	Erro normalizado
NIKHEF	324,6 min	0,58	299,3 min	0,73
SDSC01	35,9 min	0,49	376,7 min	0,89
SDSC02	50,1 min	0,51	690,2 min	0,70

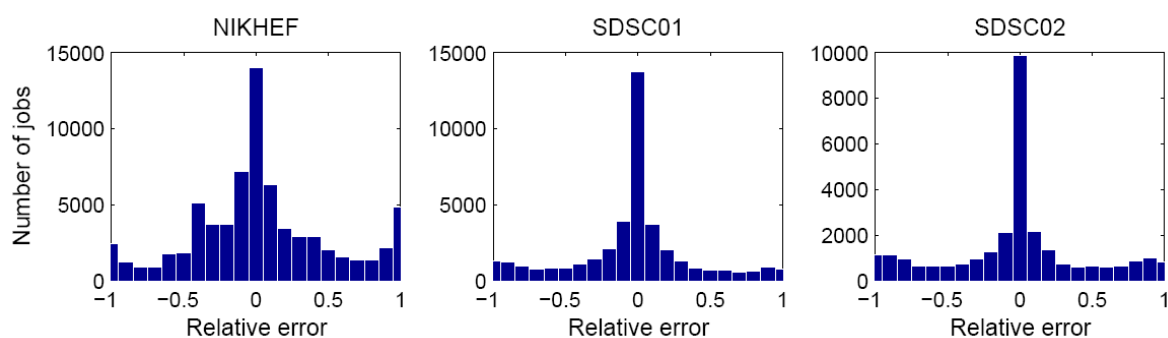


Figura 3.2. Histogramas dos erros relativos para predição do tempo de execução. Fonte: [Li and Wolters 2006].

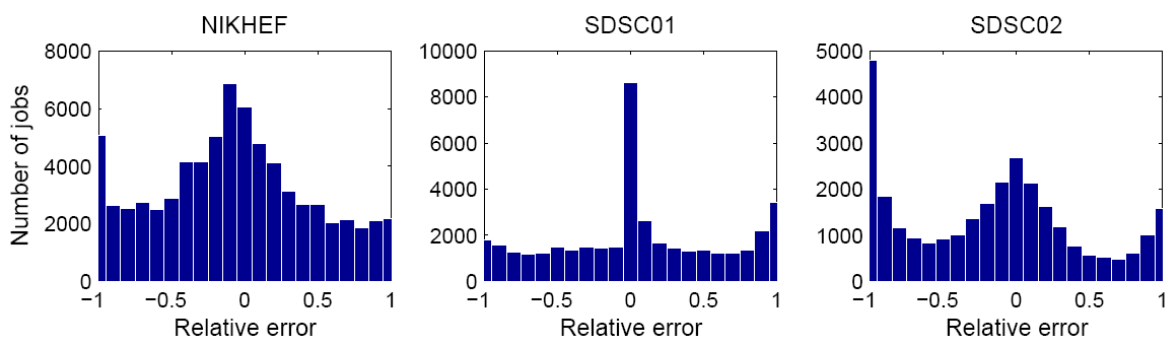


Figura 3.3. Histogramas dos erros relativos para predição do tempo de espera na fila. Fonte: [Li and Wolters 2006].

Na evolução do trabalho foram desenvolvidas e introduzidas outras técnicas: a *Adaptive selection tuning* e o uso da *M-Tree search*. A primeira técnica está relacionada com a escolha dos atributos para o cálculo da similaridade, na qual a base de casos é dividida em subconjuntos na tentativa de definir quais atributos são mais importantes para cada subconjunto. E a *M-Tree search* está relacionada com a recuperação de casos da base, na qual os casos mais similares são organizados em uma mesma sub-árvore. Essa técnica melhorou o tempo de pesquisa na base de casos de duas a oito vezes em relação à pesquisa seqüencial.

3.2.3 Predição baseada em RBC: PredCase

Outra técnica para cálculo de predições do tempo de execução de *jobs* a partir de casos passados foi desenvolvida no trabalho de doutorado de Lilian Nassif [Nassif 2006]. Como apresentado na introdução, o PredCase, componente do *middleware* MASK, utiliza o paradigma de Raciocínio Baseado em Casos (RBC) para gerar predições. No PredCase cada recurso contém uma memória de casos que evolui à medida que mais *jobs* são executados no recurso. Para cada novo *job*, os casos similares a ele são recuperados e é calculada a predição. O PredCase foi desenvolvido para realizar predições para recursos dedicados e não dedicados, porém foi validado apenas para recursos dedicados.

O PredCase possui uma base de casos organizada em três estruturas de dados, Problema, Solução e Resultado, mostrados na Figura 3.4. A estrutura de dados Problema contém os atributos que descrevem um *job* que foi executado no recurso: o nome da aplicação (Application), os argumentos da aplicação (Arguments), a data (Data) e o horário da execução (Time), o mínimo de memória necessária para sua execução (MinRam), entre outros. A estrutura de dados Solução contém atributos referentes ao cálculo da predição da execução do *job*: o tempo de execução previsto (Predicted-Time), o método utilizado (PredictionMethod) e o grau de similaridade do novo *job* com os casos passados (SimilarityDegree). A estrutura de dados Resultado contém os atributos que informam o resultado do método e outros dados referentes a execução do *job*: o tempo de execução (Duration), o erro da predição (%PredictionError), a carga de trabalho durante a execução do *job* (Workload⁶), entre outros. Juntas, as três estruturas armazenam os dados da execução e do cálculo da predição de um *job*, e seu conjunto forma a base de casos que armazenará toda a experiência do sistema. Sobre

⁶O atributo Workload[] armazena em um vetor o número de *jobs* que estão executando de forma concorrente num mesmo recurso, estes dados são coletados em intervalos regulares de tempo durante a execução do *job*.

estas estruturas o PredCase implementa três fases do RBC chamadas: Recuperação, Adaptação e Retenção (aprendizado).

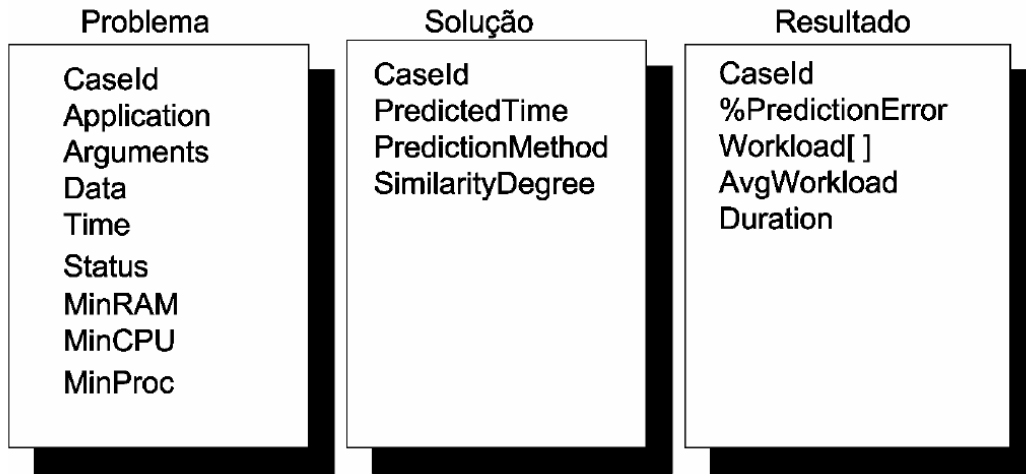


Figura 3.4. Estrutura de dados do PredCase. Fonte: [Nassif 2006].

A fase de recuperação de casos do PredCase calcula a similaridade entre um novo caso (um novo *job* submetido) com os casos da memória separando os casos mais similares. As informações dos casos necessárias para esta fase estão organizadas na estrutura de dados chamada Problema. Para esta fase foi desenvolvido o algoritmo *refined nearest-neighbor*, que tem como objetivo melhorar o desempenho do cálculo da similaridade entre o novo caso e os casos da base em relação ao algoritmo *nearest-neighbor*. Apesar do algoritmo *nearest-neighbor* ser um algoritmo típico para a fase de recuperação de casos nos sistemas baseados no RBC, ele apresenta uma grande perda de desempenho quando a base de casos cresce muito, como mostrado mais adiante na Figura 3.5. Isso porque o algoritmo compara todos os atributos de um novo caso com todos aqueles dos outros casos presentes na base. Dessa forma, aumentando o número de casos, o desempenho para o cálculo da predição diminui. Com o objetivo de melhorar seu desempenho foi desenvolvida uma modificação no algoritmo *nearest-neighbor*, que passou a ser chamado de *refined nearest-neighbor* [Nassif 2006].

O algoritmo *refined nearest-neighbor* executa o refinamento de casos de acordo com a importância do atributo no contexto da aplicação. Nele, o cálculo da similaridade do segundo atributo mais relevante ocorre somente para os casos que obtiveram alta similaridade para o primeiro atributo mais relevante. A mesma lógica é repetida para o terceiro atributo mais relevante em relação ao segundo, e assim sucessivamente para os demais atributos escolhidos [Nassif 2006]. Por exemplo, se dois casos tiverem os nomes das aplicações diferentes, caso esse seja o atributo mais relevante, então eles não

terão similaridade mínima o suficiente para continuar o cálculo da similaridade para os outros atributos, retornando zero. Caso contrário, retorna o valor da similaridade global, soma da similaridade de cada atributo. Os atributos utilizados para o cálculo da similaridade estão na estrutura de dados Problema de cada caso.

A próxima fase do PredCase, a fase de adaptação, calcula a predição do tempo de execução utilizando os casos recuperados da fase de recuperação. A partir desses β casos o PredCase calcula a média do tempo de duração, chamada de *pred*, mostrada na Equação 3.3. E depois adapta o resultado dessa média através de uma função $V = f(WP, WF)$ ⁷, mostrada na Equação 3.6. O objetivo da relação V é adaptar o valor da *pred* para a carga de trabalho atual do recurso. Para isso, V relaciona a média da carga de trabalho durante a execução do *job* passado (caso), chamada de WP, que está armazenada na estrutura de dados Resultado, e a média da carga de trabalho recente, chamada WF, mostrada nas Equações 3.4 e 3.5 respectivamente. A variável WF representa a carga de trabalho média recente calculada pelo módulo de monitoração do MASK. Nesse módulo, as cargas de trabalho dos recursos são coletadas periodicamente e registradas em uma estrutura de dados denominada Monitor. O cálculo da variável WF utiliza λ casos recuperados da estrutura de dados Monitor, no qual λ representa o número de casos que atendem à condição 3.7. O valor final previsto para o tempo de execução do *job* é dado pela Equação 3.6.

$$pred = \frac{\sum_{k=1}^{\beta} resultado_k.Duration}{\beta} \quad (3.3)$$

$$WP = \frac{\sum_{k=1}^{\beta} resultado_k.AvgWorkload}{\beta} \quad (3.4)$$

$$WF = \frac{\sum_{k=1}^{\lambda} monitor_k.AvgWorkload}{\lambda} \quad (3.5)$$

$$predAd = V * pred \quad (3.6)$$

$$Problema.Time \leq Monitor.Time \leq Problema.Time + Solucao.PredictedTime \quad (3.7)$$

A próxima fase do PredCase seguindo o ciclo do RBC é chamado de Retenção

⁷No trabalho Nassif [2006] não informa qual relação é essa, e qual seria sua unidade. Em nossos experimentos utilizamos a relação $f(WP, WF) = \frac{WF}{WP}$, a qual não tem unidade.

de casos. Nessa fase os novos casos são adicionados à base de casos, uma vez que tenham terminado sua execução com sucesso. No PredCase o armazenamento dessas informações é feita em três fases chamadas de Antes, Durante e Depois, que remetem à execução do *job* que está sendo monitorado. Na fase Antes são armazenadas, na estrutura de dados Problema, as informações relacionadas com a descrição do *job* que irá ser executado, e na estrutura de dados solução, as informações relacionadas à predição do tempo de execução do *job*. Na fase Durante é feita uma monitoração do *job*, na qual são armazenados os dados da carga de trabalho, número de *jobs* executando de forma concorrente no recurso, sendo que na fase Depois essa monitoração é resumida e armazenada na base de casos. E ainda na fase Depois são também armazenadas as informações sobre o tempo de execução do *job* e sobre a acurácia da predição. As informações das fases Durante e Depois são armazenadas na estrutura de dados Resultado.

É importante lembrar que o PredCase é um sistema distribuído e que cada recurso possui sua própria base de casos, no qual as informações armazenadas são apenas dos *jobs* que executaram nele. Dessa forma, o grau de conhecimento⁸ entre os diversos recursos que compõem uma grade computacional pode variar bastante, dependendo do número e dos tipos de *jobs* que executaram em cada recurso. A partir disso, dependendo do grau de conhecimento, recursos com a mesma configuração de hardware podem gerar predições diferentes [Nassif 2006]. A etapa de adaptação é importante para recursos não dedicados, pois a carga de trabalho que o *job* está sujeito interfere muito no seu tempo de execução. O foco do trabalho de Nassif [2006] foi em recursos dedicados e portanto a etapa de adaptação ficou para ser explorada em trabalhos futuros, espaço este que estamos investigando nesta dissertação.

Para validar o PredCase foram utilizados dois pacotes de aplicações denominados BLAST [Blast 2008] e HMMER [Hmmer 2008], que são aplicações usadas com frequência por pesquisadores do ambiente de grades computacionais. O cenário para os experimentos era composto por oito máquinas de diferentes configurações de *hardware*, no qual cada uma foi considerada um recurso. Os principais resultados foram:

- O erro da predição, medido pela fórmula 3.8, foi de 9% para recursos dedicados, ou seja, cada recurso (máquina) executa somente um *job* de cada vez. Mesmo usando máquinas de diferentes configurações, esse erro continuou o mesmo, isto demonstra, segundo a autora, que para diferentes máquinas, aplicações e durações

⁸O grau de conhecimento de um recurso está relacionado com seu histórico de execução de *jobs*, o que irá determinar o tamanho e a variedade dos tipos de casos de sua base, fatores importantes para o cálculo de uma boa predição.

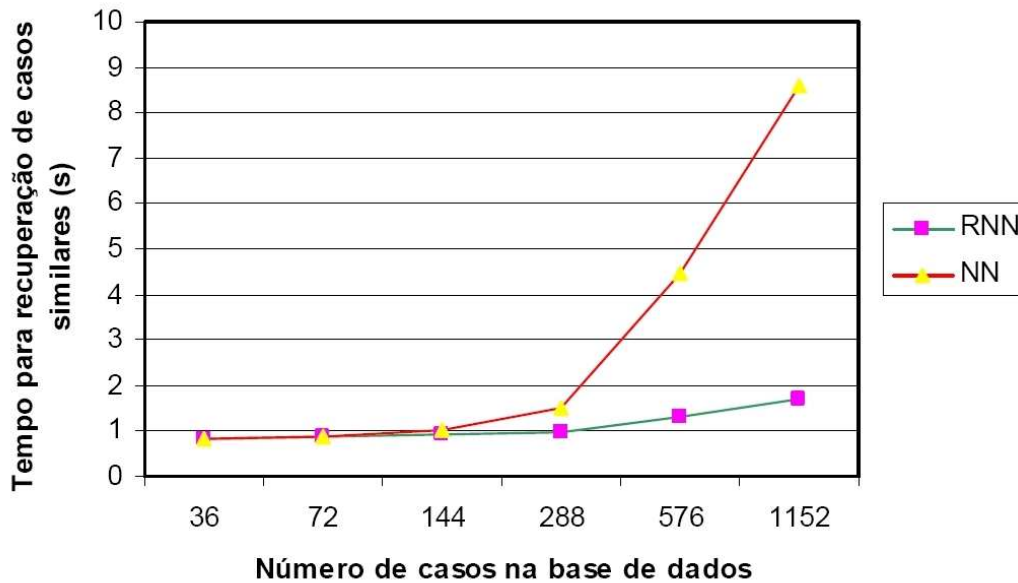


Figura 3.5. Desempenho dos algoritmos *refined nearest-neighbor* e *nearest-neighbor* para diferentes tamanho de base de casos. Fonte: [Nassif 2006].

de *jobs*, o modelo apresenta uma regularidade no erro da predição;

$$ErroPredicao = \frac{TempoPrevisto - TempoMedido}{TempoMedido} * 100 \quad (3.8)$$

- Tanto o algoritmo clássico para recuperação de casos, o *nearest-neighbor* (NN), quanto o algoritmo desenvolvido no trabalho, o *refined nearest-neighbor* (RNN), apresentam desempenhos similares em base de casos menores. Entretanto, o tempo gasto para recuperação dos casos pelo algoritmo NN aumenta substancialmente com o aumento do tamanho da base de casos, como pode ser visto na Figura 3.5, o que não acontece com o RNN.
- Quanto à relação entre o grau de similaridade e o erro da predição, foi verificado que na medida que é requerida uma maior similaridade entre um caso passado e um novo na fase de recuperação, obtém-se um menor erro de predição. Esta relação pode ser visualizada na Figura 3.6. Porém, quanto maior o grau de similaridade requerido, menor é o número de casos que se consegue recuperar da base de casos. E como para o grau de similaridade de 90% e 95% foi obtido um erro de predição bem próximo, foi escolhido usar o grau de similaridade de 90%, uma vez que não foi possível calcular a predição de alguns *jobs* quando o grau de similaridade de 95% era a opção escolhida.

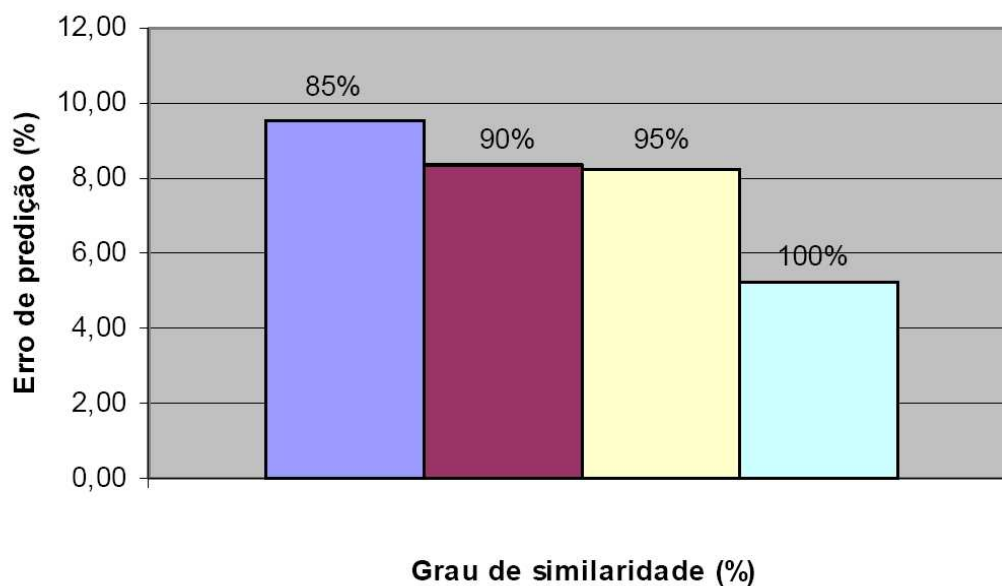


Figura 3.6. Percentual de erro médio da predição *versus* o grau de similaridade entre os casos. Fonte: [Nassif 2006].

3.3 Conclusão

Neste capítulo começamos com uma revisão da classificação das técnicas utilizadas para predição de desempenho proposto por Menascé et al. [1994], e depois apresentamos três técnicas para predição de desempenho baseadas em casos passados. A técnica para cálculo de predição, proposta por Smith et al. [1998], foi uma das primeiras técnicas a utilizar informações de *jobs* executados (casos passados), para calcular a predição do tempo de execução para um novo *job* submetido. Nessa técnica, são criados conjuntos a partir de diferentes *templates* formados por alguns atributos do *job*. Um *template* é escolhido, usando algoritmo genético ou guloso, como a melhor opção para formar conjuntos de casos similares, esse *template* pode variar para diferentes *workloads*. O cálculo da predição é feito usando uma média aritmética ou regressão linear sobre o tempo de execução dos elementos do conjunto mais similar ao novo *job*. Em outra técnica desenvolvida por Li et al. [2005], a escolha dos atributos do *job* para verificar a similaridade entre os casos da base é feita utilizando algoritmo genético. Escolhidos esses atributos os casos são recuperados e calculada a predição utilizando um método de IBL, conhecido como *Weighted Average* (n-WA). Outro trabalho de predição, chamado PredCase, desenvolvido na tese de Nassif [2006], apresenta um método baseado no paradigma RBC. Diferente dos trabalhos anteriores, no PredCase a recuperação de casos considera um conjunto fixo de atributos dos *jobs* para calcular a similaridade,

pelo qual os casos são recuperados. A partir dos casos recuperados chega a uma solução inicial, que é adaptada de acordo com a relação entre carga de trabalho recente e a carga passada em que os *jobs* foram executados. Nosso método para cálculo de predição é baseado no PredCase, diferenciando-se deste, principalmente, quanto à introdução do método n-WA para calcular a solução inicial, e a fase de adaptação desse método, como será mostrado no Capítulo 4.

Capítulo 4

Um método para predição de tempo de execução para recursos não dedicados

Neste capítulo mostraremos o desenvolvimento do nosso método para cálculo da predição para recursos não dedicados, o qual chamaremos de NdrPredCase (*Non-dedicated resources PredCase*). Nosso objetivo é calcular a predição do tempo de execução de *jobs* a partir do histórico da execução de *jobs* de cada recurso do *grid*. Este histórico contém informações sobre todos os *jobs* que já executaram no recurso, é único para cada recurso e constitui sua base de casos. Será mostrado como foram desenvolvidas as fases e a estrutura de dados para armazenamento na base de casos do RBC (Raciocínio baseado em casos). Também serão mostradas as técnicas utilizadas em nosso algoritmo para recuperação, adaptação da predição e armazenamento de casos, fases do RBC.

O escopo deste trabalho é calcular predições do tempo de execução para recursos não dedicados, também conhecidos como *time-shared*. No *time-shared*, quando um *job* é submetido num recurso ele inicia sua execução imediatamente e compartilha o recurso com todos os outros *jobs* que estão executando. Isso resulta em uma fila de *jobs* no processador, que são executados de acordo com sua política de escalonamento. Essa fila de *jobs* chamamos de carga de trabalho. Neste trabalho estamos considerando aplicações que executam em apenas um processador.

Em nosso método utilizamos o paradigma RBC por ser eficiente para o cálculo de soluções a partir de casos passados. Como mostramos na Seção 3.2, existem na literatura alguns trabalhos que já utilizam dados de *jobs* passados para calcular predição de tempo de execução de novos *jobs*. Porém o RBC foi utilizado pela primeira vez

para esta finalidade no trabalho de Nassif et al. [2007a], mostrando-se uma técnica eficiente para cálculo de predição para recursos dedicados. O RBC possui fases para o armazenamento e recuperação da informação e para adequar a solução a um determinado estado do recurso. Sua característica mais importante, em relação aos outros métodos utilizados para esse tipo de aplicação, é a adaptação de uma solução baseada em um histórico de diferentes estados do recurso para o seu estado atual. Dessa forma, o RBC é bastante atraente para utilizarmos no NdrPredCase.

O NdrPredCase estende o método desenvolvido para predição de *jobs* da tese de doutorado de Nassif [2006], método conhecido como PredCase, com o objetivo de melhorar as técnicas utilizadas no trabalho. O PredCase, embora seja um método eficiente para o cálculo de predição para recursos dedicados, ainda é ineficiente para recursos não dedicados, como mostraremos no Capítulo 5. Para melhorar esta limitação, nós analisamos seus pontos mais fracos para esta nova abordagem e desenvolvemos novamente cada fase do RBC que o PredCase implementa, e também reestruturamos sua base de casos. No NdrPredCase introduzimos algumas técnicas desenvolvidas nos trabalhos relacionados do Capítulo 3, que foram muito importantes para melhorar a acurácia do nosso método.

Esperamos com este trabalho desenvolver um método eficiente para o cálculo da predição de tempo de execução de *jobs* para recursos não dedicados, isto implica em possuir uma melhor acurácia em relação aos métodos já existentes e otimizar o tempo para cálculo da predição.

4.1 Organização dos casos

Como discutido na seção 2.2.1, um caso é uma porção de conhecimento contextualizado representando uma experiência [Kolodner 1993] *apud* [Melchioris 1999]. No NdrPredCase, os *jobs* que executaram completamente em um determinado recurso representam o conjunto de casos passados daquele recurso. Este conjunto representa uma experiência que servirá como base de conhecimento para tomar decisões futuras ou realizar cálculos. Como apresentado na seção 2.2, para a implementação do RBC é necessária uma base de casos onde serão armazenados os casos passados, formando assim a base de conhecimento para tal recurso. Cada recurso possui sua própria base de casos que armazena os dados dos *jobs* que foram a ele submetidos; cada caso representa as informações das características de um *job*, da sua execução e da solução para calcular sua predição. O tamanho da base de casos é controlado por um método específico na fase de retenção ou fase de armazenamento do paradigma RBC. Dessa forma, a memória

de todo o sistema apresenta um conjunto de bases de casos distribuídas, o que evita um possível ponto central de falha e permita o processamento distribuído. Assim a predição é calculada de forma distribuída e independente pelos recursos que compõem o *grid*.

A estrutura de dados do NdrPredCase é composta por uma única estrutura, a qual chamamos DataJob, que agrupa outras três estruturas a Problema, a Solução e a Resultado. O conjunto destas três estruturas reúne o conhecimento necessário para gerar predições em nosso método. Nelas temos todas as informações necessárias para representar cada experiência passada do tempo de execução dos *jobs* e de sua predição. Como podemos observar na Figura 3.4 da Seção 3, no PredCase, cada estrutura é independente, o que requer pelo menos três pesquisas na base para recuperar todas as informações de um caso passado. Para o NdrPredCase agrupamos as estruturas de dados Problema, Solução e Resultado em uma estrutura maior chamada DataJob, cada instância do DataJob representa um caso do RBC. Esta nova organização da base de casos permite que todas as informações de um caso sejam recuperadas em apenas uma pesquisa. Em um *grid*, cada recurso executa um conjunto de diferentes *jobs* e, portanto, cada recurso possui uma história particular quanto a execução de *jobs*. Como cada instância do DataJob representa um caso, cada recurso possui um conjunto de DataJobs representando seu conhecimento, sua memória ou base de casos.

No NdrPredCase a estrutura de dados Problema armazena a descrição do *job*, a estrutura Solução armazena os dados do cálculo da predição e a estrutura Resultado armazena as informações da execução do *job*. A estrutura DataJob agrupa as estruturas anteriores e possui um identificador, o IdJob, para diferenciar cada caso. A estrutura de dados Problema possui um conjunto de atributos que descrevem as características de um *job* que já foi executado no respectivo recurso. Esses atributos são o nome da aplicação, os argumentos da aplicação e o tamanho do arquivo de dados de entrada. Esses atributos são utilizados na fase de recuperação dos casos da base, para o cálculo da similaridade, no qual os casos mais similares são recuperados. A estrutura de dados Solução contém os dados da solução gerada, são eles: o grau de similaridade, a *raw prediction*, a predição adaptada, a predição ajustada, a predição final e um indicador de qual dessas predições foi usada, estes dados serão apresentados nas próximas seções. A estrutura de dados Resultado contém os dados da execução do *job*, que são: o tempo de execução do *job* e a média da carga de trabalho durante sua execução. Nas próximas seções iremos mostrar as fases do RBC em que cada atributo é utilizado.

O armazenamento das informações dos *jobs* nos casos é feito em duas etapas, uma antes da execução do *job* e outra depois. No momento em que um *job* é submetido a um recurso e antes de se iniciar sua execução, seus dados são armazenados na estrutura de

dados Problema e a solução da predição armazenada na estrutura Solução. Quando um *job* termina, os dados de sua execução são sumarizados e armazenados na estrutura de dados Resultado. Estas três estruturas são armazenadas no DataJob. E, finalmente, a estrutura DataJob é enviada para a base de casos. Para cada *job* submetido e executado com sucesso em um recurso, uma nova instância do DataJob é criada e armazenada na sua base. Se houver algum erro e o *job* não terminar sua execução do *job*, as informações são descartadas.

Em nosso trabalho a base de casos é organizada de forma seqüencial em uma estrutura de dados conhecida como *deque* (*double ended queue*), que é uma fila com inserção e remoção em ambos os extremos. Este tipo de estrutura facilita a adição de casos no final da fila, a pesquisa pelos casos mais recentes e o descarte dos casos antigos. Estes casos mais antigos são os primeiros da fila, e priorizando seu descarte mantém a base sempre com os casos mais recentes. A estrutura de dados *deque* ajudou a melhorar acurácia da predição e o tempo para seu cálculo, pois são utilizados os casos similares mais recentes enviados para a base de casos, sendo que nem todos os casos recuperados são utilizados para gerar a predição adaptada.

4.2 Recuperação de casos

A recuperação de casos é a primeira e uma das mais importante fases do RBC, pois todos as outras fases dependerão dos casos que serão recuperados por ela. Em nosso trabalho, ela também é a fase mais crítica quanto ao tempo gasto no cálculo da predição. Este tempo depende principalmente da quantidade de casos da base, do tipo de algoritmo utilizado para a recuperação e da forma como a base de casos é organizada. Definir um limite máximo para a quantidade de casos a ser armazenada não é uma tarefa fácil e depende da diversidade dos *jobs* que são submetidos num recurso. Bases pequenas podem não ser suficientes, pois é necessária uma quantidade mínima de casos similares que executaram com cargas de trabalhos próximas para poder adaptar a solução encontrada. Bases muito grandes tornam a fase de recuperação mais lenta, pois exigem mais do hardware para armazenamento e pesquisa. Quanto ao tipo do algoritmo para a recuperação de casos, o tempo para esta fase dependerá de como a estrutura da base de casos está organizada. Uma estrutura em árvore é mais eficiente, porém o melhor caso pode não ser encontrado. Numa estrutura linear, o melhor caso é sempre encontrado e sua implementação é simples, mas depende do tamanho da base, a qual pode comprometer o desempenho quando ela for muito grande [Nassif et al. 2007a]. O desafio da fase de recuperação está em combinar o tamanho da base, a

organização da base de casos e o algoritmo de recuperação.

Um método típico para realizar pesquisa em memórias seqüenciais, encontrado em diversas aplicações que utilizam o RBC, é o *nearest-neighbor*. O *nearest-neighbor* baseia-se na comparação entre um novo caso e todos aqueles da base de casos. Esta comparação é feita calculando-se a similaridade global entre dois casos através da soma da similaridade local entre seus atributos. Sua fórmula é dada pela Equação 4.1 [Dalfovo et al. 2002] [Watson and Marir 1994]. Calculada a similaridade global, cada técnica define qual seu valor mínimo para um caso ser recuperado, resultando em um conjunto de casos similares ao novo *job* (novo caso).

$$sim_{Global}(a, b) = \sum_{i=1}^p \omega_i sim_i(a_i, b_i) \quad (4.1)$$

onde:

$sim_{Global}(a, b)$ é a similaridade global entre os casos a e b ;

p é a quantidade dos atributos usados para verificar a similaridade entre dois *jobs*;

ω_i é o peso do atributo i ;

$sim_i(a_i, b_i)$ é a similaridade local para o atributo i entre os casos a e b .

No *nearest-neighbor* distinguem-se dois tipos de similaridade, a similaridade local e a similaridade global. A similaridade local corresponde a similaridade de cada atributo dos casos, que podem ser numéricos, simbólicos ou multi-valorados. A similaridade global corresponde ao somatório das similaridades locais de cada caso [Nassif 2006], como mostrado na Equação 4.1. Para calcularmos a similaridade local em nosso método, utilizamos algumas técnicas desenvolvidas e utilizadas nos trabalhos de Aamodt and Plaza [1994] e Nassif [2006]. As Equações 4.2, 4.3 e 4.4¹, mostram como são calculadas as similaridades locais para cada tipo de atributo numérico, simbólico e multi-valorado, respectivamente:

$$sim(a, b) = \begin{cases} \frac{a-b}{a} & \text{se } a > b; \\ \frac{b-a}{b} & \text{se } a < b. \end{cases} \quad (4.2)$$

$$sim(a, b) = \begin{cases} 1 & \text{se } a = b; \\ 0 & \text{se } a \neq b. \end{cases} \quad (4.3)$$

$$sim(a, b) = \frac{n(a \cap b)}{n(a \cup b)} \quad (4.4)$$

¹Apenas a Equação 4.4 foi desenvolvida especialmente para o NdrPredCase.

onde:

a e b são atributos de casos diferentes;

$sim(a, b)$ é a similaridade local entre os casos a e b para um determinado atributo;

n é o número de elementos do conjunto.

Com o objetivo de melhorar o desempenho durante a fase de recuperação de dados, em relação ao algoritmo *nearest-neighbor*, em nosso trabalho utilizamos o *refined nearest-neighbor*, algoritmo desenvolvido no PredCase, com algumas modificações visando melhor desempenho e melhor acurácia da predição. Os Algoritmos 1 e 2 mostram nossa implementação para o *refined nearest-neighbor*. Diferentemente do algoritmo original, desenvolvemos a distribuição de pesos de acordo com a relevância de cada atributo, e a normalização do resultado, mostrada na linha 23 do Algoritmo 2. Além do cálculo da similaridade dos argumentos do *job* que foi feita de acordo com a Equação 4.4.

O Algoritmo 1 percorre toda a base de casos adicionando os casos considerados similares na lista de casos recuperados que inicia vazia². A recuperação começa pelo caso mais recente inserido na base, mantendo a lista de casos similares ordenada por este critério. O resultado do algoritmo é uma lista dos casos que apresentam um grau mínimo de similaridade exigida, ordenados do caso mais recente ao mais antigo.

Algorithm 1 Algoritmo refined nearest-neighbor.

```

1: Entrada: baseDeCasos, descricaoDoNovoJob;
2: {Percorre os casos da base de casos}
3: for  $i = 1$  to baseDeCasos.tamanho() do
4:   {Calcula a similaridade global de acordo com o Algoritmo 2.}
5:    $dataJob \leftarrow baseDeCasos.get(i)$ ;
6:    $similGlobal \leftarrow Similaridade(descricaoDoNovoJob, dataJob.Problema)$ ;
7:   if ( $similGlobal > 0$ ) then
8:      $listaDeCasosRecuperados.add(dataJob)$ ;
9:   end if
10: end for
11: Saída: listaDeCasosRecuperados;

```

Um *job* é definido por um conjunto de parâmetros, entre os quais alguns exemplos são: o nome da aplicação, os argumentos da aplicação e os arquivos de entrada e saída de dados. Incluem-se também os requisitos mínimos necessários para a sua execução, tais como: o sistema operacional, a quantidade de memória principal, a quantidade de processadores e a velocidade da CPU [Nassif 2006], [Globus 2008] e [GridLab 2008]

²No PredCase esta lista inicia com todos os casos, sendo eliminados da lista os casos que não atingem o limite mínimo de similaridade local com o novo caso.

. Dado o objetivo do nosso trabalho, predição do tempo de execução de *jobs* para recursos não dedicados, e de acordo com o PredCase no qual nosso trabalho é baseado, foram escolhidos três atributos considerados relevantes para o cálculo da similaridade entre os casos. Estes são o nome da aplicação, os argumentos da aplicação e o tamanho do arquivo de entrada, respectivamente, de acordo com a ordem de relevância destes atributos em nossa aplicação.

Algorithm 2 Algoritmo para cálculo da similaridade.

```

1: {A entrada é composta pela descrição do novo job e da estrutura de dados Problema, que contém a descrição do caso passado.}
2: Entrada: descJob, problema;
3: similGlobal  $\leftarrow$  0;
4: {Calcula similaridade do nome da aplicação de acordo com a Fórmula 4.3.}
5: similLocal  $\leftarrow$  SimilaridadeDoNome(descJob.Nome, problema.Nome);
6: if (similLocal < LimiteSimilaridadeNome) then
7:   return  $\leftarrow$  0;
8: end if
9: similGlobal  $\leftarrow$  similLocal * pesoNome;
10: {Calcula similaridade do conjunto de argumentos do novo job e do caso passado de acordo com a Fórmula 4.4.}
11: similLocal  $\leftarrow$  SimilaridadeArgumentos(descJob.args, problema.args);
12: if (similLocal < LimiteSimilaridadeArgumentos) then
13:   return  $\leftarrow$  0;
14: end if
15: similGlobal  $\leftarrow$  similLocal * pesoArgumentos + similGlobal;
16: {Calcula similaridade do tamanho do arquivo de entrada de acordo com a Fórmula 4.2.}
17: similLocal  $\leftarrow$  SimilDado(descJob.Input.File.size, problema.Input.File.size);
18: if (similLocal < LimiteSimilaridadeInputSize) then
19:   return  $\leftarrow$  0;
20: end if
21: similGlobal  $\leftarrow$  similLocal * pesoTamanhoDado + similGlobal;
22: {Retorna a similaridade normalizada.}
23: return  $\leftarrow$  similGlobal / (pesoNome + pesoArgumentos + pesoTamanhoDado);

```

Cada cálculo da similaridade local tem um multiplicador, o qual chamamos de peso, para diferenciar a importância maior ou menor de cada atributo. Como podemos ver no Algoritmo 2, a partir destes atributos é calculada a similaridade entre um novo *job* e cada caso da base. Para o cálculo da similaridade do nome da aplicação, sendo um atributo simbólico, utilizamos a Equação 4.3. Para o cálculo da similaridade dos argumentos da aplicação separamos dois conjuntos, um com os argumentos do caso a ser calculada a similaridade e o outro com os argumentos do novo *job*. Calculamos

então a similaridade pela relação entre o número de elementos da intersecção entre os dois conjuntos e o número de elementos da união dos dois conjuntos, como mostrado na Equação 4.4. Quando os conjuntos são iguais o resultado é igual a um e quando são disjuntos o resultado é igual a zero. Já o cálculo da similaridade para o tamanho dos arquivos de entrada utiliza a equação 4.2. Como podemos ver no Algoritmo 2, o cálculo da similaridade global entre um caso da base e um novo *job* submetido segue uma ordem de importância dos atributos, que depende se a similaridade local atingiu um valor mínimo ou não. O algoritmo retorna a similaridade global entre o *job* submetido e o caso da base. Tanto a similaridade local como a global variam entre 0 e 1, fizemos isto para trabalhar com os dados normalizados e para facilitar o cálculo da função de distância como mostraremos na próxima seção.

4.3 Reúso da solução

O PredCase implementa três fases do RBC, a fase de recuperação, a de adaptação ou fase de reúso da solução, e a de retenção ou armazenamento. A fase de adaptação é a mais crítica quando utilizada em recursos não dedicados e não foi validada no PredCase. Nosso objetivo é desenvolver novos métodos para complementar o PredCase para recursos não dedicados, que inclui a previsão da carga de trabalho do recurso que irá executar o *job*. Além de estudarmos seu comportamento quando utilizado para prever a previsão do tempo de execução de *jobs* para recursos não dedicados.

No NdrPredCase foram desenvolvidas três tipos de previsão: a previsão crua, bruta, ou *raw prediction*, a previsão adaptada e a previsão ajustada. Sendo as três previsões importantes para o cálculo da previsão final.

4.3.1 *Raw prediction*

A *raw prediction* é a previsão calculada a partir do conjunto de casos similares resultado do Algoritmo 1 e de um modelo de indução. No NdrPredCase ela é a solução inicial da previsão do tempo de execução do *job*. A *raw prediction* é calculada a partir de um modelo de indução conhecido como *n-Weighted-Average* (n-WA) e do conjunto de casos recuperados, mostrado na Equação 4.5. O n-WA é um modelo de indução da IBL utilizado nos trabalhos de previsão de Li et al. [2005], Li and Wolters [2006] e Li [2007]. Em nosso método utilizamos para a *distance metric* o grau de similaridade. Para a função que mede a distância entre um *query point* e uma instância, um novo *job* e um caso da base em nosso trabalho, a chamada função distância (*distance function*),

utilizamos a similaridade global, dada pela fórmula 4.6. Na qual, quanto maior é a similaridade menor a distância entre o novo *job* e o caso passado.

A expressão para a *raw prediction* é:

$$P(q) = \frac{\sum_{i=1}^n W_i * Val(e_i)}{\sum_{i=1}^n W_i} \quad (4.5)$$

$$W_i = K(D(q, e_i)) \text{ e } K(d) = e^{-(const*d)^2},$$

onde:

q é o *query point*, ou seja, o novo *job* ou novo caso;

$P(q)$ é o resultado da *raw prediction*;

e_i é o i -ésimo caso resultado do Algoritmo 1;

$Val(e_i)$ é o tempo de execução do i -ésimo caso;

K é a *kernel function*;

D é a *distance function*;

$const$ é uma constante que controla o peso atribuído pela *kernel function*.

A *distance function* é dada por:

$$D(a, b) = 1 - sim(a, b) \quad (4.6)$$

Em nosso trabalho usamos o n-WA, que possui um limite n para o número de instâncias ou casos que irão fazer parte da solução, sendo um limite importante devido ao erro que gera quando utilizamos muitos casos antigos. A partir de observações iniciais chegamos à conclusão que calcular a predição do tempo de execução a partir de todos os casos similares da base aumenta o erro da predição. Um dos motivos é a participação com igual importância de casos com grandes diferenças da carga média de trabalho. Esta limitação aos casos mais recentes não garante que os casos mais similares farão parte do cálculo da *raw prediction*. Entretanto, em um ambiente de recursos não dedicados, o simples cálculo da similaridade baseada somente na descrição dos casos pode levar a grandes erros. Dessa forma, há um esforço para trabalhar com os casos inseridos recentemente na base.

4.3.2 Forecaster

O *forecaster* é o previsor da carga de trabalho do NdrPredCase que é usado para cálculo da predição adaptada. Esta previsão é feita a partir da média da carga do recurso monitorada recentemente. Prever a carga de trabalho em um ambiente formado por

diferentes tipos de recursos, no qual diferentes tipos de *jobs* podem ser submetidos por diferente grupos de usuários, em que a carga de trabalho apresenta grande variação, entre outros fatores característicos de um *grid*, é uma atividade sujeita a muitas incertezas, o que torna difícil o desenvolvimento de soluções eficientes.

No MASK, a carga de trabalho de um recurso é monitorada a intervalos regulares de tempo e os dados são armazenados na estrutura de dados chamada MonitorResource. Esta estrutura armazena os horários com a carga de trabalho monitorada (<Tempo, Carga de trabalho>), exemplo <1230510270902; 12>, onde 1230510270902 é o tempo em milissegundos³ e 12 é a carga de trabalho, ou seja, o número de *jobs* executando de forma concorrente. São necessários dois parâmetros para o *forecaster* gerar a previsão, a *raw prediction* e os dados da monitoração do recurso. A *raw prediction* e um multiplicador (η) informam o intervalo de tempo (*intervMonit*) necessário para recuperar os dados da monitoração, Equação 4.7. As monitorações mais recentes dentro desse intervalo de tempo são recuperadas e é calculada a média da carga de trabalho do recurso. Dessa forma, quando o *job* tem um tempo de execução menor, ele está sujeito à carga de trabalho mais próxima da atual. Quando este tempo é maior, a previsão é feita com uma maior quantidade de informação sobre o comportamento do recurso. Por exemplo para a *rawPrediction* = 100s e o $\eta = 2$, as monitorações feitas nos últimos 200s são recuperadas e a previsão da carga de trabalho é a média da carga monitorada neste intervalo de tempo.

A expressão para definir a janela de tempo do *forecaster* é:

$$intervMonit = \eta * rawPrediction \quad (4.7)$$

4.3.3 Predição adaptada

A predição adaptada é calculada a partir da lista de casos da fase de recuperação e da previsão da carga de trabalho, feita pelo *forecaster*. Após calculada a previsão da carga de trabalho, o NdrPredCase calcula a predição adaptada. A partir da lista de casos recuperados pelo Algoritmo 1, é feito um cálculo da similaridade entre as cargas de trabalho dos casos da lista e a carga prevista. Estes casos recuperados são filtrados extraindo-se apenas aqueles casos que executaram com a carga de trabalho média similar com a carga prevista. Esta similaridade é calculada para cada caso, a partir da Equação 4.8, na qual, quanto mais próximo de zero for o resultado maior é a similaridade. Também é definido um limite para a quantidade de casos que serão

³Baseado no tempo universal (UTC) que é dado pela diferença em milissegundos entre primeiro de janeiro de 1979 até a data que se quer armazenar.

recuperados, pois verificamos que quanto maior o número de casos, o coeficiente de correlação da regressão linear diminui e aumenta o erro da predição adaptada. Filtrados os casos, faz-se uma regressão linear utilizando a carga de trabalho média do recurso durante a execução do *job* e seu tempo total de processamento. A partir desta regressão linear chega-se a uma função do tempo de predição *versus* a carga de trabalho, Equação 4.9. E a partir da carga de trabalho prevista o tempo de execução do novo *job* é obtido.

A expressão para cálculo da similaridade entre a média da carga de trabalho para cada caso passado e a carga prevista pelo *forecaster* é:

$$sim(w_{est}, w_{caso_i}) = \frac{w_{est} - w_{caso_i}}{w_{est} + w_{caso_i}} \quad (4.8)$$

onde:

$sim(w_{est}, w_{caso_i})$: similaridade entre a carga de trabalho estimada e a carga de trabalho do caso passado;

w_{est} : carga de trabalho estimado pelo *forecaster*;

w_{caso_i} : carga de trabalho do *i-ésimo* caso recuperado da lista.

O tempo de execução em função da carga de trabalho obtido a partir da regressão linear é dada pela fórmula:

$$T(w) = aw + b \quad (4.9)$$

onde:

$T(w)$ é a predição do tempo de execução em função da carga de trabalho;

w é a carga de trabalho;

a e b são as variáveis obtidas pela regressão linear.

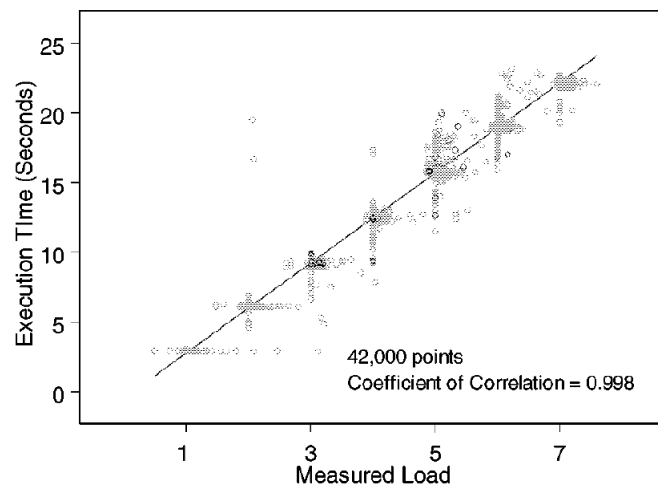


Figura 4.1. Relação entre a carga de trabalho média e o tempo de execução.
Fonte: [Dinda and O'Hallaron 2000].

A idéia de usar a regressão linear para a relação entre carga de trabalho e tempo de execução surgiu a partir de algumas observações feita pelo trabalho de [Dinda and O'Hallaron 2000]. A Figura 4.1 mostra o gráfico do tempo de execução *versus* a média da carga de trabalho experimentada durante a execução de tarefas, que consistiam de simples laços de espera, no qual chegou-se a uma relação perfeitamente linear com o coeficiente de correlação com o valor de 0.99. Segundo o artigo, os dados foram gerados rodando um número variável dessas tarefas de forma simultânea e em conjuntos. Esperar que um recurso do *grid* se comporte perfeitamente dessa forma é irreal, mas mesmo assim a análise ajudou a gerar nossa solução para calcular a predição adaptada. Pois tentamos chegar a esse resultado restringindo aos casos com carga de trabalho média bem próximas ao previsto pelo *forecaster*, limitando a uma pequena região do gráfico.

4.3.4 Predição ajustada

Nem sempre o NdrPredCase tem uma base de casos suficiente para calcular a predição adaptada, pois devido a restrições de similaridade da carga de trabalho, pode não haver casos suficientes com o grau de similaridade exigido para fazer a regressão. Quando isso ocorre é calculada a predição ajustada, pois é necessário ajustar a *raw prediction* para a carga de trabalho prevista com o objetivo de diminuir o erro de predição. A predição ajustada é uma modificação da predição adaptada do PredCase, que é realizada pela fórmula $predAd = V * pred$ [Nassif 2006], sendo $V = f(WP, WF)$, onde WP representa a carga de trabalho média dos casos passados e WF representa a carga de trabalho

média recente. Para o nosso método a fórmula foi modificada para a Equação 4.10, sendo $V = f(WP, WF)$, onde WF é a carga prevista pelo *forecaster* e WP a carga de trabalho média dos casos passados que foram utilizados para calcular a *raw prediction*.

$$predAjustada = V * rawPrediction \quad (4.10)$$

De forma resumida, os passos para o cálculo da predição pelo NdrPredCase, mostradas no Algoritmo 3, são:

1. Calcula a *raw prediction* utilizando o método n-WA, Equação 4.5, a partir da lista de casos recuperados pelo Algoritmo 1, linhas 3 e 4;
2. Prevê a carga de trabalho futura usando a *raw prediction* e os dados da monitoração dos recursos, armazenados na estrutura de dados MonitorResource, linha 5;
3. Filtra os casos recuperados pelo Algoritmo 1 com maior similaridade entre a carga de trabalho prevista pelo *forecaster* e a carga média de execução do caso passado, Equação 4.8, linha 7;
4. Calcula a regressão linear entre a carga de trabalho média e o tempo de execução dos casos filtrados, linha 9;
5. Calcula a predição adaptada a partir da carga de trabalho prevista utilizando a regressão linear, linha 11;
6. Caso a regressão seja nula, isto ocorre quando não há casos filtrados o suficiente, é calculada a predição ajustada utilizando a Equação 4.10, linhas 13 a 15 ;
7. O algoritmo retorna a predição final, ajustada ou adaptada, linhas 17.

4.4 Armazenamento

O NdrPredCase ainda não implementa a fase de revisão prevista no RBC e, portanto, os dados considerados relevantes de todos os *jobs* que executaram completamente num recurso são armazenados diretamente nas estruturas de dados Problema, Solução e Resultado, como descrito na Seção 4.1, na base de casos de tal recurso. Quando um novo caso é armazenado, se o limite máximo definido para a base é atingido, então o caso mais antigo é descartado. Esse processo de retenção das informações do *job* na base é chamado de fase de armazenamento da solução, sendo a última fase do ciclo

Algorithm 3 Algoritmo para cálculo de predição do tempo de execução em recursos não dedicados, NdrPredCase.

```

1: Entrada: descricaoDoNovoJob, baseDeCasos e monitorResource;
2: {Recuperação de casos utilizando o Algoritmo 1}
3: listaCasosRec  $\leftarrow$  RefinedNN(descricaoDoNovoJob, baseDeCasos);
4: rawPrediction  $\leftarrow$  nWA(descricaoDoNovoJob, listaCasosRec);
5: previsaoCargaTrabalho  $\leftarrow$  Forecaster(rawPrediction, monitorResource);
6: {Filtra os casos com maior similaridade de carga de trabalho}
7: listaCasosFiltrados  $\leftarrow$  FiltraCasos(listaCasosRec);
8: {Caso a regressão não seja possível, a função retorna null}
9: regressaoAdapt  $\leftarrow$  CalculaRegressaoLinear(listaCasosFiltrados);
10: if (regressaoAdapt  $\neq$  null) then
11:   predicaoFinal  $\leftarrow$  regressaoAdapt.predict(previsaoCargaTrabalho);
12: else
13:   WF  $\leftarrow$  previsaoCargaTrabalho;
14:   WP  $\leftarrow$  CalculaMediaCargaTrabalho(listaCasosRec);
15:   predicaoFinal  $\leftarrow$  PredAjustada(rawPrediction, WF, WP);
16: end if
17: Saida: predicaoFinal;

```

do RBC. No NdrPredCase os casos são armazenados de forma que os mais recentes possam ser recuperados primeiro, sendo esta característica fundamental para a fase de adaptação.

4.5 Conclusão

Neste capítulo apresentamos o desenvolvimento do NdrPredCase, nossa solução para calcular predição do tempo de execução de *jobs* para recursos não dedicados, uma extensão do PredCase. O NdrPredCase utiliza o paradigma de Raciocínio Baseado em Casos (RBC) para calcular predição a partir dos casos passados. Esses casos armazenam informações da descrição dos *jobs*, do cálculo da sua predição e de sua execução em um dado recurso, eles são organizados em uma base seqüencial em uma estrutura conhecida como *deque*. O NdrPredCase possui três fases para cálculo da predição: recuperação, reuso da solução (ou adaptação) e armazenamento. A fase de recuperação busca os casos da base mais similares ao novo *job* submetido, utilizamos o algoritmo *refined nearest-neighbor* para essa busca. A fase de reuso da solução utiliza diferentes técnicas para adaptar a solução obtida a partir dos casos passados, para a carga de trabalho prevista para o recurso. Nesta fase utilizamos diferentes técnicas: aprendizado baseado em instâncias, média aritmética e regressão linear. Uma vez que o *job* submetido foi executado com sucesso, a fase de armazenamento reterá esse caso na base.

Capítulo 5

Validação do NdrPredCase

A computação em *grid* é uma nova tecnologia para computação paralela e/ou distribuída. Ela agrega recursos heterogêneos dispersos para resolver vários tipos de aplicações da ciência, engenharia e comércio que demandam computação intensiva. Neste capítulo apresentaremos uma análise experimental do NdrPredCase. O objetivo dos experimentos é validar o NdrPredCase, verificando a acurácia da predição do tempo de execução de *jobs* e o seu desempenho. Para atender a estes objetivos iremos simular um ambiente *grid* e submeter ao mesmo uma lista de *jobs*. Para verificar a acurácia, iremos comparar o tempo previsto pelo NdrPredCase para cada *job* com o seu tempo de execução no simulador, utilizando as métricas que apresentaremos na Subseção 5.1.5. Para verificar o desempenho da nossa solução mediremos o intervalo de tempo gasto para gerar a predição.

5.1 Simulação

Para avaliar o desempenho de um ambiente *grid*, nós precisamos conduzir experimentos que possam ser repetidos e controlados, o que é difícil em um *grid* devido à sua heterogeneidade e sua natureza dinâmica. Além de gastar muito tempo e ser caro organizar um cenário de *grid* para executar os experimentos [Sulistio et al. 2005]. Devido a estas razões, nós optamos por usar um simulador.

5.1.1 GridSim

Para realizar nossos experimentos escolhemos o simulador GridSim [Buyya and Murshed 2002]. As motivações para sua escolha foram: por ser utilizado em vários trabalhos de *grid*, como por exemplo nos trabalhos de Singh et al.

[2007], Yeo and Buyya [2005] e Elmroth and Gardfjall [2005]; pela sua arquitetura bem definida; por ser de fácil utilização e alteração; desenvolvido em Java, e portanto não restrito a uma plataforma de desenvolvimento e execução da simulação; por possuir bibliotecas úteis para a implementação do nosso método, e pelo fato do MASK ter sido implementado em Java. Pesquisamos outros simuladores como o SimGrid [Casanova 2001], desenvolvido em C++, porém o GridSim atendeu melhor às nossas expectativas, além das características citadas, por ele ser mais abrangente, como, por exemplo, o GridSim implementa o escalonamento *time-shared* e *space-shared*, já o SimGrid implementa apenas o *time-shared*.

O GridSim é um *toolkit* abrangente para simular diferentes tipos de recursos heterogêneos, usuários, aplicações, *brokers* e escalonadores. Ele permite simular escalonadores de aplicações para um único ou vários sistemas distribuídos de diferentes domínios administrativos, tal como *clusters* e *grids*. Escalonadores de aplicações, também conhecidos como *brokers*, são capazes de descobrir recursos, selecionar e agregar um conjunto com vários recursos distribuídos para um usuário individual [Buyya and Murshed 2002]. A utilização de um *broker* permite ao usuário fornecer dados que otimizem o atendimento aos seus objetivos. O usuário pode informar quais são os requisitos da sua aplicação, o preço disposto a pagar pela execução do seu *job*, o tempo de execução, qualidade de serviços, entre outros.

O GridSim é baseado no SimJava [Howell and McNab 1998], um simulador de eventos discretos de propósito geral desenvolvido em JAVA. No SimJava cada sistema simulado que interage com outros é chamado de entidade [Sulistio et al. 2005]. As entidades podem representar usuários (*User*), *brokers* (*Broker*), recursos (*Resource*), serviços de informação (*Grid information service*), estatísticas e redes (*Link*, *Router*, *Packet*, entre outros), cada qual possuindo suas próprias opções de configuração, o diagrama de fluxo entre essas entidades é apresentado na Figura 5.1. Cada instância do *User* representa um usuário do *grid*, que por sua vez está conectado a uma instância do *Broker*. Um *job* do usuário é primeiro submetido para seu *Broker*, que então irá escalonar este *job* de acordo com os parâmetros definidos pelo *User*.

Para descobrir os recursos disponíveis no *grid*, o *Broker* envia uma requisição pedindo informação para o *Grid information service*, que envia uma lista de *Resources* disponíveis no *grid*. O *Broker* então, a partir da lista recebida e de um escalonador, submete o *job* a um *Resource*. Cada instância do *Resource* representa um recurso do *grid* e eles podem diferenciar entre si pelo número de processadores das máquinas, custo e velocidade de processamento, política interna de escalonamento de processo (*time-shared* ou *space-shared*), fator local da carga, fuso horário, entre outros. Os *Resources* precisam ser registrados no *Grid information service* para poder participar do *grid*, o

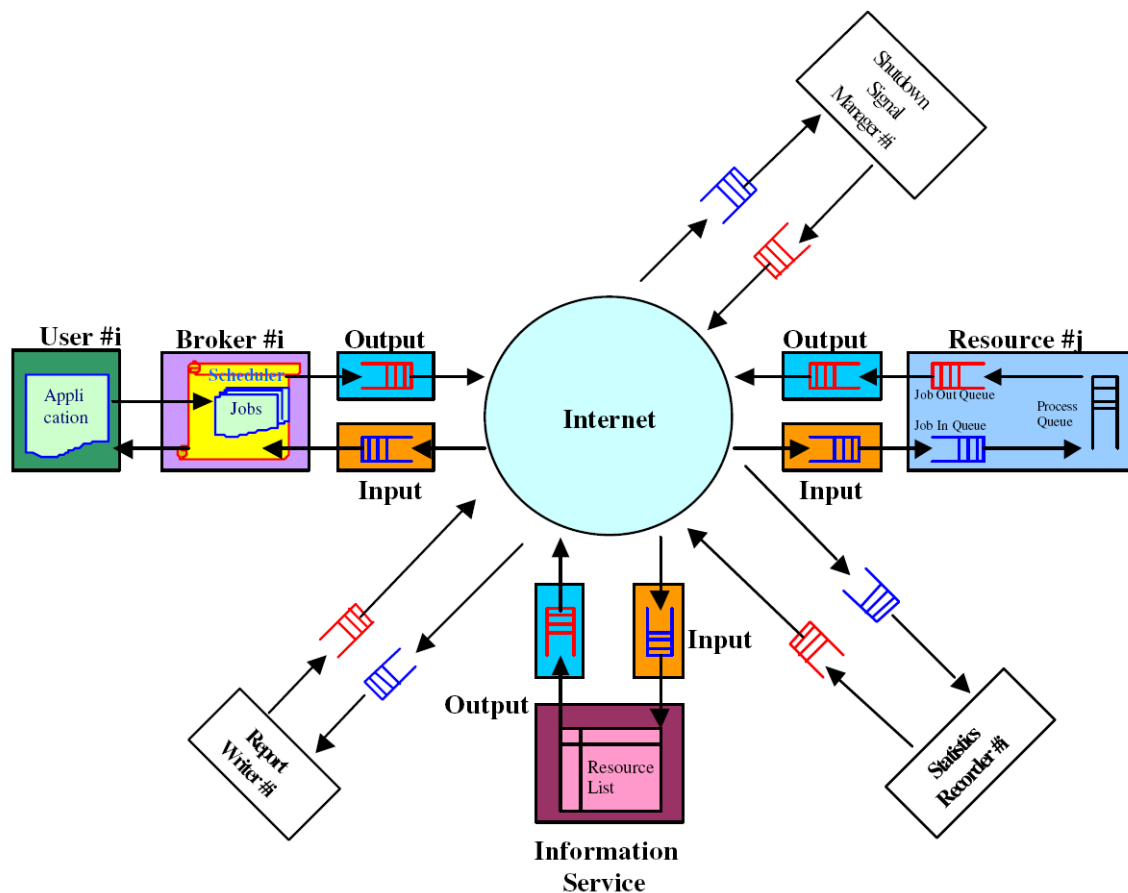


Figura 5.1. Diagrama de fluxo de comunicação entre as entidades do GridSim.
 Fonte: [Buyya and Murshed 2002].

que é feito no início da simulação. Quando um recurso termina de executar um *job*, ele o envia de volta para o respectivo *Broker* ou diretamente para o usuário, com as informações de sua execução, como tempo de execução, custo, entre outros. Durante a simulação o GridSim cria uma *thread* para cada entidade, e todas as entidades rodam em paralelo em sua própria *thread*. As entidades comunicam entre si sempre utilizando duas outras entidades desenvolvidas para controlar essa comunicação, a *Input*, para entrada de dados, e a *Output*, para saída de dados. Cada entidade possui seu próprio par de *Input* e *Output* [Buyya and Murshed 2002].

No GridSim cada tarefa pode requerer diferentes tempos de processamento e tamanhos do arquivo de entrada. Os requisitos de tais tarefas são definidos através do *Gridlet*. Um *Gridlet* é um pacote de dados que contém todos os elementos relacionados à execução *job*, tais como, o tamanho do *job*, expresso em MI (milhões de instruções), tamanho dos arquivos de entrada e saída, origem do *job*, entre outros. Esses parâmetros ajudam a determinar o tempo de execução, o tempo necessário para transferência dos arquivos de entrada e saída entre os usuários e os recursos remotos, e

também o endereço de retorno dos resultados da execução do *job* para quem o submeteu [Buyya and Murshed 2002].

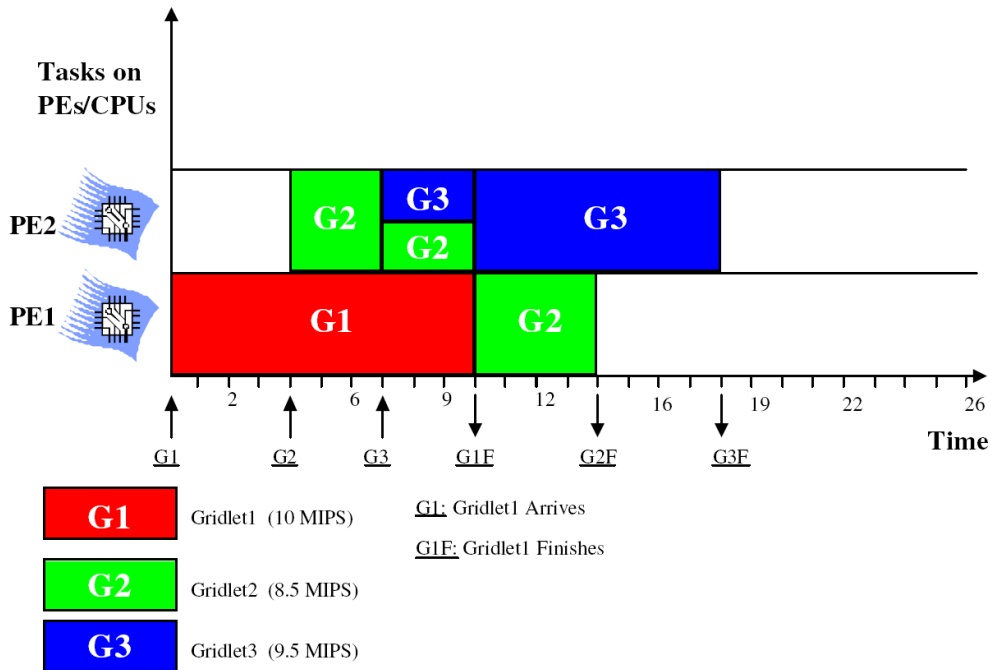


Figura 5.2. Modelo usado pelo GridSim para simular sistemas *time-shared*.
Fonte: [Buyya and Murshed 2002].

No simulador GridSim podemos criar elementos de processamento (PEs do inglês *Processing Elements*) com diferentes velocidades, que são medidas com padrões usados por *benchmarks*: MIPS ou SPEC (*Standard Performance Evaluation Corporation*). Um ou mais PEs podem juntos simular uma máquina. E uma ou mais máquinas podem juntas simular um recurso do *grid*. Portanto, um recurso do *grid* pode ser formado por um único processador, um multi-processador com memória compartilhada (SMP do inglês *Shared Memory Cluster*), ou por um *cluster* de computadores. Em nosso trabalho, como mostraremos, cada máquina é simulada como um recurso com um ou vários PEs. Esses recursos do *grid* são simulados utilizando o sistema de alocação de PEs *time-shared* ou *space-shared*. Como explicado no Capítulo 4, no *time-shared* quando um *job* é submetido num recurso ele inicia sua execução imediatamente e compartilha o recurso com todos os outros *jobs*, como mostrado na Figura 5.2. No *space-shared*, quando um *job* é submetido ele iniciará sua execução se houver um PE livre no recurso, caso contrário, será colocado na fila de espera, Figura 5.3. Em nosso trabalho estamos interessados em recursos não dedicados e portanto usamos o *time-shared*. Para implementar o *time-shared* o GridSim utiliza a política *round-robin* [Buyya and Murshed 2002].

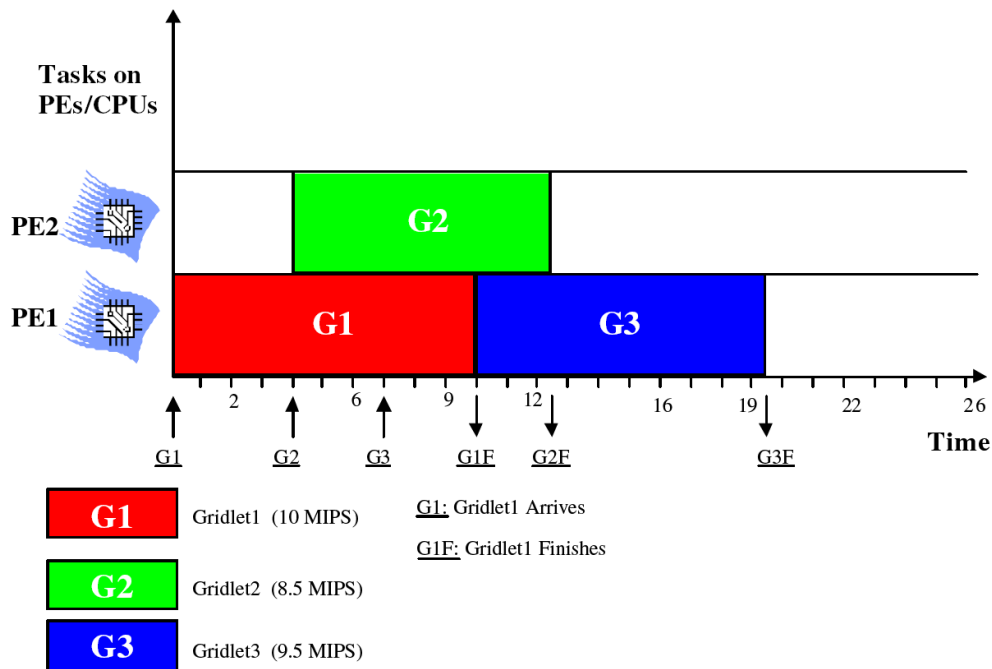


Figura 5.3. Modelo usado pelo GridSim para simular sistemas *space-shared*.
Fonte: [Buyya and Murshed 2002].

5.1.2 Simulando o MASK no GridSim

Nossa solução estende um dos componentes da arquitetura do MASK (*Multi-agent system resource broker*), o PredCase. Para criar nosso cenário, não simularemos todos os elementos da arquitetura do MASK, pois estamos interessados em avaliar somente a predição do tempo de execução de *jobs*. Uma visão geral da arquitetura do MASK é apresentada na Figura 5.4. Esta arquitetura consiste das camadas de negociação, seleção e interface [Nassif 2006].

Na camada de negociação estão agrupados os agentes e as funções relacionadas ao processo de negociação dos SLAs. Nesta camada é identificado quem está negociando, o que está sendo negociado, os tipos de negociações e como elas acontecem. Estas negociações são realizadas entre *user_agents* e *server_agents* ou entre *user_agents* e *network_agents* [Nassif 2006]. Devido ao objetivo do nosso trabalho, simulamos apenas as negociações entre os *user_agent* e os *server_agents*. O nosso método foi implementado no módulo de Predição desta camada. Ainda na camada de negociação é feita a descoberta de recursos disponíveis no *grid*, que possui um filtro para reduzir o número de recursos candidatos para a negociação. Em nossa simulação, este filtro determina de maneira aleatória quais recursos descobertos irão participar da negociação, possuindo um limite máximo para o número de negociações. O objetivo deste filtro é

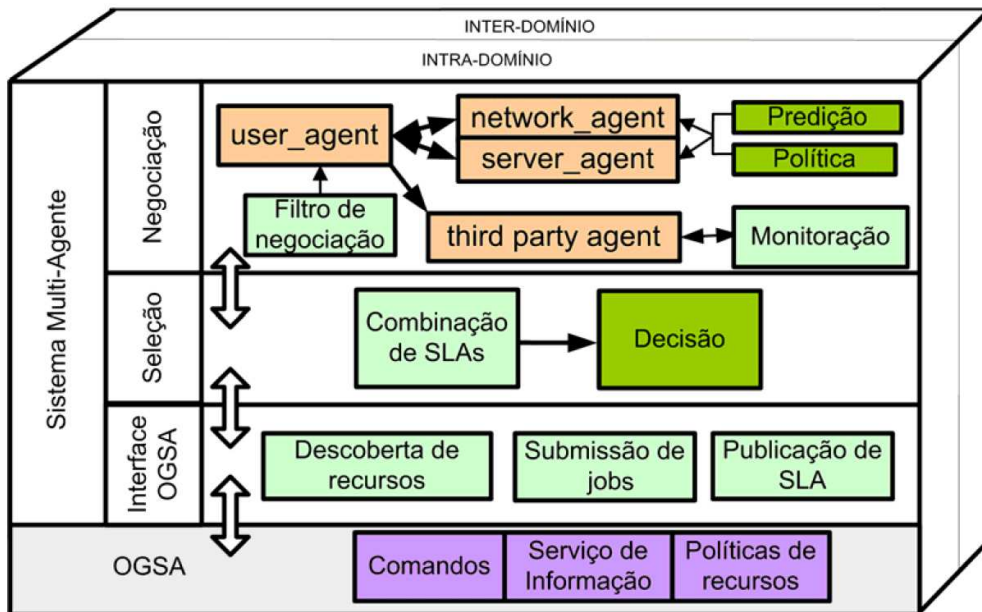


Figura 5.4. Arquitetura do MASK. Fonte: [Nassif 2006].

diminuir o número de recursos que farão parte da negociação para executar um *job*, o que melhora o desempenho da negociação em *grids* com muitos *jobs* sendo submetidos e com muitos recursos disponíveis. Como a melhor escolha destes recursos ultrapassa o escopo desse trabalho, esta escolha é feita aleatória. No GridSim desenvolvemos nosso próprio usuário e *broker*, os quais implementam o usuário do MASK e o *user_agent*, respectivamente. O *server_agent* foi desenvolvido a partir da classe *GridResource* do GridSim, adicionando a ela ferramentas para monitoração.

O módulo Monitoração da camada de negociação tem como objetivos, verificar o cumprimento do acordo do nível de serviço e coletar os dados do ambiente que contribuirão para a seleção do recurso, tais como a carga de trabalho no recurso e a taxa de sucesso na execução de *jobs*. A monitoração é realizada pelo *third_party_agent* e se inicia quando um *job* é submetido e dura até a finalização da sua execução [Nassif 2006]. Em nossa simulação, o *third_party_agent* foi desenvolvido na classe *GridResource*, e ele coleta informações somente da carga de trabalho do ambiente.

A camada de seleção é responsável pela escolha do recurso onde o *job* irá executar. A partir dos SLAs estabelecidos na camada de negociação (módulo Combinação de SLAs), como por exemplo preço, garantias e custo, essa camada toma a decisão de onde executar o *job* (módulo Decisão) [Nassif 2006]. Como não estamos considerando os possíveis problemas de redes, como congestionamentos e falhas, em nossa simulação o módulo Combinação de SLAs é desnecessário, como previsto no trabalho de Nassif

[2006] para este mesmo cenário. O módulo Decisão do MASK considera a preferência do usuário entre preço, desempenho e outros dados recebidos dos demais módulos. Porém, em nossa simulação consideramos somente o desempenho, ou seja, o recurso escolhido para submeter o *job* é aquele em que o tempo de execução previsto for o menor. Esse módulo foi implementado na entidade *Broker*.

A camada de interface é responsável pela integração do MASK com o *middleware* do *grid*, o sistema operacional, o usuário, o ambiente dos agentes e as aplicações [Nassif 2006]. Nossa simulação implementa dois dos três módulos dessa camada, o módulo Descoberta de recursos e o Submissão de *jobs*. Determinamos que todos os recursos são capazes de executar qualquer *job* e assim o módulo Descoberta de recursos retorna todos os recursos dos *grid*. Ele é implementado pelo *Grid information service* do GridSim. A submissão é feita usando uma função do GridSim chamada *gridletSubmit*, na qual os parâmetros são o *gridlet*, que representa o *job*, e o id do recurso em que o *job* irá executar.

Quando um recurso termina a execução de um *job*, ele sumariza os dados da monitoração feita durante sua execução, armazena estes dados junto com os dados de sua descrição na base de casos, e os envia para o *broker* ou para a entidade que representa o usuário (*User*). Em nossa simulação estes dados são enviados para o *User*.

5.1.3 Criação dos *jobs*

Para simular um usuário no GridSim precisamos, no mínimo, dos dados para criar os *jobs*, da quantidade desses *jobs* e dos horários que eles serão submetidos. Os dados necessários para criar um *job* no GridSim são: o identificador do *job*, o tamanho do *job* em milhões de instruções (MI), o tamanho do arquivo de entrada e o tamanho do arquivo de saída em *bytes*. Para estes dados utilizamos as aplicações da tese de Nassif [2006], capítulo 4 e Apêndice B, usados para validação do PredCase, apresentados na Tabela 5.1. Os comandos são constituintes de dois pacotes de aplicações chamados BLAST [Blast 2008] e HMMER [Hmmer 2008]. São pacotes utilizados com frequência por usuários de *grid*, mais especificamente na área de biotecnologia. O BLAST (*Basic Local Alignment Search Tool*) é um conjunto de aplicações científicas empregado em projetos de pesquisa de genomas. É formado por programas chamados *blastx*, *blastn*, *blastp*, *tblastn* e *tblastx*, incluídos no executável chamado *blastall*, além de outros programas especializados como o *blastgp*, *rpsblast* e *megablast* [Nassif 2006]. O HMMER é um pacote de aplicações usadas para seqüenciar genomas, que utiliza modelos de Markov (HMM-Hidden Markov Model) para identificar semelhanças em estruturas genéticas. Ele é formado por nove programas chamados *hmmalign*, *hmmbuild*,

Tabela 5.1. Dados para criação dos *jobs* para a validação do NdrPredCase. Cada linha representa um comando para executar um *job*. O tempo de duração foi obtido executando os comandos na máquina *Ixion*. Fonte: [Nassif 2006].

Caso	Comando	Entrada (bytes)	Duração (ms)
1	<code>hmmsearch -compat %OPT.HMM% swissprot</code>	2249	46994
2	<code>hmmsearch rrm.hmm nr</code>	0	4472390
3	<code>hmmsearch globin2.hmm swissprot</code>	0	403754
4	<code>hmmsearch fn3.hmm inputnrhmmsearch fn3.hmm inputnr</code>	31903	3348308
5	<code>hmmsearch weeviterbi_test.hmm swissprot</code>	27073	173132
6	<code>hmmsearch trace_test.hmm swissprot</code>	4313	37351
7	<code>hmmsearch fn3.hmm swissprot</code>	31903	209511
8	<code>hmmcalibrate pkin.hmm</code>	110154	24169
9	<code>hmmcalibrate globin.hmm</code>	53911	14149
10	<code>hmmsearch globin.hmm swissprot</code>	53911	352463
11	<code>hmmsearch globin.hmm yeast.aa</code>	53911	17883
12	<code>hmmsearch rrm.hmm yeast.nt</code>	0	24153
13	<code>blastall -p blastx -d nr -i nt.5706771 -e 0.1</code>	197867	30367698
14	<code>blastall -p blastp -d nr -i aa.129295</code>	406	92172
15	<code>blastall -p blastp -d nr -i aa.231729</code>	844	95582
16	<code>blastall -p blastp -d nr -i aa.1177466</code>	0	103438
17	<code>blastall -p blastp -d nr -i aa.p38398</code>	2160	555911
18	<code>blastall -p blastx -d nr -i nt.5764416 -e 0.1</code>	2143	98270
19	<code>blastall -p blastx -d nr -i nt.383410 -e 0.1</code>	20387	1249207
20	<code>blastall -p blastx -d nr -i nt.4883672 -e 0.1</code>	103810	11245595
21	<code>blastall -p blastn -d nt -i nt.383410 -e 0.1</code>	20387	907467
22	<code>megablast -d nt -i nt.5706771</code>	197867	627330
23	<code>blastall -p tblastn -d nt -i aa.129295</code>	406	803140
24	<code>blastall -p tblastn -d nt -i aa.231729</code>	844	1236514
25	<code>hmmsearch globin.hmm yeast.nt</code>	53911	197103
26	<code>hmmsearch pkin.hmm swissprot</code>	110154	819689
27	<code>myprog teste 100</code>	0	100000
28	<code>simple 30 10</code>	0	30000

`hmmcalibrate`, `hmmconvert`, `hmmemit`, `hmmfetch`, `hmmindex`, `hmmpfam` e `hmmsearch` [Nassif 2006].

A duração dos *jobs* foi obtida pela execução dedicada das respectivas aplicações numa máquina com processador Athlon 64 3200+ com 2 GHz, chamada Ixion. Para definir a quantidade e horário dos *jobs* submetidos utilizamos dois *logs* da carga de trabalho de *clusters* e *grids*, *LPC log* e *DAS-2 log*, que estão disponíveis num repositório [Workloads 2008] mantido pelo Departamento de Ciência e Computação da *Hebrew University of Jerusalem*. Esse repositório contém *logs* de *workloads* de várias entidades e universidades do mundo.

O *log* do LPC (*Laboratory of Corpuscular Physics* da Universidade *Blaise-Pascal, Clermont-Ferrand*, França [LPC 2008a]) contém informação da carga de trabalho observada durante dez meses, entre Agosto de 2004 a Maio de 2005. Neste período foram submetidos 244.821 *jobs*. O LPC é um *cluster* que faz parte do projeto EGEE (*Enabling Grids for E-science in Europe*) [EGEE 2008]. Ele é composto por 70 nós de 3GHz, *Pentium-IV dual* (140 CPUs), rodando o sistema operacional Linux [LPC 2008b]. Esse *log* foi usado apenas durante o desenvolvimento do método e não foi utilizado para a validação do NdrPredCase, para evitar que desenvolvesse um método com vícios para um determinado *log*.

O DAS-2 (*Distributed ASCI Supercomputer-2*, sendo ASCI, *Advanced School for Computing and Imaging in the Netherlands*) é um *grid* composto por cinco clusters mostrados na Tabela 5.2. As máquinas são *Pentium-III dual* com 1GHz cada. O *log* do DAS-2 contém informações dos *jobs* submetidos observadas durante janeiro de 2003 a dezembro de 2003, período em que foram submetidos um total de 432.987 *jobs* [DAS 2008]. Essas informações são: o identificador do *job*, o tempo que o *job* foi submetido, o tempo de espera para executar, o tempo de execução, memória usada entre outros, mostrado na Figura 5.5, nas colunas *job*, *submit*, *wait*, *runtime* e *mem used*, respectivamente. No Apêndice A apresentamos um estudo detalhado deste *log*.

Tabela 5.2. Clusters do DAS-2. Fonte: [DAS 2008].

#	Nome do Cluster	Localização	CPUs	<i>Jobs</i>
1	fs0	Vrije Univ. Amsterdarm	144	225.711
2	fs1	Leiden Univ.	64	40.315
3	fs2	Univ. of Amsterdam	64	66.429
4	fs3	Delft Univ. of Technology	64	66.737
5	fs4	Utrecht Univ.	64	33.795

Para relacionar os *jobs* da Tabela 5.1 com o *workload* é necessária uma fase de pré-processamento. Nesta fase, para cada linha lida do *log* do *workload*, que contém os dados do *job* que foi submetido, é criado um *job* com seu respectivo usuário e horário

job	submit	wait	run	proc	cpu	mem	proc	cpu	mem	uid	gid	qx	qp	pr	th		
			time	used	used	used	req					ues	it	job	time		
99010	11486561	1	2799	1	2785	32240	1	129600	-1	1	16	2	1	4	3	-1	-1
99011	11486616	4	4032	1	4012	35100	1	86400	-1	1	16	2	1	3	2	-1	-1
99012	11486736	3	1027	1	1015	31932	1	7200	-1	0	16	2	1	2	2	-1	-1
99013	11486737	3	3411	1	3396	31976	1	86400	-1	1	16	2	1	3	2	-1	-1
99014	11486796	3	3460	1	3444	31972	1	86400	-1	1	16	2	1	3	2	-1	-1
99015	11486797	3	3350	1	3325	34456	1	259200	-1	1	16	2	1	5	2	-1	-1
99016	11486799	3	3327	1	3305	35072	1	259200	-1	1	16	2	1	5	2	-1	-1
99017	11486844	3	2772	1	2758	32004	1	129600	-1	1	16	2	1	4	3	-1	-1
99018	11486855	3	3232	1	3217	32220	1	86400	-1	1	16	2	1	3	2	-1	-1
99019	11486857	3	3239	1	3224	32024	1	259200	-1	1	16	2	1	5	2	-1	-1
99020	11486924	2	2717	1	2702	32060	1	129600	-1	1	16	2	1	4	3	-1	-1

Figura 5.5. Exemplo do *log* utilizado para validação. Cada linha contém informações de um *job* que foi submetido.

de submissão incluído no *log*, colunas *job* e *uid* da Figura 5.5. Os comandos da Tabela 5.1 fornecem os outros dados do *job*, da qual cada linha é escolhida de forma aleatória. Ao final da leitura do arquivo do *log* todos os usuários e seus respectivos *jobs* foram criados. O tamanho do *job* é dado pelo seu tempo de execução dedicada na máquina *Ixion*, multiplicado pelo MIPS calculado para a mesma, que foi 6.217 MIPS obtido pelo software Sandra¹.

5.1.4 Cenário e parâmetros de simulação

Para validar nosso método utilizamos um número menor de CPUs para poder analisá-lo quando sujeito a diferentes e intensivas cargas de trabalho. Como o número de CPUs dos *grids* e dos *clusters* são muitos, obtidos dos *logs* utilizados, a maioria dos *jobs* acabariam executando quase que de maneira dedicada em nosso escalonador. Devido a isto, nós simulamos uma quantidade menor de CPUs para o sistema, além de simular máquinas com um número variado de *cores* e MIPS. As configurações das 18 máquinas simuladas estão descritas na Tabela 5.3 e o valor do MIPS foi obtido pelo software Sandra [Sandra 2008].

O NdrPredCase possui muitas constantes as quais precisam ser atribuídos valores, como mostrado nos algoritmos do Capítulo 4. Para nossa simulação, utilizamos os

¹Sandra é uma analisador de sistemas com módulos para comparar e testar todo o sistema e/ou seus componentes, além de obter informações sobre a CPU, o chipset, placa de vídeo, portas, rede, memória, entre outros [Sandra 2008].

Tabela 5.3. Configurações de máquinas para simulação.

Identificador	Nome	MIPS	CPUs	Configuração
1	Shiva	7.279	6 x 1	Pentium 3GHz
2	Kuja	16.954	6 x 2	Intel Core 2 duo 1.86GHz
3	Bahamut	48.438	6 x 4	Intel Core 2 quad 2.66GHz

seguintes parâmetros:

1. Quantidade máxima de negociações concorrentes para submeter um *job*: 4 negociações, entre 10% e 20% do número de recursos;
2. Tempo máximo de espera de resposta da negociação: 20 s;
3. Intervalo de tempo para monitorar o recurso e a execução do *job*: 10 s;
4. Multiplicador usado pelo *forecaster*, para definir a janela de tempo para a recuperação dos dados da monitoração da carga de trabalho recente em um recurso a partir da *raw prediction*: $\eta = 2$, Equação 4.7;
5. Número de dias de monitoração da carga de trabalho recente do recurso: 3 últimos dias;
6. Tamanho máximo da base de casos: 10.000 casos ;
7. Valores limites, mínimos, para o cálculo da porcentagem de similaridade², utilizado pelo Algoritmo 2:
 - Nome da aplicação: 1,0 (100%);
 - Parâmetros, ou argumentos, da aplicação: 0,9 (90%);
 - Tamanho do arquivo de entrada: 0,9 (90%);
8. Pesos atribuídos para cálculo da similaridade, indicando a importância de cada atributo, utilizados pelo Algoritmo 2:
 - Nome da aplicação: 1.0;
 - Parâmetros, ou argumentos, da aplicação: 1.0;
 - Tamanho do arquivo de entrada: 0.5;

²A tese de Nassif [2006] sugere que um caso deve ser recuperado quando sua similaridade for maior que 90%.

9. Valor máximo aceito para a relação entre a carga de trabalho média do caso e a carga prevista, usado no cálculo da predição adaptada: $sim(w_{est}, w_{caso_i}) < 0.1$, Equação 4.8 ;
10. Constante da *kernel function* do n-WA para o cálculo da *raw prediction*: $const = 3$, Equação 4.5 ;
11. Número máximo de casos passados usados para cálculo da *raw prediction*: $n = 10$ (10-WA), Equação 4.5 ;
12. Número máximo de casos passados para participar do cálculo da predição adaptada: 10;
13. Relação utilizada para o cálculo da predição adaptada do PredCase e para a predição ajustada do NdrPredCase: $f(WP, WF) = WF/WP$, Equação 4.10.

Alguns destes parâmetros, itens 4, 6, 9, 10, 11 e 12, são mais sensíveis para o cálculo da predição. Seus valores foram calculados durante simulações realizadas antes da simulação final. Analisamos os resultados das predições calculadas com diferentes valores para cada um destes parâmetros, e fixamos os valores que geraram os melhores resultados.

5.1.5 Métricas para validação do NdrPredCase

Para validar a acurácia da predição utilizamos duas métricas, o erro absoluto normalizado (α) e o erro relativo (δ) [Li 2007]. Estas duas métricas foram utilizadas nos trabalhos de predição de Li et al. [2005], Li and Wolters [2006] e Li [2007]. A métrica erro absoluto normalizado é calculada pela relação entre a média dos erros absolutos da predição e a média do tempo de execução do *job*, Equação 5.1. O erro relativo é a relação entre a diferença do tempo estimado (t_{est}) e o tempo real de execução (t_{real}) e a soma desses dois valores, Equação 5.2. A partir do erro relativo é possível construir um histograma no qual podemos visualizar quantos erros das previsões estiveram mais próximas de zero, bem como a variação do erro.

Erro absoluto normalizado:

$$\alpha = \frac{\sum_{i=1}^N \frac{|t_{est_i} - t_{real_i}|}{N}}{\sum_{i=1}^N \frac{t_{real_i}}{N}} \quad (5.1)$$

onde:

α = erro absoluto normalizado;

N = número total de *jobs* que executaram com sucesso;

t_{est} = tempo estimado para a execução do *job*;

t_{real} = tempo real de execução do *job*, *runtime*.

Erro relativo:

$$\delta = \frac{t_{est} - t_{real}}{t_{est} + t_{real}} \quad (5.2)$$

5.2 Resultados e análise da predição adaptada

Nesta seção iremos apresentar os resultados obtidos e uma análise dos mesmos. Restringimos a análise aos resultados da predição adaptada, sendo ela nosso objetivo principal. Diferenciamos dois grupos de *jobs*: um primeiro grupo com os *jobs* que executaram com uma média de carga de trabalho de até dez *jobs* executando de forma concorrente, o qual chamamos de cenário 10, e um segundo grupo com uma média de até cem *jobs* concorrentes, o qual chamamos de cenário 100. Em ambos cenários os *jobs* executaram em diferentes recursos, simulados com diferentes configurações de *hardware*. As predições analisadas foram calculadas com uma base de casos maior que 5.000 casos, isto para evitar que as condições iniciais das simulação interferissem nos resultados.

O objetivo da divisão em dois grupos, relacionada com o número médio de *jobs* concorrentes, foi estudar a acurácia do PredCase e o NdrPredCase quando submetidos a diferentes cargas de trabalho. O desempenho para o cálculo da predição foi praticamente o mesmo para os dois grupos, ficando em média 2,5 ms quando executado em uma máquina Dell com 4GB de memória principal e processador Intel Core 2 Quad de 2,66GHz, sendo um valor próximo a outras técnicas de predição, porém com bases de tamanho menor, além de ser bem menor do que o tempo gasto pelo PredCase. Do total de 432.986 *jobs* submetidos em 18 diferentes recursos, 107.982 foram descartados para evitar erros devido a condições iniciais da simulação, uma destas condições foi o tamanho mínimo da base de casos para que deveria ter pelo menos 5.000 casos na base de cada recurso, condição mínima para analisar a predição calculada, somente para esta condição foram descartados 90.000 casos.

5.2.1 Análise do cenário com no máximo dez *jobs* concorrentes (cenário 10)

Para os casos em que a quantidade máxima da média de *jobs* concorrentes é igual a dez, foram obtidos os seguintes resultados:

- Para 16,74% das predições realizadas, do total de 325.004 execuções de *jobs* analisados, foi possível calcular a predição adaptada do NdrPredCase. Isto pode variar de acordo com a quantidade e capacidade dos recursos disponíveis, com a quantidade de *jobs* submetidos e sua taxa de chegada, com o tempo de execução dos *jobs*, entre outros fatores do cenário da simulação e do *log do workload*;
- 78,00% dos *jobs* que utilizamos para este experimento executam com menos de 1.000 segundos, a Figura 5.6 mostra a distribuição do tempo de execução (*runtime*) dos *jobs* analisados;

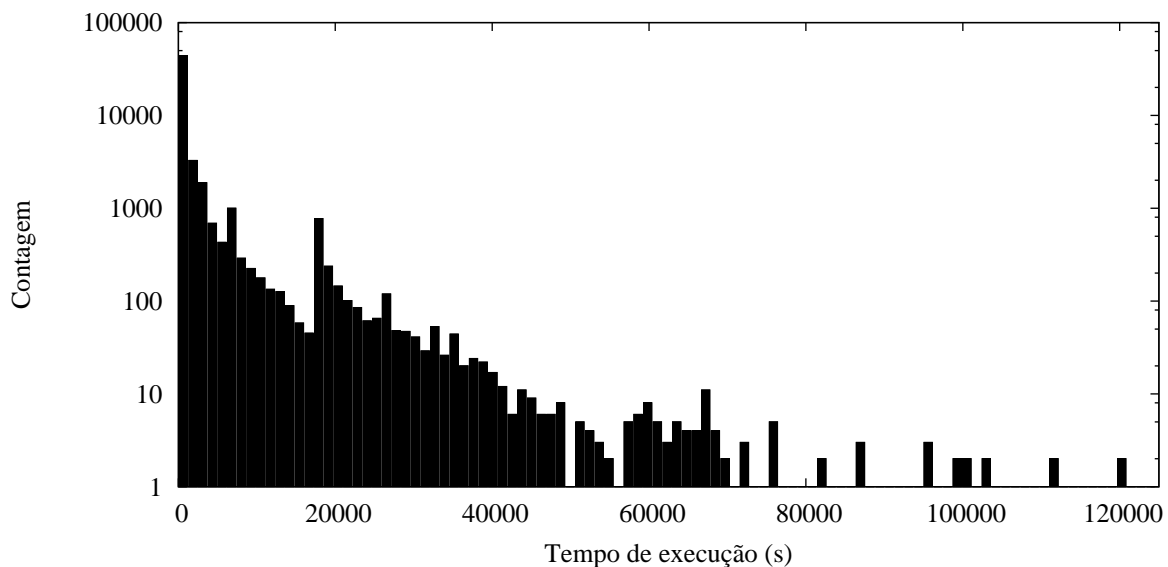


Figura 5.6. Distribuição do tempo de execução dos *jobs* analisados para o cenário 10.

Valores muito altos ou muito baixos quando comparado com a maioria dos valores em um conjunto são chamados de *outliers*. *Outliers* representam um problema quando não são causados por um real fenômeno do sistema. Nesse caso devem ser ignorados porque podem produzir um modelo inválido. Porém, se for possível a ocorrência de um *outlier* em um sistema real, ele deve ser incluído no sistema pois sua exclusão pode produzir um modelo inválido. Decidir quais *outliers* devem ser ignorados e quais devem ser incluídos faz parte da arte da avaliação de desempenho e requer um estudo cuidadoso do sistema que está sendo modelado [Jain 1991].

Para o PredCase consideramos *outliers* aqueles erros absolutos maiores do que 15.000 segundos, os quais correspondem a 2,10% do total das predições calculadas. Para o NdrPredCase os erros maiores do que 4.500 segundos são considerados *outliers*, os

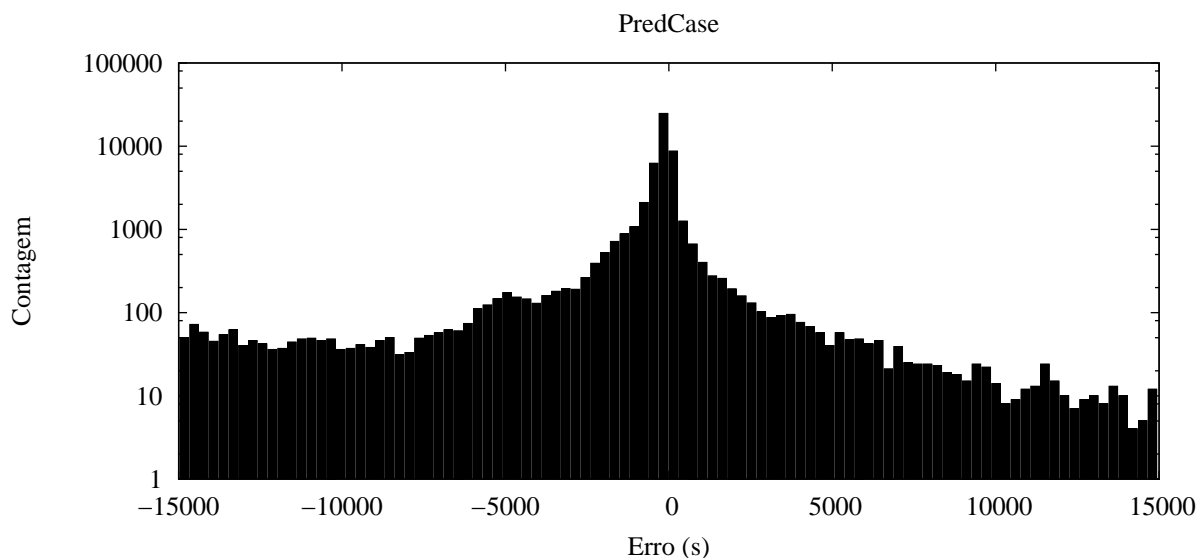


Figura 5.7. Distribuição da diferença entre o tempo previsto pelo PredCase e o tempo real da execução dos *jobs* analisados, sem *outliers*.

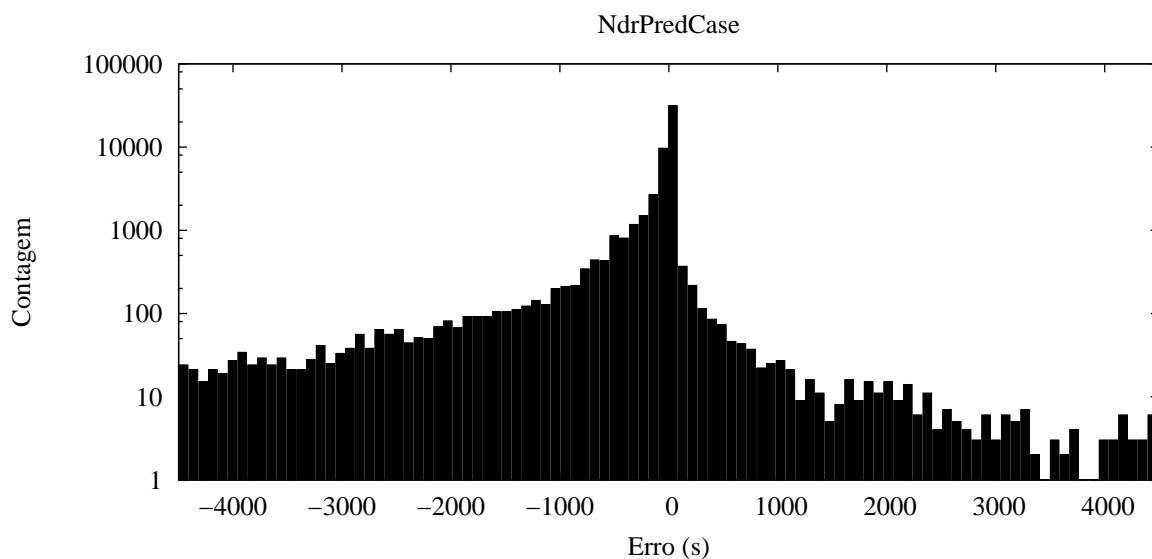


Figura 5.8. Distribuição da diferença entre o tempo previsto pelo NdrPredCase e o tempo real da execução dos *jobs* analisados, sem *outliers*.

quais correspondem a 2,20% do total de previsões adaptadas calculadas. Pelas Figuras 5.7 e 5.8, que quantificam os erros da previsão, podemos verificar que a distribuição dos erros não se aproxima da distribuição normal, pois são assimétricas, dificultando o cálculo do intervalo de confiança. Porém podemos constatar que o desvio padrão do PredCase é maior que o do NdrPredCase o que resultará em um resultado pior tanto

Tabela 5.4. Predições calculadas pelos métodos PredCase e NdrPredCase. Para os *jobs* que executaram com uma média de carga de trabalho de até dez *jobs* concorrentes. $\bar{x}_{|e|}$ representa a média dos erros absolutos com seu respectivo desvio padrão σ e α representa o erro absoluto normalizado.

Métrica	PredCase	NdrPredCase
$\bar{x}_{ e }$	882,21 s ($\sigma = 2117,87$ s)	174,67 s ($\sigma = 511,91$ s)
α	0,63	0,13

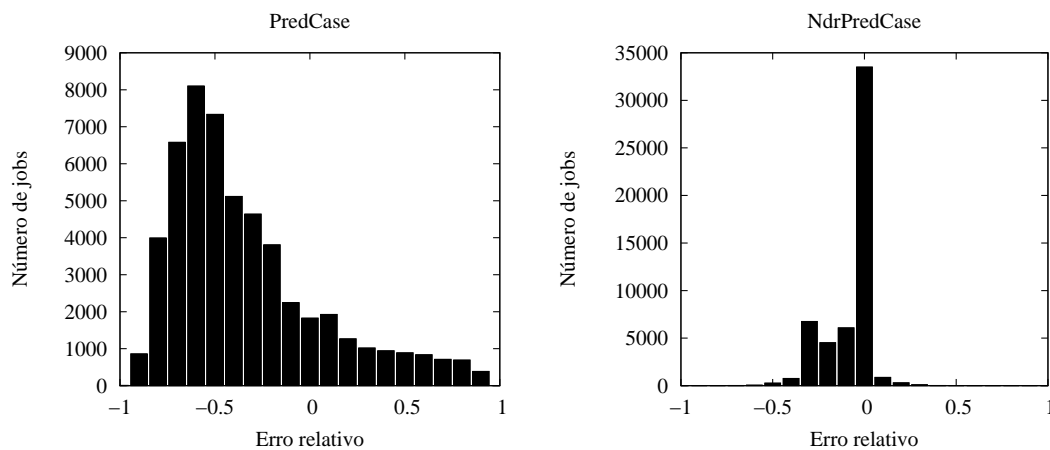


Figura 5.9. Erro relativo do PredCase e do NdrPredCase.

do erro absoluto relativo quanto do erro relativo.

A Tabela 5.4 e a Figura 5.9 apresentam os resultados encontrados. Apesar da média dos erros absolutos nos dois métodos apresentar um alto valor do desvio padrão e ter forte influência dos valores com maior tempo de execução, o NdrPredCase apresenta um erro médio absoluto menor para o cálculo da predição do que o PredCase, o erro absoluto normalizado ficou em 0,63 para o PredCase e 0,13 para o NdrPredCase. Com um maior erro médio absoluto e também com um maior desvio padrão, o PredCase apresenta os erros relativos piores daqueles do NdrPredCase, Figura 5.9. Este possui a maior parte dos erros relativos entre -0,05 e 0,05, com um total de aproximadamente 62% dos *jobs* avaliados, contra aproximadamente 3,7% do PredCase. Lembrando que o cálculo do erro relativo não tem relação direta com o valor da média absoluta dos erros e com a média do tempo de execução dos *jobs*.

5.2.2 Análise do cenário com no máximo cem *jobs* concorrentes (cenário 100)

Para os casos em que a quantidade máxima da média de *jobs* concorrentes é igual a cem, foram obtidos os seguintes resultados:

- Para 36,23% das predições realizadas, do total de 325.004 execuções de *jobs* analisados, foi possível calcular a predição adaptada. É uma fração maior do que na análise anterior por considerar uma quantidade maior do número de *jobs*;
- 77,00% dos *jobs* que utilizamos para este experimento executam com menos de 5.000 segundos, a Figura 5.10 apresenta a distribuição do tempo de execução dos *jobs* analisados;

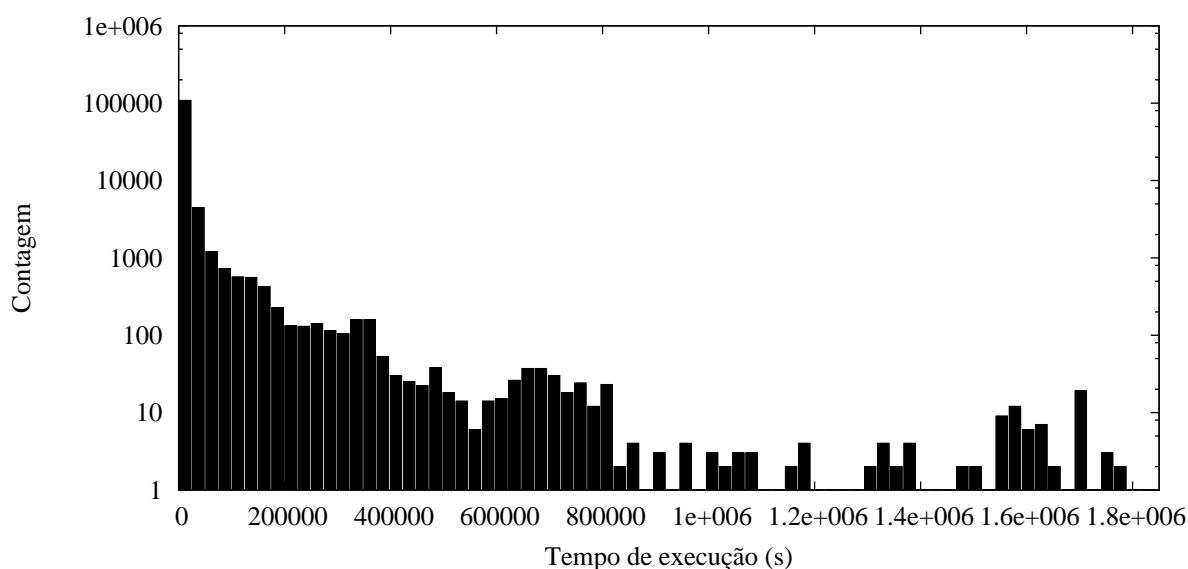


Figura 5.10. Distribuição do tempo de execução dos *jobs* analisados para o cenário 100.

A partir das Figuras 5.11 e 5.12 retiramos os valores que consideramos *outliers*. Para o PredCase são considerados *outliers* aqueles erros absolutos maiores do que 25.000 segundos, os quais correspondem a 4,39% do total das predições calculadas. Para o NdrPredCase os erros maiores do que 15.000 segundos são considerados *outliers*, e correspondem a 4,91% do total de predições adaptadas calculadas.

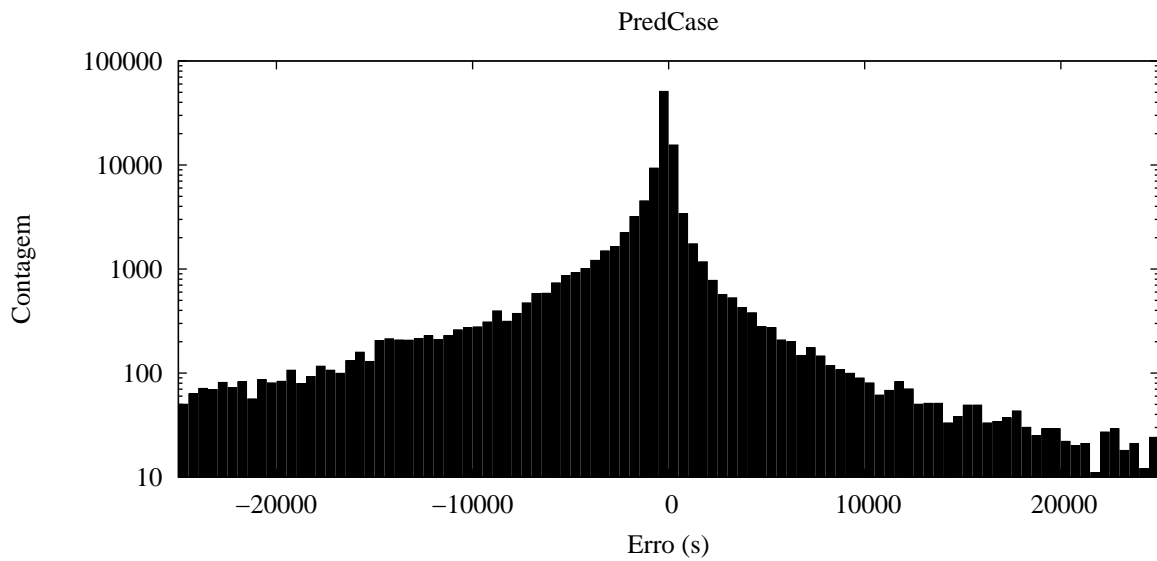


Figura 5.11. Distribuição da diferença entre o tempo previsto pelo PredCase e o tempo real da execução dos *jobs* analisados.

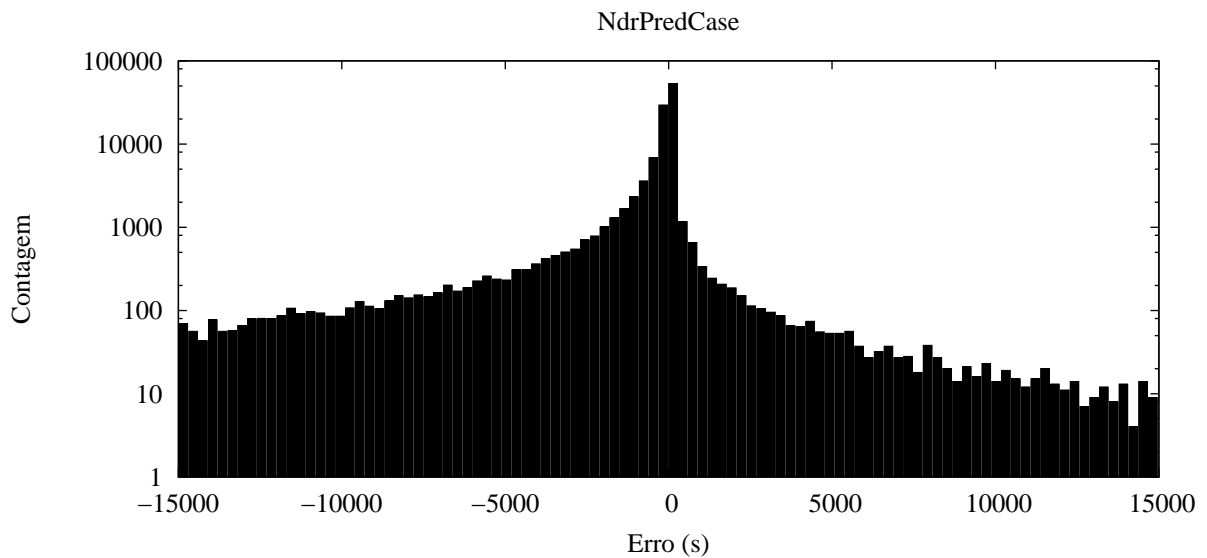


Figura 5.12. Distribuição da diferença entre o tempo previsto pelo NdrPredCase e o tempo real da execução dos *jobs* analisados.

A Tabela 5.5 mostra os resultados encontrados. Em relação à análise anterior, cenário 10, as médias dos erros e os respectivos desvios padrão continuam com valores altos, com o NdrPredCase mantendo o erro médio absoluto e desvio padrão menores do que o PredCase. Já o erro absoluto normalizado (α) melhorou muito para o PredCase, e o NdrPredCase teve uma leve piora, indicando que o PredCase trabalha melhor quando

submetido a maiores cargas de trabalho do que quanto submetido a menores cargas, ao contrário do NdrPredCase. Lembrando que o PredCase também é um método para calcular previsões para recursos não dedicados. Quanto aos erros relativos do NdrPredCase, Figura 5.13, apesar de terem diminuído levemente, aproximadamente 58% estão na faixa entre -0,05 e 0,05, os erros a partir dessa faixa estão mais decrescentes. Para o PredCase, seus erros relativos, apesar de ainda estarem longe do ideal, apresentaram uma melhora significativa com uma proporção de 8,5% de erros na faixa entre -0,05 e 0,05.

Tabela 5.5. Resultados do cálculo da previsão pelos métodos PredCase e o NdrPredCase.

Métrica	PredCase	NdrPredCase
$\bar{x}_{ e }$	1.745,28 ($\sigma = 3.621,03$)	706,57 ($\sigma = 1.933,86$)
α	0,36	0,16

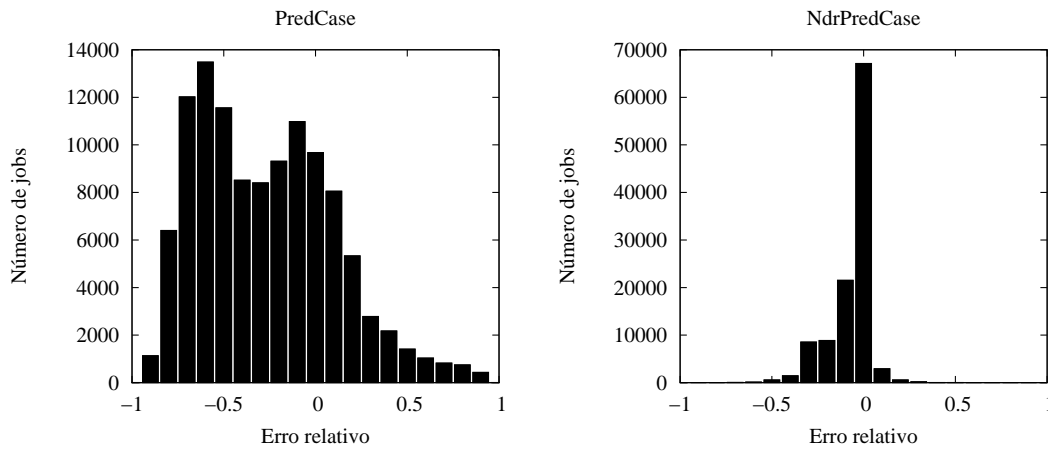


Figura 5.13. Erro relativo do PredCase e do NdrPredCase.

As Figuras 5.14 e 5.15 mostram os gráficos do comportamento do erro absoluto normalizado (α) do PredCase e do NdrPredCase, respectivamente. Podemos observar, que para o PredCase o erro é muito grande quando poucos *jobs* estão executando de forma concorrente, diminuindo quando esta quantidade aumenta. Já para a previsão adaptada do NdrPredCase o erro aumenta quando a carga de trabalho aumenta, porém tende a oscilar entre 0,16 e 0,18, quando ultrapassa os 25 *jobs* concorrentes. A Figura 5.16 mostra a distribuição da carga de trabalho, na qual podemos observar que grande parte dos *jobs* executam com uma carga menor do que 26 *jobs* concorrentes.

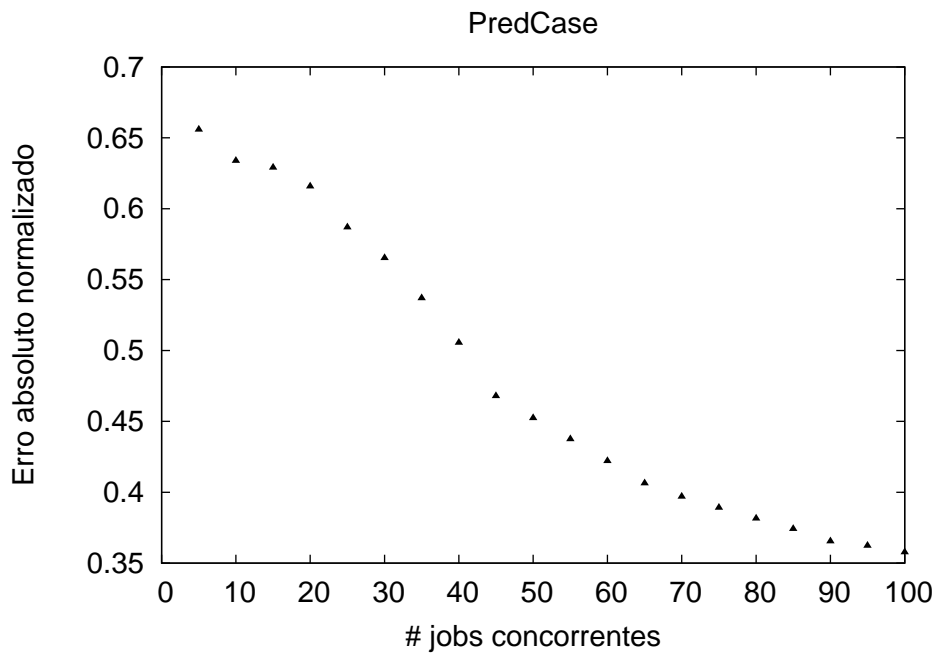


Figura 5.14. Comportamento do erro absoluto normalizado (α) do PredCase.

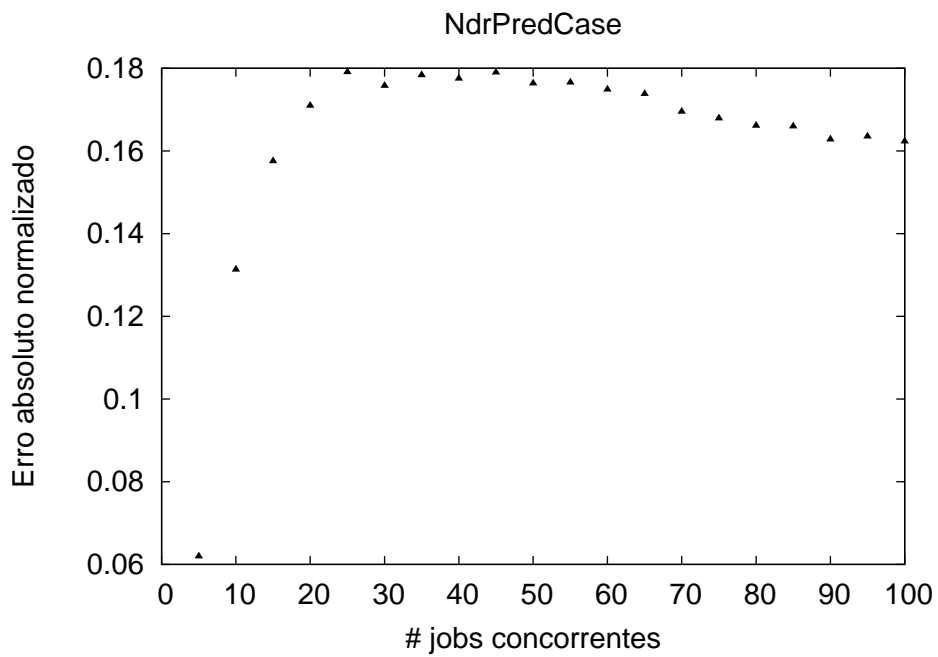


Figura 5.15. Comportamento do erro absoluto normalizado (α) do NdrPred-Case.

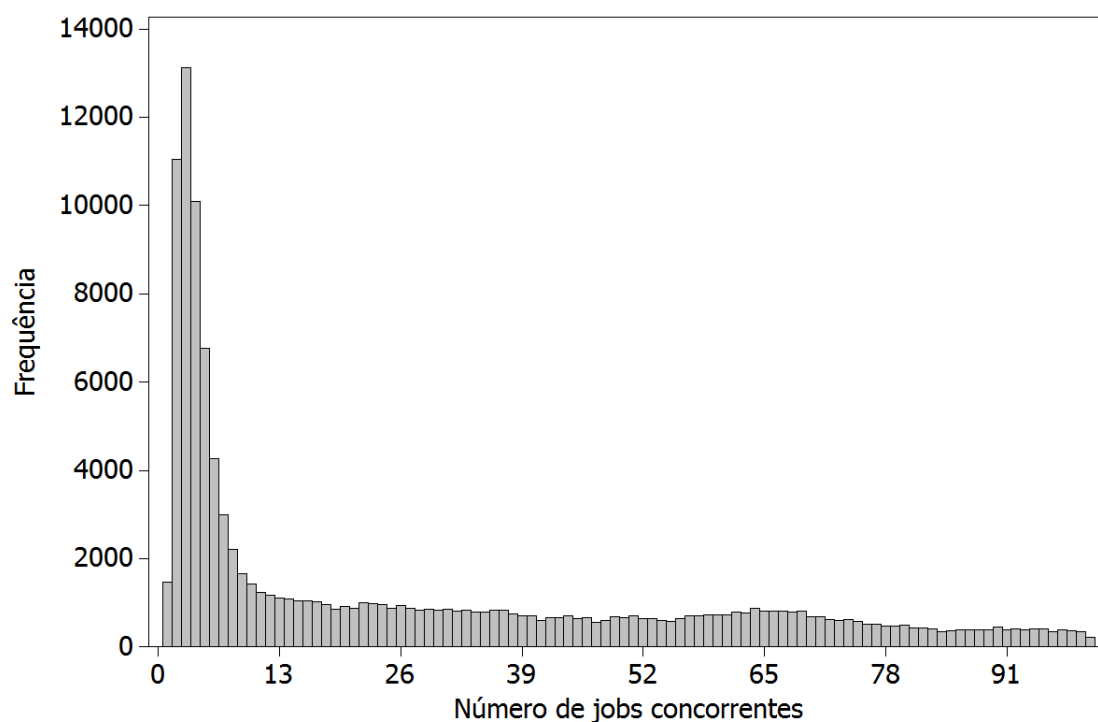


Figura 5.16. Distribuição da carga de trabalho média durante a execução dos *jobs*.

5.3 Conclusão

Neste capítulo apresentamos uma análise experimental do PredCase e do NdrPredCase. Para fazer esta análise simulamos parte da arquitetura do MASK no simulador GridSim, principalmente os módulos que se relacionam com o cálculo da predição. Apresentamos alguns modelos utilizados pelo GridSim e suas principais entidades que o compõem e como essas entidades e outras novas foram desenvolvidas para implementar o MASK. Configuramos um cenário com três tipos de máquinas (recursos) do nosso laboratório, que possuem diferentes números de elementos de processamento (ou *core*). Para a criação dos *jobs* utilizamos uma lista de aplicações que foram utilizados na tese de doutorado de Nassif [2006], que possui comandos de dois pacotes de aplicações chamados BLAST [Blast 2008] e HMMER [Hmmer 2008]. Para submissão dos *jobs* utilizamos o *log* do *workload* DAS-2 [DAS 2008].

A análise dos resultados se restringiu à predição adaptada. Para a análise da acurácia da predição nos baseamos principalmente em duas métricas: erro absoluto normalizado e erro relativo. Com o objetivo de estudar o PredCase e o NdrPredCase com diferentes cargas de trabalhos dividimos a nossa análise em dois grupos, um com os

jobs que executaram com uma média da carga de trabalho de até dez *jobs* concorrentes e outro com até cem *jobs* concorrentes. Nos experimentos verificamos que o erro médio absoluto da predição tem um desvio padrão muito alto o que prejudica a análise pela métrica erro absoluto normalizado. Porém, o cálculo do erro relativo não possui relação direta com a média e podemos analisar de forma mais segura os resultados. Para as duas métricas o NdrPredCase foi superior ao PredCase nos dois grupos, possuindo um erro absoluto normalizado menor e os erros relativos ficando em sua maior parte próximos a zero. Entretanto, quando se aumenta o número de *jobs* concorrentes, o NdrPredCase tem seu conjunto de resultados levemente piorado, enquanto o PredCase melhora consideravelmente. Quanto ao desempenho, o nosso método gastou em média 2,5 ms para cálculo da predição nos dois grupos analisados.

Como mostrado no Capítulo 4 o NdrPredCase possui vários parâmetros e previsores que precisam ser analisados para responder várias perguntas que surgem, como por exemplo:

- Para a previsão da carga de trabalho:
 - Qual a relação entre o erro da previsão da carga de trabalho, feito pelo *Forecaster*, com o erro da predição adaptada?
 - Qual a influência da variação dos dados monitorados com o erro da previsão da carga de trabalho?
 - Existe um índice de tendência central melhor do que a média dos valores monitorados utilizados pelo *Forecaster*?
- Para a carga de trabalho durante a execução dos *jobs*:
 - Qual a relação entre a variação da carga de trabalho com o tempo de execução de um mesmo *job*?
 - A média é o melhor indicador para avaliar a carga de trabalho durante a execução do *job*?
- Para o cálculo da predição:
 - Outros métodos como o LWR (*Locally weighted regression*), mostrado na Seção 2.1, ponderado pela carga de trabalho, podem gerar melhores predições adaptadas do que a regressão linear?
 - Qual a relação entre os indicadores de variação da regressão linear com a acurácia da predição adaptada?

- Quando a predição adaptada não for possível de ser calculada, qual o melhor previsor, a *raw prediction*, a predição ajustada ou o PredCase?
- Qual seria o ponto ótimo que uma menor similaridade entre as cargas de trabalho prevista e dos casos recuperados para o cálculo da predição adaptada, iria gerar uma melhor predição final? Apesar que isso diminuiria o coeficiente de determinação da regressão linear utilizada pela predição adaptada. Entretanto, seria possível calcular a predição adaptada para mais *jobs*, uma vez que ela apresenta melhor acurácia do que as outras predições calculadas.

Estas questões têm uma grande influência na acurácia da predição do NdrPredCase e respondê-las irá garantir um melhor método para o cálculo da predição de *jobs* para recursos não dedicados. Apesar de não estar no escopo deste trabalho a busca de respostas para essas questões a continuidade de pesquisa seria bastante interessante e poderia chegar a novos e melhores resultados.

Gostaríamos de comparar nossa solução com outras para recursos não dedicados, como o trabalho de Liu et al. [2002]. Entretanto, como a parte experimental desse trabalho não é bem explorada pois baseia seus resultados em um valor médio sem pelo menos calcular sua variação seria necessário implementar tais métodos para poder compará-los com a nossa solução. Outro problema é quanto à nossa análise, ela foi feita apenas quando é possível calcular a predição adaptada, sendo necessário um estudo de todos os resultados gerados, ou seja, fazer uma análise da predição final.

Capítulo 6

Conclusão

O trabalho desenvolvido abordou o problema de predição do tempo de execução de *jobs* para recursos não dedicados para grades computacionais (*grids*). *Grids* são estruturas heterogêneas, dinâmicas e com diversos recursos que podem estar distribuídos geograficamente, e com diferentes máquinas integrando o sistema. Sua utilização por parte dos usuários também é bastante diversificada, sendo difícil traçar um perfil de utilização do sistema por um longo período de tempo, tanto em relação ao número de *jobs* quanto aos tipos de aplicações submetidas. Devido a essa complexidade torna-se difícil desenvolver escalonadores de *jobs* e *brokers* baseados na configuração do recurso ou pela combinação de modelos de aplicação, para obter um melhor escalonamento, ou uma melhor seleção de recurso para submeter um *job*. Para contornar este problema, alguns escalonadores e *brokers* utilizam a predição do tempo de execução de *jobs* como a principal informação para atender seus objetivos. Essa predição pode fornecer de forma indireta informações sobre qual recurso possui o melhor poder de processamento, qual está mais ocioso, entre outras. Assim, um escalonador ou um *broker* poderão escolher o recurso que tem uma maior probabilidade de executar o *job* em menor tempo.

Nosso método, denominado NdrPredCase, calcula a predição do tempo de execução de *jobs* para recursos não dedicados utilizando uma base de casos passados. O NdrPredCase foi desenvolvido a partir do paradigma Raciocínio Baseado em Casos, no qual são definidas as fases que orientam quanto a recuperação, adaptação da solução e armazenamento. Também foi utilizada a técnica de Aprendizado Baseado em Instâncias que pondera a participação de cada caso da base em relação ao novo caso de acordo com sua similaridade. O NdrPredCase estende o módulo PredCase do MASK, um *middleware* com função de *broker* desenvolvido na tese de doutorado de Nassif [2006].

Para a validação do NdrPredCase nós restringimos a avaliar somente a predição

adaptada e seu desempenho, devido a sua importância em adaptar a solução obtida para diferentes cargas de trabalho. Utilizamos um simulador de *grid* chamado GridSim, para implementar parte da arquitetura do MASK relacionada ao cálculo da predição. Dividimos a análise dos resultados em dois grupos com diferentes limites para o número médio de *jobs* concorrentes durante a execução do *job*, para verificar o comportamento do método quando submetido a diferentes cargas de trabalho. Os resultados foram muito bons nos dois grupos estudados mostrando um método mais eficiente e com melhor desempenho do que o PredCase, para recursos não dedicados. O desempenho para o cálculo da predição também teve uma grande melhora gastando em média 2,5 ms para seu cálculo. Estes resultados indicam que o NdrPredCase atende nossas expectativas quando o cálculo da predição adaptada for possível, ou seja, quando existe uma base de casos com número suficiente de casos passados.

Para desenvolvimento do NdrPredCase houve várias tentativas que resultaram em ganho quase nulo ou em perda da acurácia da predição. Entre estas tentativas destacam-se:

- A análise da taxa de chegada de *jobs* durante o dia e durante a semana, a partir do *log* da carga de trabalho de um *grid*, com o objetivo de chegar a um previsor para esta carga não gerou bons resultados. A carga de trabalho a que o *grid* está sujeito pode não refletir nos recursos que o compõe, pois os recursos sofrem influência do escalonador, do número de recursos disponíveis, das políticas, do tamanho dos *jobs*, entre outros. Isto faz com que haja uma grande diferença entre a carga de trabalho de cada recurso e aquela observada para o *grid*, sendo muito difícil chegar a um único método para todos as diferentes configurações dos recursos;
- Ainda para prever a carga de trabalho, desenvolvemos dois métodos baseados na média e na regressão linear para gerar previsões a partir do estudo dos seus indicadores de qualidade, como variância, coeficiente de determinação, entre outros. Porém a regressão não funcionou, pois percebemos que a carga de trabalho varia muito, fazendo previsões de cargas de trabalho muito grandes ou até mesmo negativas. Portanto, mantemos apenas o previsor baseado na média da monitoração da carga de trabalho;
- Desenvolvemos um método com o objetivo de ajustar uma nova solução a partir do erro das soluções passadas, porém ainda não conseguimos obter uma relação matemática entre o erro da nova solução com aqueles das soluções passadas.

Resumindo as contribuições deste trabalho foram:

- Desenvolvemos um método para calcular predição para o tempo de execução de *jobs* para recursos não dedicados;
- Implementamos e analisamos o PredCase para recursos não dedicados e comparamos com a nossa solução;
- Apresentamos os trabalhos mais relevantes da literatura para geração de predição baseado em execuções passadas;
- Desenvolvimento de um método para predição distribuída, validado em um simulador de *grid*;
- Desenvolvemos grande parte da arquitetura do MASK, baseada em agentes, em um simulador de *grid*.

6.1 Trabalhos futuros

Este trabalho sugere pesquisas em diversas direções a partir do método desenvolvido.

O NdrPredCase depende fortemente de métodos que sumarizam a variação dos dados e indicam a tendência central. Depende também de modelos de previsão para gerar soluções a partir de um histórico de dados para prever a carga de trabalho e para calcular as predições. Um trabalho interessante seria estudar isoladamente o comportamento dos dados utilizados por cada fase do cálculo da predição, e a partir desse estudo desenvolver um método que possa alternar entre, por exemplo, os diferentes índices de análise de tendência central como média, mediana e moda. Também seria interessante um estudo para verificar a eficiência de diferentes modelos como o *Locally Weighted Regression* (LWR) em troca da regressão linear para a predição adaptada, ou autoregressão para prever a carga de trabalho de intervalos de tempos pequenos. Além de estudar os indicadores da regressão linear, verificando aqueles que interferem mais no modelo.

Na fase de adaptação o NdrPredCase restringe a um número de casos constante que serão recuperados baseados na similaridade da carga de trabalho e dos atributos. Entretanto, nada garante que os casos mais similares quanto à carga de trabalho e também quanto aos atributos sejam recuperados. Organizar a base de casos em um *metric space* utilizando o método *M-Tree* para pesquisar os casos poderia reduzir esta limitação, além de melhorar o desempenho em bases de casos muito grande, pois nesses métodos não é necessário percorrer todos os casos da base. Outro problema da base de

casos é quanto ao descarte de casos: sempre que se atinge um limite definido os casos mais antigos são descartados. Apesar desta solução manter os casos mais recentes na base, quando grandes quantidades de um mesmo tipo de *job* são submetidos em um intervalo curto de tempo, submissão em *bags*, a base fica muito especializada para aquele tipo de *job*. Isto ocasiona a perda da experiência adquirida para outros tipos de *jobs* que poderão eventualmente ser submetidos novamente, ocasionando grandes erros de predição. Neste caso, seria interessante um método de descarte que relacionasse os principais tipos de *jobs* submetidos, sua freqüência e a quantidade desses tipos na base de casos. Ou até mesmo avaliar a acurácia da predição e o desempenho do método quando nenhum caso é descartado.

Uma outra abordagem interessante seria estudar técnicas para as fases de revisão da solução atreladas ao armazenamento de casos. Analisando quais casos geraram boas predições, se um caso merece ser armazenado, se não, qual seria a possível modificação neste caso para ele ser armazenado, como previsto no paradigma RBC. Analisar também a permanência dos casos passados, algoritmos para remoção de casos de acordo com regras pré-determinadas, tais como remover casos não usados ou casos com influência negativa na predição.

Nossa análise de resultados abordou somente a predição adaptada, em um cenário com constantes definidas. Seria interessante realizar experimentos analisando a relação entre cada uma dessas constantes para o valor final da predição, como, por exemplo: quantos casos devem participar do cálculo da *raw prediction* e da predição adaptada; qual o valor ótimo a que podemos diminuir o limite da similaridade exigida para carga de trabalho para a predição adaptada sem comprometer o resultado final. Quanto maior esta limitação, menor é o número de predições que são adaptadas.

Outros estudos interessantes seriam: adição de mais atributos para o cálculo da similaridade, como o usuário ou o grupo a que ele pertence; desenvolvimento de um método capaz de identificar em uma nova solução problemas que causaram insucesso em soluções passadas; analisar se existe uma relação entre o erro da predição e o erro do previsor de carga de trabalho.

Outra extensão para este trabalho seria a qualidade da predição do tempo de execução do *job*. Os trabalhos de predição baseados em históricos de casos calculam a predição a partir de uma base que pode ter mais ou menos casos similares ao *job* submetido, podendo gerar melhores ou piores predições respectivamente. Definir uma métrica para a qualidade que fosse passada ao *broker* ou ao escalonador junto com a predição iria permitir a ele escolher o menor tempo de execução com a melhor qualidade, diminuindo o erro da predição.

Apêndice A

Um estudo de caso do *grid* DAS-2

Neste apêndice apresentamos um estudo do *log* da carga de trabalho do *grid* DAS-2, quanto à sua utilização pelos usuários, tipos de *jobs* submetidos e sua taxa de chegada, além de outras características. Nosso objetivo é conhecer a carga de trabalho em um *grid* para estudo de sua previsão e para realizar a validação do nosso modelo.

O trabalho de Li et al. [2004] apresenta uma análise dos *traces* dos *jobs* executados no supercomputador *multi-cluster* DAS-2 [DAS 2008] durante todo o ano de 2003. O DAS-2 consiste de cinco clusters, chamados fs0, fs1, fs2, fs3 e fs4, localizados em cinco universidades holandesas, usado principalmente para pesquisas científicas. Seu objetivo principal é o de garantir uma resposta rápida e manter processadores disponíveis para os pesquisadores das universidades participantes, e não o de atender a grande quantidade de usuários. O DAS-2 utiliza uma política especial para escalonar seus *jobs* na qual não é permitido que um recurso execute mais de um *job* ao mesmo tempo, ou seja, cada *job* roda de maneira dedicada (*space shared*).

A média de utilização do DAS-2 durante o ano de 2003, mostrado na Figura A.1, possui uma tendência de variação diária na utilização dos *clusters*, que tem grande relação com a quantidade de *jobs* que chegam durante o dia, mostrada na Figura A.2. Os autores afirmam que apesar da taxa de utilização do *cluster* ser menor durante a noite, *jobs* noturnos costumam requerer mais processadores e mais tempo de execução que os *jobs* submetidos durante o dia.

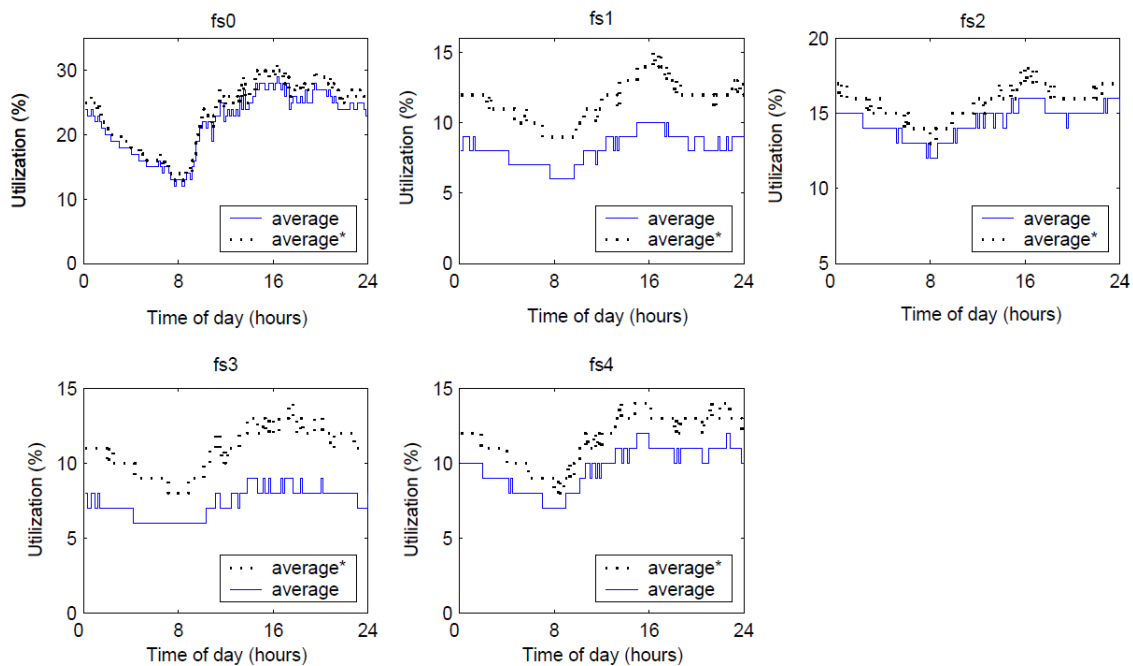


Figura A.1. Utilização do DAS-2 em função do horário do dia, observada durante o ano de 2003. "Average" é a média da utilização em todos os dias no ano. "Average*" é a média da utilização para todos os dias ativos no ano, dos quais são excluídos o intervalo de tempo que o sistema ficou ocioso e os dias sem chegada de *jobs*. Fonte: [Li et al. 2004].

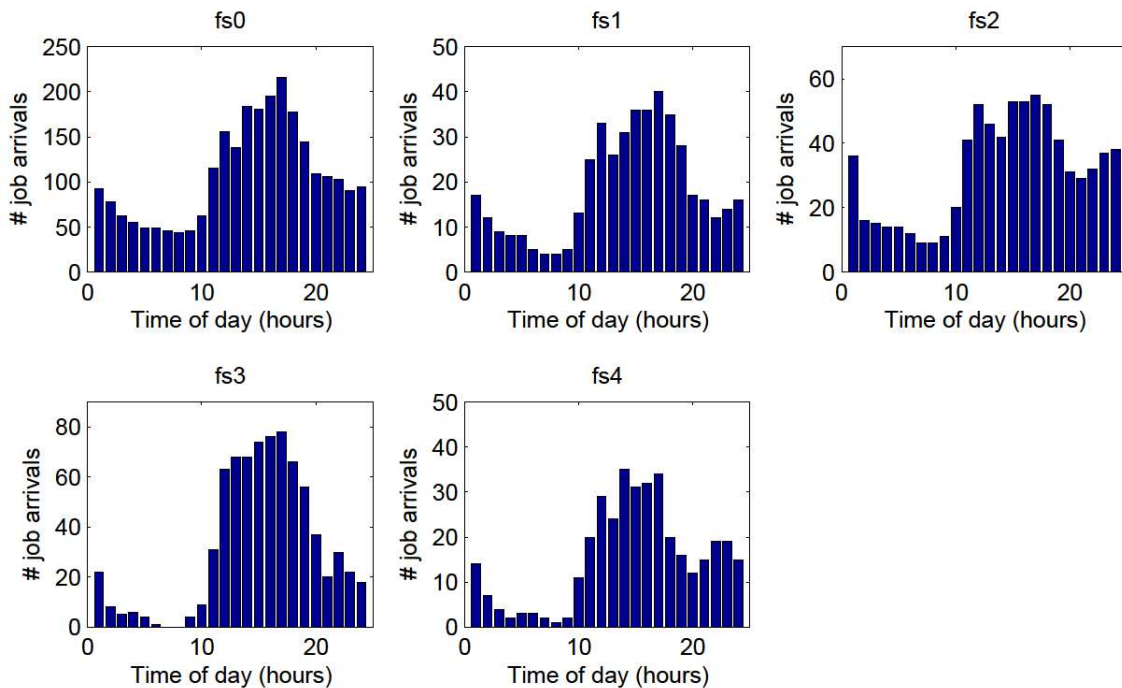


Figura A.2. Ciclo diário da chegada de *jobs* durante os dias úteis da semana no DAS-2. Fonte: [Li et al. 2004].

Sobre a taxa de chegada de *jobs*, os autores afirmam que é esperado que tenham três níveis de ciclos: anual, semanal e diário. No ciclo anual os *jobs* não são distribuídos igualmente. A carga de trabalho é maior durante alguns meses específicos e a taxa de chegada de *jobs* em tais meses é maior duas ou mais vezes do que sua média anual. Eles justificam isso devido às diferentes atividades dos usuários e grupos de pesquisa sobre os *clusters*, que são mais ativos em diferentes períodos do ano. No ciclo semanal todos os *clusters* compartilham as mesmas características: na quarta-feira a média da taxa de chegada de *jobs* é maior e decresce à partir daí, o sábado e domingo possuem as menores taxas de chegada de *jobs*. Os autores justificam que isso é normal uma vez que as pessoas trabalham mais durante os dias da semana do que nos finais de semana. E ainda afirmam que o ciclo mais importante é o diário, uma vez que os *clusters* possuem uma distribuição similar da carga de trabalho diária durante os dias da semana, diferenciando horários de picos, intermediários e com menos chegadas de *jobs*. Segundo os autores esse comportamento é similar à outros *clusters*, porém com horários de picos diferenciados. Sobre o tempo entre chegadas de *jobs*, os autores explicam que geralmente nenhuma distribuição é capaz de modelá-lo bem.

Quanto ao tamanho do *job*, número de processadores que precisam para executá-lo, os autores observam que há um grande número de *jobs* submetidos com tamanho dois e potência de dois (*power-of-two*). E que isso se deve a poucos usuários mais ativos nos *clusters* e submetendo os mesmos *jobs*.

O tempo de execução dos *jobs* variam entre 374 e 2.427 segundos, que é menor do que os autores esperavam em comparação a outros *clusters*. Porém o coeficiente de variação (CV) fica entre 5.3 e 16, que eles afirmam ser maior do que outros sistemas de produção.

Os autores observaram que em outros *clusters* com *jobs* grandes, aqueles que precisam de maior número de processadores, gastam mais tempo de execução do que os *jobs* menores. Para o DAS-2, eles não podem afirmar se há alguma correlação entre o tamanho do *job* e seu tempo de execução. Já para o tempo requisitado pelo usuário para executar o *job* e o seu tempo real de execução a correlação é alta. Os autores afirmam que, *jobs* com grandes requisições de tempo para sua execução geralmente são os que gastam mais tempo.

Segundo os autores, o comportamento dos usuários tem grande influência na carga de trabalho dos *clusters* estudados. Tipicamente um *cluster* contém um conjunto de usuários com diferentes níveis de atividades e períodos de trabalho. Uns poucos usuários e tipos de aplicações tendem a dominar. O comportamento dos usuários são uniformes e previsíveis para um curto intervalo de tempo, colaborando para calcular previsões melhores, para serem usadas em escalonadores de desempenho

[Downey and Feitelson 1999]. Uma vez que o comportamento dos usuários tem significativo impacto sobre a modelagem da carga de trabalho, técnicas e modelos têm sido propostas para capturar esse comportamento [Calzarossa and Serazzi 1994]. No DAS-2 cada grupo tem mais atividades no *cluster* de sua própria instituição.

Referências Bibliográficas

- Aamodt, A. and Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications*, 7(1):39–59.
- Abel, M. and Castilho, J. (1996). Um Estudo sobre Raciocínio Baseado em Casos. *Porto Alegre: UFRGS*.
- Amdahl, G. M. (2000). Validity of the single processor approach to achieving large scale computing capabilities. pages 79–81.
- Bichindaritz, I. (2006). Memory Organization as the Missing Link Between Case-Based Reasoning and Information Retrieval in Biomedicine. *Computational Intelligence*, 22(3-4):148–160.
- Blast (2008). BLAST: Basic Local Alignment Search Tool. <http://blast.ncbi.nlm.nih.gov/Blast.cgi>. Último acesso em: 04/09/08.
- Buyya, R. and Murshed, M. (2002). GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220.
- Calzarossa, M. and Serazzi, G. (1994). Construction and use of multiclass workload models. *Perform. Eval.*, 19(4):341–352.
- Casanova, H. (2001). Simgrid: A toolkit for the simulation of application scheduling. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 430, Washington, DC, USA. IEEE Computer Society.
- Chris Atkeson, Andrew Moore, S. S. (1997). Locally weighted learning. *AI Review*, 11:11–73.
- Dalfovo, O., da Silva Minella, C., Fenili, R., and Selhorst, M. (2002). Raciocínio Baseado em Casos utilizando a Dieta do Tipo Sangüíneo. Blumenau, SC, Brasil.

- DAS (2008). The DAS2 5-Cluster Grid Logs. http://www.cs.huji.ac.il/labs/parallel/workload/l_das2/index.html. Último acesso em: 14/10/08.
- Dinda, P. A. and O'Hallaron, D. R. (2000). Host load prediction using linear models. *Cluster Computing*, 3(4):265–280.
- Downey, A. B. and Feitelson, D. G. (1999). The elusive goal of workload characterization. *SIGMETRICS Perform. Eval. Rev.*, 26(4):14–29.
- EGEE (2008). Enabling Grids for E-science in Europe. <http://public.eu-egge.org>. Último acesso em: 15/10/08.
- Elmroth, E. and Gardfjall, P. (2005). Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 221–229, Washington, DC, USA. IEEE Computer Society.
- Foster, I. (2002). What is the Grid? A Three Point Checklist. *Grid Today*, 1(6):22–25.
- Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200.
- Globus (2008). The globus alliance. <http://www.globus.org/>. Último acesso em: 21/04/08.
- Gray, J. and Shenoy, P. (2000). Rules of thumb in data engineering. In *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, page 3, Washington, DC, USA. IEEE Computer Society.
- GridLab (2008). GridLab: A Grid Application Toolkit and Testbed. <http://www.gridlab.org/>. Último acesso em: 21/04/08.
- Hmmer (2008). HMMER: Biosequence analysis using profile hidden Markov models. <http://hmmer.janelia.org/>. Último acesso em: 04/09/08.
- Howell, F. and McNab, R. (1998). SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling. In *Proceedings of the First International Conference on Web-based Modelling and Simulation*, San Diego, CA.
- Irving, M., Taylor, G., and Hobson, P. (2004). Plug in to grid computing. *Power and Energy Magazine, IEEE*, 2(2):40–44.

- Jain, R. (1991). *The art of computer systems performance analysis*. Wiley.
- Kapadia, N. H., Fortes, J. A. B., and Brodley, C. E. (1999). Predictive application-performance modeling in a computational grid environment. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 6, Washington, DC, USA. IEEE Computer Society.
- Kolodner, J. (1993). *Case-based reasoning*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.
- Lewis, L. (1995). *Managing Computer Networks: A Case-Based Reasoning Approach*. Artech House, Inc., Norwood, MA, USA.
- Lewis, L. (1999). *Service Level Management for Enterprise Networks*. Artech House, Inc. Norwood, MA, USA.
- Li, H. (2007). Machine learning for performance predictions on space-shared computing environments. *International Transactions on Systems Science and Applications, invited paper*. invited paper.
- Li, H., Groep, D., and Wolters, L. (2004). Workload characteristics of a multi-cluster supercomputer. In *In Job Scheduling Strategies for Parallel Processing*, pages 176–193. Springer Verlag.
- Li, H., Groep, D., and Wolters, L. (2005). Efficient response time predictions by exploiting application and resource state similarities. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 234–241, Washington, DC, USA. IEEE Computer Society.
- Li, H. and Wolters, L. (2006). An investigation of grid performance predictions through statistical learning. In *In proceedings of 1st Workshop on Tackling Computer System Problems with Machine Learning Techniques (SysML), in conjunction with ACM Sigmetrcs*, Saint-Malo, France.
- Liu, C., Yang, L., Foster, I., and Angulo, D. (2002). Design and evaluation of a resource selection framework for grid applications. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 63, Washington, DC, USA. IEEE Computer Society.
- LPC (2008a). Laboratory of Corpuscular Physics da Universidade Blaise-Pascal. <http://clrwww.in2p3.fr>. Último acesso em: 15/10/08.

- LPC (2008b). The LPC Log. http://www.cs.huji.ac.il/labs/parallel/workload/l_lpc/index.html. Último acesso em: 14/10/08.
- Melchior, C. (1999). *Raciocínio Baseado em Casos Aplicado ao Gerenciamento de Falhas em Redes de Computadores*. PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.
- Menascé, D., Almeida, V., and Dowdy, L. (1994). *Capacity planning and performance modeling: from mainframes to client-server systems*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.
- Moore, A. W., Schneider, J., and Deng, K. (1997). Efficient locally weighted polynomial regression predictions. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 236–244, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Nabrzyski, J., Schopf, J., and Węglarz, J. (2003). *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers.
- Nassif, L. N. (2006). *Seleção distribuída de recursos em grades computacionais usando raciocínio baseado em casos e políticas de granularidade fina*. PhD thesis, Universidade Federal de Minas Gerais.
- Nassif, L. N., Nogueira, J. M., Karmouch, A., Ahmed, M., and de Andrade, F. V. (2007a). Job completion prediction using case-based reasoning for grid computing environments: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(9):1253–1269.
- Nassif, L. N., Nogueira, J. M., and Vinicius de Andrade, F. (2007b). Distributed resource selection in grid using decision theory. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 327–334, Washington, DC, USA. IEEE Computer Society.
- Nassif, L. N., Nogueira, J. M., and Vinicius de Andrade, F. (2009). Resource selection in grid: a taxonomy and a new system based on decision theory, case-based reasoning, and fine-grain policies. *Concurrency and Computation: Practice and Experience*, 21(3):337–355.
- NIKHEF (2008). NIKHEF: National Institute for Subatomic Physics. <http://www.nikhef.nl/>. Último acesso em: 28/07/08.

- Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.
- Sandra (2008). SisSoftware Sandra. <http://www.sissoftware.co.uk/sandra/>. Último acesso em: 10/09/08.
- SDSC (2008). The San Diego Supercomputer Center (SDSC) Blue Horizon log. http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_blue/. Último acesso em: 28/07/08.
- Singh, G., Kesselman, C., and Deelman, E. (2007). A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 117–126, New York, NY, USA. ACM.
- Smith, W., Foster, I. T., and Taylor, V. E. (1998). Predicting application run times using historical information. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142, London, UK. Springer-Verlag.
- Sulistio, A., Poduvaly, G., Buyya, R., and Tham, C. (2005). Constructing A Grid Simulation with Differentiated Network Service Using GridSim. In *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*.
- Vogel, A., Kerhervé, B., von Bochmann, G., and Gecsei, J. (1995). Distributed multimedia and qos: A survey. *IEEE MultiMedia*, 2(2):10–19.
- Watson, I. and Marir, F. (1994). Case-Based Reasoning: A Review. *The Knowledge Engineering Review*, 9(4):327–354.
- Wilson, D. R. and Martinez, T. R. (1997). Improved heterogeneous distance functions. *CoRR*, cs.AI/9701101.
- Workloads (2008). Parallel Workloads Archive. <http://www.cs.huji.ac.il/labs/parallel/workload/>. Último acesso em: 14/10/08.
- Yeo, C. S. and Buyya, R. (2005). Service level agreement based allocation of cluster resources: handling penalty to enhance utility. In *Proceedings of the 7th IEEE International Conference on Cluster Computing*, pages 27–30.