UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

"Seleção distribuída de recursos em grades computacionais usando raciocínio baseado em casos e políticas de granularidade fina"

Lilian Noronha Nassif

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para obtenção do título de Doutora em Ciência da Computação.

Orientador: Prof. José Marcos Silva Nogueira

Belo Horizonte, 29 de junho de 2006

UNIVERSIDADE FEDERAL DE MINAS GERAIS

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

FOLHA DE APROVAÇÃO

Seleção Distribuída de Recursos em Grades Computacionais Usando Raciocínio Baseado em Casos e Políticas de Granularidade Fina

LÍLIAN NORONHA NASSIF

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

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

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

PROF. CLODOVEU AUGUSTO DAVIS JÚNIOR
Instituto de Informática - PUC-MG

PROF. EDMUNDO ROBERTO MAURO MADEIRA
Instituto de Computação - UNICAMP

PROF. ANTONIO ALFREDO FERREIRA LOUREIRO
Departamento de Ciência da Computação - UFMG

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 29 de junho de 2006.

Dedicatória

Dedico esta tese à minha filhinha Isabela, que me deu tantas motivações para prosseguir. Enquanto eu tentava fazer uma tese ela fez grandes avanços como andar, falar, ler, escrever e tocar instrumentos musicais. Uma autêntica pesquisadora!

Agradecimentos

Agradeço ao professor José Marcos Nogueira pela sua fundamental orientação neste trabalho e lhe sou grata também por ter mostrado todos os caminhos de pesquisa que percorri durante o doutorado. Diversos artigos, apresentações em conferências, intercâmbio com pesquisadores e elaboração de projetos de pesquisa fizeram parte da minha formação nesse período e solidificaram o trabalho aqui conduzido. Agradeço-lhe imensamente pelas oportunidades e pelos seus comentários criteriosos e precisos nos documentos revisados.

Agradeço ao professor Ahmed Karmouch, que me recebeu na Universidade de Ottawa, no Canadá, para a realização do doutorado sanduíche. O prof. Karmouch me orientou regularmente nesse período com comentários diretos e observações pertinentes, redirecionando o tema de tese com objetividade.

Agradeço ao Dr. Roger Impey, chefe de pesquisa no *High Performance Distributed Computing* (HPDC) do *National Research Council Canada* (NRC), por ter me recebido nessa instituição e permitido a condução inicial da presente pesquisa no maior ambiente de grade computacional do Canadá. Agradeço a todos os colegas do HPDC, Dr. Gabriel Mateescu, Darcy Quesnel, Jean-Claude e Ratilal pelos comentários nas reuniões do meu projeto. Em especial agradeço ao Dr. Mohamed Ahmed, que acompanhou de perto a minha pesquisa dentro do NRC.

Agradeço aos professores do DCC/UFMG por terem contribuído na minha formação acadêmica, em especial agradeço ao prof. Loureiro, pelo incentivo no início do trabalho e por transmitir com entusiasmo, a beleza da pesquisa na ciência da computação. Agradeço também à profa. Linnyer pelo apoio e liberação de equipamentos para os experimentos da tese.

Agradeço ao professor Ricardo Machado Ruiz, da Faculdade de Ciências Econômicas da UFMG, pelos valiosos comentários ao capítulo 6 desta tese.

Agradeço à Prodabel, pelo suporte institucional e a todos os diretores dessa empresa por desenvolverem um programa de qualificação de pessoal ao nível de doutorado. Agradeço aos colegas de trabalho da Prodabel pelo incentivo. Agradeço à Capes pelo apoio financeiro ao meu doutorado sanduíche.

Agradeço ao meu marido Pedro Ernesto, que me apoiou incondicionalmente nessa empreitada, superando a saudade decorrente dos meses de distância quando fui realizar o doutorado sanduíche. Agradeço-lhe por ter me acalmado nas diversas crises que enfrentei durante o doutorado. Agradeço à minha mãe por ter me acompanhado ao Canadá e por ter tomado conta do meu anjinho Isabela. Agradeço à minha querida irmã, Dra. Elaine, pela ajuda prestada em diversas ocasiões do meu dia-a-dia e pelos comentários feitos ao texto desta tese. Agradeço ao Maurízio, ao meu pai e aos meus enteados pela torcida.

Agradeço ao colega Flávio Vinícius por ter contribuído com este trabalho. Agradeço à grande amiga Karla Albuquerque, pelos recíprocos desabafos compreendidos. Agradeço aos colegas do DCC, aos familiares e aos velhos amigos da PUC pelo apoio.

Agradeço a Deus, por ter me dado saúde e persistência e por ter colocado todas essas pessoas no meu caminho.

Resumo

Uma das funcionalidades principais para virtualização do serviço de computação em grade é a seleção do recurso adequado para executar um *job*. Tal funcionalidade pode ser obtida com um *broker* de recursos. A seleção de recursos feita pelos *brokers* atuais ainda apresenta desafios para fazer a melhor escolha, principalmente quando são considerados muitos fatores. Neste trabalho, abordamos o problema de seleção de recursos em grades computacionais cujo decisor é o usuário. Lidamos com esse problema considerando a preferência do usuário por determinados objetivos na seleção de um recurso, tais como o desempenho esperado para executar uma aplicação, a restrição de acesso ao recurso, o custo de execução da aplicação no recurso e a confiabilidade do recurso. Para cada objetivo, utilizamos técnicas diferentes e as conjugamos em um modelo de teoria da decisão. Ao considerarmos o desempenho na seleção de um recurso, utilizamos a técnica de raciocínio baseado em casos, que leva em conta execuções passadas de *jobs* similares para prever o tempo de execução de um novo *job*. Nesse modelo de predição, desenvolvemos um novo algoritmo de recuperação de casos para memórias desestruturadas baseado em seqüência de relevância e distância geométrica dos atributos do caso. Resultados mostram que o nosso modelo de predição tem boa acurácia e eficiência no tempo para realizar uma predição. O algoritmo de recuperação de casos apresenta melhor desempenho que outras abordagens com o crescimento da base de casos. Ao considerarmos a restrição de acesso ao recurso como fator na seleção, desenvolvemos um modelo de verificação distribuída de restrições de acesso baseado em políticas de granularidade fina. Diferentemente das políticas globais que se aplicam a todos os recursos de uma organização virtual, as políticas de granularidade fina estabelecem regras específicas para recursos e usuários. Nesse caso, a verificação prévia evita a escolha de um recurso que não permitirá o acesso quando da submissão do job. Resultados mostram que o modelo distribuído de verificação de acesso executa mais rapidamente que abordagens centralizadas Ao considerarmos o custo na seleção, analisamos esse fator como um dos atributos de um acordo de nível de serviço estabelecido entre recursos e usuários. Ao considerarmos a confiabilidade do recurso na seleção, utilizamos dados históricos resultantes de cada execução no ambiente, que registra a probabilidade do recurso executar o *job* no prazo previsto e no custo negociado. O modelo de decisão é formalizado utilizando a teoria da utilidade multiatributo, que relaciona os objetivos acima mencionados e que expressa a preferência do usuário em proporções diferentes para cada objetivo. A solução completa é implementada de forma distribuída utilizando sistema multiagentes, que age como um *broker* de recursos. Todos os modelos da tese foram avaliados em um ambiente real, apresentando funcionamentos adequados e bons resultados de desempenho.

Abstract

One of the main functionalities for virtualization service on grid computing is suitable resource selection for job execution. Resource brokers provide this functionality. Resource selection on current brokers still presents challenges for achieving the best solution in the decision-making process, especially when considering many factors. In our work, the user decides the resource selection in grid computing. We approach this problem considering user preference for specific resource selection objectives such as expected performance for an application execution, resource access restriction, execution application cost, and resource reliability. For each objective, we employ different techniques and combine them in a decision theory model. By considering performance in the selection process, we use the case-based reasoning technique based on similar past job executions to predict a new job time execution. In this prediction model, we develop a new case retrieval algorithm for flat memories, which is based on relevant sequence and geometric distance for case attributes. Results show that our prediction model is accurate and efficient in the prediction process. The case retrieval algorithm also presents better performance than other approaches as the case base increases. With the resource access restriction as a selection factor, we develop a fine-grain policy-based model for distributed resource access verification. Unlike a global access policy, which applies to all resources in a virtual organization, a fine-grain policy establishes rules for specific resources and users. In this case, a previous access restriction verification prevents a resource selection which may denies access to a requisition, resulting in an unsuccessful submission. In addition, the model developed is based on standard policies that avoid redundancy in access control management. Results show that our distributed model runs faster than centralized approaches and presents indexes analyzing the efficiency of each approach based on machines, requisitions, and access restrictions heterogeneities. We consider the resource cost in the selection process as an attribute in service level agreements between resources and users. By considering the resource reliability in the selection process, we use historical data from each execution in the environment, which register the job execution resource probability in the predicted time and with the negotiated cost. The decision model is completely formalized using the multi-attribute utility theory, which relates the important objectives above and allows different proportions of user preferences for each objective. The complete solution is distributed and implemented using a multi-agent system, acting as a resource broker. All models of this thesis are analyzed in a real environment, presenting appropriate functional behaviors and positive performance results.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AAA	*Authentication, Authorization, Accounting*
ACL	*Access Control List*
ARC	*Advanced Resource Connector*
ARMS	*Agent-based Resource Management System*
API	*Application Program Interfaces*
ASP	*Application Service Provision*
BLAST	*Basic Local Align Search Tool*
CAS	*Community Authorization Service*
CERN	*Conseil Européen pour la Recherche Nucléaire*
CoG	*Commodity Grid*
CGC	Condor Gerenciamento Central
CPU	*Central Processing Unit*
DF	*Directory Facilitator*
DNA	*Desoxyribo Nucleic Acid*
DNS	*Domain Name System*
EGA	*Enterprise Grid Alliance*
FIPA	*Foundation of Intelligent Physical Agents*
FSLA	*Final Service Level Agreement*
GEODISE	*Grid Enabled Optimisation and Design Search for Engineering*
GGF	*Global Grid Forum*
GHS	*Grid Harvest Service*
GJD	GRMS *JobDescription*
GNOME	*GNU Network Object Model Environment*
GNU	*GNU's Not UNIX*
GPRS	*General Packet Radio Service*
GrADS	*Grid Application Development Software*
GRASP	*Grid-based Application Service Provision*
GRB	*Grid Resource Broker*
GRIDBUS	*GRID computing and BUSiness technologies*
GRIDRPC	*GRID Remote Procedure Call*
GRMS	*GridLab Resource Management System*
JADE	*Java Agent DEvelopment framework*
HMM	*Hidden Markov Models*
HPC	*High Performance Computing*
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
IAREQ	Índice de Aceitação de Requisitos
IARES	Índice de Aceitação de Restrições
IC2D	*Interactive Control and Debugging of Distribution*
ICMP	*Internet Control Message Protocol*
IETF	*Internet Engineering Task Force*

I/O	*Input/Output*
IP	*Internet Protocol*
IPG	*Information Power Grid*
KDE	K *Desktop Environment*
KFLOPS	*thousand(K) FLoating-point Operations Per Second*
LDAP	*Lightweight Directory Access Protocol*
MAS	*Multi-Agent System*
MASK	*Multi-Agent System broKer*
MAUT	*Multi-Attribute Utility Theory*
MDS	*Monitoring and Directory Service*
MUE	Máxima Utilidade Esperada
MIPS	*Million Integer-instructions Per Second*
MPI	*Message Passing Interface*
NCBI	*National Center for Biotechnology Information*
NEES	*Network for Earthquake Engineering Simulation*
NN	*Nearest-Neighbor algorithm*
NSLA	*Network Service Level Agreement*
OASIS	*Organization for the Advancement of Structured Information Standards*
OCR	*Overhead* de comunicação na rede
OGSA	*Open Grid Service Architecture*
OSPF	*Open Shortest Path First*
OVRA	*Overhead* de Verificação de Restrições de Acesso
PACE	*Performance Analysis and Characterization Environment*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PERMIS	*PrivilEge and Role Management Infrastructure Standards validation*
POLAR	*Policy-based Access Restriction Model*
PSMRG	Problema de Seleção do Melhor Recurso em Grade computacional
QTL	*Quantitative Trait Loci*
RAM	*Random Access Memory*
RMI	*Remote Method Invocation*
RBC	Raciocínio baseado em casos
RNN	*Refined Nearest-Neighbor algorithm*
RSL	*Resource Specification Language*
RSVP	*Resource reSerVation Protocol*
SARAH	*Security-Assured Resource Allocation Architecture*
SLA	*Service Level Agreement*
SMA	Sistema Multiagente
SSAHA	*Sequence Search and Aligment by Hashing Algorithm*
SSLA	*Server Service Level Agreement*
TCP	*Transmission Control Protocol*
TFLOPS	one Trillion FLoating-point Operations Per Second
TILAB	*Telecom Itália Lab*
TSR	Tempo de Seleção de um Recurso para uma requisição
UDP	*User Datagram Protocol*
UFMG	Universidade Federal de Minas Gerais
UNICORE	*Uniform Interface to Computing Resources*
VGrADS	*Virtual Grid Application Development Software*

VO	*Virtual Organization*
VOMS	*Virtual Organization Membership Service*
WLAN	*Wireless Local Area Network*
WSLA	*Web Service Level Agreement*
XACML	*eXtensible Access Control Markup Language*
XML	*eXtensible Markup Language*

Capítulo 1

Introdução

"When the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special purpose appliances."
Gilder Technology Report, June 2000

1.1 Grade computacional

Grade computacional[1] é uma infra-estrutura que envolve o uso colaborativo e integrado de computadores, redes, bases de dados e instrumentos científicos pertencentes a diferentes organizações [Foster *et al.* 2001]. O enfoque da grade computacional é a resolução de problemas para o compartilhamento de recursos entre organizações multi-institucionais. A grade computacional é uma tecnologia emergente que promete mudar a forma como abordamos problemas computacionais complexos. Assim como a Internet revolucionou a forma do compartilhamento de informações, a grade computacional, similarmente, tende a revolucionar o compartilhamento de poder computacional e de armazenamento.

De 1986 a 2000, a velocidade dos computadores foi multiplicada por 500 enquanto a velocidade das redes de computadores foi multiplicada por 340.000 [Stix 2001]. No entanto, ainda permanecem desafios para computações científicas avançadas. Um exemplo disso são os dados produzidos pelos detectores de partículas do Laboratório Europeu de Partículas Físicas, no CERN (*Conseil Européen pour la Recherche Nucléaire*), onde até

[1] O termo "grade computacional" será utilizado no texto como tradução de "*computational grid*".

2005 vários petabytes (quatrilhões de bytes) de dados por ano foram produzidos. Essa grande quantidade de informações é da ordem de um milhão de vezes a capacidade de armazenamento de um microcomputador pessoal com capacidade de 100GB de armazenamento. Além da necessidade de armazenamento é preciso ainda realizar a análise desses dados, que irão requerer cerca de 20 teraflops (um trilhão de operações de ponto flutuante por segundo) de poder computacional [Foster 2003].

Uma das formas de resolver essa alta demanda de armazenamento e processamento é através da utilização de *clusters*. Um *cluster* é um conjunto de computadores, algumas vezes centenas deles, interconectados por uma rede de comunicação de alta velocidade da ordem de gigabits/segundo. Os computadores individuais podem ser monoprocessados ou multiprocessados. Um *cluster* provê alta disponibilidade e serviços escaláveis – tais como mecanismos de busca oferecidos para usuários da Internet – pela replicação ou particionamento do processamento através dos processadores do *cluster* [Coulouris *et al.* 2005].

Uma diferença significativa entre grade computacional e computação *cluster* é que, embora *cluster* ofereça alto poder computacional, ele, diferentemente da grade computacional, não ultrapassa os limites entre organizações, restringindo sua escalabilidade [Cao 2001].

A escalabilidade em grade computacional tem várias dimensões: 1) poder computacional e capacidade de armazenamento através de agregação de recursos; 2) heterogeneidade de recursos; 3) distribuição geográfica e organizacional; 4) multiplicidade de perfis de uso para diferentes comunidades de usuários da grade computacional [Johnston *et al.* 1999].

A associação de *clusters* e grades computacionais pode resolver problemas de demanda computacional muito alta. Um exemplo disso é o ambiente TeraGrid que agrupa 40 teraflops de poder computacional e aproximadamente 2 petabytes de armazenamento. Os elementos do TeraGrid se interconectam através de uma rede dedicada de 10-40 Gbps que interliga 8 entidades diferentes em uma organização virtual [TeraGrid 2005].

Uma organização virtual (*Virtual Organization* - VO) é um conjunto de indivíduos e instituições sujeitas a regras de compartilhamento bem definidas. Recursos compartilhados em uma VO são computadores, dados, *software* e instrumentos científicos [Foster *et al.* 2001].

Tanto o campo acadêmico como o empresarial possuem hoje demandas que apontam para um modelo de grade computacional. Para a comunidade científica tais necessidades são dadas pelo exemplo da comunidade de pesquisadores que conduzem experimentos de partículas físicas. A colaboração em pesquisas científicas nessa área é altamente vantajosa, uma vez que os experimentos geram dados na ordem de petabytes por ano e os poucos laboratórios no mundo que possuem detectores de partículas estão geograficamente distribuídos.

A demanda pela tecnologia de grade computacional para organizações empresariais existe pelas seguintes razões:

- É necessário realizar investimentos constantes em equipamentos para se obter poder computacional compatível com a necessidade de aplicações emergentes;
- É necessário criar uma infra-estrutura de tecnologia da informação que seja escalável, flexível e preparada para executar aplicações distribuídas;
- É preciso reduzir as despesas operacionais, um objetivo constante de qualquer organização empresarial;
- O desenvolvimento acelerado (por exemplo, pela descoberta de novos medicamentos) exige que seja reduzido o tempo para colocar produtos no mercado e a computação distribuída pode reduzir o tempo total de processamento;
- É preciso que haja segurança nas transações e compartilhamento de informações e recursos na interação entre empresas;

Diversas organizações[2] estão utilizando a computação em grade em áreas tais como a pesquisa científica colaborativa, a descoberta de medicamentos, a análise de risco financeiro e a definição de produtos. A tecnologia de grade computacional possibilita a interação entre organizações ao superar as restrições de integração de dados e de computação, além de reduzir custos por meio da otimização da utilização dos recursos. A colaboração entre entidades utilizando grade computacional favorece a resolução de problemas de computação que estão além da capacidade individual de uma única organização.

[2] Alguns exemplos de organizações são referenciados nos projetos de grades computacionais apresentados no Apêndice A.

A idéia da grade computacional é fazer com que o uso da computação passe a ser um serviço virtualizado, assim como acontece com outros serviços que a humanidade hoje tem à sua disposição, a saber: o fornecimento de energia elétrica, de água e de informação. A rápida disponibilidade desses recursos exemplificam o conceito de virtualização, que é entendida como as funções escondidas por detrás de uma interface que oculta os detalhes de como tais funções são implementadas. Não virtualizar o poder computacional passa a ser tão estranho quanto pensar que cada residência opere sua própria planta de energia elétrica, reservatório de água e fontes de informação (por exemplo: televisão, jornal, revista e Internet). Por que seria diferente para a computação? [Foster 2003].

Para tornar essa virtualização uma realidade, é necessário o uso de um *software* que permita o compartilhamento de infra-estruturas já existentes, preserve os sistemas dos participantes de um compartilhamento e os agrupe em uma estrutura de organização virtual. Esse tipo de *software* precisa oferecer serviços básicos de autenticação, autorização, descoberta de recursos e controle do movimento dos dados [Foster 2003]. Tais requisitos constituem a base para o desenvolvimento de um *middleware* para grade computacional. Entende-se *middleware*, no contexto de grade computacional, como uma camada de serviços intermediária entre os recursos computacionais, aplicações e interface com o usuário, que provê segurança, acesso e troca de informações consistentes entre recursos gerenciados localmente por diferentes métodos. Existem várias implementações de *middleware* para grade computacional, dentre as principais destacam-se o Globus [Globus 2005] e o UNICORE (*Uniform Interface to Computing Resources*) [Romberg 2000]. Ambos seguem padrões e são de código aberto. Destacam-se também no cenário brasileiro o Integrade e o Ourgrid (maiores detalhes no Apêndice A).

1.2 Motivação

Em um modelo computacional onde os computadores estão interligados em rede, um usuário é capaz de utilizar qualquer computador da malha de conexões, desde que ele possua uma conta para autenticação. Nesse modelo, o usuário escolhe um computador específico, autentica-se e depois utiliza o recurso para determinado processamento. Dessa forma, não há virtualização do poder computacional, pois o usuário sabe exatamente onde está sendo feito o processamento. Na grade computacional, o conjunto de computadores é

visto como um só computador, de forma que não é necessária a autenticação do usuário para cada computador em que ele deseja executar um aplicativo. Para que a escolha dos recursos compartilhados seja um serviço virtualizado para o usuário, é preciso que haja um mecanismo de seleção automática de recursos. A seleção de recursos em grade computacional é uma das atividades iniciais do processo de execução de aplicativos. É onde se pode prover a virtualização de uso do conjunto dos recursos compartilhados.

A submissão de um *job* (tarefa) em grade computacional pode ser realizada das seguintes maneiras: por meio de *brokers* (agentes intermediadores do processo de seleção); utilizando linha de comandos, onde o usuário especifica o recurso desejado de forma similar ao ambiente de rede; ou por meio de portais específicos de uma comunidade de grade computacional.

Os métodos de escolha de recursos disponíveis atualmente nos *brokers* para grade computacional ainda não conjugam as informações necessárias para o *broker* fazer a melhor escolha. Há na literatura[3] métodos que mapeiam as necessidades dos usuários com as características dos recursos e escolhem a máquina mais rápida ou a primeira máquina pesquisada que atende aos requisitos. Entretanto, alguns problemas decorrem de uma seleção desse tipo. Primeiro, sob o ponto de vista de desempenho, corre-se o risco de sempre escolher a mesma máquina, aquela considerada possuidora das melhores características computacionais. Isso a torna mais utilizada que as demais, que ficam com seus recursos compartilhados ociosos. Segundo, sob o ponto de vista econômico, em ambientes onde os recursos têm preços proporcionais às suas capacidades, escolher sempre a máquina com melhores características computacionais pode resultar em custos mais altos.

Um aspecto importante de um processo de decisão sobre a seleção de recursos é a predição do tempo de execução de *jobs*. A predição de desempenho é uma técnica essencial para atingir uma boa seleção no ambiente de grade computacional. Sem realizar predição não é possível oferecer ao usuário uma expectativa de quando a execução do *job* irá terminar e nem é possível controlar a carga de trabalho para um tempo futuro.

Outro aspecto ainda fraco nas soluções atuais é a falta de associação dos mecanismos de seleção de recursos com as políticas locais dos recursos; por exemplo, a seleção do recurso sem a prévia verificação da permissão do usuário em utilizar um recurso a um determinado

[3] O capítulo 2 apresenta uma revisão bibliográfica sobre diversas abordagens que realizam seleção de recursos em grade.

preço, qualidade de serviço e horário. Tal dissociação entre a seleção de recursos e a verificação de restrições de acesso pode levar a submissões fracassadas de *jobs*. Os *jobs* podem ser direcionados para um recurso que, embora possa ser o mais poderoso do ambiente, não permite que o usuário execute uma aplicação para o horário desejado e nem abaixo de determinado custo. De fato, passa a ser difícil para o usuário selecionar um recurso em grade, se o mesmo não oferece mecanismos de seleção que atendam a certos critérios.

Sendo o ambiente de grade computacional uma instância da computação distribuída, é preciso usar uma técnica de desenvolvimento da solução que seja autônoma, distribuída, dinâmica e robusta. A tecnologia de agentes e de sistemas multiagentes atende a todos esses requisitos. Agentes autônomos são sistemas computacionais que residem em ambientes dinâmicos complexos, percebem e agem dinamicamente neste ambiente e realizam um conjunto de tarefas para os quais eles foram projetados [Maes 1990]. Um sistema multiagente é uma coleção de agentes de *software* que cooperam ou competem entre si, ou alguma combinação de cooperação e competição que visa uma infra-estrutura comum, resultando em um sistema, ao invés de um conjunto descoordenado de agentes autônomos [Ferber 1999]. A tecnologia de agentes é padronizada pela *Foundation for Intelligent Physical Agents* (FIPA) [FIPA 2002] e, quando a utilizamos, aplicamos os métodos para comunicação e segurança entre agentes.

No momento da decisão sobre a seleção de um recurso, diversas perguntas são feitas pelo usuário, tais como: Quais recursos estão disponíveis? Quais recursos atendem aos requisitos da aplicação? Estou autorizado a executar essa aplicação nesse recurso? Qual o tempo previsto para executar essa aplicação? Qual a garantia para que a aplicação rode nesse período? Qual a máquina que roda pelo menor preço? Qual a probabilidade da máquina não executar no tempo previsto? Qual a probabilidade da garantia ser aplicada caso a previsão não se confirme?

Perguntas como essas indicam que a decisão do usuário é uma tarefa árdua e que a necessidade de um *broker* é indispensável. A nossa abordagem, descrita nesta tese, busca responder a todas essas perguntas, oferecendo ao usuário uma forma mais simples de interação com os sistemas em grades computacionais.

1.3 Objetivos

O objetivo da presente tese é estudar e prover uma solução para o problema de seleção de recursos em grades computacionais. Estamos particularmente interessados em possibilitar uma seleção direcionada ao atendimento das necessidades do usuário com relação a desempenho, política de acesso, garantia do serviço e custo. Essas necessidades devem ser atendidas de forma conjugada e como complemento aos requisitos da aplicação. Técnicas diferentes devem ser usadas para cada requisito e a combinação de tais técnicas deve ser representada através de um modelo comum. Para isso projetamos um sistema multiagente que tem como tarefa a seleção de um recurso com base nos acordos de nível de serviço (do inglês *Service Level Agreements* - SLAs) estabelecidos entre agentes que representam usuários e agentes que representam recursos. Durante a negociação, módulos de predição de término de execução do *job* e de política de restrição de acesso desempenham papéis essenciais.

A predição de tempo de execução de um *job* é baseada em execuções passadas similares. A predição permite associar garantia ao provimento do serviço de execução do *job*. A relação entre a carga de trabalho durante a execução dos *jobs* passados e a carga de trabalho recente deve ser considerada em uma predição adaptada. O aprendizado do processo de predição depende da inclusão de novos casos e soluções na base de casos. Para prover eficiência ao processo de predição desenvolvemos, dentro do modelo de predição, um novo algoritmo de recuperação de casos passados e incluímos procedimentos de aprendizado.

A verificação de restrição de acesso é outro requisito desafiador para a seleção de recurso. A restrição de acesso deve ser verificada de forma dinâmica e integrada com as políticas declaradas pelo proprietário do recurso. A falta da verificação de restrições de acesso pode incorrer em cenários onde *jobs* são submetidos para recursos que não permitem tais execuções. Isso pode levar a re-submissões do mesmo *job* para outros recursos que também impedem a execução por uma ou outra política. Para essa abordagem, nosso objetivo é definir um modelo de verificação de restrições de acesso baseado em políticas de granularidade fina de forma dinâmica, que permita inclusive a negociação de métricas de níveis de serviço flexíveis que possam estar restringindo o acesso ao recurso.

A escolha da melhor combinação entre predição de desempenho e restrições de acesso é orientada pelas preferências definidas pelo usuário e depende da conjugação das incertezas sobre os possíveis resultados. Para a tomada da decisão de seleção do melhor recurso em grade utilizamos técnicas da teoria da decisão, que associam probabilidades às preferências envolvidas.

1.4 Contribuições

As principais contribuições da tese são:

- A definição de um modelo de predição para a execução de *jobs* em grades computacionais utilizando a técnica de Raciocínio Baseado em Casos (RBC) de forma distribuída, técnica essa ainda não utilizada para esse fim em grades computacionais. Com o uso de modelo de predição é possível definir garantias para o provimento do serviço e escalonamento de *jobs* para tempo futuro. Também definimos um novo algoritmo de recuperação de casos passados baseado na distância geométrica entre atributos e com ordenação por relevância. Adaptamos soluções passadas utilizando a relação de cargas de trabalho passadas e recentes. Resultados experimentais mostram para esse modelo uma rápida execução e uma boa acurácia da predição de tempo de execução de *jobs*;
- A definição de um modelo de verificação de restrições de acesso baseado em políticas de granularidade fina e distribuídas. A verificação de políticas de granularidade fina é acoplada ao processo de negociação do serviço. Resultados experimentais mostram que a seleção desacoplada da verificação de políticas locais leva a re-submissões de *jobs* e demora do processo de seleção.
- A definição de um modelo de decisão para o problema de seleção do melhor recurso em grades computacionais utilizando a teoria da decisão. 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 direciona a seleção e é modelada em uma função multiatributo para preço e

desempenho. O recurso selecionado é aquele que maximiza a função utilidade do usuário. Resultados experimentais mostram que a melhor solução está associada aos objetivos do decisor e às variáveis de incerteza associadas ao modelo de seleção;

- A definição de um sistema multiagente (SMA) para integrar os modelos de predição, política, negociação e seleção acima relacionados. O SMA atua de forma paralela e distribuída no ambiente de grade computacional para oferecer um processamento da seleção de recursos em tempo real.

Outras contribuições de menor relevância são também apresentadas na tese, como a descoberta de recursos em grades computacionais e a monitoração da execução de *jobs*, ambas utilizando sistema multiagente. Tais funções são essenciais para o funcionamento geral da nossa solução, mas são complementares para o problema de seleção de recursos conforme aqui investigado.

1.5 Organização do documento

Os capítulos da tese estão organizados da seguinte forma:

Capítulo 2: Apresenta conceitos sobre grades computacionais, agentes e sistemas multiagentes. Revisa a literatura com relação ao problema de seleção de recursos em grades computacionais comparada com a nossa abordagem e sumariza esse estudo através de uma taxonomia sobre seleção de recursos nesse ambiente.

Capítulo 3: Descreve o núcleo do sistema multiagente, apresentando a arquitetura, o modelo de negociação, a interface do sistema e um resumo dos principais módulos e funções.

Capítulo 4: Apresenta o modelo de predição, considerado essencial para obter um acordo com níveis de serviços esperados. A técnica RBC é utilizada de forma distribuída e um novo algoritmo é desenvolvido na etapa de recuperação de casos passados.

Capítulo 5: Apresenta o modelo de verificação de restrições de acesso integrado ao processo de negociação do sistema multiagente. Descreve o conjunto de possibilidades dinâmicas na negociação considerando as políticas locais definidas pelo proprietário do recurso.

Capítulo 6: Apresenta o modelo de decisão sobre o melhor recurso da grade computacional apropriado para a execução de um *job* considerando os resultados obtidos nos módulos de predição e de política. Realiza o processamento distribuído para ponderar resultados e seleciona uma proposta final considerando a probabilidade de ocorrência e a preferência do usuário.

Capítulo 7: Faz um resumo dos objetivos e resultados alcançados e projeta possibilidades de continuação da pesquisa a partir do novo patamar alcançado.

Capítulo 2

Revisão da Literatura

2.1 Introdução

Neste capítulo, faremos uma revisão da literatura sobre a tecnologia de grades computacionais, agentes e sistemas multiagentes. Faremos também uma análise dos trabalhos relacionados com o problema da seleção de recursos em grade.

A tecnologia de agentes é considerada uma das mais promissoras abordagens para a construção de sistemas complexos, pela sua natureza autônoma, distribuída e dinâmica. Os sistemas multiagentes, executando funções de *middleware* em um ambiente de grade computacional, oferecem muitas vantagens, dentre elas: 1) a arquitetura multiagente é escalável; 2) os agentes podem representar diferentes objetivos dos diferentes participantes de uma organização virtual; 3) os métodos de comunicação, autenticação e migração dos agentes são bem definidos e padronizados, e dispensam a reelaboração desses métodos dentro do *middleware* de grade computacional.

Os trabalhos relacionados com o problema da seleção de recursos em grades computacionais são aqui apresentados e discutidos. Esse estudo possibilitou a definição de uma taxonomia sobre seleção de recursos nesse tipo de ambiente.

2.2 Características e desafios das grades computacionais

Esta seção apresenta o conceito de grades computacionais e a arquitetura dessa tecnologia, organizada em camadas. Ao descrever cada camada, expomos as funcionalidades associadas e os correspondentes desafios de pesquisa para implementar a concepção para grades computacionais definida pelos autores aqui estudados.

2.2.1 Definições básicas de grade computacional

Os fundamentos da tecnologia de grades computacionais são descritos em três artigos clássicos sobre o tema:

1) *Anatomy of the Grid: Enabling Scalable Virtual Organizations* [Foster *et al.* 2001]. Nesse artigo, os autores definem grade computacional, propondo um conceito inicial sobre sua arquitetura. O artigo também descreve a aplicabilidade da tecnologia de grade e outras tecnologias contemporâneas;
2) *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration* [Foster *et al.* 2002]. Esse segundo artigo descreve a arquitetura e a infra-estrutura de grades computacionais, tendo sido desenvolvido no *Global Grid Forum* [GGF 2005] pelo grupo de trabalho de infra-estrutura de serviços em grade;
3) *What is the Grid? A Three Point Checklist* [Foster 2002]. Nesse artigo é apresentada uma lista de verificação das características de um ambiente de grade computacional.

A grade computacional é definida [Foster & Kesselman 2004] como *um sistema que coordena recursos distribuídos usando interfaces e protocolos padronizados, abertos e de propósito geral para entregar serviços de qualidade não trivial*. Os elementos principais dessa definição são assim analisados [Foster & Kesselman 2004]:

- *Coordena recursos distribuídos* – Uma grade computacional integra e coordena recursos e usuários que estão sob diferentes domínios e aborda temas de segurança, política, pagamento, afiliação, entre outros.
- *Usa interfaces e protocolos padronizados, abertos e de propósito geral* – Uma grade computacional é construída a partir de protocolos e interfaces

padronizados e abertos de propósitos múltiplos, relacionados com temas de autenticação, autorização, descoberta e acesso a recursos.

- *Entrega serviço de qualidade não trivial* – Uma grade computacional pode ser constituída de recursos para serem usados de forma coordenada para prover serviços com qualidades variadas. Uma grade computacional pode co-alocar vários tipos de recursos que atendam a demandas complexas do usuário, de forma que a utilidade combinada do sistema é significativamente maior que a soma das partes.

2.2.2 Arquitetura, funcionalidades e desafios das grades computacionais

A arquitetura de grades computacionais define os componentes fundamentais da tecnologia e especifica a função e interação entre os componentes. Essa arquitetura é apresentada em uma estrutura em camadas, conforme mostra a Figura 2-1 [Foster & Kesselman 2004].

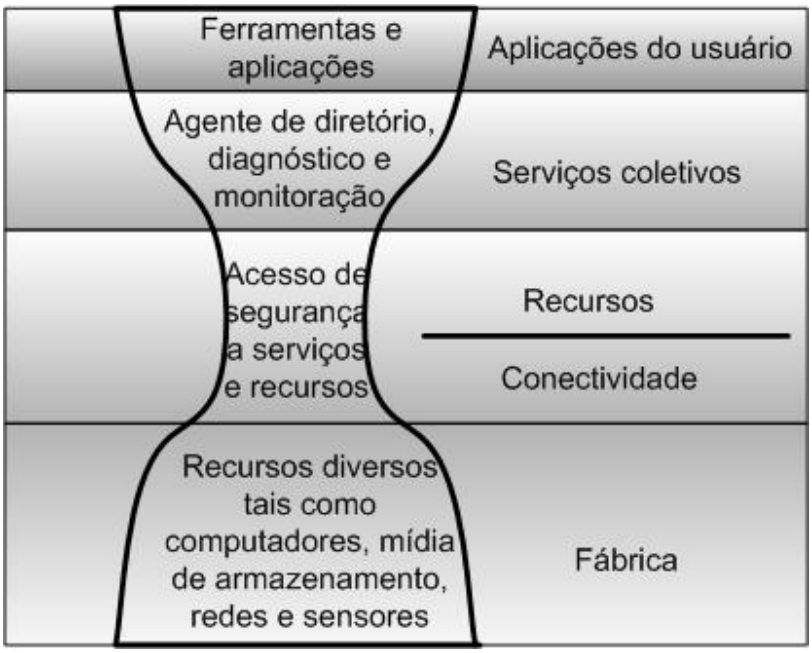

Figura 2-1: Arquitetura de grade computacional
Fonte: [Foster & Kesselman 2004]

A arquitetura de grade computacional tem a forma de uma ampulheta, pois no gargalo da ampulheta deve haver um número reduzido de protocolos. Um conjunto de comportamentos de alto nível (parte acima do gargalo) pode ser mapeado para muitas e

diferentes tecnologias (parte abaixo do gargalo). As camadas da arquitetura são descritas a seguir.

Camada de Fábrica

A camada de fábrica compreende os recursos para os quais os acessos compartilhados são mediados pelos protocolos da grade computacional. Exemplos dos elementos dessa camada são os diversos recursos, que podem ser físicos ou lógicos, tais como sistemas de armazenamento, recursos de rede, sensores, computadores ou *clusters* de computadores. Os componentes de fábrica executam as operações locais que são específicas dos recursos como resultado das solicitações de compartilhamento das operações de mais alto nível.

Os desafios nessa camada estão relacionados com a implementação de mecanismos internos nesses recursos, que permitam, por um lado, a descoberta de sua estrutura, estado e capacidades e por outro lado, o controle da qualidade de serviço, como mostram os exemplos a seguir:

- Recursos computacionais: nos recursos computacionais é necessária a definição de novos mecanismos para iniciar, monitorar e controlar a execução de processos. Funções internas são necessárias para determinar as características de *hardware* e *software* assim como as informações de estado relevantes, tais como carga de trabalho corrente.

- Recursos de armazenamento: nos recursos de armazenamento é necessário o desenvolvimento de novos mecanismos para enviar e recuperar arquivos. Geralmente são mecanismos para leitura, escrita e execução de arquivos remotos.

- Recursos de rede: nos recursos de rede é importante o desenvolvimento de mecanismos de gerenciamento que forneçam controle sobre os recursos alocados para as transferências na rede. Funções internas devem ser providas para determinar as características e a carga na rede.

Camada de Conectividade

A camada de conectividade define os protocolos de comunicação e de autenticação para transações específicas de grades computacionais. Os protocolos de comunicação permitem a troca de dados entre os recursos da camada de fábrica. Os protocolos de autenticação fornecem mecanismos de segurança de acesso e criptografia para verificar a identidade de usuários e recursos.

Os protocolos de comunicação de grade incluem transporte, roteamento e nomeação. Tais protocolos são atualmente projetados a partir da pilha de protocolos *Tranmission Control Protocol / Internet Protocol* (TCP/IP), tais como IP, *Internet Control Message Protocol* (ICMP), TCP, *User Datagram Protocol* (UDP), *Domain Name System* (DNS), *Open Shortest Path First* (OSPF), *Resource ReSerVation Protocol* (RSVP), dentre outros. Desafios futuros devem abordar o projeto de novos protocolos de comunicação específicos para grades computacionais que atendam a demandas da dinâmica de redes específicas como, por exemplo, redes ópticas ou redes sem fio.

Os protocolos de segurança também estão sendo baseados em padrões, dentro do contexto da suíte de protocolos Internet. Os temas de segurança para grades computacionais que são mais importantes atualmente referem-se a:

1) autenticação única – um usuário deve se autenticar somente uma vez, dispensando autenticações sucessivas para acessos a recursos ou domínios administrativos diferentes;
2) delegação - um usuário deve poder delegar a execução de um programa para os recursos onde ele tem autorização de uso. O programa deve poder opcionalmente ser capaz de delegar seus direitos para outro programa;
3) integração com soluções de segurança local – as soluções de grades computacionais devem interoperar com soluções de segurança local;
4) relacionamento de confiança baseada no usuário – caso o usuário tenha permissão para executar programas nos recursos A e B, não deve ser obrigatório que A e B interajam.

Camada de Recursos

A camada de recursos é construída sobre os protocolos de comunicação e de autenticação da camada de conectividade, e seu papel é definir protocolos para negociação, inicialização, monitoração, controle, contabilização e pagamento de operações compartilhadas em recursos individuais de forma segura. As implementações desses protocolos da camada de recurso são baseadas nas funções da camada de fábrica para acessar e controlar os recursos locais. Os protocolos da camada de recursos concentram-se nos recursos individuais e ignoram o estado global. Duas classes principais de protocolos da camada de recursos são definidas:

- Protocolos de informação – são usados para obter informação sobre a estrutura e o estado de um recurso, como por exemplo, sua configuração, carga de trabalho corrente e política de uso, por exemplo, custo.

- Protocolos de gerenciamento – são usados para negociar acesso a recursos compartilhados, especificando, por exemplo, os requisitos do recurso e as operações a serem executadas, tais como criação de processo e acesso a dados. No projeto desse tipo de protocolo é preciso ter um ponto de aplicação de política assegurando que as operações solicitadas sejam consistentes com a política do recurso a ser compartilhado. Temas que precisam ser considerados incluem a contabilização de uso do recurso e pagamento pelo uso.

Embora muitos desses protocolos possam ser imaginados, a camada de recursos e a camada de conectividade formam o gargalo do modelo de grade computacional e, portanto, devem ser limitadas a um conjunto pequeno de protocolos.

Camada de Serviços Coletivos

A camada de serviços coletivos endereça problemas de descoberta, seleção e alocação de recursos, segurança, política e contabilização. A camada de serviços coletivos contém protocolos e serviços não associados a um recurso específico. Essa camada pode implementar soluções para uma coleção de recursos. Pelo fato dos componentes coletivos serem construídos acima da camada de recurso e, portanto, por participarem de uma camada mais "larga" do modelo, a camada de serviços coletivos pode implementar uma grande variedade de serviços sem adicionar novos requisitos aos recursos que estão sendo compartilhados. Por exemplo:

1) Serviços de diretório – permitem aos participantes de uma VO descobrir a existência de recursos compartilhados;

2) Serviços de co-alocação, escalonamento e *brokering* – permitem aos participantes de uma VO requisitar a alocação de um ou mais recursos para uma proposta específica e escalonar as tarefas nos recursos apropriados;

3) Serviços de monitoração e diagnóstico – monitoram recursos para detectar falha, intrusão e sobrecarga;

4) Serviços de réplica de dados – gerenciam o armazenamento de recursos para maximizar o desempenho no acesso aos dados.

Camada de Aplicações

A camada de aplicações na arquitetura de grade computacional compreende as aplicações do usuário que operam dentro de um ambiente de uma VO. As aplicações são construídas nos termos dos serviços oferecidos em cada camada da arquitetura.

As ferramentas no nível da camada de aplicação devem oferecer mecanismos para que os usuários consigam escrever e executar suas aplicações no ambiente de grade computacional. Isso porque os usuários não dispõem de habilidade, tempo e motivação para aprender os detalhes dos serviços de informação da grade e fazer desses serviços a base para a tomada de decisão sobre a seleção de recursos. As ferramentas no nível da aplicação devem evitar a desconexão entre a grade computacional e a comunidade de usuários.

De forma resumida, os desafios para a efetivação do ambiente de grades computacionais estão relacionados com os problemas advindos do compartilhamento de recursos entre organizações diferentes. Destacam-se na camada de fábrica os problemas de descoberta de recursos (incluindo descoberta da estrutura, estado e capacidade do recurso), a monitoração da execução de processos no recurso (incluindo carga de trabalho e estado da fila) e a movimentação de dados entre recursos (incluindo mecanismos de leitura, escrita, transferência e execução remota) [Nishandar 2004]. Na camada de conectividade, destacam-se os problemas de definição de protocolos de comunicação para grades computacionais [Czajkowski *et al.* 2002] e os temas de segurança (incluindo única autenticação do usuário no ambiente, delegação de execução entre processos distribuídos e relacionamento de confiança entre recursos que autorizam o mesmo usuário a utilizar o recurso) [Bertino *et al.* 2004]. Na camada de recursos, destacam-se os problemas de definição de novos protocolos que interajam com a camada de fábrica para obter informações dos recursos (incluindo sua configuração, carga de trabalho e política) [Raman *et al.* 2003] e consigam gerenciá-los (incluindo aplicação da política, contabilização e pagamento) [Cao *et al.* 2002]. Na camada coletiva destacam-se os problemas que implementam soluções para uma coleção de recursos, tais como serviços de diretório, serviços de co-alocação, escalonamento, seleção, monitoração e réplica de dados [Castellano *et al.* 2004]. Na camada de aplicação destacam-se os problemas de definição de padrões e ferramentas para o desenvolvimento de aplicações distribuídas e os problemas de interfaces facilitadoras de uso da grade computacional pelos usuários [García *et al*. 2005].

2.3 Agentes e Sistemas Multiagentes

As tecnologias de agentes e sistemas multiagentes são utilizadas na tese como técnicas para a implementação dos modelos desenvolvidos. Cabe nesta seção destacar os conceitos relacionados a essas tecnologias.

2.3.1 Agentes

Existem várias definições de agentes que esclarecem que um agente não é um simples programa. Em [Franklin & Graesser 1996] são apresentadas dez definições diferentes. Apresentamos aqui algumas definições de agentes que julgamos serem as mais adequadas.

Definição de agentes segundo [Ferber & Gasser 1991]: *Um agente é uma entidade real ou virtual, capaz de agir em um ambiente e se comunicar com outros agentes; é movido por um conjunto de tendências (sejam objetivos individuais a atingir ou uma função de satisfação a otimizar); possui recursos próprios; é capaz de perceber seu ambiente (de modo limitado); dispõe (eventualmente) de uma representação parcial desse ambiente; possui habilidade e oferece serviços; pode eventualmente se reproduzir; seu comportamento tende a atingir seus objetivos utilizando as habilidades e os recursos que dispõe e levando em conta os resultados de suas funções de percepção e comunicação, bem como suas representações internas.*

Definição de agentes segundo [Wooldridge & Jennings 1995]: *"... um hardware ou (mais frequentemente) um sistema de computação baseado em software que possui as seguintes propriedades:*

- *Autonomia: agentes operam sem a intervenção de seres humanos ou outros, e possuem algum tipo de controle sobre suas ações e sobre seus estados internos;*
- *Habilidade social: agentes interagem com outros agentes (e possivelmente seres humanos) via algum tipo de linguagem de comunicação de agentes;*
- *Reatividade: agentes percebem seus ambientes (que podem ser o mundo físico, um usuário via uma interface gráfica, uma coleção de outros agentes, a Internet, ou talvez todos esses combinados) e respondem de tempos em tempos às mudanças que ocorrem nesses elementos;*

- *Pró-atividade: agentes são capazes de exibir um comportamento direcionado a objetivo ao tomarem uma iniciativa."*

Definição de agentes inteligentes segundo [Gilbert 1997]: " *Um agente inteligente é um software que auxilia e age em nome de outros. Os agentes inteligentes permitem que pessoas deleguem trabalhos para os agentes de software. Os agentes podem realizar tarefas repetitivas, sumarizar dados complexos, aprender e recomendar ações."*

Os agentes fornecem aos projetistas e desenvolvedores uma forma de estruturar aplicações com componentes comunicativos e autônomos e guiar a construção de ferramentas de *software* e infra-estrutura para suportar projetos. Por isso oferecem uma nova e apropriada maneira de desenvolver sistemas complexos, principalmente em ambientes abertos e dinâmicos [Luck *et al.* 2005].

Um agente é capaz de oferecer auxílio a um usuário para realizar determinado trabalho. Uma forma de potencializar esse auxílio é utilizar um grupo de agentes que trabalham juntos para atingir um objetivo em comum.

2.3.2 Sistemas Multiagentes

Um sistema multiagente (SMA) é uma coleção de agentes de *software* que cooperam ou competem entre si ou alguma combinação de cooperação e competição visando uma infra-estrutura comum, resultando em um sistema, ao invés de, simplesmente, um conjunto descoordenado de agentes autônomos.

Algumas das características básicas de um sistema multiagente são que cada agente tem uma visão limitada do estado do mundo, não há um controle global, os dados são descentralizados e a computação é assíncrona [Sycara 1998].

Os sistemas multiagentes oferecem modelos robustos para representar ambientes do mundo real com um grau apropriado de complexidade e dinamismo.

Um sistema multiagente é aplicado a sistemas que possuem os seguintes elementos [Ferber 1999]:

a) Um ambiente, *E*, que é um espaço;

b) Um conjunto de objetos, *O*. É possível em um dado momento, associar qualquer objeto com uma posição em *E*. Esses objetos podem ser percebidos, criados, destruídos ou modificados por agentes (definidos a seguir);

c) Um conjunto de agentes, *A*, que são as entidades ativas do sistema;

d) Um conjunto de relações *R*, que interconectam objetos e agentes;

e) Um conjunto de operações, *Op*, fazendo com que os agentes de *A* percebam, produzam, consumam, transformem e manipulem objetos de *O*;

f) Operadores com a tarefa de representar a aplicação dessas operações e as reações do mundo a essa tentativa de modificação.

Os sistemas multiagentes são considerados por alguns autores como a abordagem moderna para a Inteligência Artificial Distribuída (IAD) [Weiss 2001]. O domínio de pesquisa da IAD nasceu no final da década de 1970 e compreende o estudo de modelos e técnicas para resolver a classe de problemas cuja distribuição física ou funcional é inerente. A partir da década de 1990, com o avanço das redes de computadores, o domínio da IAD também considera questões como distribuição, heterogeneidade e abertura do sistema. A metáfora da inteligência passa a ser o comportamento social, onde modelos, arquiteturas e implementações são estabelecidos para que um conjunto de agentes inteligentes execute ações de modo coordenado, surgindo assim os sistemas multiagentes [Rezende 2003].

As aplicações dos sistemas multiagentes são inúmeras. Destacamos a seguir, quatro categorias principais, classificadas por [Ferber 1999]:

- Resolução de problemas: Esta área envolve agentes que são puramente *software* e não possuem estrutura física real. Contrasta com as aplicações para robótica coletiva (a ser vista adiante). No ambiente de problemas de resolução distribuída, a perícia está distribuída entre todos os agentes, sendo que cada um possui habilidades restritas em relação ao problema completo a ser resolvido. No ambiente de resolução de problemas distribuídos, o problema é distribuído e os agentes possuem as mesmas habilidades.
- Simulação multiagente: Os modelos de simulação consistem em analisar propriedades do mundo real e são geralmente construídos na forma de relações matemáticas entre variáveis representando valores físicos medidos da realidade. Tais modelos numéricos possuem limitações para alguns casos, por exemplo, pelo nível de análise de uma população em relação às decisões tomadas pelos indivíduos. A simulação multiagente oferece a representação individual, seus comportamentos e suas interações, sendo possível representar um fenômeno como sendo o resultado das interações de uma sociedade de agentes que possuem autonomia operacional diferente.

- Construção de mundos artificiais: Para essa categoria não são utilizadas aplicações, no sentido exato da palavra, uma vez que não é buscada a resolução específica de um problema. Na construção de mundos artificiais é possível analisar mecanismos de interação de forma mais detalhada que uma aplicação poderia fazer, por exemplo, a análise de um protocolo de cooperação, ou a compreensão da influência de uma sociedade regulamentada.
- Robótica coletiva: A robótica coletiva refere-se à criação de uma sociedade de robôs que cooperam para atingir uma missão. São utilizados agentes concretos em um ambiente real. Os robôs são classificados como robôs celulares e robôs móveis. Robôs celulares referem-se à construção de robôs com base modular onde cada parte do robô é um agente, formando um sistema multiagente, e esse conjunto constitui um único agente. O movimento do robô envolve a coordenação de um conjunto de agentes. Na categoria de robôs móveis devem existir no mínimo dois robôs que devem coordenar seus movimentos e cooperar em tarefas tais como intervir para auxiliar pessoas e explorar espaços. A coordenação de veículos, inclusive aviões, também se refere a essa área de aplicação.

2.4 Seleção de recursos em grades computacionais

A seleção de recursos é um serviço da camada coletiva da arquitetura de grades computacionais. Esse serviço pode ser implementado através de soluções particulares que complementam um *middleware* de grade. O Globus, por exemplo, não possui um *broker* de recursos e potencialmente pode utilizar diferentes *brokers* para selecionar um recurso. A seguir, descrevemos os trabalhos relacionados com a seleção de recursos em grades computacionais e faremos comentários sobre suas diferenças, vantagens e desvantagens em relação à nossa proposta.

Condor é um sistema de gerenciamento de carga de trabalho especializado para computação intensiva de *jobs* [Condor 2005]. O Condor provê mecanismos de enfileiramento de *jobs*, políticas de escalonamento e um esquema de prioridade, além de monitoração e gerenciamento de recursos. O Condor possui um mecanismo denominado *matchmaking* [Raman *et al.* 1998], que expressa atributos delimitadores de uso do recurso de forma individual e executa combinações entre as requisições de *jobs* e os recursos

disponíveis para fazer a seleção. O mecanismo consiste na combinação simétrica entre anúncios feitos por recursos e usuários e corresponde à idéia de anúncios de classificados.

O trabalho de [Raman *et al.* 2003] estende a filosofia da seleção de recursos utilizada no Condor, mantendo a adoção de mecanismos de *matchmaking* utilizando anúncios. A extensão feita no trabalho permite a combinação de várias requisições de *jobs* com restrições de acesso aos recursos. Dessa forma é possível retornar um conjunto de recursos que atendam ao anúncio, ao invés de retornar somente um recurso, como é o caso do mecanismo original. Nessa abordagem, quanto maior o número de pares de máquinas analisados, maior o tempo para a definição de seleção do recurso. O trabalho de [Liu & Foster 2004] também estende o mecanismo de *matchmaking* com o uso de anúncios. Nessa solução, os usuários especificam restrições para recursos e os proprietários dos recursos especificam restrições para o consumidor. Os autores interpretam a combinação desses fatores como um problema de satisfação de restrições e exploram as tecnologias resolvedoras de restrições para implementar operações de combinação.

A nossa abordagem se diferencia em vários aspectos dos trabalhos conduzidos por [Raman *et al.* 1998], [Raman *et al.* 2003] e [Liu & Foster 2004]. A nossa abordagem de combinação de restrições de acesso é descentralizada, enquanto tais trabalhos centralizam esse processamento; a nossa abordagem identifica os recursos que atendem aos requisitos[4] antes de verificar as restrições de acesso, enquanto tais trabalhos não diferenciam requisitos de restrições e processam a combinação de restrições para todos os recursos independentemente se o recurso atende aos requisitos mínimos. A segunda grande diferença entre as abordagens que utilizam mecanismos de *matchmaking* e a nossa é que as políticas fazem parte dos anúncios, não existindo implementação das estruturas de políticas que sigam um padrão de definição e verificação de políticas. A nossa abordagem verifica as políticas usando os mecanismos *Policy Decision Point* (PDP) e *Policy Enforcement Point* (PEP). PDP e PEP são especificações do *Internet Engineering Task Force* (IETF) e representam uma forma comum de abstrair os diferentes componentes funcionais de um sistema de autorização. Um PDP é um mecanismo de processamento que avalia políticas baseadas nas requisições. Um PEP é tipicamente um elemento onde uma política é aplicada e que gera requisições em um PDP [Godik & Moses 2003]. A utilização de padrões de

[4] Estamos considerando como requisitos, as características físicas mínimas requeridas pela aplicação, tais como memória, velocidade de CPU e quantidade de processadores.

verificação de políticas torna a nossa abordagem mais flexível de ser utilizada por organizações que já seguem esses padrões e evita repetir essas restrições de acesso dentro de soluções específicas. O terceiro ponto que diferencia os três trabalhos em relação ao nosso é que a abordagem que apresentamos realiza predições para execução do *job*, enquanto as demais não fazem uso dessa informação para realizar a seleção do recurso.

Em [Song & Hwang 2004], os autores argumentam que a segurança dos ambientes de grades computacionais pode ser aumentada se forem analisadas as taxas de sucesso da execução de *job*s anteriormente executados nessas plataformas. Essa análise é feita considerando que os recursos podem ser alterados por códigos maliciosos podendo danificar outras aplicações. A alocação do recurso é feita considerando a informação de segurança para a execução de aplicações no recurso. Baseado em um modelo de confiança com lógica *fuzzy,* é proposto um esquema denominado *Security-Assured Resource Allocation Architecture* (SARAH) para assegurar a segurança distribuída em vários recursos de grades computacionais. A abordagem pretende integrar a confiança com a otimização de recursos para acomodar todas as aplicações de usuários com baixa perda de *job*s e curto tempo de espera. A proposta é alcançar alto desempenho com baixo custo. O algoritmo provê a distribuição da carga de trabalho e o tempo de execução estimado em um vetor de alocação que designa *m* recursos para executar uma seqüência de *n job*s. Essa abordagem é direcionada para o balanceamento de carga, enquanto a nossa solução é centrada nos objetivos do usuário. Para o balanceamento de carga é preciso agrupar todas as demandas para depois distribuí-las. No entanto, estamos tratando de um cenário onde, de forma dinâmica e em tempo real, o usuário seleciona o melhor recurso para a sua aplicação.

Em [Alunkal *et al.* 2003] os autores desenvolvem um serviço de reputação para auxiliar no processo de seleção de recursos. A reputação é computada usando uma classificação de confiança fornecida pelos usuários dos serviços. Essa abordagem de uso de reputação é popularmente usada em aplicações de vendas pela Internet, onde compradores registram como foram realizadas as transações comerciais e atribuem notas que resultam na reputação dos vendedores. O processo desenvolvido em [Alunkal *et al.* 2003] inclui a seleção de recursos confiáveis que satisfazem requisitos da aplicação de acordo com uma métrica de confiança pré-definida. Um dos aspectos que a nossa abordagem trata é o estabelecimento da garantia da entrega do serviço e a probabilidade de aplicá-la. Portanto, a nossa solução emprega dados históricos e estatísticos para verificar a confiabilidade do provedor de

serviços, enquanto em [Alunkal *et al.* 2003] essa reputação é construída a partir de dados informados pelos usuários do serviço. Tais dados são subjetivos e a disponibilidade dos mesmos depende da presteza do usuário em fornecê-los. Nossa abordagem, por outro lado, lida com dados precisos, registrados a cada execução de *job* realizada no recurso.

[Lee *et al.* 2004] escolhem um conjunto de servidores que oferecem o menor tempo de execução do aplicativo e, depois da seleção já realizada, dispõem de mecanismos de tolerância a falhas para prover a garantia ao serviço. São feitas migrações do aplicativo entre servidores na ocorrência de falhas. Na nossa abordagem, a garantia associada ao serviço é estabelecida e aplicada em caso da predição da entrega do serviço não se concretizar. Diferentemente da nossa solução, a escolha de [Lee *et al.* 2004] desconsidera os requisitos de políticas locais de acesso aos recursos, o que a torna adequada somente quando todos os recursos permitem acessos iguais para todos os usuários. A nossa abordagem está preparada para a seleção que considera diferentes restrições de acesso para os diferentes recursos e usuários do ambiente compartilhado.

O trabalho de [Elmroth & Tordsson 2004] concentra-se em selecionar o recurso que ofereça o menor tempo total de entrega, incluindo o tempo para transferir os arquivos para o recurso, o tempo em espera na fila, o tempo de processamento e o tempo de transferência dos resultados para o local de destino informado. As predições para tais tempos de execuções são baseadas em *benchmarks* que o usuário informa como referência anterior. O usuário informa que através do *benchmark* de nome b_i foi obtido um resultado $r_{i,}$ sendo que a aplicação executou em tempo t_i. Baseando-se nessas informações, o seletor estima de forma linear o tempo de execução para os recursos de interesse. Não são apresentados valores de erros de predição usando tal mecanismo. Essa abordagem requer que o usuário conheça um conjunto relevante de resultados de *benchmarks*. A nossa abordagem utiliza apenas a descrição do *job* para realizar predições, o que a torna mais simples e dispensa que o usuário conheça um *benchmark* que esteja associado à sua aplicação.

O sistema *Grid Harvest Service* (GHS) fornece um escalonamento dinâmico e auto-adaptativo [Sun & Wu 2003]. A máquina é escolhida com base no tempo de execução da aplicação, e um modelo probabilístico faz a predição do desempenho. O modelo de predição GHS emprega padrões de uso do recurso desconsiderando o tipo da aplicação, enquanto nossa abordagem faz predições baseadas em casos passados similares à aplicação a ser executada. Com isso a nossa abordagem consegue taxas de erros menores que o GHS.

O objetivo do GHS é minimizar o tempo de execução da aplicação, dividindo a tarefa em sub-tarefas para processamento paralelo na grade computacional. Entretanto, o nosso objetivo é selecionar o melhor recurso para executar um *job* que não é preparado para distribuição do processamento, mostrando que as abordagens, nesse ponto, buscam selecionar recursos para classes de aplicações diferentes. O GHS foca nas predições para aplicações de longa duração (com tempo de execução acima de 8 horas), enquanto o nosso processo de predição é adequado para aplicações de qualquer duração, o que o torna mais flexível para predições de aplicações diferentes. A implementação corrente do GHS não está preparada para uso em grades computacionais, mas a nossa solução está disponível para realizar a predição de duração de um *job* quando um usuário submete um *job* para o ambiente compartilhado, o que a torna mais apropriada à solução do problema de seleção de recursos em grades.

O sistema *Agent-based Resource Management System* (ARMS) é um sistema multiagente para a descoberta e gerenciamento de recursos [Cao *et al.* 2002]. O ARMS utiliza a ferramenta *Performance Analysis and Characterization Environment* (PACE) para executar predições para tempo de execução de *jobs* [Nudd *et al.* 2000]. Agentes são associados aos recursos e são organizados em uma estrutura hierárquica, com a máquina de melhores características computacionais na raiz. O PACE, em uma máquina central, calcula o tempo de execução da aplicação para cada máquina. Se uma máquina não atende aos requisitos da aplicação, a requisição de execução do *job* é enviada para uma máquina localizada em posição hierárquica superior. No ARMS, a requisição de execução do *job* é enviada de forma randômica para os agentes da estrutura; entretanto, nossa abordagem somente envolve agentes no processo de negociação que atendem aos requisitos do *job,* como pode ser visto no capítulo 4. O processo de predição do PACE requer modelos especiais para cada aplicação e para cada *hardware*. Nossa técnica de predição dispensa tais modelos, já que obtém o desempenho da aplicação usando casos passados similares executados no *hardware* local e registrados em uma base de casos local que é incrementada a cada execução de um novo *job,* tornando-se, nesse processo de aprendizado, cada vez mais confiável. Nossa abordagem simplifica e flexibiliza o processo de inclusão de novas aplicações e novos recursos de *hardware* no ambiente da grade computacional.

Jang *et al.* (2004) apresentam um sistema planejador de recurso que usa a predição de desempenho baseado nos dados históricos para identificar recursos apropriados. O trabalho

faz a interface entre dois sistemas, Pegasus (*Planning for Execution in Grids*) [Deelman *et al.* 2004] e Prophesy, uma infra-estrutura de modelagem de desempenho [Taylor *et al.* 2003]. O Pegasus mapeia a descrição do fluxo de trabalho em recursos disponíveis da grade computacional. O Prophesy é uma infra-estrutura para análise de desempenho e modelagem distribuída. O Prophesy envia para o Pegasus uma lista de servidores e uma lista de probabilidades de tempo de execução para cada um deles. O Pegasus usa essa informação para decidir como distribuir *jobs* entre recursos da grade computacional. O Prophesy mantém em uma base de dados centralizada os dados de desempenho, características do sistema e detalhes das aplicações. Tanto o Prophesy quanto a nossa abordagem usam dados históricos para fazer predição; o Prophesy faz uma predição centralizada para todos os servidores indicados pelo Pegasus, enquanto a nossa abordagem faz uma predição descentralizada em cada servidor candidato. As bases de dados da nossa abordagem são menores por serem distribuídas, e é dispensável armazenar dados sobre as características dos servidores, já que cada um possui a sua base particular de casos.

O trabalho em [Smith *et al.* 1998] agrupa *jobs* com argumentos idênticos em um molde, criando um conjunto de moldes. Os *jobs* são classificados em mais de uma categoria de molde. A predição de desempenho é computada usando média simples ou regressão linear. A principal diferença entre essa abordagem e a nossa está no conceito de similaridade e nos algoritmos de recuperação de casos. Enquanto [Smith *et al.* 1998] classificam os *jobs* similares em mais de uma categoria de *job*, a nossa abordagem computa funções de similaridade entre os argumentos dos *jobs*. O sistema descrito [Smith *et al.* 1998] recupera casos usando algoritmos gulosos ou genéticos, enquanto nossa abordagem utiliza um eficiente e novo algoritmo (descrito no Capítulo 4), elaborado a partir da modificação de um algoritmo *nearest-neighbor* original. Nossa abordagem objetiva a predição do tempo de execução de um *job* em cada máquina candidata para selecionar a melhor máquina para executar o *job*, o que possibilita atender a um cenário de tempo real. Quesitos como restrições de acesso ao recurso não são considerados em [Smith *et al.* 1998] como fator para a seleção do recurso, o que pode levar à escolha de recursos que não permitem a execução de um *job*.

De modo geral, o que caracteriza melhor a nossa abordagem e a diferencia das demais são os seguintes aspectos:

- Conjugamos quatro conjuntos de fatores indispensáveis para a seleção do recurso, sendo eles:
    1. requisitos mínimos do recurso para executar a aplicação (capacidade da CPU, quantidade de processadores, memória, dentre outros);
    2. atendimento aos objetivos e preferências do usuário (desempenho, preço e garantia);
    3. restrições de acesso definidas por cada recurso;
    4. evidências passadas sobre o provimento do serviço;
- Utilizamos técnicas apropriadas para os diferentes fatores, ou seja, um modelo de predição e um modelo para verificação de restrições de acesso;
- Utilizamos a teoria da decisão para a seleção, pois conjuga vários fatores, é orientada pela preferência do usuário e possibilita análise sob incerteza. Os trabalhos relacionados apresentam visões parciais da solução;
- A nossa abordagem utiliza processamento paralelo e distribuído para todos os modelos aqui elaborados.

2.5 Uma taxonomia para seleção de recursos em grades computacionais

Nos ambientes de grades computacionais, a seleção de recursos depende de quem está tomando a decisão, como está sendo feita a tomada de decisão, quantos recursos estão sendo selecionados e quais objetivos estão sendo atendidos. Para esse fim, esta seção apresenta uma taxonomia para a seleção de recursos em grades computacionais. Esta taxonomia irá auxiliar no entendimento das diferentes abordagens e suas motivações para a seleção de recursos.

Decisores

A entidade que toma decisão no processo de seleção de recursos é um agente do ambiente compartilhado que precisa direcionar a execução da aplicação para um ou mais recursos do ambiente de grade computacional. Chamamos esse agente de "decisor". Os decisores desse ambiente podem ser o proprietário do recurso, o usuário ou o administrador do conjunto de recursos compartilhados. Os decisores tomam decisões segundo os objetivos descritos na seção 2.5.4.

Tomada de decisão

A tomada de decisão pode ser realizada de forma centralizada ou descentralizada. A decisão centralizada é realizada sempre no mesmo recurso detentor das informações e executor do procedimento que decide sobre a seleção de um recurso; e a decisão descentralizada pode ser realizada por qualquer recurso do ambiente compartilhado. A decisão pode ser orientada pela preferência expressa pelo decisor para um ou múltiplos objetivos.

Quantidade de recursos selecionados

No ambiente de grade computacional existem aplicações preparadas para execução em vários processadores simultaneamente e também aplicações monotarefas, cujo processamento não pode ser distribuído entre vários recursos. Portanto, dependendo da aplicação e do número de processadores solicitados para a sua execução, a seleção de recursos poderá ser feita para 1 ou *n* recursos.

Objetivos

Objetivos distintos podem orientar a seleção de recursos, basicamente sob os pontos de vista do proprietário do recurso, do usuário e do administrador do conjunto de recursos. Um objetivo básico repete-se para cada decisor, onde, no mínimo, o recurso deve possuir as características solicitadas pela aplicação. Outros objetivos adicionais podem conduzir a uma melhor seleção se forem consideradas as variáveis que atendem aos objetivos do decisor.

Caso a seleção de recursos fique a cargo do proprietário do recurso, os objetivos principais podem ser a obtenção de maior lucro e a redução da ociosidade da máquina. Caso a seleção fique a cargo do usuário, os objetivos considerados podem ser a obtenção de menor preço, a execução em menor tempo e a maior confiabilidade dos recursos. Sendo a decisão tomada por um administrador do ambiente compartilhado, os possíveis objetivos podem ser o balanceamento de carga, o desempenho, a justiça entre solicitações, a possibilidade de acesso ao recurso e a confiabilidade do recurso.

Quando a seleção é orientada pelo balanceamento da carga de trabalho dentre os diversos recursos do ambiente compartilhado, pode-se estar querendo alcançar justiça e maior disponibilidade do ambiente.

Quando a seleção é orientada pelo desempenho, estamos considerando que o seletor de recursos pode ou não utilizar alguma técnica de predição de desempenho (relacionadas na seção 2.5.5). Se nenhuma técnica de predição for utilizada, como é o caso do [Condor 2005], o recurso pode ser selecionado baseando-se em suas características de desempenho, tais como *Million Integer-instructions Per Second* (MIPS) e *thousand(K) FLoating-point Operations Per Second* (KFLOPS). No entanto, quando se utiliza alguma técnica de predição, devem ser observadas as seguintes condições: se a predição está considerando que o recurso está ou não dedicado para a tarefa de execução da aplicação; se a predição é voltada para uma tarefa monoprocessada ou multiprocessada; se a predição é realizada de forma centralizada ou distribuída.

Quando a seleção é orientada pelo custo, pode haver funções específicas desenvolvidas para maximizar ou minimizar o custo, ou mesmo encontrar uma boa relação custo e desempenho. A maximização do custo pode estar associada a uma situação onde o proprietário do recurso é quem toma a decisão e a minimização do custo pode ser ilustrada pela situação onde o usuário ou o administrador do ambiente tomam a decisão.

Quando a seleção é orientada pela confiabilidade de um recurso, pode-se ter enfoques diferentes para representar confiança, dentre eles a confiança de término da prestação do serviço no prazo e a confiança na integridade da aplicação e dos dados. As abordagens que se baseiam na integridade do recurso para realizar a seleção utilizam como métricas a reputação do recurso, a capacidade de defesa do recurso a ataques maliciosos e o histórico de ataques ao recurso. Seguindo esse objetivo, o seletor poderá evitar a escolha de recursos não confiáveis. As abordagens que se baseiam na confiabilidade da execução do serviço normalmente devem se basear nos dados históricos ou informações providas pelos usuários para caracterizar a correta execução da aplicação, a ocorrência de falhas e a probabilidade de término da execução do serviço no prazo previsto.

Quando a seleção é orientada pelos requisitos mínimos, existem mecanismos que verificam as configurações das máquinas para atender às necessidades das aplicações. Essa verificação pode ser feita pela combinação de anúncios [Condor 2005] ou pelo uso de um serviço de diretório [Nassif *et al.* 2005b]. Também se distinguem pelo processamento que pode ser feito de forma centralizada [Condor 2005], distribuída [Nassif et al. 2005b] ou hierárquica [Cao *et al.* 2001].

Técnicas

Quando a seleção tem como objetivo o desempenho, pode-se utilizar técnicas de predição de desempenho capazes de prever em quanto tempo um *job* será executado. Existem várias técnicas de predição de desempenho[5] que foram aplicadas nos trabalhos revisados, dentre elas, simulação, modelo analítico [Cao 2001], modelo probabilístico [Sun & Wu 2003], dados históricos [Jang et al. 2004] e *benchmark* [Elmroth & Tordsson 2004]. Principalmente quando se usa dados históricos, identificamos também o uso de técnicas de inteligência artificial (IA), dentre elas, redes neurais [Castellano *et al.* 2004], lógica *fuzzy* [Sang & Hwang 2004], raciocínio baseado em casos [Nassif *et al.* 2005a] e algoritmos genéticos [Smith *et al.* 1998]. Essas técnicas de IA também foram identificadas quando a confiabilidade é o objetivo principal associado à seleção de recursos [Alunkal et al. 2003]. Na grande maioria dos trabalhos relacionados, identificamos que os mecanismos de verificação de requisitos da aplicação e de restrições de acesso ao recurso são feitos utilizando-se um processo de combinação das características solicitadas com as características providas pelos recursos. Esse mecanismo também pode ser provido pela utilização de um modelo baseado em políticas, conforme apresentado no Capítulo 5 desta tese.

2.5.1 Taxonomia

A taxonomia caracteriza e classifica as abordagens que realizam seleção de recursos no contexto de grades computacionais. A taxonomia proposta consiste na organização dos sistemas de seleção de recursos em grades computacionais segundo os tipos de decisores, os objetivos da seleção, o número de recursos selecionados e o modo como é feita a tomada de decisão. A Figura 2-2 apresenta a taxonomia proposta, onde cada sistema de seleção de recursos em grades computacionais pode ser classificado. A classificação é feita seguindo o fluxo indicado no início da Figura 2-2, onde, para cada fator de classificação, é exigido o preenchimento completo das características (pela indicação do conector lógico "e") ou o preenchimento de pelo menos uma das características (pela indicação do conector lógico "ou"). Quando o sistema não considera alguns objetivos, o caminho indicado como

[5] A seção 4.3 apresenta mais detalhes sobre técnicas de predição de desempenho.

tracejado deve ser seguido. Dessa forma, um único caminho é construído para cada sistema classificado.

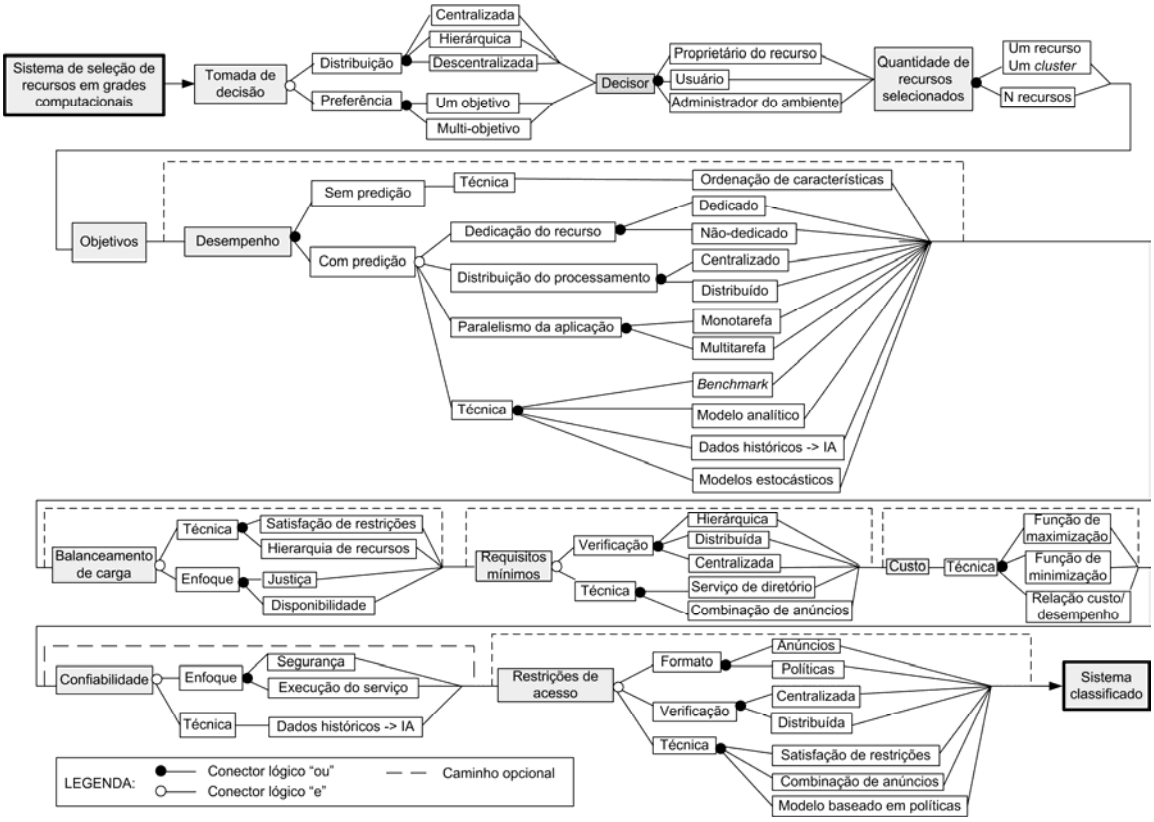

Figura 2-2: Taxonomia para sistemas de seleção de recursos em grades computacionais

2.5.2 Classificação dos sistemas existentes segundo a taxonomia proposta

A Tabela 2-1 apresenta um exemplo de classificação dos trabalhos relacionados segundo a taxonomia proposta para os objetivos e as correspondentes técnicas utilizadas em cada solução.

Tabela 2-1: Classificação dos sistemas segundo taxonomia do decisor, objetivos e técnicas utilizadas

Sistemas	Objetivo	Técnica utilizada
Condor [Condor 2005]	Acesso ao recurso	Combinação de anúncios
	Desempenho	Ordenação das características
Modelo de satisfação de restrições [Liu & Foster 2004]	Acesso ao recurso	Satisfação de restrições
Gangmatching [Raman *et al.* 2003]	Acesso ao recurso	Combinação de anúncios
Serviço de reputação do recurso [Alunkal *et al.* 2003]	Confiabilidade	Dados históricos (fornecido por usuários)
Gerenciador de recursos [Lee *et al.* 2004]	Confiabilidade	Dados históricos -> IA-> Algoritmo genético
SARAH [Song & Hwang 2004]	Confiabilidade Custo	Dados históricos -> IA -> Lógica *fuzzy* Relação custo/desempenho
Algoritmo de *brokering* de recurso [Elmroth & Tordsson 2004]	Desempenho	*Benchmark*
Modelo de seleção de recurso [Castellano *et al.* 2004]	Desempenho	Dados históricos-> IA-> Redes neurais
GHS [Sun & Wu 2003]	Desempenho	Modelos estocásticos
ARMS [Cao *et al.* 2002]	Desempenho	Modelo Analítico
	Balanceamento de carga	Hierarquia de recursos
Sistema planejador de recursos [Jang *et al.* 2004]	Desempenho	Modelo analítico Dados históricos
Algoritmo genético [Smith *et al.* 1998]	Desempenho	Dados históricos-> IA ->algoritmo genético
MASK [Nassif *et al.* 2006]	Acesso ao recurso	Modelo baseado em políticas;
	Desempenho	Dados históricos ->IA->RBC
	Confiabilidade	Dados históricos
	Custo	Relação custo/desempenho

Embora a revisão da literatura aponte esforços para a seleção de recursos em grade computacional com enfoques isolados voltados para o desempenho, confiabilidade, custo e acesso ao recurso, ainda é discutível a obtenção da melhor seleção quando múltiplos atributos são preferenciais. Por isso, é importante a busca por melhores soluções que utilizem novas técnicas e que analisem os fatores relevantes para a seleção de recursos no ambiente de grade computacional de forma integrada.

A nossa abordagem, construída através de um sistema multiagente, realiza a seleção satisfazendo a diversos objetivos: requisitos mínimos da aplicação (velocidade do processador, memória e quantidade de processadores), acesso ao recurso, desempenho, disponibilidade e custo. Essas informações, associadas à preferência do usuário, conduzem a uma melhor seleção de recurso no ambiente da grade. A Figura 2-3 apresenta a classificação da nossa abordagem segundo a taxonomia proposta, onde o caminho da classificação aparece destacado. Dessa forma é possível verificar o conjunto de fatores que

serão verificados e o conjunto de fatores que não serão verificados na tese. O problema da seleção do melhor recurso em grade computacional é aqui resolvido pela teoria da decisão e será apresentado formalmente no Capítulo 6.

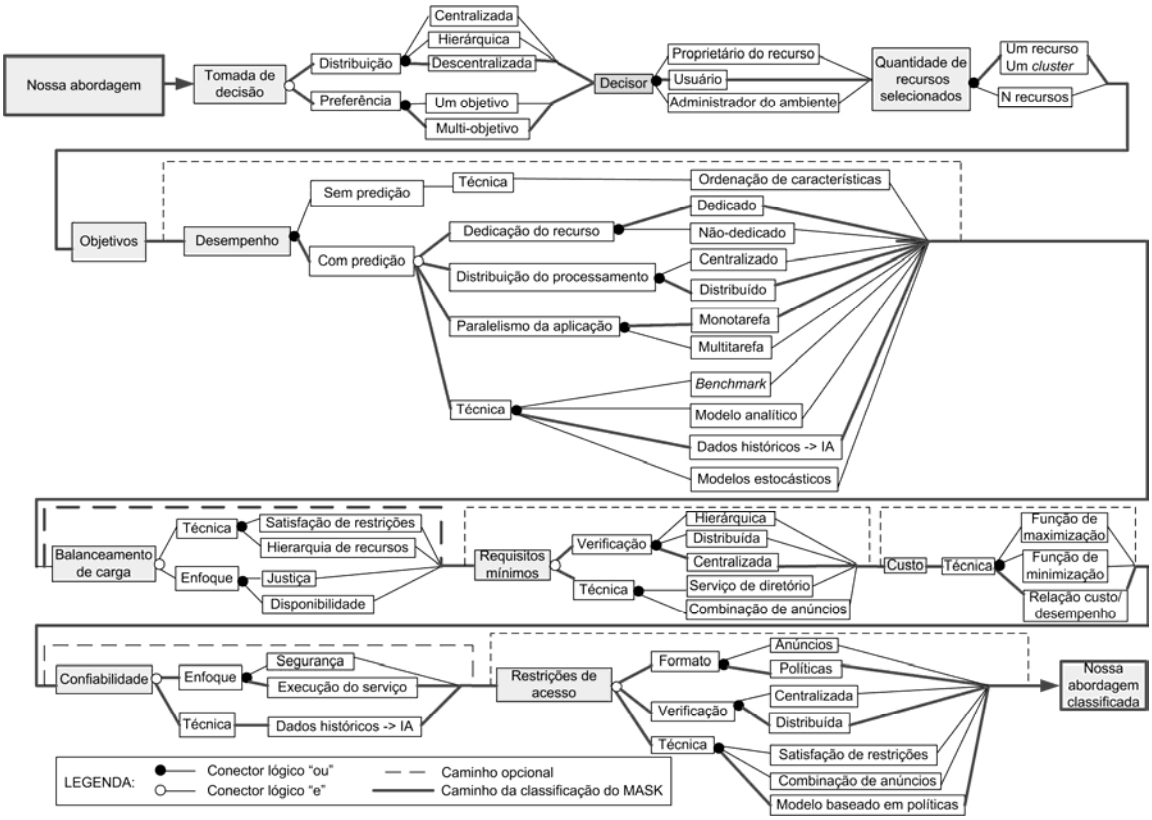

Figura 2-3: Classificação da nossa abordagem segundo taxonomia proposta

2.6 Conclusões

A tecnologia de grades computacionais é um novo tipo de computação distribuída com novas características e desafios. O compartilhamento de recursos entre diversas entidades e indivíduos gera uma série de demandas para organizar esse tipo de ambiente, dentre elas, a descoberta, seleção e gerenciamento de recursos, além de segurança.

A nossa abordagem foca no problema de seleção de recursos, que é um desafio presente na camada de serviços coletivos da arquitetura de grade. Embora o problema da seleção de recursos em grade computacional seja tratado em diferentes abordagens, nenhuma delas articula de forma completa os fatores importantes desse contexto e possibilita seleção pela preferência multiatributo. Desenvolvemos neste capítulo uma taxonomia para a seleção de

recursos em grade de acordo com o decisor da seleção e seus objetivos e classificamos as abordagens estudadas de acordo com a taxonomia proposta.

Destacamos ainda neste capítulo os fundamentos das tecnologias de agentes e sistemas multiagentes. Por serem as grades computacionais ambientes distribuídos e sendo os sistemas multiagentes apropriados para a resolução de problemas cuja distribuição física ou funcional é inerente, iremos explorar a conjugação dessas tecnologias nessa tese com o objetivo de atingir eficiência na solução do problema de seleção de recursos em grade.

Capítulo 3

MASK: *Multi-Agent System broKer* para grades computacionais

3.1 Introdução

Seja o cenário onde um usuário deseja submeter um *job* para uma das máquinas de um ambiente de grade computacional e deve selecionar, em tempo real, a máquina que melhor atenda às suas expectativas. O *job* aqui tratado é um *batch job* não-interativo e pode abranger qualquer tipo de processamento (intensivo em CPU, intensivo em I/O ou intensivo em RAM) e pode ser de curta ou longa duração. Uma solução para esse problema deve considerar os seguintes requisitos:

a) escalabilidade: a solução deve ser escalável para atender à comunidade de usuários de grades computacionais, que pode atingir grande número de usuários e equipamentos heterogêneos e distribuídos;
b) custo de processamento em tempo de execução: a solução deverá selecionar o melhor recurso em tempo de execução;
c) preferências do usuário: a solução deverá considerar a preferência do usuário na decisão;
d) estabelecimento de SLA: a solução deverá considerar que cada execução do *job* pode ser uma transação negociada entre recurso e usuário, com previsão de quanto

tempo o serviço será executado, o custo envolvido e a garantia para provimento do serviço;

e) evidências e aprendizado: a solução deverá possuir processos de aprendizado e utilizar o conhecimento adquirido para melhorar o processo de tomada de decisões.

Este capítulo apresenta o *middleware* MASK (*Multi-Agent System Resource broKer* - para grades computacionais) de seleção de recursos compartilhados entre as entidades de uma organização virtual para executar uma determinada aplicação submetida pelo usuário. A escolha do recurso é considerada a melhor possível pelo fato de ponderar a predição para o término da execução do *job*, a política de acesso ao recurso, o preço associado à execução do serviço e a preferência do usuário entre preço e desempenho. O MASK utiliza a tecnologia de multiagentes e funciona como um *middleware* entre o usuário e o ambiente de grade computacional. A arquitetura do MASK é dividida em camadas funcionais, sendo uma de negociação, uma de seleção de recursos e outra de interface com entidades externas.

Esse capítulo está organizado da seguinte maneira: a seção 3.2 apresenta a arquitetura do MASK; a seção 3.3 descreve sucintamente os componentes da camada de negociação; a seção 3.4 descreve os componentes da camada de seleção; a seção 3.5 apresenta como a arquitetura MASK está interfaceada com usuários, o *middleware* da grade e os recursos. Finalmente a seção 3.6 apresenta comentários finais sobre a estrutura do sistema multiagente MASK.

3.2 Arquitetura funcional do sistema multiagente

Uma visão geral sobre a arquitetura do sistema multiagente MASK é apresentada na Figura 3-1. A arquitetura consiste das camadas de negociação, seleção e interface. Na camada de negociação são agrupados os agentes e as funções relacionadas ao processo de negociação de SLAs. A camada de seleção combina os SLAs estabelecidos pelos agentes e decide onde executar o *job*. A camada de interface integra os agentes com o usuário, com a plataforma de agentes e com uma arquitetura aberta de serviço da grade - *Open Grid Service Architecture* (OGSA) [Globus 2005]. A arquitetura MASK tem dois níveis hierárquicos diferentes associados com aspectos do intra-domínio e do inter-domínio, tais como segurança e escalonamento.

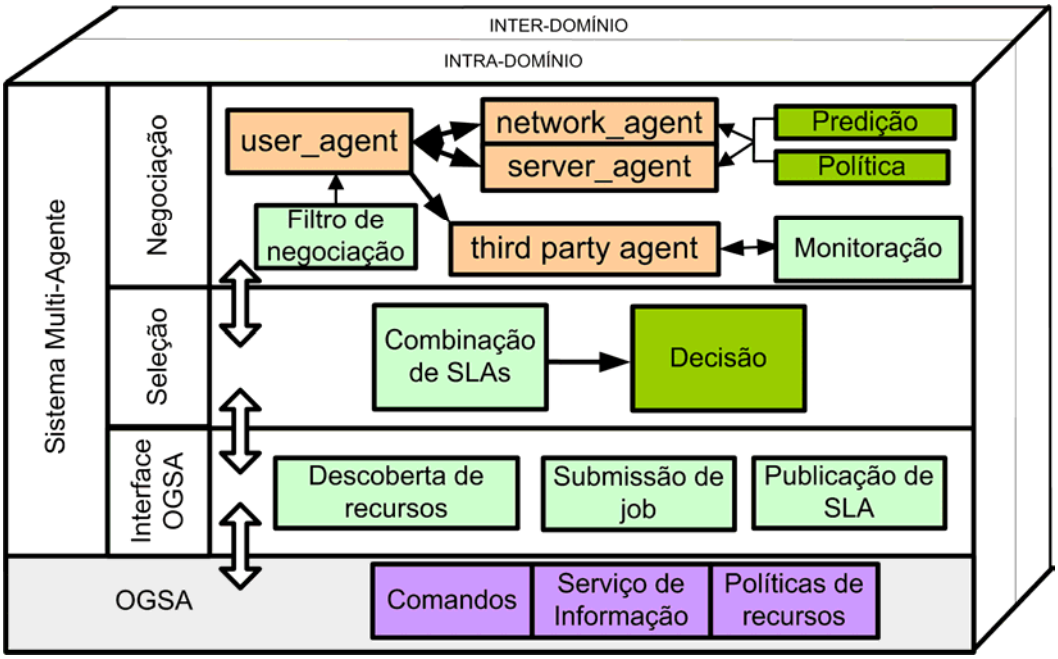

Figura 3-1. Arquitetura do MASK

No MASK existem agentes que representam usuários (*user_agents*), agentes que representam recursos (*server_agents* e *network_agents*) e agentes neutros (*third_party_agents*) que gerenciam os acordos alcançados entre os agentes do ambiente. Tal solução foi integrada com o Globus, considerado o padrão *de facto* para grades computacionais.

As próximas seções deste capítulo apresentam todas as funções presentes na arquitetura MASK (conforme Figura 3-1). Algumas dessas funções são complementares e de menor importância na solução, como por exemplo, o filtro de negociações (seção 3.3.1), a monitoração de *jobs* (3.3.4) e a interface (3.5). Os principais módulos da solução e, portanto as maiores contribuições da tese, são os modelos discutidos para os módulos de predição, de verificação de restrições de acesso e de seleção. Tais modelos serão descritos em detalhes nos capítulos 4, 5 e 6.

3.3 Camada de negociação

Na camada de negociação é identificado quem está negociando, o que está sendo negociado, os tipos de negociações e como as negociações acontecem. As negociações são realizadas entre *user_agents* e *server_agents* ou entre *user_agents* e *network_agents*. Entre

os *user_agents* não há uma modelagem de comportamento voltada para a competição pelos recursos. Os agentes negociam o preço para acessar o recurso, a garantia para a execução do *job* e a métrica de avaliação da garantia. Os elementos da negociação são representados pela tupla <*u,x,y,pr,tj,m,g*,**J**>, onde *u* representa o agente do usuário; *x* representa o domínio a qual *u* pertence; *y* representa o recurso consultado na negociação; *pr* representa o preço negociado; *tj* representa o tempo para início do *job*; *m* representa a métrica de qualidade de serviço; *g* representa a garantia associada à métrica *m*; e **J** representa a descrição do *job*. As negociações podem ser bilaterais, de múltiplos temas e encadeadas [Nassif *et al.* 2004b]. Uma negociação bilateral ocorre quando existem duas partes envolvidas. Em uma negociação de múltiplos temas, diferentes aspectos são negociados. Em uma negociação encadeada, há dependências nas negociações que ocorrem em seqüência. A negociação entre *user_agent* e network_agent é uma negociação encadeada, pois possui dependência do preço e horário já estabelecidos na negociação entre *user_agent* e server_agent.

Uma negociação bilateral de múltiplos temas pode ser executada de várias formas, tais como: 1) negociação de múltiplos temas como um pacote [Zhang & Lesser 2002] (Figura 3-3, possibilidade 1); 2) negociação de cada tema separadamente, em seqüência, descartando algumas propostas (Figura 3-3, possibilidade 2); e 3) negociação de cada tema separadamente, em execução paralela, checando interseção dos resultados de cada tema (Figura 3-3, possibilidade 3). A negociação de múltiplos temas como um pacote (possibilidade 1 da Figura 3-3) é a negociação mais apropriada, uma vez que estamos considerando que a negociação de um tema pode influenciar a negociação de outros.

Uma negociação é denominada encadeada quando é necessária a transferência de dados no ambiente. Neste caso, existem dependências entre o *user_agent* e o *network_agent* e a negociação prévia realizada entre o *user_agent* e o *server_agent*. Tais dependências são chamadas de precedência de localização e precedência de tempo e de preço.

Existe precedência de localização entre os recursos onde os dados estão localizados e os recursos para onde os dados serão transferidos. Os enlaces de comunicação que interconectam esses pontos podem ser representados como uma matriz $M(D_i,S_j)=[L_{ij}]$, onde D_i representa um servidor onde o dado requisitado pelo usuário está disponível, S_j representa um servidor selecionado pelo *user_agent* para negociar, e L_{ij} é um enlace de comunicação de dados que conecta D_i a S_j. Os L_{ij} são ordenados baseando-se na largura de

banda e *n* elementos são selecionados deste conjunto. Os *network_agents* representam esses *n* elementos a serem negociados com o *user_agent* em uma negociação encadeada.

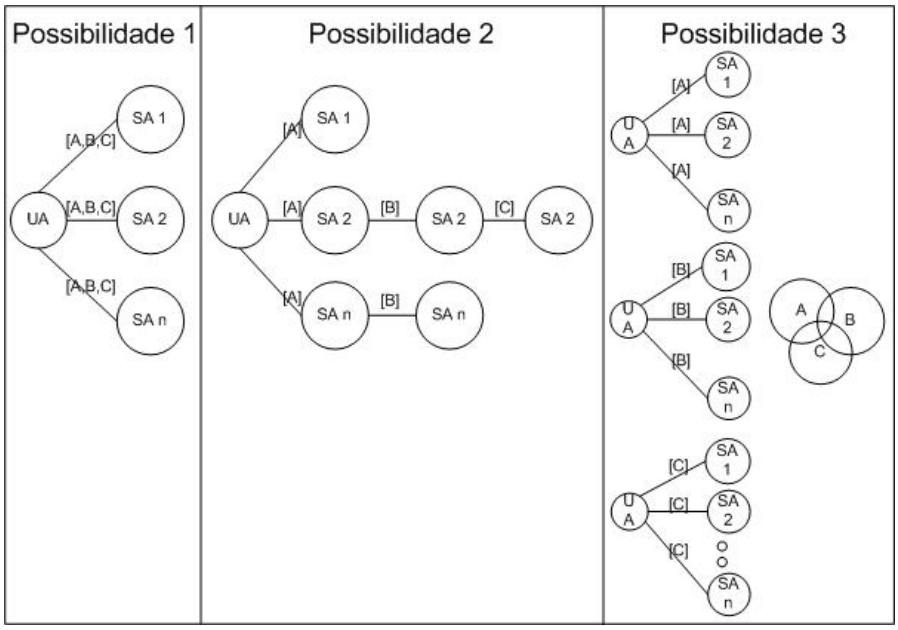

Figura 3-2. Negociação de vários temas (A,B,C) entre *user_agent*(UA) e server_agent(SA)

Existe precedência de tempo em uma negociação encadeada, uma vez que o tempo para terminar um *network*_SLA (NSLA) tem que ser inferior ao tempo para iniciar um *server*_SLA (SSLA); portanto, nsla.término < ssla.início. Existe precedência de preço em uma negociação encadeada, uma vez que o preço a ser negociado tem de ser menor ou igual ao preço máximo definido pelo usuário, diminuído do preço já negociado no SSLA; portanto, nsla.preço <= preçoMax – ssla.preço.

Os procedimentos referentes ao filtro de negociação, negociação baseada em política, predição de desempenho e monitoração da execução do *job* são procedimentos complementares ao processo de interação entre os agentes durante a negociação e são descritos a seguir.

3.3.1 Filtro de negociação

O filtro de negociação é um procedimento usado pelos *user_agents* para reduzir o número de servidores candidatos para negociar. Em um ambiente de grade computacional com muitos equipamentos pode-se reduzir o número de candidatos considerando um

conjunto de equipamentos que tenham maior probabilidade de serem os escolhidos. Dentro desse conjunto destacam-se as máquinas mais rápidas, as máquinas logicamente mais próximas ao dado a ser processado e as máquinas mais ociosas. Esses fatores devem ser associados com o total de máquinas do ambiente e o total de máquinas que atendem aos requisitos solicitados pelo usuário de forma a filtrar as máquinas mais promissoras à seleção.

Os fatores que influenciam o filtro das negociações podem ser definidos como RS= (L, Perf, I, C, PS, CS), onde RS representa o conjunto de recursos selecionados para a etapa de negociação; L representa a localização do servidor em relação à localização dos dados; Perf representa o desempenho da máquina identificado pela velocidade da CPU; I representa o grau de ociosidade da máquina; C representa o grau de confiabilidade do recurso; PS representa o percentual de servidores a serem selecionados para a negociação; e CS representa o conjunto de servidores candidatos. A seleção é feita utilizando uma função $U=(L*p_1 + Perf*p_2 + I*p_3 + C*p_4)$, onde pesos são associados às características. Um conjunto de PS*CS servidores candidatos são selecionados, considerando aqueles que obtiveram resultados mais elevados na ordenação de U.

3.3.2 Predição de desempenho de recursos

A predição de desempenho em grades computacionais é um dos muitos e importantes desafios a serem superados, já que a predição é a base para a construção de um SLA usado para monitorar e garantir o nível do serviço. A predição de desempenho prevê durante quanto tempo um recurso irá executar um *job*.

Na nossa solução, durante o processo de negociação, o server_agent prevê a duração da execução de um *job* baseando-se no tempo de execução de *jobs* similares para o recurso considerado. O MASK usa uma técnica de predição para saber quando um *job* irá terminar e, dessa forma, pode associar garantias em caso de descumprimento do tempo de execução previsto nos SLAs negociados. Utilizamos a técnica de raciocínio baseado em casos (RBC) que prevê o desempenho de um recurso na execução de um *job* baseado em casos passados. Se um *job* com características similares já rodou na mesma máquina, é possível prever o tempo de execução de outro *job*. Com a predição, é possível estimar quantos SLAs são associados ao mesmo tempo em cada recurso e melhorar o processo de escalonamento [Smith *et al.* 1998].

O capítulo 4 descreve em detalhes como a predição está inserida no sistema multiagente MASK.

3.3.3 Verificação de restrições de acesso baseada em políticas

A negociação e a escolha do recurso para a execução de um *job* precisam considerar fortemente as políticas locais definidas para cada equipamento. É através das restrições de acesso ao recurso que são descritas as restrições do tipo "a que horas o usuário *u* pode usar o recurso *y*" e "qual o preço para a computação do *job j* no recurso *y*". A escolha do recurso deve estar condicionada à permissão de uso do mesmo. Portanto, em nossa solução, os agentes negociam o acesso aos recursos conforme suas políticas locais. Este procedimento é usado pelos *server_agents* ou *network_agents* para elaboração de contrapropostas baseadas nas restrições de acesso do recurso.

O capítulo 5 descreve detalhadamente como a política está inserida no sistema multiagente MASK.

3.3.4 Monitoração da execução do *job*

Monitorar a execução do *job* é fundamental para verificar o cumprimento do acordo do nível de serviço. Também é importante coletar os dados do ambiente que possam contribuir para a seleção do recurso, tais como a carga de trabalho no servidor e a taxa de sucesso na execução de *jobs*. O procedimento de monitoração aqui desenvolvido acompanha o comportamento do *job* desde a sua submissão até o final de sua execução. Em ambientes heterogêneos como o de grades computacionais, diversos usuários podem estar submetendo *jobs* para vários recursos simultaneamente. No corrente trabalho definimos e implementamos a monitoração do *job* necessária à complementação do processo de predição e da aplicação da garantia do serviço oferecido. A seguir descreveremos o módulo de monitoração.

Tão logo o acordo na negociação seja estabelecido, os agentes que representam usuários e servidores registram esse acordo em estruturas de dados locais. Tais estruturas contêm a identificação da negociação, o horário para submissão do *job* e a previsão de duração da execução.

Após a seleção, o agente que representa o usuário (*user_agent*) altera o seu estado para inativo e somente retorna ao estado ativo quando chegar o horário da execução do *job*. O

user_agent, nesse momento, submete o *job* e envia uma mensagem para o agente que monitora *jobs* na máquina escolhida (*third_party_agent*) para monitorar a carga de trabalho paralela àquele processo. Quando o *user_agent* recebe o arquivo resultante da execução, ele informa ao *third_party_agent* para finalizar a monitoração da carga de trabalho paralela a esse *job* e analisar o cumprimento dos níveis de serviço. Portanto, ao final da execução do *job*, o *third_party_agent* registra a duração da execução do *job*. O capítulo 4, que aborda a predição do tempo de execução de *jobs*, apresenta em detalhes a composição das estruturas de dados que são atualizadas pelo módulo de monitoração.

3.4 Camada de seleção

A camada de seleção é responsável pela decisão de onde executar o *job* no ambiente da grade computacional, ou seja, é responsável pela escolha do recurso entre aqueles disponíveis na grade. Ela combina os SLAs estabelecidos na camada de negociação e toma a decisão de onde executar o *job* conforme descrito a seguir.

3.4.1 Combinação de SLAs

Este módulo é utilizado quando existe a negociação encadeada, ou seja, quando o *user_agent*, além de negociar com o *server_agent*, também negocia com o network_agent. Caso a negociação encadeada seja desnecessária pelo fato de não haver transferência de dados na rede, esse módulo é dispensável e o módulo de decisão é suficiente para realizar a seleção do recurso. O presente módulo combina SLAs estabelecidos na camada de negociação e que estão interconectados e associados ao provimento do mesmo serviço. O procedimento de combinação de SLAs é modelado utilizando o grafo G, direcionado e acíclico G=(V,E), onde cada vértice em V representa um SLA. Cada borda denota a precedência entre SLAs no tempo escalonado e acumula parâmetros de SLA. A Figura 3-4 mostra um exemplo do grafo que combina SLAs. As camadas representam FSLA (Final SLA), NSLA (*Network* SLA) e SSLA (*Server* SLA). A raiz do grafo representa o SLA que será selecionado pelo módulo de decisão. Um nó é representado pela tupla <Id,*pr,***Q,** *tj,ej***,G**>, onde Id representa a identificação do SLA; *pr* representa o preço negociado; Q representa um conjunto de métricas de qualidade de serviço; *tj* representa o horário de início do serviço descrito no SLA; *ej* representa o horário de término do serviço descrito no SLA; e **G** representa um conjunto de garantias.

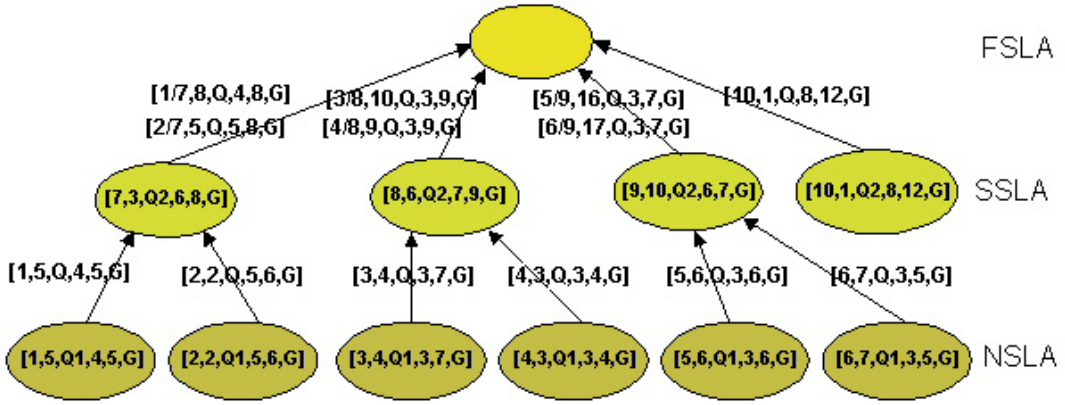

Figura 3-3. Grafo de precedência temporal para combinação de SLAs

O momento de início de um serviço descrito no Final_SLA (FSLA) é o momento de início do serviço descrito no NSLA; o momento de término do serviço descrito no FSLA é o momento de término do SSLA; portanto, `fsla.início = nsla.início` e `fsla.término = ssla.término`. O preço do FSLA é o preço acumulado para todos os SLAs interconectados, definido como `fsla.preço = nsla.preço + ssla.preço`. Os conjuntos **Q** e **G** de um FSLA são compostos de diferentes métricas de qualidade de serviço e de garantias estabelecidas no NSLA e no SSLA respectivamente, onde `fsla.qos = {(nsla,ssla).qos}` e `fsla.garantia = {(nsla,ssla).garantia}`.

3.4.2 Decisão

O módulo de decisão define onde o *job* será executado. Para a tomada de decisão é considerada a preferência do usuário entre preço e desempenho e outros dados recebidos dos demais módulos. Este módulo submete um *job* para o recurso selecionado utilizando comandos Globus ou comandos do sistema operacional. Os SLAs selecionados são publicados no *Monitoring and Directory Service* (MDS) do ambiente Globus. O capítulo 6 descreve em detalhes o módulo de decisão e a seção 3.5.3 descreve como os SLAs são publicados no ambiente Globus.

3.5 Camada de Interface

A camada de interface é responsável pela integração do MASK com o *middleware* de grade computacional, o sistema operacional, o usuário, o ambiente dos agentes e as aplicações. O MASK está situado entre o *middleware* da grade e as aplicações, mas

também tem acesso direto à camada de *software* dos recursos. A Figura 3-5 ilustra como o sistema multiagente aqui desenvolvido pode ser visualizado na arquitetura de grade computacional.

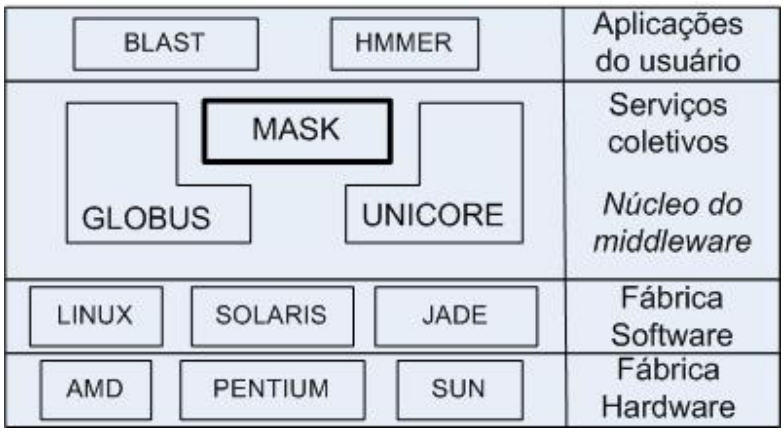

Figura 3-4: Integração entre o MASK e a arquitetura de grade computacional

A camada de interface agrupa o conjunto de mecanismos que interoperam o MASK com o os demais componentes da arquitetura da grade computacional por meio dos seguintes procedimentos: 1) submissão de *jobs*; 2) descoberta de recursos candidatos à negociação; 3) publicação de informações de SLA; 4) verificação de políticas de restrição de acesso utilizando a estrutura de políticas locais do servidor (tal interação é detalhada no capítulo 5). A próxima seção apresenta os aspectos de implementação de tais mecanismos.

3.6 Implementação do MASK

O MASK foi desenvolvido utilizando Jade [Jade 2005], um ambiente robusto para a implementação de agentes (discutido com mais detalhes no Apêndice B). As implementações dos modelos de predição, de verificação de restrições de acesso e de decisão são apresentadas nos capítulos 4, 5 e 6 respectivamente. A seção 3.6.1 apresenta o fluxo de comunicação entre os agentes do MASK e as seções 3.6.2 a 3.6.4 apresentam as implementações das funções presentes na camada de interface da arquitetura do MASK.

3.6.1 Fluxo de comunicação entre agentes e interface do MASK

A Figura 3-2 ilustra o fluxo de comunicação entre os agentes do MASK e sua interface com os ambientes Jade e Globus. Inicialmente o usuário configura o *user_agent* provendo informação sobre preço, qualidade, horário de execução e argumentos do *job*. O *user_agent* usa o serviço de informações do Jade, denominado *Directory Facilitator* (DF), para descobrir recursos que combinem com a descrição dos requisitos do *job*. O *user_agent* filtra um grupo de recursos para negociar com os respectivos agentes que os representam. A negociação envolve envio de propostas pelo *user_agent* e de contrapropostas pelos *server_agents* e *network_agents*. O objetivo de cada negociação é estabelecer um SLA que define o nível do serviço a ser prestado e as penalidades em caso de descumprimento de um acordo feito entre o agente provedor de serviços e o agente cliente de serviços. O *server_agent* negocia com o *user_agent* de acordo com as restrições definidas nas políticas locais do recurso. Além de negociar com *server_agents*, o *user_agent* também negocia com um grupo de *network_agents* associados aos enlaces de rede que conectam os locais onde os dados estão disponíveis com os locais onde os servidores selecionados pelo *user_agent* estão localizados. Um grupo de SLAs é escolhido como referência para o provimento do serviço (detalhes sobre a escolha do grupo de SLAs são discutidos na seção 3.4). Um agente neutro (*third_party_agent*) gerencia todos os SLAs estabelecidos. Tais SLAs estabelecidos entre agentes tornam-se informações públicas no Globus pelo seu serviço de diretório, o *Monitoring and Directory Service* (MDS).

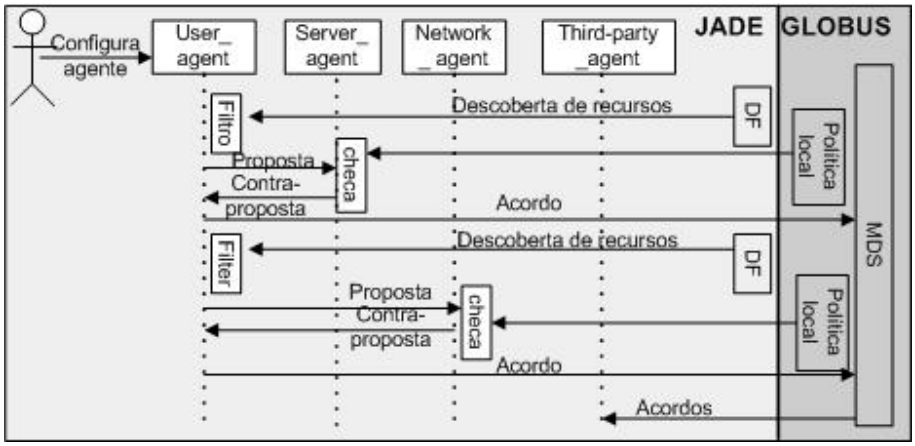

Figura 3-5. Fluxo de comunicação no MASK interfaceando Jade e Globus

3.6.2 Submissão de *jobs*

Um *job* é uma tarefa que deve ser executada em um recurso computacional e é caracterizado por uma série de informações, tais como o nome da aplicação e seus argumentos, o diretório onde se encontram os programas e os dados, o diretório de saída padrão, o tipo da máquina e a memória necessária.

Usando Globus, a submissão de *jobs* pode ser feita com diferentes tipos de comandos, entre eles: 1) globus-*job*-run: para *jobs* interativos; 2) globus-*job*-submit: para *batch jobs*; 3) globusrun: para *job-scripts*; 4) mpirun: para submissão de *jobs* em paralelo; e 5) globus-sh-exec: para executar um *shell script* em máquina remota. As informações associadas a tais comandos devem ser descritas apropriadamente utilizando uma *Resource Specification Language* (RSL). Em outros sistemas de submissão de *jobs*, tais como o *GridLab Resource Management System* (GRMS) [GRMS 2005], a descrição da informação relacionada ao *job* é feita em XML, chamada GRMS *JobDescription* (GJD). Exemplos de descrições escritas em RSL e GJD são apresentados na Figura 3-6.

Exemplo do RSL - Globus	(&(resourceManagerContact="sinbaram6.hpcnet.ne.kr:2119/jobmanager:/O=Grid/O=Globus/CN=cluster.hpcnet.ne.kr") (count=4) (label="subjob 0") (environment=(GLOBUS_DUROC_SUB*JOB*_INDEX 0)) (directory=/edun/globus/Gtest/M3-4s4w4) (executable=/edun/globus/Gtest/M3-4s4w4/mpa8-17c))
Exemplo do GJD - GRMS	< grms*job* appid=”appid”> <simplejob> <executable type=”single”count=”1”> <file name=exec-file” type=”in”> <url> file:////bin/tar</url> </file> <arguments> <value>cfv</value> <value> file.tar</value> <file name=”report” type=”in”> <url> gsiftp://rage1.man.poznan.pl/~/examples/report</url> </file> <file name=”report.tar”type=”out”> <url> gsiftp://rage1.man.poznan.pl/~/examples/report.tar</url> </file> </arguments> </executable> </simplejob> </grmsjob>

Figura 3-6: Exemplo de descrição de *job*s utilizando Globus e GRMS

Para a submissão de *jobs*, o MASK utiliza uma descrição de *job* em formato XML e possui também uma interface gráfica que pode converter tais informações para qualquer que seja a forma da descrição do *job* utilizado pelo *middleware* da grade computacional. No caso desta tese, o MASK está preparado para utilizar o Globus e, portanto, converter as informações de um *job* para RSL. Por não possuirmos o Globus instalado em todas as máquinas do nosso experimento, usamos o MASK como um *middleware* para grade computacional e convertemos os dados da interface para a submissão de *jobs* diretamente para os comandos do sistema operacional da máquina alvo.

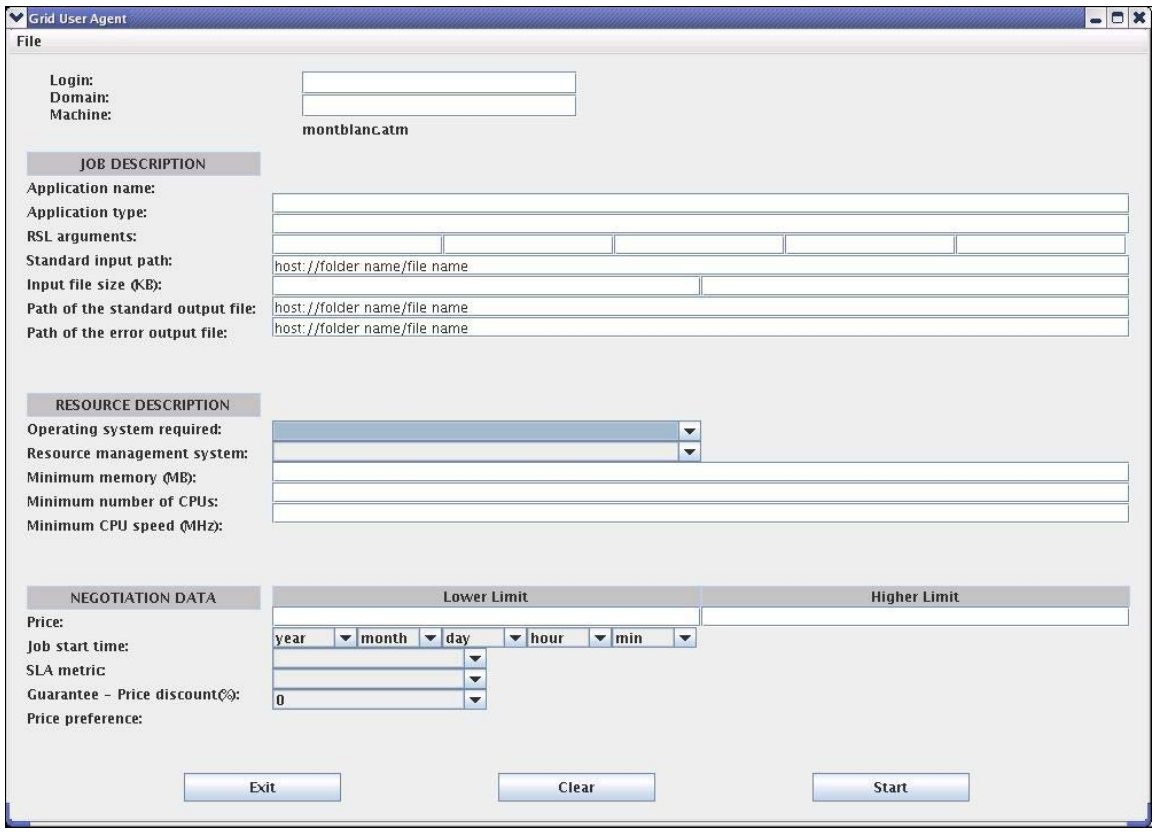

Figura 3-7: Interface do MASK para submissão de *jobs*

Na interface gráfica do MASK, apresentada na Figura 3-7, existem três grupos de informações que são utilizados para diferentes finalidades, sendo eles: 1) *Job description* – com essa informação é possível realizar a predição do tempo de execução do *job*. Essa descrição de um novo *job* é comparada com execuções passadas de *jobs* pelo módulo de predição (descrito no Capítulo 4); 2) *Resource description* – com tais informações é feita a descoberta de recursos candidatos à negociação, contendo a descrição mínima do equipamento solicitado; 3) *Negotiation Data* – com tais informações é possível realizar a

negociação de preço, horário de execução e garantia do serviço conforme a política de acesso de cada equipamento envolvido na negociação através do módulo de verificação de restrição de acesso (descrito no Capítulo 5).

3.6.3 Descoberta de recursos candidatos à negociação

Na nossa solução, a descoberta de recursos que atendem aos requisitos da aplicação é o primeiro passo na negociação entre os agentes. Dentre os diversos equipamentos disponíveis no ambiente compartilhado, somente aqueles que atendem aos requisitos mínimos de velocidade de CPU, memória e quantidade de processadores são escolhidos pelo *user_agent* para iniciar negociações.

As principais características da descoberta de recursos são o registro de serviço, o anúncio de serviço, a descoberta de serviço e a interoperabilidade [Cao 2001]. Os recursos podem ser computadores, aplicações, circuitos de comunicação ou informações. As perguntas que podem ser respondidas por um sistema de descoberta de serviços são: 1) quais serviços estão disponíveis? 2) como o usuário descreve o serviço que ele necessita? 3) usuários e serviços precisam estar registrados? 4) como os provedores de serviço anunciam seus serviços?

Destacamos aqui dois mecanismos usados em soluções diferentes para a finalidade de descoberta de recursos no ambiente de grade computacional, sendo eles o *matchmaking* do Condor e o MDS do Globus.

```
Request=[ user=”maria”;
        Requirements= other.type==”machine”  && Other.mips > 2000;
        Rank= other.memsize]
Resource=[name=”bonete”;
        Type=”machine”;
        Mips=2371;
        Memsize=512M;
        Requirements=DayTime()>’18:00’&&member(other.user,{“maria”, “josé”})]
```

Figura 3-8: Exemplo de ClassAd (Condor)

O processo de descoberta de recursos do Condor chama-se *matchmaking* e consiste na combinação simétrica entre anúncios feitos por recursos e usuários. As requisições dos usuários e as descrições dos provedores são expressas usando a mesma sintaxe denominada ClassAds (ou anúncios)[Liu & Foster 2004]. A Figura 3-8 apresenta exemplos de ClassAds, onde, no primeiro classificado, a usuária “maria” requisita uma máquina com velocidade superior a 500 MHz e solicita uma ordenação de máquinas pela capacidade de

memória. O segundo classificado descreve uma máquina chamada “bonete” com Mips=2371, 512 MB de memória e indica que os usuários “maria” e “josé” podem acessar o recurso após as 18h.

No ambiente Globus, o *Monitoring and Discovery System* (MDS) publica periodicamente os serviços oferecidos pelo ambiente da grade. O esquema da versão MDS 2.4 é baseado no protocolo *Lightweight Directory Access Protocol* (LDAP), uma coleção de definições de tipos de atributos e classes de objetos. O modelo de informações de recursos MDS estrutura os componentes lógicos e físicos de um recurso computacional como uma hierarquia de elementos. O MDS publica informações estáticas para servidores, tais como sistema operacional, tipo de CPU e quantidade de processadores, e informações dinâmicas tais como carga de trabalho média, espaço disponível em disco e latência da rede. Um conjunto de programas para prover informações é acoplado ao sistema MDS e um usuário desenvolvedor do ambiente Globus pode estender o esquema MDS original.

Nas camadas inferiores ao *middleware* MASK estão o ambiente Jade e o ambiente Globus (Figura 3-5). Em ambos existem estruturas de serviço de diretório onde os serviços desses ambientes são anunciados. Portanto, não desenvolvemos um novo mecanismo para a descoberta de recursos, mas aplicamos uma nova solução de interface entre agentes e grades computacionais, onde utilizamos o serviço de diretório dos agentes para descobrir os recursos da grade que são candidatos ao processo de negociação.

No ambiente Jade, o serviço de diretório, chamado *Directory Facilitator* (DF), é um registro centralizado de entradas que associam descrições de serviços a identificação de agentes [Jade 2005]. Na interação entre agentes é importante que os mesmos compartilhem de uma lista de ontologias, protocolos, linguagens e serviços que são caracterizados por propriedades. A nossa solução utiliza a estrutura de propriedades já existente no DF para anunciar as características das máquinas compartilhadas. Os agentes que representam servidores (*server_agents*) rodam comandos locais de identificação de propriedades do recurso (identificado pelo método CmdExec da Figura 3-9). Com o CmdExec o server_agent executa comandos no sistema operacional para obter dados sobre o sistema operacional, memória, espaço em disco, quantidade de processadores e nome da máquina que ele representa. Tais informações são definidas como propriedades dos *server_agents* e esses, assim que iniciam a execução na plataforma de agentes, registram os seus serviços no DF contendo tais propriedades. Dessa forma, o *user_agent* faz uma consulta ao DF

procurando por *server_agents* que possuam as propriedades que atendam às suas necessidades e identifica o conjunto de máquinas candidatas à negociação. Na Figura 3-9, apresentamos o método de registro do agente no DF contendo as propriedades utilizadas nessa fase de descoberta.

```
private void registra()
{
   DFAgentDescription dfd = new DFAgentDescription();
   dfd.setName(getAID());
   dfd.addServices(this.confService());
   try {  DFService.register(this, dfd );}
   catch (FIPAException fe) { fe.printStackTrace(); }
}
private ServiceDescription confService()
{
   String SO, resMan, memoria, cpuProc, cpuSpeed;
   CmdExec dados;
   dados = new CmdExec ("uname");
   SO = dados.saida();
   dados = new CmdExec ("./comandos/memoria");
   memoria = dados.saida();
   dados = new CmdExec ("./comandos/cpuProc");
   cpuProc = String.valueOf(dados.getNumeroResultado());
   dados = new CmdExec ("./comandos/cpuSpeed");
   cpuSpeed = dados.saida();
   Property propSO = new Property("SO",SO);
   Property propMem = new Property("memoria",memoria);
   Property propCP = new Property("cpuProc",cpuProc);
   Property propCS = new Property("cpuSpeed",cpuSpeed);
   Property propMaq = new Property("maquina", this.maquina);
   ServiceDescription sd  = new ServiceDescription();
   sd.setType("server");
   sd.setName(getLocalName());
   sd.addProperties(propSO);
   sd.addProperties(propMem);
   sd.addProperties(propCP);
   sd.addProperties(propCS);
   sd.addProperties(propMaq);
   return sd;
 }
```

Figura 3-9: Registro de propriedades pelo server_agent no DF

Para diminuir o número de máquinas no processo de negociação entre agentes, existem diversos parâmetros que podem ser considerados na descoberta de recursos:

1) do ponto de vista do usuário, é melhor escolher um conjunto de máquinas mais rápidas;

2) do ponto de vista do balanceamento de carga, é melhor escolher as máquinas mais ociosas;
3) do ponto de vista da confiabilidade do serviço oferecido, é melhor escolher as máquinas com melhor índice de confiabilidade;

Tais definições podem ser facilmente incorporadas no procedimento de filtragem de negociação (seção 3.3.1) usando o serviço de diretório do Jade, onde propriedades tais como ociosidade e confiabilidade podem ser publicadas.

3.6.4 *Service Level Agreement* como resultado da negociação

O objetivo da negociação entre os agentes é alcançar um SLA que será definido utilizando as estruturas dos agentes e publicado no serviço de informações do *middleware* Globus, MDS [Globus 2005]. Na nossa implementação, o SLA está estruturado a partir da adaptação do esquema apresentado na padronização do *Web Service Level Agreement* (WSLA) [Ludwig *et al.* 2003] [Dan *et al.* 2002], conforme mostra a Figura 3-10.

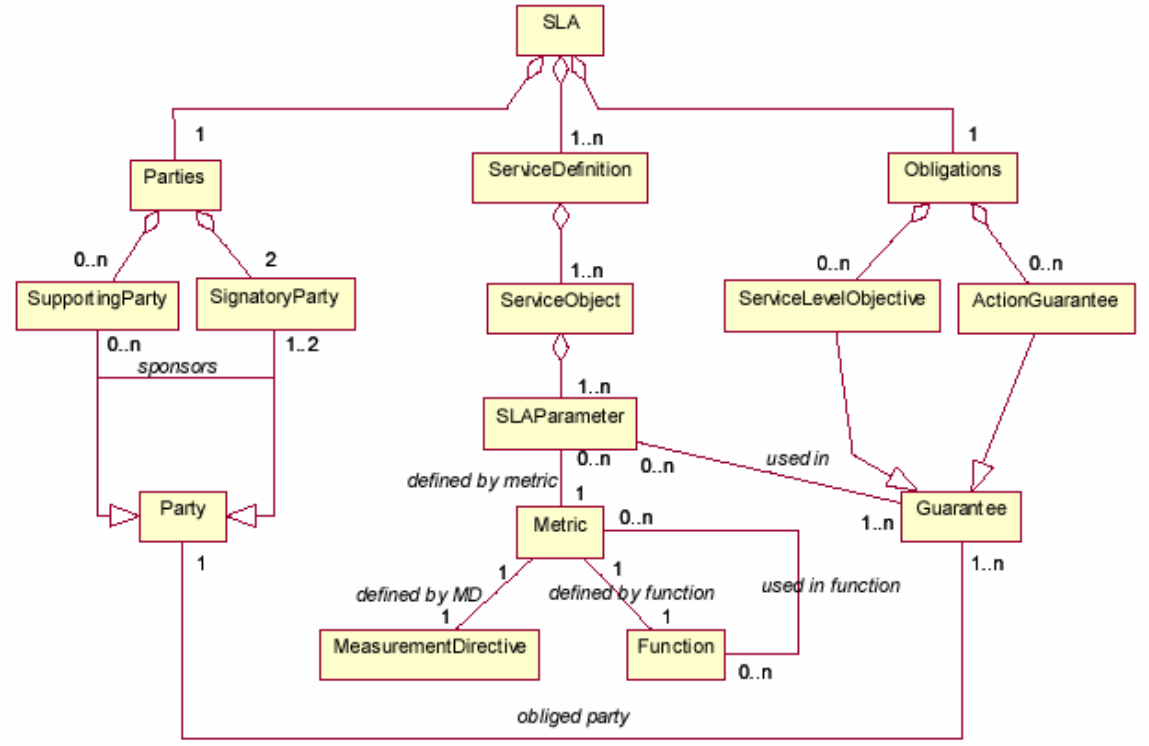

Figura 3-9: Estrutura do *Web Service Level Agreement*
Fonte: [Dan *et al.* 2002]

Os WSLAs representam acordos entre um provedor de serviço e um cliente. Tais acordos definem as obrigações das partes envolvidas. Basicamente, um acordo descreve

que o provedor de serviço deve oferecer garantias para determinados serviços tais como disponibilidade, tempo de resposta e vazão. Cada acordo contém seções denominadas Partes, Serviços e Garantias. A seção Partes descreve as partes envolvidas no acordo. A seção Serviços descreve uma visão comum do serviço que é compartilhada pelas partes do contrato. A seção Garantia expressa um consentimento para manter um estado particular do serviço por um determinado período. A Figura 3-11 apresenta alguns desses atributos que foram adicionados ao esquema do MDS e a Figura 3-12 apresenta a saída do provedor de informações que desenvolvemos para publicar, no MDS, os SLAs acordados pelo sistema multiagente.

SLA	**Alguns atributos SLA para a extensão do esquema MDS**	**Exemplo da instanciação do SLA usando uma negociação feita pelo MASK**
Partes envolvidas	SLA-Parties-ServiceProvider SLA-Parties-ServiceConsumer	SrvAg_montblanc.atm.dcc.ufmg.br UsrAg_itapeva.atm.dcc.ufmg.br
Serviço	SLA-Service-Object-Schedule-Period SLA-Service-Object-Schedule-Interval SLA-Service-Object-SLAParameter-Metric SLA-Service-Object-SLAParameter-Metric	9h 1h Tempo de execução Preço
Garantia	SLA-Guarantees-ServiceLevelObjective-Obliged SLA-Guarantees-ServiceLevelObjectiveEvaluationEvent SLA-Guarantees-ServiceLevelObjective-Expression	10% desconto no preço Fim da execução Se tempo de execução > tempo de predição

Figura 3-11: Alguns atributos de SLA utilizados no MASK que são publicados no MDS

O *Web Service Level Agreement* (WSLA) define SLAs para serviços Web e para cenários de gerenciamento inter-domínio ou gerenciamento de redes, sistemas e aplicações em geral. O WSLA é descrito em uma linguagem baseada em *eXtensible Markup Language* (XML) [Box *et al.* 2000].

Existem outros padrões, além do WSLA, que definem a estrutura de um SLA. Um exemplo é o WS-Agreement [Dan *et al.* 2004]. A abordagem WS-Agreement também foi estudada e considerada para descrever um SLA alcançado pelos agentes em MASK. No entanto, os padrões WSLA e WS-Agreement possuem focos diferentes. O WS-Agreement é parte do esforço que vem sendo realizado pelo *Global Grid Forum* (GGF) [GGF 2005] para padronizar acordos de serviços *Web* e serviços de grade computacional entre um cliente e um provedor de serviço. O WSLA define como os acordos podem ser gerenciados durante o serviço e considera terceiros (*third parties*) para assumir a responsabilidade de avaliar e

gerenciar os níveis de serviços especificados. A tendência é que os padrões WSLA e WS-Agreement tenham convergência e que partes da estrutura WSLA sejam utilizadas em diversas padronizações do GGF. No atual momento, o WSLA está mais completo e inteligível como uma estrutura a ser aproveitada em uma solução como a nossa. O WS-Agreement e outros padrões do GGF ainda estão em construção. Por isso adotamos a estrutura WSLA para gerar o padrão de saída de SLAs no MASK.

```
nassifl@hpc129219:/opt/globus-2.4/etc> /home/nassifl/informationprovider/grid-info-SLA-core -
devclassobj -devobjs -dn Mds-Host-hn=hpc129219.iit.nrc.ca,Mds-Vo-name=local,o=grid -validto-s
ecs 21600 -keepto-secs 240800
dn: Service-Level-Agreement=agent, Mds-Host-hn=hpc129219.iit.nrc.ca,Mds-Vo-name=local,o=grid
objectclass: UoOAgentSLA
objectclass: UoOAgentSLAPartiesServiceProvider
objectclass: UoOAgentSLAPartiesServiceConsumer
objectclass: UoOAgentSLAPartiesSupportingParty
objectclass: UoOAgentSLAServiceObjectSLAParameter-Metric
objectclass: UoOAgentSLAServiceGuaranteesServiceLevelObjective
UoO-Agent-SLA-ID:15012204103598
UoO-Agent-SLA-Parties-ServiceProvider-Contact: hpc129219://nrc/agent/agentnetwork2
UoO-Agent-SLA-Parties-ServiceConsumer-Contact: ontario://nrc/agent/agentlilian
UoO-Agent-SLA-Parties-SupportingParty-Contact: cobra://nrc/agent/third-party-agent
UoO-Agent-SLA-Parties-SupportingParty-Role: Manager
UoO-Agent-SLA-Service-Object-Schedule-Period:Mon Jun 28 08:50:00 EDT 2004
UoO-Agent-SLA-Service-Object-SLAParameter-Metric-Name:bandwidth
UoO-Agent-SLA-Guarantees-ActionGuarantee-Expression: if transfer_time > transfer_prediction
Mds-Service-hn: hpc129219.iit.nrc.ca
Mds-Service-port: 2135
Mds-Service-Ldap-timeout: 30
Mds-Service-admin-contact: Mohamed.Ahmed@nrc.cnrc-gc.ca
Mds-Service-Executable-PID: 9616
Mds-Service-Path: /opt/globus-2.4
Mds-Service-admin-comment: This is an MDS 2.2 deployment.
Mds-Service-Ldap-suffix: Mds-Vo-name=local,o=Grid
Mds-Service-Ldap-suffix: Mds-Vo-name=site,o=Grid
Mds-validfrom: 200406281407172
Mds-validto: 200406282007172
Mds-keepto: 200407010900372
```

Figura 3-12. Saída do provedor de informação com parâmetros SLA

3.7 Conclusões

Para a seleção do melhor recurso para executar um *job* no ambiente de grade computacional é necessário considerar diversos fatores. Uma solução para esse problema deve considerar que: o ambiente é distribuído e necessita de escalabilidade; a seleção deve ser feita em tempo de execução, no momento em que o usuário deseja submeter um *job*; o usuário tem preferências variadas com relação ao tempo de execução e ao preço para executar sua aplicação; o usuário deseja saber com antecedência qual a previsão para a entrega do serviço, qual o custo envolvido e qual a garantia para provimento do serviço. A

solução apresentada aqui se constitui de um *middleware* denominado MASK, que considera essas diversas variáveis.

O MASK é um sistema multiagente que faz o papel de *broker* de recursos para usuários da grade computacional. O MASK é estruturado em uma arquitetura em camadas de negociação, seleção e interface. Dentro da camada de negociação apresentamos o fluxo de informações entre os agentes, os tipos de negociações que ocorrem entre eles, o filtro de candidatos para negociação e o resultado da negociação. Ainda dentro da camada de negociação apresentamos resumidamente os modelos de predição e de verificação de restrições de acesso a serem explorados com mais detalhes nos capítulos 4 e 5, respectivamente. Na camada de seleção apresentamos como são combinados os SLAs alcançados na camada de negociação e descrevemos sucintamente o modelo de decisão que será apresentado em detalhes no capítulo 6. Na camada de interface descrevemos como os *jobs* são submetidos, como os recursos são descobertos e como os SLAs alcançados no ambiente dos agentes são publicados no serviço de diretório do ambiente de grade computacional.

Os componentes da arquitetura do *middleware* MASK foram implementados e testados. Os capítulos 4 e 5 apresentam experimentos que validam os modelos de predição e de verificação de políticas separadamente. O capítulo 6 valida o funcionamento geral do MASK e realiza experimentos para validação do modelo de seleção de recursos.

Capítulo 4

Predição de tempo de execução de *jobs*

4.1 Introdução

Um fator importante na tomada de decisão sobre o melhor recurso para executar um *job* é o desempenho oferecido pela máquina. Uma abordagem possível é escolher a máquina considerando alguma de suas características computacionais influentes no desempenho, tal como a velocidade do processador. Se essa for a estratégia de todos os selecionadores, a máquina de maior poder computacional poderá se tornar muito utilizada, enquanto as demais máquinas do ambiente compartilhado poderão permanecer ociosas. Outros fatores também podem influenciar no desempenho de uma máquina para executar determinado *job*, tais como sua carga de trabalho, a velocidade de leitura e escrita no disco para computação de I/O intensivo e o tamanho da memória cache. Os trabalhos de [Elmroth & Tordsson 2004] e [Cao *et al.* 2002] consideram o desempenho da máquina como fator para a seleção de um recurso utilizando técnicas de *benchmark* e de simulação. Tais técnicas, embora teoricamente apresentem maior acurácia, conforme nossa revisão bibliográfica da seção 4.3, possuem algumas limitações para o cenário que estamos abordando e também apresentam um custo computacional. Primeiro porque, conforme anteriormente argumentado, para um ambiente heterogêneo como o de grades computacionais, novos equipamentos e aplicações podem ser incluídos no ambiente sem nunca terem sido

submetidos a um *benchmark*, não sendo possível obter uma predição para tais recursos. Segundo porque o uso de simulação para produzir uma predição de execução de um recurso pode exigir habilidade do usuário para construir um modelo de simulação ou requerer modelos previamente construídos para cada equipamento e cada aplicação que for executar no ambiente. Tais requisitos tornam essas alternativas complicadas para uso em tempo real por um usuário não especialista que deseja submeter um *job* no ambiente compartilhado ou pela inclusão de novos equipamentos e aplicações no ambiente da grade.

Este capítulo apresenta uma nova abordagem para predição de execução de *jobs* em grades computacionais utilizando o paradigma raciocínio baseado em casos (RBC) [Kolodner 1993]. O objetivo deste capítulo é apresentar um modelo de predição como parte do *middleware* MASK apresentado no capítulo 3. O modelo de predição, chamado PredCase, fornece ao sistema multiagente um dos parâmetros para a escolha do melhor servidor para executar um *job*.

Um sistema RBC utiliza casos passados para recomendar uma solução para um novo caso. Cada descrição de *job* é considerada um caso e a idéia chave é verificar se um servidor específico já rodou um caso similar. O tempo gasto na execução de um caso passado auxilia no cálculo do tempo para executar um novo *job*. Baseando-se nessas idéias, o capítulo apresenta algoritmos de recuperação de casos envolvendo distância geométrica e de relevância, assim como algoritmos de adaptação conforme a carga de trabalho do recurso.

As demais seções deste capítulo estão organizadas como se segue. A seção 4.2 descreve a técnica RBC e referencia as técnicas de predição existentes. A seção 4.3 apresenta o modelo de predição incluído no sistema multiagente MASK. A seção 4.4 apresenta experimentos com o PredCase e a seção 4.5 conclui o capítulo.

4.2 Raciocínio baseado em casos

Raciocínio baseado em casos é uma técnica que adapta soluções passadas para novas demandas usando casos passados para explicar, criticar e interpretar novas situações para um novo problema. O RBC sugere um modelo de raciocínio que incorpora a resolução e o aprendizado do problema e integra casos no processo de memória [Kolodner 1993].

Um caso é um pedaço contextualizado do conhecimento representando uma experiência que ensina uma lição fundamental para alcançar os objetivos do raciocinador. Casos

representam conhecimento específico para situações específicas, ou seja, representam conhecimento no nível operacional, onde é explicitado como uma tarefa foi conduzida, ou como um conhecimento foi aplicado, ou como estratégias específicas foram utilizadas para alcançar determinado objetivo [Kolodner 1993].

O ciclo básico do RBC, ilustrado na Figura 4-1, inclui as fases denominadas Recuperação, Adaptação, Revisão e Retenção. Os casos passados de resolução de problemas são armazeados em uma base de casos. Na fase Recuperação, o sistema RBC compara um novo problema com casos passados e recupera casos similares que sugerem soluções para o novo problema. Na fase Adaptação, o sistema modifica a solução passada para construir uma solução para o novo problema. Na fase Revisão, o sistema RBC revisa a solução proposta. Na fase Retenção, o novo caso e sua solução são incluídas na biblioteca de casos [Kolodner 1993][Lewis 1995].

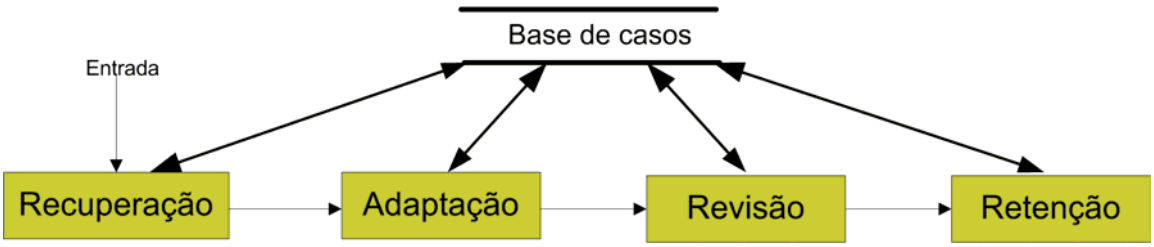

Figura 4-1: Estrutura genérica para raciocínio baseado em casos

As seguintes premissas fazem parte do modelo RBC [Kolodner 1993]:

- Fazer referência a casos passados é vantajoso para lidar com situações que são recorrentes. A referência a situações prévias similares é frequentemente necessária para lidar com as complexidades das novas situações;
- Qualquer forma de raciocínio requer que a situação seja elaborada em detalhes e seja representada com clareza e com vocabulário apropriado para permitir ao decisor adquirir o conhecimento que ele precisa e raciocinar sobre ele;
- Como o caso passado pode não ser o mesmo que o novo caso é preciso adaptar a solução passada para que seja utilizada na nova situação. A adaptação compensa as diferenças entre o a situação passada e a nova;

- O aprendizado acontece como uma conseqüência natural do raciocínio. Se um novo problema é derivado da resolução de um problema complexo, então, por conseqüência, um novo procedimento é aprendido para lidar com essa classe de situações.

A técnica RBC foi introduzida em ferramentas comerciais no início da década de 90 e desde então tem sido usada em aplicações dos mais variados domínios, dentre eles [AIAI 2003]:

- Diagnóstico: O diagnóstico baseado em casos recupera casos passados cuja lista de sintomas são similares ao novo caso e sugerem diagnósticos baseados na melhor combinação de casos combinados;
- *Help-Desk*: Os sistemas de diagnóstico baseado em casos são usados na área de atendimento ao cliente lidando com problemas relacionados a um produto ou serviço;
- Suporte à decisão: Na tomada de decisão, quando encontrado um problema complexo, as pessoas geralmente procuram por problemas análogos para encontrar possíveis soluções;
- Projeto: Sistemas de apoio a projetistas industriais e arquitetônicos têm sido desenvolvidos para apoio no processo de partes do projeto geral.

Este capítulo apresenta a implementação RBC para o problema de predição de tempo de execução de *jobs* em grade computacional e adapta algoritmos para os processos básicos da técnica RBC.

4.3 Técnicas de predição de desempenho

As técnicas de predição de desempenho diferem em três aspectos principais: complexidade, acurácia e custo. [Menascé *et al.* 1994] relatam as seguintes técnicas para a predição de desempenho: 1) regras práticas; 2) análise de tendências; 3) *benchmarks;* e 4) modelos de desempenho (analíticos e de simulação).

As regras práticas são usadas para determinar a capacidade geral de um sistema como uma função de utilização de um componente-chave. Normalmente, as regras práticas são extraídas das informações dos fornecedores e da experiência das pessoas. Um exemplo popular de regra prática é: "a utilização média da CPU não deve ultrapassar 80%". As

regras práticas são fáceis de usar e têm baixo custo, no entanto possuem sérias limitações, pois não consideram as interações entre componentes do sistema.

A análise de tendências prevê o que acontecerá em um sistema baseado nos dados históricos de seu comportamento passado. Frequentemente, as predições são feitas usando extrapolação linear de tendências recentes.

A técnica de *benchmark* traz resultados precisos para o comportamento de um sistema caso a carga de trabalho e os recursos do sistema sejam representados corretamente no *benchmark.* Uma análise de ambientes distribuídos, compostos de equipamentos de múltiplos fabricantes, torna essa técnica complicada e de alto custo.

Para a solução de modelos de desempenho existem duas abordagens: modelos de simulação e analíticos. 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 medidas de desempenho. Para que esses modelos sejam matematicamente tratáveis, eles são menos detalhados que os modelos de simulação. Eles são menos precisos, mas rodam mais eficientemente.

A nossa solução utilizou as técnicas de modelo de desempenho analítico e dados históricos para a predição.

4.4 Arquitetura do modelo de predição de execução de jobs baseado em casos passados

A Figura 4-2 reapresenta a arquitetura MASK que foi descrita no capítulo 3, destacando o módulo de predição. O módulo de predição é utilizado pelos agentes que representam os recursos (*server_agent*s e network_agents) durante o processo de negociação do provimento do serviço com agentes que representam usuários (*user_agents*). O foco desta seção é um dos módulos da arquitetura MASK, denominado PredCase, um modelo de predição que utiliza a técnica de *raciocínio baseado em casos*.

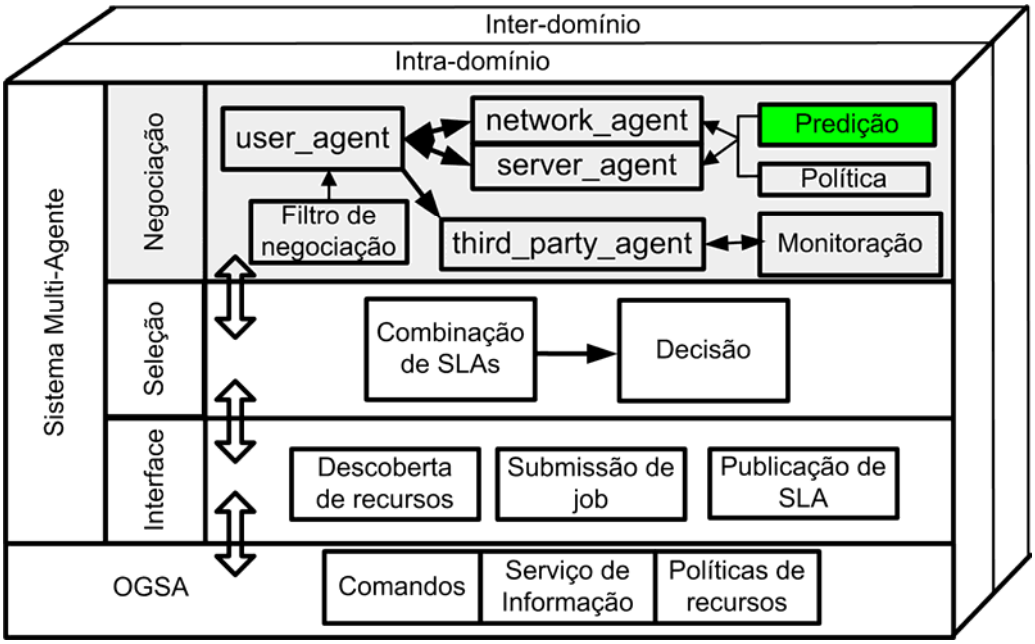

Figura 4-2: Arquitetura do sistema multiagente – destaque para o módulo de predição

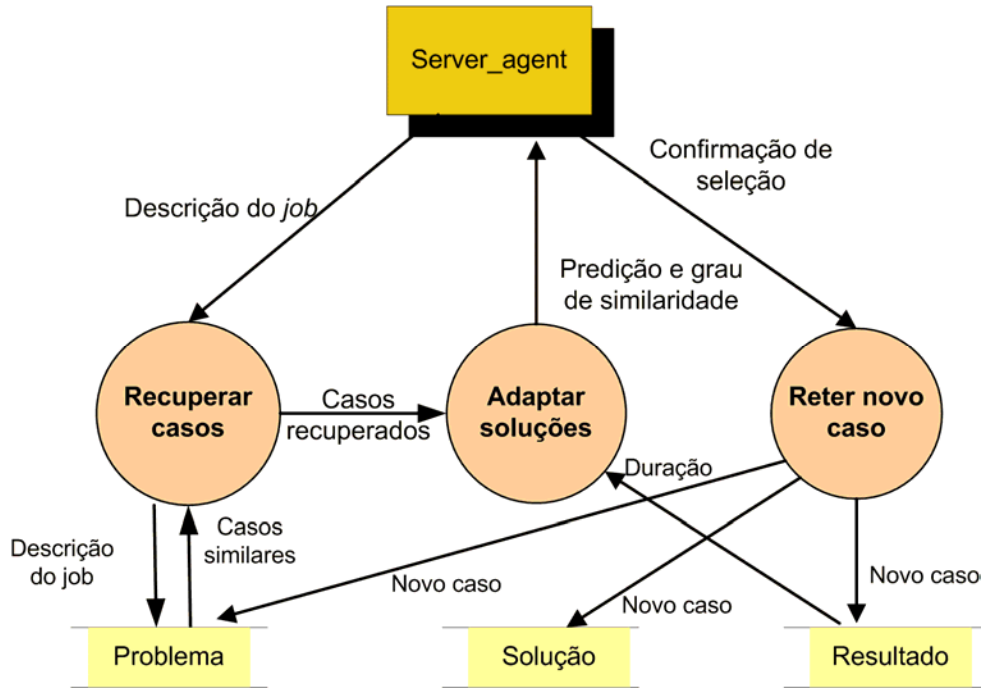

Figura 4-3: Fluxo de dados no PredCase

O fluxo de dados do módulo PredCase, representado pela caixa "predição" na Figura 4-2, é apresentado na Figura 4-3. O PredCase é composto de três processos principais, chamados recuperar casos, adaptar soluções e reter novo caso. Esses processos rodam em

cada *server_agent* que representa um servidor do ambiente da grade computacional e acessam a base de casos do servidor. Faz parte também do PredCase a base de casos, composta de três estruturas de dados, sendo uma para a descrição de casos passados (Problema), uma para a descrição da predição de um *job* (Solução) e uma com dados sobre a execução passada de um *job* (Resultado). As próximas seções fornecem descrições detalhadas do PredCase referentes à organização da base de casos e aos processos de recuperação, adaptação e retenção de casos.

4.4.1 Organização da base de casos

A organização da memória de casos auxilia na melhoria da eficiência da fase de recuperação de casos. Duas principais classes de memórias têm sido propostas: desestruturadas – ou *flat*, e estruturadas. Na memória *flat*, todos os casos são do mesmo nível, a fase de recuperação processa todos os casos e o algoritmo clássico denominado *nearest-neighbor* é o método comum usado para a recuperação de casos. As memórias estruturadas são organizadas por generalizações ou abstrações. Elas podem ser organizadas, por exemplo, pelo uso de hierarquias de camadas, árvore de decisão ou árvores B. Memórias organizadas de forma estruturada facilitam a avaliação de uma nova situação e permitem o controle por indexação [Bichindaritz 2005].

Uma pesquisa em uma lista estruturada é diferente de uma pesquisa em uma estrutura de árvore complexa. Cada uma tem suas vantagens e desvantagens. Na memória *flat*, embora requeira mais tempo de computação, o melhor caso é sempre encontrado. Em uma estrutura de árvore complexa, menos tempo de computação é requerido, porém a recuperação do melhor caso não é garantida [Kolodner 1993].

No MASK, a memória de casos é organizada sequencialmente e um novo algoritmo de pesquisa para essa estrutura de dados é apresentado na próxima seção. Os sistemas RBC controlam o tamanho da memória de casos na fase Retenção, onde o tamanho da memória pode ser ajustado pelos algoritmos de remoção de casos, desenvolvidos de acordo com regras pré-determinadas. Por exemplo, uma regra poderia ser “remova casos não usados e remova casos que tenham influência negativa na predição do tempo de execução”. Para esse propósito, atributos apropriados precisam ser incluídos na estrutura de dados.

No MASK, cada recurso possui seu próprio grupo de casos previamente executados. A estrutura de casos distribuída evita um ponto central de falha e permite o processamento

distribuído. Consequentemente, os agentes que representam recursos calculam predições de forma distribuída.

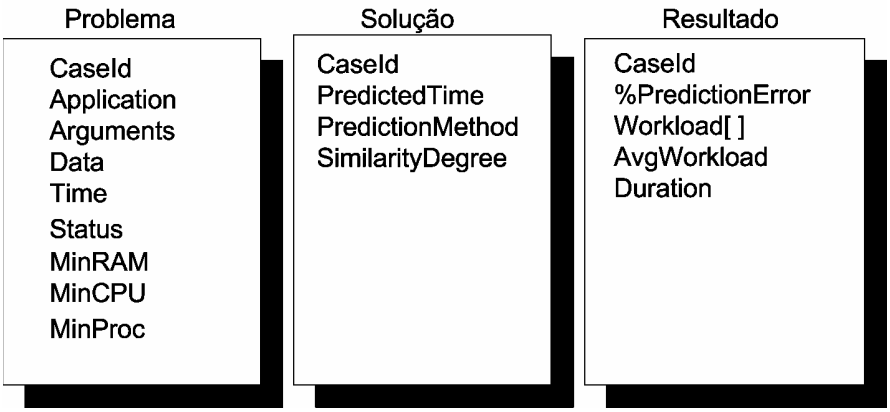

Figura 4-4: Componentes da base de casos do PredCase

Cada recurso possui estruturas de dados chamadas Problema, Solução e Resultado, como ilustrado na Figura 4-4. A estrutura de dados Problema contém atributos que descrevem um *job* que já foi executado no recurso, incluindo a identificação do caso (*CaseId*); o nome da aplicação (*Application*); os argumentos da aplicação (*Arguments*); o tamanho do arquivo de dados (*Data*); o horário de execução do *job* (*Time*); o estado do caso (*Status*); e os requisitos mínimos para executar o *job*, como memória (*MinRAM*), velocidade do processador (*MinCPU*) e quantidade de processadores (*MinProc*). A estrutura de dados denominada Solução contém atributos referentes à predição de execução do *job*, incluindo a identificação do caso (*CaseId*); a predição de execução do *job* (*PredictedTime*); o método de predição utilizado (*PredictionMethod*); e o grau de similaridade do caso em relação aos casos passados similares (*SimilarityDegree*). A estrutura de dados Resultado contém os atributos que verificam a execução do *job*, incluindo a identificação do caso (*CaseId*); a porcentagem de erro na predição (%*PredictionError*); o vetor de registro periódico da carga de trabalho durante a execução do *job* (*Workload*[]); a carga de trabalho média durante a execução do *job* (*AvgWorkload);* e a duração real da execução do *job (Duration)*.

4.4.2 Recuperação de casos

Esta seção inicialmente descreve os conceitos e técnicas envolvidas na recuperação de casos no modelo RBC. Depois apresenta como o PredCase os implementa. Finalmente, a

seção mostra alguns exemplos do comportamento da nossa abordagem e também outras abordagens.

4.4.2.1 Recuperação de casos no RBC

Na fase de recuperação de casos, os casos passados são selecionados de acordo com o grau de similaridade em relação a um novo caso. Para o cálculo de similaridade, existe uma distinção entre similaridade global e local. A similaridade global corresponde à similaridade entre casos e a similaridade local corresponde à similaridade entre atributos do caso [Kolodner 1993]. A similaridade local para diferentes atributos depende da computação de uma função f_i. Uma função pode converter símbolos diferentes em números. A similaridade também depende de uma faixa de valores. Por exemplo, se o valor da função é 1, significa total similaridade, ou se o valor da função é 0 significa nenhuma similaridade. O número de casos recuperados depende de uma marca limite, T, tal que o caso é recuperado somente se a similaridade entre ele e o novo caso é maior do que T. Os seguintes métodos são comumente usados para o cálculo da similaridade local [Aamodt & Plaza 1994].

Numérico $$\mathrm{sim}(a,b) = 1 - \frac{|a-b|}{\text{faixa}}$$

Simbólico $$\mathrm{sim}(a,b) = \begin{cases} 1 \text{ se } a = b \\ 0 \text{ se } a \neq b \end{cases}$$

Multi-valorado $$\mathrm{sim}(a,b) = \frac{\mathrm{card}(a) \cap \mathrm{card}(b)}{\mathrm{card}(a \cup b)}$$

Taxonomia $$\mathrm{sim}(a,b) = \frac{h(\text{nodo comum}(a,b))}{\min(h(a),h(b))}$$

Onde:

a e *b* são atributos de casos diferentes;

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

faixa é o valor absoluto da diferença entre os limites superior e inferior do conjunto;

card é a cardinalidade (tamanho) do conjunto de atributos;

h = altura(número de níveis) da árvore da taxonomia.

Uma vez que a similaridade local tenha sido calculada para cada característica, o sistema RBC deve calcular a similaridade global. Uma maneira de medir a similaridade global entre dois casos A e B com *p* características é dada a seguir:

$$\mathrm{sim}(A,B) = \sum_{i=1}^{p} \omega_i \mathrm{sim}_i(a_i b_i)$$

Onde:

ω_i é o peso do atributo *i;*

sim_i é a similaridade local calculada para o atributo *i.*

Um algoritmo muito popular chamado *nearest-neighbor* recupera casos em sistemas RBC. Neste algoritmo, a soma das características ponderadas de um novo caso é comparada com casos históricos [Kolodner 1993]. O algoritmo seleciona os casos que apresentam mais características em comum com o novo caso.

4.4.2.2 Recuperação de casos no PredCase

No PredCase, o processo de recuperação de casos é feito comparando um novo caso com os casos disponíveis na estrutura Problema. O novo caso e os casos passados possuem os mesmos atributos descritos na Figura 4-4.

Embora o algoritmo *nearest-neighbor* possa ser usado para recuperar casos nos sistemas RBC, seu uso apresenta degradação de desempenho quando a base de casos torna-se muito grande. O algoritmo compara um novo caso com todos os outros casos presentes na base de dados. Aumentando-se o número de casos resulta em problemas de desempenho, uma vez que os cálculos ocorrem em tempo de execução.

Esta seção propõe uma nova solução para o problema de desempenho na recuperação de casos, um algoritmo chamado *refined nearest-neighbor*, que é uma variação do algoritmo *nearest-neighbor*. O novo algoritmo executa o refinamento de casos de acordo com a importância do atributo no caso. Ao invés de associar pesos para todos os atributos e calcular a similaridade para todos os casos, ele calcula a distância geométrica de cada atributo seguindo uma seqüência de atributos relevantes. Com essa modificação, o cálculo do grau de similaridade do segundo atributo mais relevante ocorre somente para 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 relevantes.

No ambiente em que estamos focando, os casos a serem recuperados da base são *jobs* que já foram executados em determinado recurso. O *job* aqui considerado é um *batch job* não-interativo que pode ser intensivo em CPU, intensivo em I/O ou intensivo em RAM e que também pode ser de longa ou curta duração. Um *job* é formado por um conjunto de parâmetros e baseado nos trabalhos [GRMS 2005] e [Globus 2005] destacamos alguns exemplos de sua composição: nome da aplicação; argumentos da aplicação; arquivo de dados; diretório de entrada padrão; diretório de saída padrão; diretório de erro padrão; e requisitos mínimos necessários para a execução, tais como sistema operacional, memória principal, quantidade de processadores e velocidade da CPU.

O algoritmo *refined nearest-neighbor* usa uma generalização para parâmetros de *jobs* em grades computacionais, apresentada pelo vetor E_i de atributos relevantes ordenados, i, onde E_i=[*application, arguments, data*]. Este algoritmo refinado evita cálculos exaustivos para toda a base de casos, pois nem sempre o cálculo de similaridade é necessário para todos os atributos. Por exemplo, calcular a similaridade para os atributos *arguments* pode não fazer sentido quando os atributos *application* de um novo caso e de um caso passado são diferentes. Ou seja, para aplicações diferentes não tem sentido comparar seus argumentos.

Cada fase do refinamento determina a seleção de casos para um valor específico do limite T_i que varia de acordo com o atributo E_i. Um caso C_j permanece na base de casos C se a similaridade local acumulada, SimLA entre C_{ij} (o atributo i do caso j na base C) e o novo caso NC_i (o atributo i do novo caso), é maior que T_i ($SimLA_i(C_{ij},NC_i) > T_i$). A similaridade local, SimL, denota a similaridade entre o atributo i de um novo caso NC_i e o atributo i de um caso passado específico, C_{ij}, de uma base de casos C. SimL é calculada por uma função f_i, descrita adiante. A similaridade global é equivalente a SimLA quando essa acumula a similaridade local para todos os atributos. As equações (1), (2) e (3) abaixo descrevem tais medidas de similaridade:

$$SimL(C_{ij},NC_i) = f_i \quad (1)$$

$$SimLA(C_{ij},NC_i) = SimLA(C_{(i-1)j},NC_{(i-1)}) + SimL(C_{ij},NC_i) \quad (2)$$

$$SimG(C_j,NC) = \sum_{i=1}^{k} SimL(C_{ij},NC_i) \quad (3)$$

De acordo com a equação (1), o cálculo da similaridade local usa diferentes funções, f_i. Tais funções são associadas aos atributos i do vetor E_i=[*application, arguments, data*]. As

definições dessas funções são apresentadas como se segue, onde *application* é um atributo simbólico, *argument* é um atributo multi-valorado, e *data* é um atributo numérico. Na definição da função para atributos multi-valorados há uma recorrência à definição da função $f_i(C_{ij},NC_i)$. Portanto, deve-se analisar novamente se o atributo é simbólico, numérico ou multi-valorado. O atributo *arguments* do vetor E_i é um atributo multi-valorado formado por uma série de argumentos simbólicos. Dessa forma, a definição da função para o atributo multi-valorado é a soma das funções para os atributos simbólicos relativos aos seus argumentos dividido pelo maior número de atributos contidos em C_{ij} ou em NC_i.

$$f_i(C_{ij},NC_i) = \begin{cases} 0 \text{ se } C_{ij} \text{ é simbólico e } C_{ij} \neq NC_i \\ 1.0 \text{ se } C_{ij} \text{ é simbólico e } C_{ij} = NC_i \\ \dfrac{\sum_{y=1}^{card(NC_i)} f_i(C_{i[y]j},NC_{i[y]})}{\max(card(NC_i),card(C_{ij}))} \text{ se } C_{ij} \text{ e } NC_i \text{ são multi-valorados} \\ \dfrac{|C_{ij} - NC_i|}{C_{ij}} \text{ se } C_{ij} \text{ e } NC_i \text{ são numéricos} \end{cases}$$

O algoritmo *refined nearest-neighbor,* que calcula as similaridades locais entre atributos usando o vetor E_i é apresentado na Figura 4-5. Esse é um algoritmo de propósito geral para qualquer aplicação submetida para execução *batch* usando um conjunto de parâmetros; um desses parâmetros é o arquivo de dados cujo tamanho tem influência no tempo de execução da aplicação.

```
Begin
  E=[application, arguments, data]
  T=[1,1.8,2.6]
  Read C, NC
  For j= 1 to length(C) do          // for each case j in base C
     For E[i] = 1 to 3 do           // for each attribute i in vector E
         Calculate Local Similarity: SimL(Cij,NCi)
         Calculate Accumulated Local Similarity: SimLA(Cij,NCi)
          if SimLA(Cij,NCi) < T[i]
              Remove j from C;  break;
          endif
      Endfor
  Endfor
End
```

Figura 4-5: Algoritmo *refined nearest-neighbor*

4.4.2.3 Exemplos de cálculos de similaridades

As Tabelas 4-1 e 4-2 mostram o cálculo da similaridade local para 14 casos usando o algoritmo *nearest-neighbor* e o algoritmo *refined nearest-neighbor* respectivamente. As similaridades locais foram alcançadas ao aplicar as funções f_i definidas anteriormente. Usando o algoritmo *nearest-neighbor*, as similaridades locais são calculadas para todos os atributos incluídos no vetor E. Depois de todos os cálculos, o algoritmo seleciona os casos com graus de similaridade global superiores a um limite específico. No algoritmo *refined nearest-neighbor* os casos são eliminados da base de casos quando a similaridade local acumulada para cada atributo não alcançar um determinado limite. Como resultado, menos cálculos ocorrem no algoritmo *refined nearest-neighbor* comparado com o algoritmo *nearest-neighbor.*

Tabela 4-1: Exemplo de cálculo da similaridade local usando o algoritmo *nearest-neighbor*

Caso\ Atributo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Aplicação	0	0	0	1	1	0	1	0	1	0	0	1	0	0
Argumentos	0	0	0	0.8	1	0	0.8	0	0.8	0	0	0.8	0	0
Dado	0.8	0.4	0.4	0.4	1	0.6	1	0.4	0.8	0.8	0.4	0.8	0.4	0.6
Similaridade global	**0.8**	**0.4**	**0.4**	**2.2**	**3**	**0.6**	**2.8**	**0.4**	**2.6**	**0.8**	**0.4**	**2.6**	**0.4**	**0.6**

Tabela 4-2: Exemplo de cálculo da similaridade local usando o algoritmo *refined nearest-neighbor*.

Caso\ Atributo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Aplicação	0	0	0	1	1	0	1	0	1	0	0	1	0	0
Argumentos				0.8	1		0.8		0.8			0.8		
Dado				0.4	1		1		0.8			0.8		
Similaridade global				**2.2**	**3**		**2.8**		**2.6**			**2.6**		

Limite:1
Limite: 1.8
Liimite: 2.6

4.4.2.4 Outras abordagens de recuperação de casos

A literatura apresenta outras abordagens projetadas para evitar pesquisas exaustivas na base de casos. Em [Schaaf 1996], a base de dados armazena distâncias de similaridade pré-computadas de aspectos específicos entre vários objetos. Para um novo caso A, o algoritmo escolhe um objeto de teste F da base de dados. Se F é dissimilar a A, indica que um objeto F' similar a F também é dissimilar a A. Portanto, o algoritmo elimina objetos similares a F e conseqüentemente reduz o espaço de pesquisa a cada iteração do algoritmo. Em [Montani

& Portinale 2005], os autores levam em consideração as dependências temporais entre casos. Uma classificação preliminar reduz o espaço de pesquisa de recuperação de casos baseado em uma única característica. Um algoritmo de recuperação de casos é então organizado como se segue: 1) os casos com alta similaridade local são extraídos (considerando-se uma característica por vez) e a interseção dos conjuntos recuperados é computada; 2) a similaridade global é também computada, considerando a média ponderada das distâncias locais. Na nossa abordagem de recuperação de casos, a base de casos é filtrada baseada em uma seqüência de atributos relevantes. Essa seqüência evita o uso de pesos para atributos na computação da similaridade global. Diferente das abordagens acima, o algoritmo *refined nearest-neighbor* associa limites aos atributos ao invés de associar aos casos. Essa associação evita computação desnecesssária da similaridade local para alguns atributos. Pelo fato de termos mudado o uso tradicional dos pesos e limites no cálculo de similaridades e por termos utilizado memória *flat*, o algoritmo *refined nearest-neighbor* é uma nova abordagem que sempre encontra os casos mais similares.

4.4.3 Adaptação de soluções

Esta seção descreve os conceitos e técnicas envolvidas na adaptação de soluções no modelo RBC e apresenta como o PredCase os implementa.

4.4.3.1 Adaptação de soluções no RBC

A fase de adaptação usa as soluções aplicadas nos casos obtidos na fase de recuperação. A fase de adaptação procura por diferenças entre o caso novo e o caso passado e aplica fórmulas ou regras sugerindo uma solução adaptada [Lewis 1995]. Se as diferenças são irrelevantes, então a solução do caso passado é transferida sem modificações para o novo caso. Esse é um tipo trivial de reuso de casos. No entanto, para alguns sistemas, a cópia da solução é inadequada e uma adaptação é requerida [Aamodt & Plaza 1994].

Em geral, existem quatro decisões principais que precisam ser executadas no processo de adaptação:

- 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.

As adaptações podem ser feitas através de métodos de substituição e de transformação. O método de substituição é o processo de escolher e instalar uma reposição de alguma parte de uma solução passada. Um grupo completo de componentes pode ser substituído simultaneamente. O método de transformação converte uma solução passada em uma solução que funciona em uma nova situação ao se inserir, remover ou modificar alguns elementos [Schaaf 1996].

A fase de adaptação depende do domínio da aplicação. Em alguns domínios, tal como medicina, essa fase é considerada como um dos principais desafios para se aplicar a técnica RBC. Até o momento, não foi proposto na literatura nenhum método ou algoritmo geral para adaptação de soluções.

4.4.3.2 Adaptação de soluções no PredCase

No nosso modelo de predição os resultados das execuções de casos passados são recuperados da estrutura de dados denominada Resultado. Uma adaptação estrutural é realizada utilizando-se regras que são diretamente aplicadas nas soluções armazenadas nos casos identificados como similares. A predição da duração do *job* para um novo caso, *pred*, é uma média simples da duração de execução de *jobs* de casos passados similares, como descrito na equação (4)[6]. Uma base de casos, C, provida pela fase de recuperação de casos, é completamente pesquisada para calcular *pred*, onde ß= length(C).

A predição apresentada em (4) é adequada quando os casos passados e o novo caso são executados sob a mesma carga de trabalho. Por exemplo, a carga de trabalho é a mesma em um recurso dedicado para a tarefa de execução de um *job*. No entanto, em uma máquina não dedicada, o processo de predição precisa considerar a carga de trabalho. Neste caso, é necessário adaptar a predição *pred,* incluindo a relação entre as cargas de trabalho passadas e recentes. Essa predição adaptada, *predAd,* é calculada associando o parâmetro V (*predAd*=V**pred*). V adapta a intensidade da carga de trabalho de acordo com o horário de submissão do caso passado similar e do novo caso, computada pela fórmula V=f(WP,WF), onde WP representa a carga de trabalho média dos casos passados (equação 5) e WF representa a carga de trabalho média recente (equação 6) detectada pelo módulo de

[6] A notação "estrutura_de_dados.atributo" usada nas equações desta seção denota qual atributo é usado de cada estrutura de dados.

monitoração. Nesse módulo, um agente coleta cargas de recurso periodicamente e as registra regularmente em uma estrutura de dados denominada Monitor. WF usa α casos recuperados da estrutura de dados Monitor onde α representa o número de casos em Monitor que atendem à condição (7).

$$pred = \frac{\sum_{k=1}^{\beta} resultado_k.Duration}{\beta} \quad (4)^7$$

$$WP = \frac{\sum_{k=1}^{\beta} resultado_k.AvgWorkload}{\beta} \quad (5)$$

$$WF = \frac{\sum_{k=1}^{\alpha} monitor_k.AvgWorkload}{\alpha} \quad (6)$$

$$Problema.Time \leq Monitor.Time \leq Problema.Time + Solução.PredictedTime \quad (7)$$

4.4.4 Retenção de casos

A fase de retenção de casos adiciona novos casos à base de dados e analisa a permanência dos casos passados. Os atributos da base de casos são atualizados para as seguintes fases de execução do *job*: 1) antes da execução; 2) durante a execução; e 3) depois da execução.

Antes da execução do *job*, a base de casos distribuída é manipulada somente pelo servidor escolhido pelo *user_agent* para executar o *job*. O novo caso é incluído nas estruturas de dados Problema, Solução e Resultado. O estado do caso é definido como "aberto", já que o caso ainda está incompleto, faltando informações. Enquanto o caso apresentar o estado "aberto" o novo caso não é utilizado para predições por outros novos casos. Os dados calculados na fase de adaptação do caso, tais como a predição adaptada (*predAd)*, o método de predição e o grau de similaridade, preenchem os atributos *PredictedTime, PredictionMethod* e *SimilarityDegree* na estrutura de dados Solução.

Durante a fase de execução do *job*, a estrutura de dados Resultado é atualizada de acordo com os dados obtidos pelo módulo de monitoração do sistema multiagente, descrito no capítulo 3.

Depois da execução do *job*, o *server_agent* registra o tempo real da execução do *job* e a porcentagem de erro na predição. O estado do caso é então alterado para "fechado".

---

[7] Uma opção otimizada do cálculo da predição é associar pesos maiores na duração da execução passada para os casos que têm maior grau de similaridade com o novo caso.

A Tabela 4-3 apresenta a atualização dos atributos da base de casos apresentada na Figura 4-4 de acordo com as fases de execução do *job* em um recurso.

Tabela 4-3: Inclusão dos atributos de um novo caso na base para as fases de execução do *job*

Estrutura de dados	**Fases de execução do *job***		
	Antes	**Durante**	**Depois**
Problema			
CaseId	X		
Application	X		
Arguments	X		
DataSize	X		
Time	X		
Status	Aberto		Fechado
MinRAM	X		
MinCPU	X		
MinProc	X		
Solução			
CaseId	X		
PredictedTime	X		
PredictionMethod	X		
SimilarityDegree	X		
Resultado			
CaseId	X		
%PredictionError			X
Workload [ ]		X	
AvgWorkload			X
Duration			X

4.5 Validação do modelo de predição

Esta seção apresenta a validação do PredCase. Os objetivos dos experimentos são validar o modelo de predição e verificar: 1) a acurácia e a eficiência do modelo de predição; 2) o desempenho do algoritmo *refined nearest-neighbor*; 3) os valores limites que definem a similaridade entre casos.

Para atender ao objetivo 1, utilizamos como estratégia a comparação do tempo previsto com o tempo real de execução de *jobs*. Comparamos com dados da literatura, o erro de predição obtido e o tempo necessário para realizar predições. Tais métricas demonstram a eficiência do nosso modelo.

Para atender ao objetivo 2, utilizamos como métrica o tempo de recuperação de casos pelo algoritmo *refined nearest-neighbor* e pelo algoritmo *nearest-neighbor*. Comparamos o tempo de recuperação de casos desses dois algoritmos para diferentes tamanhos de base de casos.

Para atender ao objetivo 3, verificamos a relação entre valores de limite de similaridade e erro de predição. Essa comparação visa encontrar a porcentagem de características iguais entre casos capaz de trazer um erro de predição aceitável.

Na seção 4.5.1, apresentamos as aplicações utilizadas nos experimentos. Na seção 4.5.2, analisamos a acurácia da predição do PredCase. Na Seção 4.5.3, avaliamos a eficiência do algoritmo *refined nearest-neighbor*. Na seção 4.5.4, analisamos os valores aceitáveis para definir os limites que estabelecem que um caso novo é similar a um caso passado. Finalmente, na seção 4.5.5, comparamos os nossos resultados com os obtidos na literatura.

4.5.1 Aplicações e máquinas utilizadas nos experimentos

Para validar o PredCase utilizamos dois pacotes de aplicações denominados BLAST e HMMER [Blast 2005] [Eddy 2001]. Tais pacotes são frequentemente utilizados por pesquisadores do ambiente de grade computacional, 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 [Blast 2005]. O BLAST é formado por programas chamados blastx, blastn, blastp, tblastn e tblastx (incluídos no executável denominado blastall) e outros programas especializados tais como blastpgp, rpsblast e megablast. Esses programas usam diferentes algoritmos para pesquisar seqüências de DNA (*Desoxyribo Nucleic Acid*) ou proteínas em bases de dados procurando por similaridade entre eles. A Tabela 4-4 mostra uma lista reduzida de comandos BLAST submetidos em nosso experimento, onde as colunas Arg1 até Arg4 representam os argumentos das aplicações: Arg1 seleciona o tipo do programa BLAST; Arg2 indica diferentes bases de dados (nt, nr, ecoli, swissprot e htgs); Arg3 representa diferentes consultas (*queries*) à base. O tamanho da consulta é mostrada na última coluna; e Arg4 apresenta os parâmetros para o arquivo de saída. O Apêndice B2 apresenta mais informações sobre o BLAST.

Tabela 4-4: Exemplo de comandos BLAST utilizados nos experimentos

Application	Arg 1	Arg 2	Arg 3	Arg 4	Data (bytes)
blastall	-p blastn	-d nt	-i nt.5764416	-o out.5764416.blastn	2143
blastall	-p blastx	-d nr	-i nt.5706771	-o out.5706771.blastn	197867
blastall	-p blastn	-d ecoli	-i test.txt	-o out.test	852
blastall	-p blastx	-d swissprot	-i nt.5706771	-o out.5706771.mb	197867
megablast		-d htgs	-i nt.ests	-o out.ests_htgs.mb	815529
blast	-p blastp	-d nr	-i aa.p38398	-o out.p38398.blastp	2185

O HMMER é um seqüenciador de genomas que utiliza modelos de Markov (HMM - *Hidden Markov Model*) para identificar semelhanças em estruturas genéticas. O pacote HMMER é formado por nove programas chamados hmmalign, hmmbuild, hmmcalibrate, hmmconvert, hmmemit, hmmfetch, hmmindex, hmmpfam e hmmsearch. A Tabela 4-5 apresenta uma lista reduzida de comandos HMMER submetidos no nosso experimento, onde as colunas Arg1 a Arg4 representam os argumentos dos programas. Nos exemplos da Tabela 4-5, os argumentos %OPT.HMM% e Globin.hmm representam diferentes *queries* e swissprot, yeast.aa, Optiontests.fa e Optionstests.nslx, diferentes bases. O Apêndice B2 apresenta mais informações sobre o HMMER.

Tabela 4-5: Exemplos de comandos HMMER utilizados nos experimentos

Application name	**Arg1**	**Arg2**	**Arg3**	**Arg4**	**Data (bytes)**
hmmalign	-m	%OPT.HMM%	Optiontests.fa		8318
hmmbuild	-F	-n	%OPT.HMM%	Optionstests.nslx	8318
hmmsearch	globin.hmm	swissprot			54185
hmmsearch	globin.hmm	yeast.aa			54185

Essa amostra diferenciada de aplicações foi utilizada para demonstrar a usabilidade do PredCase para a aplicação alvo que definimos para o nosso ambiente, ou seja, o PredCase pode ser utilizado por *jobs* construídos para qualquer aplicação seguida de um conjunto de argumentos para executar em modo *batch*. A partir de tutoriais e testes incluídos no próprio pacote das aplicações BLAST e HMMER [Kägström *et al.* 1995] [Sosa *et al.* 2002], além de estudo e pesquisa de como utilizar tais aplicações [Cruz 2003] [Eddy 2001], construímos um conjunto de comandos para serem submetidos no nosso experimento.

Um conjunto diferenciado de máquinas foi utilizado nos experimentos desta seção. Tínhamos limitação no número de máquinas disponíveis no nosso ambiente para efetivação dos testes. As máquinas estão relacionadas na Tabela 4-6 e se encontram nos laboratórios do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (UFMG).

Tabela 4-6: Configuração das máquinas dos experimentos para validar o PredCase

Máquina	RAM (MB)	CPU (MHz)	Modelo
Hulk	1024	1700	Intel Pentium 4 CPU mod 1
Caridade	512	1800	AMD Athlon 2500+ mod 10
Alegria	512	1666	AMD Athlon 2000+ mod 8
Itapeva	512	1800	AMD Athlon 2500+ mod 10
Montblanc	512	1800	AMD Athlon 2500+ mod 10
Sajama	1024	2000	AMD Athlon(tm) 64 Processor 3200+ mod 47
Palermo	1024	3000	Intel® Pentium® 4 CPU 3.00GHz mod 4
Ibituruna	512	1666	AMD Athlon 2000+ mod 8

4.5.2 Acurácia da predição

O experimento para verificar a acurácia da predição realizada pelo PredCase foi executado em duas etapas. O objetivo da primeira etapa, além de verificar a acurácia, é também demonstrar que o modelo de predição é distribuído e que máquinas com processadores de velocidade semelhantes possuem predições diferentes. A estratégia usada para essas verificações consiste na escolha do número de equipamentos e das suas características. Para esse experimento foi utilizado apenas um tipo de aplicação. A segunda etapa, além de verificar a acurácia, objetiva mostrar que o modelo de predição pode ser usado para aplicações diferentes. Para essa etapa, utilizamos apenas uma máquina e aumentamos o número de casos submetidos.

Os experimentos envolveram um total de seis computadores (incluídos na Tabela 4-6) com diferentes capacidades e que estavam dedicados aos experimentos. A Figura 4-6 apresenta um ambiente típico do MASK, onde cada *server_agent* representa um recurso e cada *user_agent* representa um usuário. O PredCase e outro módulo do MASK referente à verificação de restrições de acesso rodam em cada *server_agent*. O PredCase, destacado na

Figura 4-6, é composto dos processos de Recuperação, Adaptação e Retenção e das estruturas de dados Problema, Solução, Resultado e Monitor.

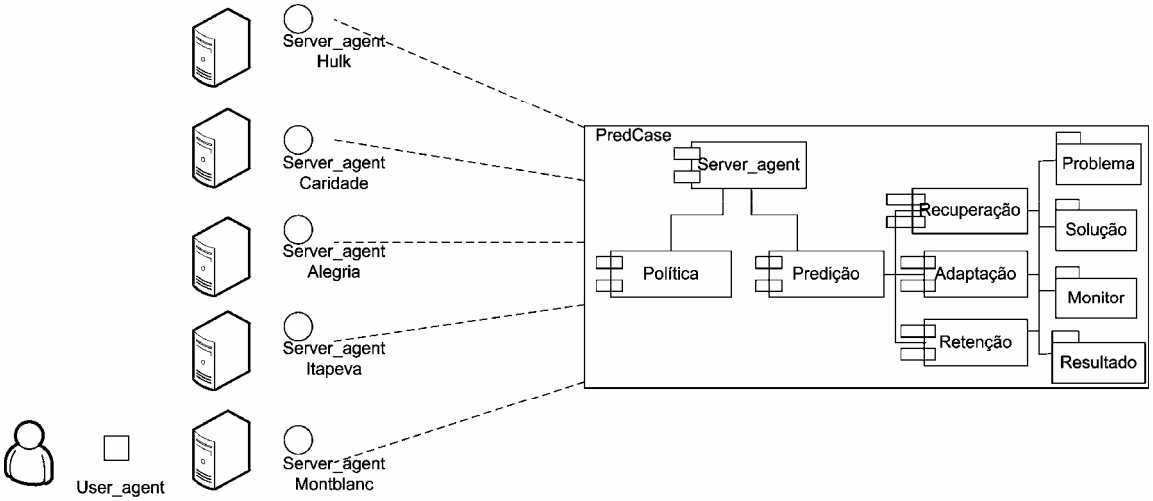

Figura 4-6: Estrutura RBC distribuída e sistema multiagente

Nos experimentos, o *user_agent* envia requisições de execução de *jobs* para todos os *server_agents*. Cada *server_agent* calcula o tempo de predição de execução para o *job* recebido usando o PredCase e retorna o tempo estimado para o *user_agent*. O *user_agent* seleciona a máquina que possui a predição do menor tempo de execução para cada caso[8]. Se para um determinado caso o *user_agent* não receber proposta de nenhuma das máquinas, pelo fato de não existir nas bases de casos consultadas um caso similar em 90% (valor esse discutido na seção 4.5.4) com o novo caso, então, o *job* não é submetido.

Os cálculos de erro de predição são feitos utilizando a equação (8) que relaciona o tempo previsto e o tempo real de execução do *job*.

$$\text{ErroPredição} = \frac{\text{TempoPrevisto} - \text{TempoMedido}}{\text{TempoMedido}} \text{ x } 100\% \qquad (8)$$

A acurácia da predição depende da exclusividade do recurso no momento da execução do *job*. Mais da metade das máquinas utilizadas nos experimentos possui 512 MB de RAM, enquanto algumas bases de dados das aplicações para grade computacional utilizadas têm tamanho superior a 1GB. Dessa forma, ao se executar uma única aplicação na máquina,

[8] No capítulo 6, quando MASK é completamente apresentado, o processo de seleção considera outros fatores.

essa utilizava a área de *swap* freqüentemente e apresentava uma duração regular para a execução do *job*. No entanto, ao executarmos simultaneamente dois ou mais *jobs* na mesma máquina, sendo que cada *job* utilizava uma base de dados acima de 1GB, a máquina também utilizava a área de *swap*, mas apresentava resultados de duração de execução não regulares (ou seja, a cada execução apresentava durações completamente diferentes, dependendo do tamanho das bases de dados utilizadas simultaneamente). Por tais razões optamos por executar cada *job* de forma exclusiva nas máquinas do experimento, de forma a não comprometer a predição de execução.

Tabela 4-7: Casos previamente executados nas máquinas do experimento (etapa 1)

Caso	Comando	Tempo de execução (ms)				
		Itapeva	Montblanc	Hulk	Alegria	Caridade
1	Blastall –p blastn –d nt –i nt.5764416	461296	584494	-	-	415342
2	Blastall –p blastn –d “nt refseq_genomic.00” –i test2.txt	-	-	-	-	-
3	Blastall –p blastn –d nr –i nt.5706771	-	-	-	-	-
4	blastall -p blastn -d nt –I nt.2655203	-	2433429	6386397	-	2144729
5	Blastall –p blastp –d pdbaa –I examplenrc	-	-	-	-	-
6	Megablast –d nt –i nt.5706771	-	1098131	1941443	-	894298
7	blastall -p blastn -d nr -i QUERY	-	-	-	-	-
8	Megablast –d htgs –i nt.ests	1474043	-	-	-	751257
9	Blastall –p blastp –d nr –i aa.129295	61510	185141	355599	80530	55688
10	Blastall –p blastn –d refseq_genomic –i test.txt –o test.out	-	-	-	-	-
11		-	-	-	-	-
12	Blastall –p blastp –d nr –i aa.1177466	141983	260551	197336	184511	143564
13	Blastall –p blastp –d nt –i aa.p38398	191295	816415	276221	243704	196323
14	Blastpgp –i seq1 –B align1 –j 2 –d nr	-	-	-	-	-
15	blastall -i ff.chd -d yeast -p psitblastn -R ff.chd.ckp	-	-	-	-	-
16	Blastall –p blastx –d nr –i nt.5764416 –e 0.1	131163	165832	245946	198399	128378
17	Blastall –p blastx –d nr –i nt.3283410	1988786	2647980	4151941	3570879	2250200
18	Blastall –p blastx –d nr –i nt.4883672	20223346	23130532	25508851	24097680	21424454
19	Blastall –p blastx –d nr –i nt.5706771	-	-	-	47183036	-
20	Blastall –p tblastn –d nt –i aa.231729	-	1703085	1848934	-	586824
21	Blastall –p tblastn –d nt –i aa.231729	-	1433872	2205072	-	805755
22	Blastall –p tblastn –d htgs –i aa.1177466	-	1532435	-	-	1083540
23	Blastall –p tblastn –d htgs –i p38398	-	1757578	-	-	1259730

Na primeira etapa dos experimentos, utilizamos as máquinas Hulk, Caridade, Itapeva, Alegria e Montblanc. Isso porque as máquinas Itapeva, Caridade e Montblanc possuem as mesmas características computacionais e gostaríamos de verificar o comportamento de MASK nessa situação, onde não é facil deduzir qual é a máquina mais rápida do ambiente,

já que elas possuem a mesma velocidade de processamento. A máquina Montblanc estava configurada como a plataforma dos agentes e a partir dela todos os agentes foram inicializados remotamente e todos os *jobs* foram submetidos pela interface do MASK para um usuário local.

Tabela 4-8: Predição de tempo de execução para cada máquina (etapa 1)

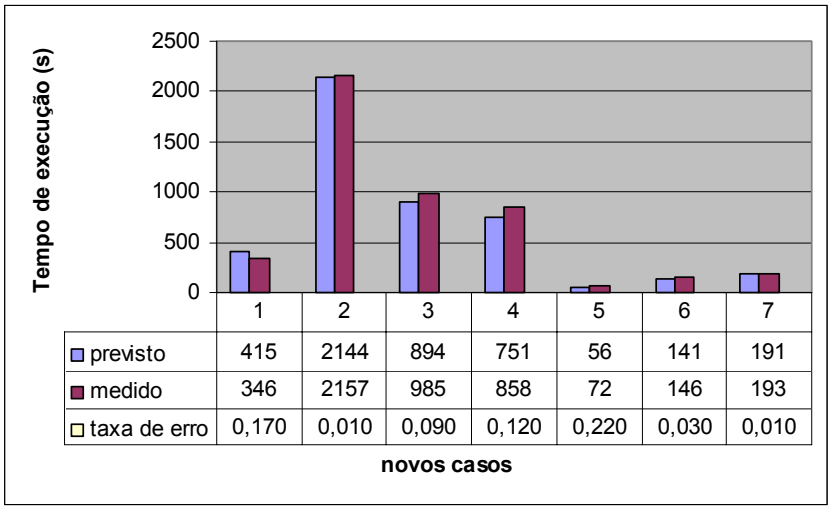

Figura 4-7: Tempo previsto e tempo medido na execução de *jobs* (etapa 1)

Casos	Predição do tempo de execução (s)					Máquina selecionada
	Montblanc	Alegria	Hulk	Caridade	Itapeva	
1	584	-	-	415	461	Caridade
4	2433	-	6384	2144	-	Caridade
6	1098	-	1941	894	-	Caridade
8	1474	-	-	751	-	Caridade
9	185	81	356	56	61	Caridade
12	260	185	197	143	141	Itapeva
13	816	244	276	196	191	Itapeva

Para essa primeira etapa de experimentos, cada máquina continha um conjunto de casos para a aplicação BLAST previamente executados conforme mostra a Tabela 4-7, que é preenchida com "-" para os casos que nunca foram executados na correspondente máquina. Foram re-submetidos para o PredCase os 13 primeiros casos da Tabela 4-7. Para cada caso submetido, o *user_agent* escolhe a máquina que prevê a execução do *job* no menor período de tempo. A Tabela 4-7 apresenta a seleção para os casos submetidos. Os casos 2, 3, 5, 7, e 10 nunca foram executados em nenhuma das máquinas do experimento e, portanto não foram incluídos na Tabela 4-8. A Figura 4-7 apresenta o tempo de execução comparado com o tempo previsto para cada máquina selecionada pelo MASK. Nestes experimentos,

verificou-se uma média de erro de predição de aproximadamente 9%, calculada pelo uso da equação (8).

Na segunda etapa dos experimentos, verificamos a acurácia da predição para o total de *jobs* em uma única máquina, no caso a máquina Sajama, que já possuía em sua base de casos um conjunto de *jobs* previamente executados. Para essa segunda etapa, além da aplicação BLAST, utilizamos a aplicação HMMER e alguns programas simples escritos em *shell script* e na linguagem C, totalizando 39 casos. A Tabela 4-9 apresenta os casos submetidos para a máquina Sajama, a correspondente porcentagem do erro de predição e a duração da execução do *job*.

Nas duas etapas de testes, o erro de predição foi de aproximadamente 9%. Isso demonstra que para máquinas diferentes, aplicações diferentes e execuções de *jobs* com durações diferentes, o modelo apresenta uma regularidade no cálculo do erro de predição.

Tabela 4-9: Validação do PredCase na máquina Sajama (etapa 2)

Caso	Comando	Erro de predição	Duração (ms)
1	hmmsearch --compat %OPT.HMM% swissprot	0,15	46994
2	hmmbuild -F %OPT.HMM% Optiontests.slx	0	3
3	hmmalign %OPT.HMM% Optiontests.fa	0	4
4	hmmalign --outformat selex %OPT.HMM% Optiontests.fa	0,25	4
5	hmmalign --outformat clustal %OPT.HMM% Optiontests.fa	0,25	4
6	hmmsearch rrm.hmm nr	-0,26	4472390
7	hmmsearch globin2.hmm swissprot	-0,01	403754
8	hmmbuild -F -n foo %OPT.HMM% Optiontests.nslx	0,05	20
9	hmmbuild -F -g %OPT.HMM% Optiontests.nslx	0,05	20
10	hmmbuild -F --wvoronoi %OPT.HMM% Optiontests.nslx	-0,01	22
11	hmmcalibrate %OPT.HMM%	0,07	1284
12	hmmsearch fn3.hmm input/nr	-0,13	3348308
13	hmmemit -o %OPT.OUT% %OPT.HMM%	0	3
14	hmmsearch weeviterbi_test.hmm swissprot	0,06	173132
15	hmmsearch trace_test.hmm swissprot	-0,01	37351
16	hmmsearch fn3.hmm swissprot	-0,09	209511
17	hmmcalibrate pkin.hmm	-0,01	24169
18	hmmcalibrate globin.hmm	-0,10	14149
19	hmmsearch globin.hmm swissprot	-0,12	352463
20	hmmsearch globin.hmm yeast.aa	-0,07	17883
21	hmmsearch rrm.hmm yeast.nt	-0,08	24153
22	blastall -p blastx -d nr -i nt.5706771 -e 0.1	-0,25	30367698
23	blastall -p blastp -d nr -i aa.129295	-0,13	92172
24	blastall -p blastp -d nr -i aa.231729	-0,16	95582
25	blastall -p blastp -d nr -i aa.1177466	-0,26	103438
26	blastall -p blastp -d nr -i aa.p38398	-0,20	555911
27	blastall -p blastx -d nr -i nt.5764416 -e 0.1	-0,23	98270

28	blastall -p blastx -d nr -i nt.3283410 -e 0.1	-0,07	1249207
29	blastall -p blastx -d nr -i nt.4883672 -e 0.1	-0,10	11245595
30	blastall -p blastn -d nt -i nt.3283410 -e 0.1	-0,04	907467
31	megablast -d nt -i nt.5706771	0,00	627330
32	blastall -p tblastn -d nt -i aa.129295	-0,04	803140
33	blastall -p tblastn -d nt -i aa.231729	-0,19	1236514
34	hmmsearch %OPT.HMM% Optiontests.fa	-0,16	12
35	hmmsearch -A 0 %OPT.HMM% Optiontests.fa	0,14	8
36	hmmsearch globin.hmm yeast.nt	-0,01	197103
37	hmmsearch pkin.hmm swissprot	-0,007	819689
38	myprog teste 100	0	100000
39	simple 30 10	0	30000

4.5.3 Desempenho do algoritmo *refined nearest-neighbor*

Nesta seção, o desempenho do algoritmo *refined nearest-neighbor* (RNN) é comparado com o desempenho do algoritmo *nearest-neighbor* (NN) pela observação do tempo de recuperação de casos em ambos. Um determinado *job*, J1 (blastall –p blastx –d nr –i nt.5764416 -e 0.1) foi submetido 10 vezes para ambos os algoritmos disponíveis na máquina Palermo. Para cada submissão, diferentes tamanhos de bases de casos foram usados: 36, 72, 144, 288, 576 e 1152 respectivamente. O aumento da base foi conseguido a partir da replicação dos casos de uma base inicial de 36 casos. A Figura 4-8 apresenta os tempos de recuperação de casos dos algoritmos RNN e NN para 6 diferentes tamanhos de bases de casos.

De acordo com o gráfico da Figura 4-8, quando a base de casos é pequena, tanto o NN quanto o RNN apresentam desempenhos similares. No entanto, o tempo que o algoritmo NN recupera casos aumenta substancialmente com o aumento do tamanho da base de casos. A tendência geral para ambos os algoritmos é apresentar um aumento no tempo de recuperação de casos com o aumento de número de casos na base e com o aumento do número de casos similares. Tais degradações de desempenho podem ser contornadas com diferentes melhorias: o aumento da base de casos no RNN pode ser controlado pela revisão da permanência de casos na base já previsto de ser realizado pela fase de Retenção do modelo RBC; e o aumento dos casos similares pode ser controlado pela definição de recuperação de k-casos em ambos os algoritmos, o que poderia resultar em um novo algoritmo *k-refined nearest-neighbor*, por exemplo.

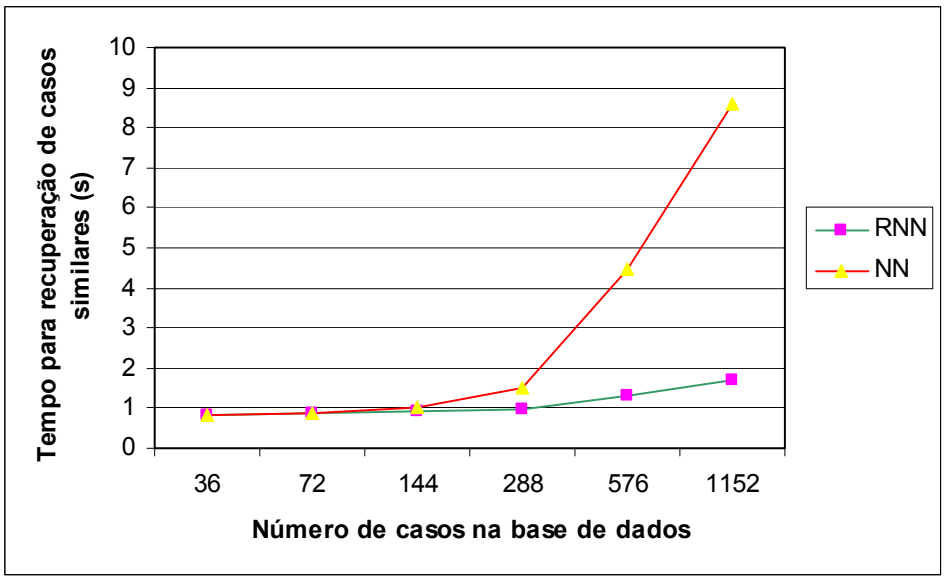

Figura 4-8: Desempenho dos algoritmos RNN e NN para diferentes tamanhos de base de casos

4.5.4 Limites que definem similaridade entre casos

O objetivo desta seção é analisar quais os valores que definem que um caso pode ser considerado similar a outro. Para esse fim, utilizamos valores crescentes para o limite que representa o grau de similaridade e comparamos, para cada limite, o percentual de erro na predição obtida. Um conjunto de 49 comandos foi executado três vezes na máquina Palermo requerendo o cálculo de predição com os valores de 85%, 90%, 95% e 100% de similaridade entre o caso novo e os casos passados. Esses valores foram alterados no código do algoritmo *refined nearest-neighbor* para analisar qual o percentual de erro de predição obtido quando se variava esse parâmetro.

A Figura 4-9 apresenta a média de erro de predição verificada para o conjunto dos *jobs* submetidos na máquina Palermo. Verificou-se que, à medida que se aumenta o limite de similaridade entre o caso passado e o novo caso, obtem-se menor erro de predição. A média de erro de predição que se baseia em casos idênticos ao novo caso, ou seja, grau de similaridade igual a 100%, é de 5%. No entanto, quando o grau de similaridade é 100%, alguns *jobs* não conseguiram obter predição, pelo fato de não haver um caso idêntico na base consultada. O erro de predição obtido com 95% de grau de similaridade foi muito próximo ao erro obtido com de 90% de similaridade. No entanto, quando o grau de similaridade foi definido como 95%, alguns *jobs* também não conseguiram obter predição.

Dessa forma, optamos por utilizar um grau de 90% de similaridade entre os casos passados e o novo caso, permitindo calcular a predição com casos similares, mas nem sempre idênticos e obtendo uma média de erro de predição em torno de 9%. Conforme comentários anteriores, esse grau de similaridade é adaptável dentro do MASK e, em uma situação com muitos casos representativos para a comunidade de usuários na base de casos, pode-se definir esse parâmetro para um valor maior e assim obter melhor acurácia.

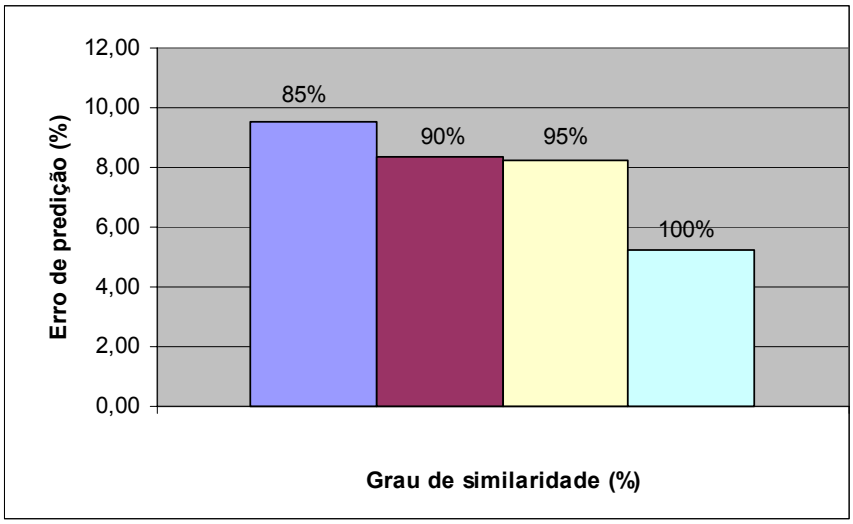

Figura 4-9: Percentual de erro médio de predição X grau de similaridade entre casos

4.5.5 Comparação de resultados com soluções correntes

Esta seção compara nossa solução com diferentes abordagens para a predição de execução de *jobs* em ambientes de grades computacionais. Duas importantes métricas de desempenho são consideradas: acurácia da predição e tempo de cálculo da predição.

Em [Elmroth & Tordsson 2005], os autores calculam a predição entre 2s e 16s. Infelizmente, eles não apresentaram resultados sobre a acurácia da predição. Em [Sun & Wu 2005], o erro de predição é de 25% se os *jobs* são executados em uma única máquina em menos de 1 hora e o erro de predição é de 10% se os *jobs* têm duração acima de 8 horas. O tempo de cálculo da predição é de 0.662s. Em [Cao *et al.* 2002], o erro de predição é de 12% e o tempo para efetuar o cálculo da predição é de 2s. Em [Liu & Foster 2002], quando a máquina está dedicada à execução do *job*, o erro médio de predição é de 6.2%. Quando a máquina não está dedicada para a execução do *job*, o erro médio de predição é de 59%. Esses valores são válidos somente para a aplicação Cactus. Nos nossos experimentos,

identificamos um erro médio de predição de 9% e o tempo de cálculo da predição foi abaixo de 1s para base de casos com 300 casos.

A acurácia da predição é fortemente dependente da máquina estar dedicada para a tarefa de execução do *job*. De acordo com as abordagens comparadas, o tempo de cálculo da predição é curto, obtido geralmente em poucos segundos. Embora cada uma dessas abordagens utilize como métricas de desempenho a acurácia e o tempo do cálculo da predição, existem limitações de uso dessas abordagens para o cenário que aqui apresentamos, conforme os problemas de funcionalidades destacados na revisão bibliográfica do capítulo 2.

4.6 Conclusões

Este capítulo apresentou um novo modelo para predição de duração da execução de *jobs* no ambiente de grade computacional. Foi utilizado o paradigma do raciocínio baseado em casos, RBC, para o cálculo da duração de execução de um novo *job*. Os cálculos são baseados em submissões de casos passados similares.

O modelo de predição aqui investigado, PredCase, faz parte do sistema multiagente, MASK, onde agentes negociam entre si para alocar recursos no ambiente de grade computacional. Com erros de predição relativamente baixos, o PredCase pode fornecer ao usuário uma boa indicação de quando o *job* irá terminar e permitir o escalonamento do *job* para tempo futuro. Como parte do modelo de predição, foi desenvolvido um algoritmo denominado *refined nearest-neighbor,* que relaciona técnicas de seqüência de relevância e distâncias geométricas; adaptamos as soluções de predição usando substituição de soluções passadas.

PredCase foi implementado com sucesso e testado para um pequeno porém significativo conjunto de casos, considerando a nossa pequena infra-estrutura de grades computacionais e a falta de usuários especialistas de aplicações para grades computacionais no nosso ambiente. Pelo fato de não haverem usuários especialistas, cada inclusão de novas aplicações e correspondentes *jobs* nos demandou um esforço de aprendizado das mesmas para instalação, configuração e elaboração de novos comandos.

Quanto à acurácia da predição, obtivemos uma média de erro de 9% quando consideramos que um caso novo e um caso passado são similares em 90% das suas características. Esse erro médio pode ser reduzido se a predição for baseada em casos mais similares. Os resultados de predição obtidos pela nossa abordagem foram compatíveis com os da literatura e relativamente melhores para as condições do nosso escopo, ou seja, para cálculo de predição de execução para qualquer aplicação *batch* de qualquer duração, demandando recursos dedicados, onde a predição deve ser obtida em tempo real na submissão de um *job* para a grade.

Quanto ao desempenho na recuperação de casos passados, verificamos que o novo algoritmo *refined nearest-neighbor* tem desempenho superior ao algoritmo original *nearest-neighbor* quando ocorre aumento do número de casos na base. Obtivemos um tempo de recuperação abaixo de 1s para bases contendo 300 casos e abaixo de 2 s para bases contendo 1200 casos. Para as mesmas condições, o algoritmo *nearest-neighbor* obteve um tempo de recuperação 50% maior para bases contendo 300 casos e 450% maior para bases contendo 1200 casos.

Os resultados demonstraram que o paradigma RBC associado à tecnologia multiagente é uma solução dinâmica para a predição de desempenho distribuída em ambientes heterogêneos. A nossa abordagem de predição de desempenho de execução de *jobs* apresentou boa acurácia tornando-a um importante componente no processo de seleção de recursos.

Capítulo 5

Restrições de acesso baseadas em políticas de granularidade fina

5.1 Introdução

A evolução das futuras gerações de infra-estrutura de grades computacionais requer avanços fundamentais nos modelos e mecanismos de segurança para sistemas distribuídos em larga escala [Lorch *et al*. 2003a]. Tais avanços precisam superar o uso de políticas no nível macro, que aplicam regras gerais para um grande número de usuários e recursos, e buscar o estabelecimento de políticas nos níveis elementares, que aplicam restrições para usuários e recursos específicos com alto nível de detalhe. Uma granularidade menor de políticas de uso dos recursos demanda novas soluções para o processo de seleção de recursos para executar *jobs*.

Uma política é um método de ação baseado em um conjunto de condições que guiam e determinam decisões presentes e futuras. No contexo de grades computacionais, políticas são implementadas e executadas dentro de um contexto particular, tal como segurança, gerenciamento de carga de trabalho e qualidade de serviço, e fornecem um conjunto de regras para administrar, gerenciar e controlar o acesso a recursos [Foster & Kesselman 2004].

O controle de acesso para um usuário de grade computacional pode ser feito nos âmbitos de políticas globais e locais. As políticas globais estabelecem regras genéricas para um conjunto de usuários que acessam recursos de determinada organização. Tais abstrações são implementadas, por exemplo, nas ferramentas *Virtual Organization Membership Service* (VOMS) [Alfieri *et al.* 2003], *Community Authorization Service* (CAS) [Cannon *et al.* 2003] e *Privilege and Role Management Infrastructure Standards Validation* (PERMIS) [Permis 2005].

As políticas locais estabelecem regras específicas para recursos e usuários. As implementações de controle de acesso associadas a cada recurso podem levar a um refinamento das políticas. Esse refinamento pode definir, por exemplo, os horários que determinado usuário ou grupo de usuários pode usar um recurso para a execução de *jobs*. Um direito a acesso é dito "refinado" (ou de granularidade fina) se for aplicável a um dado usuário e a um dado objeto. Direitos de acesso "não refinados" (ou de granularidade grossa) são aqueles aplicados a todos os membros em um grupo de usuários ou a todos os objetos de um conjunto [Lorch *et al.* 2003a].

O controle de acesso baseado em políticas de granularidade fina auxilia o uso de grades computacionais onde cada proprietário do recurso compartilhado tem a possibilidade de controlar o acesso ao seu recurso. De forma similar, os usuários também ganham em flexibilidade para escolher o recurso para a execução de seus *jobs* [Bertino *et al.* 2004].

Atualmente, nos *middlewares* para grades computacionais que não utilizam *brokers* para seleção, como é o caso do Globus [Globus 2005], os usuários escolhem um recurso e submetem *jobs* para o mesmo sem saber se possuem autorização, sem conhecer as condições de provimento do serviço e sem saber a carga de trabalho no recurso destino. Essa falta de informação na submissão pode incorrer em re-submissões até que seja encontrado um recurso que autorize a requisição do usuário. Além disso, sem uma prévia negociação de uso do recurso antecedendo a submissão de um *job* não há como estabelecer um acordo especificando as condições de prestação de um serviço, como o preço de uso do recurso e as penalidades para o descumprimento da prestação do serviço no prazo previsto.

Além desses problemas de submissão direta ao recurso, existem ainda desafios a serem superados pelos atuais *brokers,* que realizam a verificação das restrições de forma centralizada, utilizam uma linguagem própria para descrever as restrições e impõem um *overhead* de processamento da verificação das restrições.

Esse é o caso do Condor [Raman *et al.* 1998], que trata todas as variáveis do problema de seleção de recursos como restrições e utiliza um procedimento comum de análise, o algoritmo *matchmaking* descrito no capítulo 2. Embora as restrições estejam distribuídas no ambiente Condor, o *matchmaking* é feito de forma centralizada e todas as restrições são verificadas. O Condor verifica todas as restrições e não apresenta, em uma situação de negação do acesso, qual a restrição que negou o acesso dentre *n* verificadas. Essa falta de informação faz com que o usuário tenha que fazer uma longa análise nos arquivos de *log* gerados pelo Condor para encontrar o motivo da negação de seu acesso. Outro problema decorrente da igualdade de processamento entre as restrições aparece na situação onde o algoritmo *matchmaking* continua a verificar os parâmetros para um recurso que já atingiu a carga de trabalho máxima e que, portanto não poderá executar o *job* no horário solicitado. O Condor declara as restrições em um formato próprio chamado ClassAd. Portanto, para organizações, que descrevem políticas de acesso em outro formato, utilizarem o Condor, é preciso redefinir as restrições de acesso no formato de ClassAd. Tais redefinições de restrições de acesso causam redundância e dificultam o gerenciamento coerente das informações.

Neste capítulo, propomos um modelo, denominado POLAR (*Policy-based access restriction*), para verificação de restrições de acesso baseadas em políticas de granularidade fina. O conjunto de restrições de acesso tratadas neste capítulo inclui a autorização do usuário ao recurso, a delimitação de valores para os parâmetros de SLA e a carga de trabalho máxima aceita pelos recursos. Procedimentos diferentes verificam essas restrições, cujo processamento é distribuído e acoplado ao modelo de negociação ponto-a-ponto descrito no capítulo 3.

Na nossa abordagem, a verificação das políticas tem que anteceder a submissão, a fim de evitar o cancelamento de *jobs* devido à problemas de restrições de acesso. A negociação do provimento do serviço considera várias restrições, tais como o preço para utilizar o recurso, as garantias[9] envolvidas no descumprimento do serviço e a carga de trabalho no recurso. As regras que definem essas restrições estão estruturadas no mesmo padrão de definição de políticas. O POLAR distingue as restrições evitando a verificação completa dos parâmetros de negociação dos recursos.

[9] A garantia ao serviço é representada aqui como a penalidade envolvida em caso de descumprimento do serviço acordado.

O modelo apresentado é parte integrante da solução MASK, que objetiva a escolha do melhor recurso da rede para submissão de um *job* e cuja estrutura geral foi apresentada no capítulo 3.

As demais seções deste capítulo estão assim organizadas: a seção 5.2 apresenta a declaração das políticas e os requisitos do modelo; a seção 5.3 apresenta o modelo para verificação de restrições de acesso; a seção 5.4 apresenta a validação do POLAR; e a seção 5.5 conclui o capítulo.

5.2 Declaração de políticas e requisitos do modelo

Para estabelecer restrições de acesso em um ambiente de grade compartilhado é fundamental ter uma linguagem de declaração de políticas capaz de representar um conjunto de regras, tais como as regras R1 a R4 dadas a seguir:

- R1 – O usuário *u* pode executar um *job* no recurso *y* entre os horários *a* e *b*;
- R2 – A carga máxima de trabalho no recurso *y* é *v*;
- R3 – O preço mínimo aceitável para executar qualquer ação no recurso *y* é *pr*;
- R4 – Uma garantia (desconto) *g* é aplicada sobre o preço *pr* quando a métrica *m* não é cumprida.

No caso dessas regras dadas como exemplo, o conjunto de variáveis envolvidas na verificação de restrição de acesso aqui apresentada é dado pela tupla (*u*,*y*,*a,b,v*,**N**), onde **N** pode agrupar diversos parâmetros relacionados à negociação de um SLA. Neste trabalho instanciamos 3 desses parâmetros, **N** = {*pr,g,m*}, onde *pr* representa o preço, *g* representa a garantia aplicada caso haja descumprimento do serviço negociado e *m* representa a métrica de referência para analisar a qualidade do serviço.

No modelo POLAR, cada recurso da grade computacional possui sua própria definição de políticas, descritas em formato *eXtensible Access Control Markup Language* (XACML) [Godik & Moses 2003]. Optamos pelo uso do XACML porque ele é um padrão específico para a definição de restrições de acesso, evitando desta forma esquemas XML proprietários. Além disso, esse padrão possui flexibilidade suficiente para definir regras, tais como as regras R1 a R4 exemplificadas nesta seção.

A título de ilustração, a Figura 5-1 apresenta a descrição em XACML das regras para uma instância de R1, onde é estabelecido que o usuário *maria@dcc.ufmg.br* tem acesso a um recurso no período entre 8h e 17h. As restrições para parâmetros de SLA estão descritas

na Figura 5-2, onde o acesso será negado caso o preço oferecido seja menor que 100, atendendo à regra de formato R3. Instâncias das regras R2 e R4 podem ser declaradas de forma similar à R3.

```
<Rule RuleId=
  ”EveryoneDuringBusinessHours”Effect=”Permit”>
 <Condition FunctionId=
  ”http//:research.sun.com/projects/xacml/names/function#time-in-range”>
  <Apply FunctionID=
    ”urn:oasis:names:tc:xacml:1.0:function:time-one-and-only”>
   <ResourceAttributeDesignator DataType=
     ”http://www.w3.org/2001/XMLSchema#time”
      AttributeId=” urn:oasis:names:tc:xacml:1.0:resource:StartJob”/>
  </Apply>
  <AttributeValue DataType=
    “http: //www.w3.org/2001/XMLSchema#time”>08:00:00</AttributeValue>
  <AttributeValue DataType=
    “http: //www.w3.org/2001/XMLSchema#time”>17:00:00</AttributeValue>
 </Condition>
</Rule>

<Rule RuleId=
  ”EmployeesAlways”Effect=”Permit”>
  <Target>
   <Subject>
    <SubjectMatch MathId=“urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match”>
     <AttributeValue DataType=
       “http:// //www.w3.org/2001/XMLSchema#string”>maria@dcc.ufmg.br</AttributeValue>
        <AttributeId=“urn:oasis:names:tc:xacml:1.0:function:subject:subject-id”/>
    </SubjectMatch>
   </Subject>
  <Resources></Resources>
 <Actions></Actions>
</Target>
</Rule>
```

Figura 5-1: Exemplo de declaração de políticas em XACML para delimitar autorização de usuário

```
<Rule RuleId="PriceInTheRange" Effect="Deny">
 <Target>
  <Subject></Subject>
   <Resources>
    <Resource>
     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
      <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#integer"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:Price"/>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">60</AttributeValue>
     </ResourceMatch>
    </Resource>
  </Resources>
 </Target>
</Rule>
```

Figura 5-2: Exemplo de declaração de políticas em XACML para delimitar parâmetros de SLA

Os requisitos principais para a definição do POLAR são: 1) cada recurso pode ter um nível de granularidade de acesso que pode ser diferente da política global definida para uma organização virtual; 2) um acordo de nível de serviço deve obedecer limites para o oferecimento do serviço baseando-se nas restrições definidas pelo recurso; 3) uma carga de trabalho máxima deve ser definida para cada recurso; 4) a descrição de políticas e o mecanismo de verificação devem ser distribuídos para atender às características do ambiente de grade computacional de forma autônoma; 5) a verificação das políticas de acesso deve anteceder a submissão do *job*; 6) o modelo de definição e verificação de políticas deve seguir padrões.

5.3 Modelo para verificação de restrições de acesso

A Figura 5-3 apresenta a arquitetura e o funcionamento do POLAR integrado ao processo de negociação do *middleware* MASK. A integração é feita através de negociação entre um *user_agent* e *n server_agents*. A partir da configuração do *user_agent* feita pelo usuário, o *user_agent* envia os seguintes parâmetros para os *server_agents*: <usuário, domínio, horário de início do *job*, preço, garantia, descrição do *job*[10]>. Cada *server_agent* gera dinamicamente requisições em formato XACML contendo esses parâmetros para serem validados no módulo *Policy Decision Point* (PDP). Os *server_agents* analisam a proposta enviada através dos seguintes passos, onde os fluxos se referem à Figura 5-3: 1) verificam a autorização do usuário para utilizar o recurso (fluxos 1, 2 e 3); 2) verificam a carga máxima aceitável no recurso (fluxos 5 e 6); 3) verificam as restrições do recurso referente a parâmetros de SLA, instanciados aqui pelo preço de uso do recurso e garantia no descumprimento da entrega do serviço (fluxos 7n, 8n e 9n); 4) elaboram contraproposta (fluxo 11); e retornam aceitação (fluxo 10), rejeição (fluxos 4 e 7) ou contraproposta (fluxo 12) conforme resultados obtidos nos fluxos anteriores.

[10] A descrição do *job*, **J**, não é utilizada por POLAR e sim pelo módulo de predição (Capítulo 4). O fluxo completo das informações entre os módulos é apresentado no Capítulo 6.

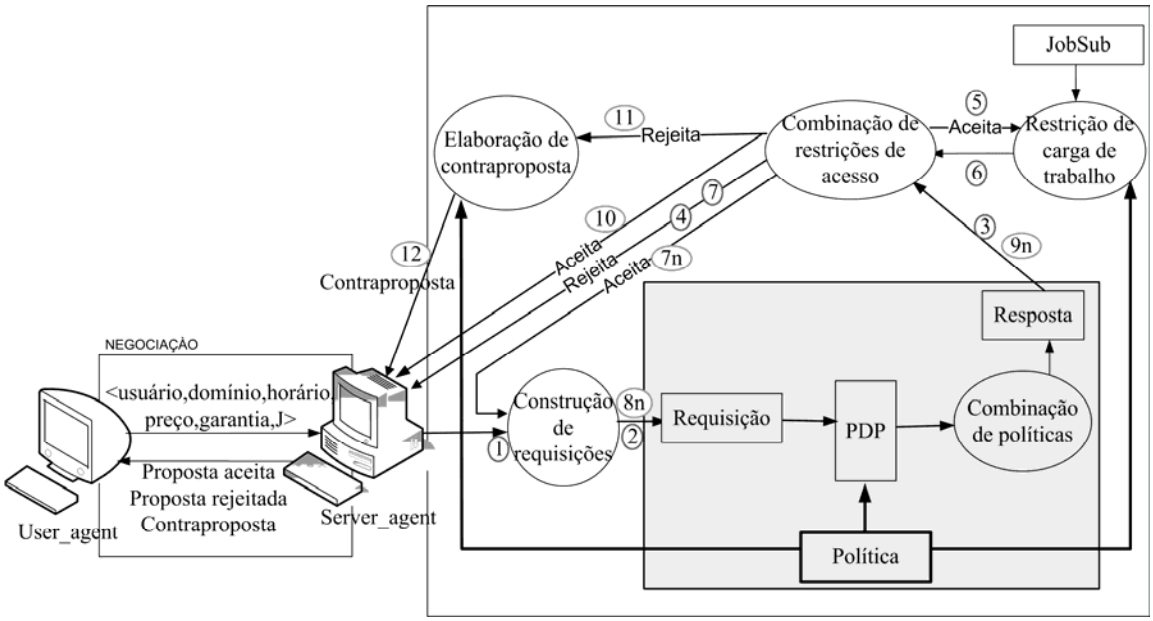

Figura 5-3: Arquitetura do POLAR integrado ao processo de negociação

O *server_agent* envia cada requisição individualmente para um PDP, que confronta um dos arquivos de requisição contra o arquivo de políticas e retorna respostas dentro do seguinte conjunto de opções: {negado, permitido, não aplicável}. O uso de um único arquivo de requisição contendo todos os atributos enviados no processo de negociação não identifica em qual atributo está sendo associada uma resposta do tipo "negado" e qual será a regra que proíbe o acesso ao recurso. Portanto, no modelo aqui elaborado, avançamos gradativamente na elaboração individual das requisições para serem confrontadas com as políticas disponíveis. Primeiramente, o *server_agent* elabora uma requisição contendo apenas os atributos de autenticação (identificação do usuário e horário), que corresponde ao fluxo 2 da Figura 5-3. Caso a resposta seja positiva ("permitido"), o *server_agent* elabora arquivos de requisição para o conjunto de parâmetros de SLA (instanciado por preço e garantia, que correspondem aos fluxos 7n, 8n e 9n da Figura 5-3) e checa tais arquivos individualmente contra o arquivo de política (repetindo os fluxos para cada parâmetro de SLA negociado). Dessa forma é possível verificar se, por exemplo, o preço sugerido pelo *user_agent* está dentro da faixa aceitável de valores aceitos pelo recurso. Os fluxos 7n, 8n e 9n podem ser executados *n* vezes para *n* variáveis de negociação nesse modelo, tornando-o genérico para mais variáveis que compõem um SLA.

A área sombreada da Figura 5-3 representa o contexto de uso do XACML para uma variedade de ambientes e aplicações [Godik & Moses 2003]. O contexto XACML descreve

uma representação canônica para as entradas e saídas do PDP. As seções a seguir apresentam a descrição de cada mecanismo presente no modelo da Figura 5-3: construção de requisições, combinação de políticas, combinação de restrições de acesso, elaboração de contraproposta e restrição de carga de trabalho.

5.3.1 Construção de requisições

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <Subject>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
         DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
   <AttributeValue>maria@dcc.ufmg.br</AttributeValue>
  </Attribute>
 </Subject>

 <Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
         DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>Sajama</AttributeValue>
  </Attribute>
 <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:StartJob"
         DataType="http://www.w3.org/2001/XMLSchema#time">
   <AttributeValue>08:30:00</AttributeValue>
  </Attribute>
 <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:Price"
         DataType="http://www.w3.org/2001/XMLSchema#integer">
   <AttributeValue>60</AttributeValue>
  </Attribute>
   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:Metric"
         DataType="http://www.w3.org/2001/XMLSchema#integer">
   <AttributeValue>response time</AttributeValue>
  </Attribute>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:Guarantee"
         DataType="http://www.w3.org/2001/XMLSchema#integer">
   <AttributeValue>30</AttributeValue>
  </Attribute>
 </Resource>

 <Action>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>RunJob</AttributeValue>
  </Attribute>
 </Action>
</Request>
```

Figura 5-4: Exemplo de uma requisição XACML incluindo a proposta enviada pelo *user_agent*

No POLAR, enquanto a descrição da política é única para cada servidor da grade computacional (representada pela estrutura de dados "Política" da Figura 5-3), as descrições de requisições (representadas pela estrutura de dados "Requisição" da Figura 5-3) são dinâmicas e construídas pelo *server_agent* a cada proposta de negociação recebida. A Figura 5-4 apresenta um exemplo de uma requisição em XACML construída pelo *server_agent* a partir uma proposta recebida pelo *user_agent*. Nessa proposta, conforme destacado na Figura 5-4, o usuário maria@dcc.ufmg.br deseja executar um job às 8h30min na máquina Sajama, ao preço máximo de 60 e garantia mínima de 30, caso o *job* não seja completado no período previsto.

5.3.2 Combinação de políticas

O mecanismo "Combinação de políticas", na Figura 5-3, executa um algoritmo que determina um resultado após verificar quais regras se aplicam à requisição enviada para o PDP. Os algoritmos padronizados pelo XACML que fazem combinação de regras são: "*deny overrides*", "*permit overrides*", "*first applicable*", e "*only one applicable*" [Godik & Moses 2003]. Caso o algoritmo de combinação utilizado seja o "*deny overrides*", se existir uma regra definida na estrutura Política que se aplique à requisição e o resultado dessa aplicação seja "*negar*" o acesso, então essa regra irá cancelar o efeito das demais e o resultado final será negar o acesso. No entanto, se o algoritmo utilizado for o "*permit overrides*", para o conjunto de regras em Política, se alguma regra permitir o acesso, então o resultado da combinação das regras deve ser permitir o acesso. Quando o algoritmo "*first-applicable*" é utilizado, todas as regras são analisadas em ordem até encontrar uma que se aplique à requisição e dessa forma a pesquisa pára e utiliza-se o resultado de permitir ou negar o acesso conforme o resultado definido pela regra aplicada. Para o algoritmo "*only-one-applicable*", pesquisa-se no conjunto de regras se existe apenas uma regra que se aplique à requisição analisada, retornando o resultado dessa avaliação. Se houver mais de uma regra aplicável, o resultado será indeterminado. Se não houver nenhuma regra que se aplique, o resultado será "*não aplicável*". Utilizamos o algoritmo "*deny-overrides*", pois o objetivo da verificação é saber se a requisição será negada pelo recurso e isso será verdade se existir pelo menos uma regra que tem como resultado "*deny*".

5.3.3 Combinação de restrições de acesso

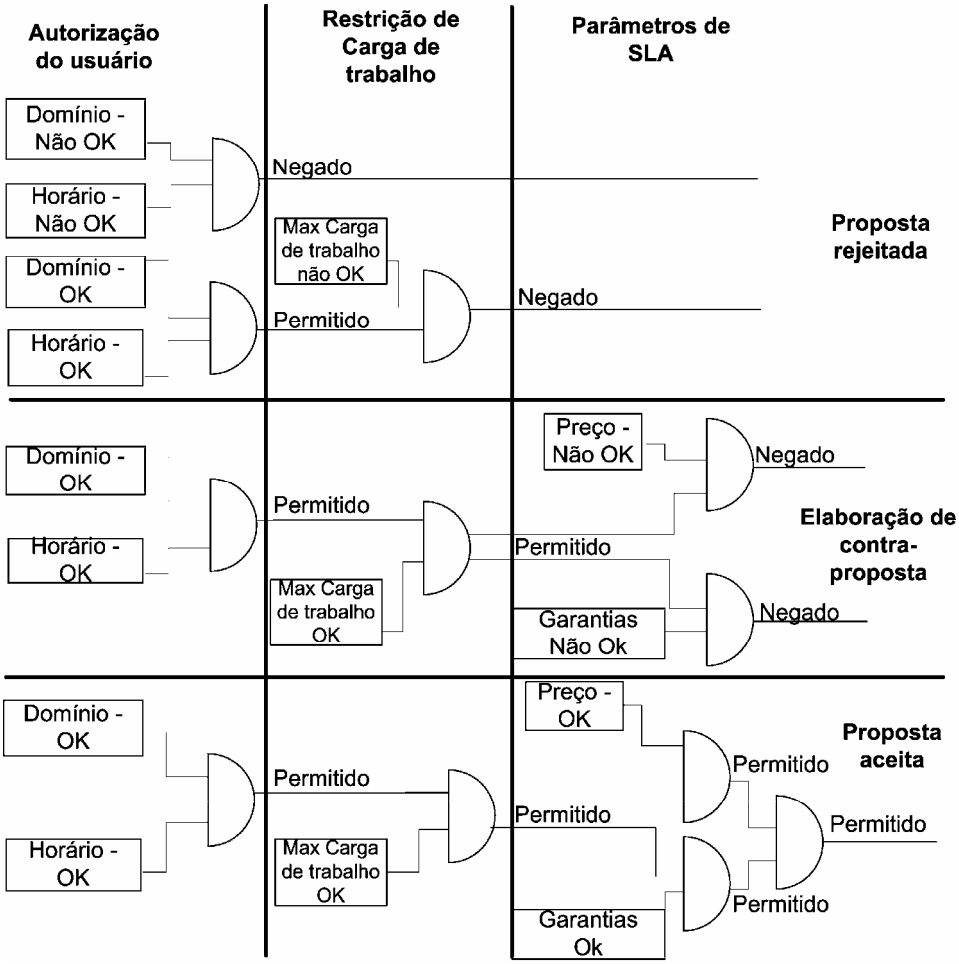

Figura 5-5: Mecanismo de combinação de restrições de acesso

A Figura 5-5 apresenta um mecanismo de avaliação do conjunto dos confrontos entre requisições e políticas e que corresponde ao processo "Combinação de restrições de acesso" da Figura 5-3. Tal mecanismo utiliza uma simbologia similar à de circuitos digitais combinatórios. Três resultados diferentes decorrem dessa avaliação: 1) a proposta pode ser rejeitada; 2) o *server_agent* pode elaborar uma contraproposta; e 3) a proposta pode ser aceita na íntegra. Caso o usuário não tenha autorização para executar o *job* no horário sugerido, a proposta será totalmente rejeitada. De modo semelhante, mesmo se o usuário tiver autorização para executar o *job*, mas o recurso tiver atingido a carga máxima de trabalho permitida, a proposta também será rejeitada. Se o usuário tiver autorização e a carga de trabalho máxima ainda não tenha sido atingida, porém um dos parâmetros de SLA estiver fora dos limites aceitáveis pelo recurso, o *server_agent* deverá elaborar uma

contraproposta para tais parâmetros. Caso o usuário seja autorizado, a carga máxima ainda não tiver sido atingida e todos os parâmetros estiverem dentro dos limites, então a proposta é aceita da forma originalmente enviada pelo *user_agent*.

5.3.4 Elaboração de contraproposta

Uma contraproposta é realizada quando um ou mais parâmetros negociáveis de SLA não são aceitos (Figura 5-5 – camada do meio). O valor da contraproposta para cada parâmetro é definido previamente para o recurso da grade computacional através da estrutura de dados Política (Figura 5-3) e é utilizado pelo *server_agent* para compor a contraproposta.

A contraproposta é elaborada pela alteração dos valores de preço e garantia. Um cenário que descreve essa possibilidade é dado a seguir: Um usuário envia para negociação um valor de preço p=20. No entanto, o preço mínimo definido na política associada ao recurso R consultado na negociação é p=40. Essa informação é extraída pelo *server_agent* do arquivo de políticas conforme exemplo de declaração na Figura 5-2. Seja uma situação onde todos os demais atributos são positivamente verificados no arquivo de políticas de R, exceto esse valor de preço. Portanto, na elaboração da contraproposta, o *server_agent* que representa R retorna para o *user_agent* os argumentos que foram aprovados, ou seja, que atendem às regras estabelecidas, e troca o valor do preço para p=40. Caso o usuário tenha instruído o seu agente (*user_agent*) para negociar até um preço máximo igual a 40, o *user_agent* pode considerar a proposta enviada pelo *server_agent* de R como uma proposta que atende à sua demanda.

5.3.5 Restrição de carga de trabalho

Existem várias formas de definir um limite máximo de uso relacionado à carga de trabalho em um recurso. Por exemplo, um dono de um recurso pode definir que os usuários da grade computacional podem usar somente 70% da capacidade de CPU e 50% da capacidade de memória de um recurso; ou o dono do recurso pode especificar o número máximo de *jobs* executáveis simultaneamente no mesmo recurso. Cada métrica definida como a referência delimitadora de uso com relação à carga de trabalho demanda um mecanismo específico para verificar se a carga de trabalho atingiu o valor máximo.

No nosso ambiente, o recurso deve estar dedicado à execução do *job* para manter a acurácia no modelo de predição descrito no capítulo 4. Portanto, neste capítulo, a restrição de carga de trabalho é associada ao número máximo de *jobs* simultâneos em um recurso. Outros *brokers*, tais como Condor, também destinam a execução de um novo *job* somente para um recurso livre, que não esteja rodando outro *job*. O nosso modelo permite que outras métricas relacionadas a carga de trabalho e o correspondente mecanismo de monitoração substituam a métrica e o mecanismo de controle de carga de trabalho que implementamos.

No atual mecanismo de restrição de carga de trabalho do POLAR, a monitoração dos *jobs* concorrentes é realizada através da estrutura de dados JobSub, que contém os atributos J, StartJ e EndJ, onde J representa um *job* já agendado para submissão no recurso, StartJ representa o horário previsto de início do *job* J e EndJ representa o horário previsto de término de execução do *job* J. EndJ é computado a partir da predição *predAd* enviada pelo módulo de predição, adicionado de StartJ, tal que EndJ=predAd+ StartJ.

A estrutura de dados JobSub é analisada para verificar quantos *jobs* simultâneos estão escalonados para o mesmo período de execução de um novo *job* J. Seja j cada valor individualmente pesquisado em JobSub e i uma instância para um novo *job* J. Na estrutura de dados JobSub, são considerados *jobs* simultâneos ao *job* J(i) se StartJ(j) ou EndJ(j) <= EndJ(i).

Todo novo *job* escalonado para o recurso é inserido de forma ordenada na estrutura de dados JobSub. Dessa forma, a pesquisa por *job*s concorrentes só ocorre enquanto EndJ(i) <= StartJ(j).

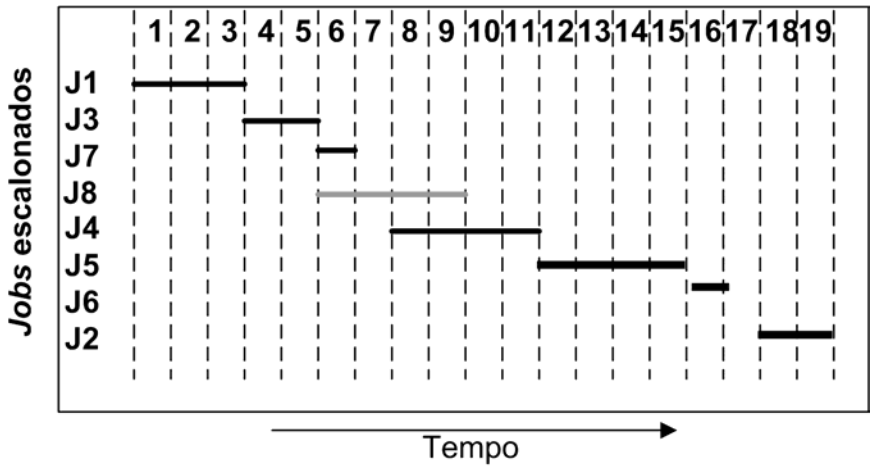

Figura 5-6: Exemplo de visualização do mapa temporal para *jobs* escalonados para um determinado recurso

Na Figura 5-6 apresentamos um mapa temporal para os *jobs* já escalonados para execução em um determinado recurso, onde o eixo das abscissas representa a duração do *job*. Os *jobs* foram ordenados pelo horário de início da execução e no exemplo dessa figura é permitida a execução de vários *jobs* simultâneos.

Dois algoritmos simples são associados à fase de restrição de carga de trabalho, sendo um referente à pesquisa e outro referente à inserção ordenada do *job* na estrutura de dados. O algoritmo de pesquisa é apresentado abaixo (Figura 5-7). No algoritmo de inserção ordenada, a ordenação se dá apenas pelo horário de início, não importando qual o horário final de execução do *job*.

```
Search-for-simultaneous-jobs(i)
{
  j=1
  Result = true
  Num_jobs = 0
  While StartJ(j) < EndJ(i) do
        If  StartJ(j) < StartJ(i) < EndJ(j) or StartJ(j) < EndJ(i) < EndJ(j)
              Num_jobs = Num_jobs + 1
              If Num_jobs > Max_Workload
                    Result= false
                    Break;
              End if
              j= j + 1;
        End if
 End While
 Return Result
}
```

Figura 5-7: Algoritmo *search-for-simultaneous-jobs*

Seja um exemplo de um novo *job*, J(8), ao cogitar o processamento onde já existem 7 *jobs* já escalonados para tempo futuro (conforme Figura 5-6). O job J(8) não poderá ser escalonado no recurso desse exemplo, pois outros *jobs,* J(7) e J(4), já possuem reserva para a execução naquele período.

5.3.6 Negociação integrada à verificação de restrições de acesso

A Figura 5-3 apresenta ainda um mecanismo de negociação associado ao *user_agent* e ao *server_agent*. Nesse mecanismo, o *user_agent* comporta-se da seguinte forma:

1) Elabora uma proposta de negociação baseada nos dados informados pelo usuário (Figura 3-7);

2) Filtra os *server_agents* candidatos conforme mecanismo apresentado em 3.3.1 e envia a proposta elaborada para *n server_agents*;

3) Aguarda a resposta dos *server_agents* até um tempo máximo pré-estabelecido no MASK;

4) Caso receba uma contraproposta, o *user_agent* analisa se o valor recebido enquadra-se na faixa de valores aceitáveis previamente configurados pelo usuário via interface (Figura 3-7);

5) Dentre as várias propostas aceitas pelos *server_agents*, o *user_agent* decide sobre a melhor opção (essa decisão é tratada no capítulo 6) e informa para o *server_agent* escolhido sobre a sua decisão;

No mecanismo de negociação, os *server_agents* comportam-se da seguinte forma:

1) Para cada requisição recebida é criado um *server_agent* (com função de negociação para cada requisição) com identificação única e que irá executar o modelo descrito na Figura 5-3.

2) Após o envio da resposta ao *user_agent* (fluxos 4, 7, 10 ou 12 da Figura 5-3), o *server_agent* aguarda resposta até um limite máximo de tempo previamente configurado no MASK para retirar ou manter a reserva de execução feita na estrutura JobSub.

3) A reserva feita na negociação anterior é considerada na análise de carga de trabalho para as próximas negociações.

O módulo de monitoração é executado após o término do módulo de negociação, conforme descrito na seção 3.3.4.

5.4 Validação do POLAR

O POLAR foi implementado utilizando a plataforma de desenvolvimento de agentes Jade [Jade 2005] (mais detalhes no Apêndice B) e a versão XACML da SUN [Sun 2003].

Esta seção apresenta a validação do POLAR. Os objetivos dos experimentos para validação do modelo de verificação de restrições de acesso são: 1) o funcionamento do modelo; e 2) o desempenho do modelo.

Para atender ao objetivo 1, mostramos o resultado da execução do POLAR com detalhes sobre os resultados intermediários obtidos em cada um de seus módulos para um conjunto misto de máquinas, restrições e requisições de acesso. Essa verificação é funcional e visa ilustrar as operações básicas do POLAR.

Para atender ao objetivo 2, comparamos o POLAR com duas outras abordagens. A comparação foi feita utilizando três métricas: 1) o tempo em que cada abordagem realiza uma seleção; 2) o *overhead* de verificação de restrições de acesso; e 3) o *overhead* de comunicação na rede. Essas métricas são importantes para destacar a eficiência obtida por abordagens centralizadas e descentralizadas na verificação de restrições de acesso local. Os resultados obtidos são generalizados pela introdução de índices que analisam o ambiente misto de restrições, requisições e configuração das máquinas.

Na seção 5.4.1, apresentamos a configuração do ambiente de teste, capaz de atender aos objetivos dos experimentos, que foram projetados para realizar a verificação funcional (seção 5.4.2) e a verificação de desempenho (seção 5.4.3) do POLAR. Na seção de verficação de desempenho, a nossa abordagem é comparada com uma abordagem ingênua, que seleciona recursos aleatoriamente, e com o Condor, que realiza a combinação de restrições de forma centralizada.

5.4.1 Configuração do ambiente

Cada recurso utilizado nos experimentos possui um conjunto de políticas de granularidade fina associadas. As políticas agrupam regras que delimitam o acesso por vários parâmetros. Os termos "requisições", "requisitos" e "restrições de acesso" são utilizados com os seguintes sentidos:

- Requisições: são propostas que um *server_agent* recebe de um *user_agent* durante um processo de negociação de recursos. As requisições consideradas nos experimentos incluem os seguintes atributos: preço máximo para utilizar o recurso, horário desejado para execução do *job*, garantia mínima desejada, usuário e domínio que deseja executar o *job*, CPU e RAM mínimos necessários para executar o *job*;
- Requisitos: são características físicas do recurso necessárias para executar um *job*. Os requisitos considerados nos experimentos são: a velocidade mínima do processador, a quantidade mínima de memória e o número de processadores. Os requisitos fazem parte das requisições e devem ser comparados com as características dos recursos.
- Restrições de acesso: são as limitações de uso definidas em forma de política de granularidade fina pelo proprietário do recurso. As restrições de acesso consideradas nos experimentos são denotadas pela tupla [preço, garantia, usuário,

horário permitido], especificando o preço mínimo para executar um *job* no recurso; a garantia máxima provida em caso de descumprimento do nível do serviço acordado; o usuário autorizado a executar *jobs* no recurso; o horário permitido para o usuário executar um *job* no recurso.

Para verificar o comportamento do POLAR e de outras abordagens, construímos um conjunto heterogêneo de restrições (Tabela 5-2) e de requisições (Tabela 5-3) com valores aleatórios, para um conjunto também heterogêneo de máquinas do nosso experimento (Tabela 5-1).

Na Tabela 5-2, a coluna "Preço mínimo" refere-se ao preço mínimo para executar um *job* no recurso; a coluna "Garantia (desconto)" significa um desconto no preço original negociado se o tempo de execução for maior que o tempo previsto; a coluna "Usuário" descreve qual usuário tem autorização para acessar o recurso durante o intervalo de tempo estabelecido pela coluna "Horário permitido", outros domínios da organização virtual poderão acessar o recurso sem restrição de horários. A Tabela 5-3 apresenta as requisições dos usuários para submissão de *jobs* contendo somente dados relacionados à verificação de restrições de acesso.

Tabela 5-1: Configuração das máquinas no experimento de validação do POLAR

Máquina	Quantidade Processadores	RAM (MB)	CPU (MHz)	Modelo do Processador
Alegria	1	512	1666	AMD Athlon 2000+ mod 8
Amizade	1	512	1666	AMD Athlon 2000+ mod 8
Caridade	1	512	1800	AMD Athlon 2500+ mod 10
Confiança	1	512	1666	AMD Athlon 2000+ mod 8
Ibituruna	1	512	1666	AMD Athlon 2000+ mod 8
Itapeva	1	512	1800	AMD Athlon 2500+ mod 10
Jumento	2	512	2400	Intel(R) Pentium(R) 4 CPU 2.40GHz
Montblanc	1	512	1800	AMD Athlon 2500+ mod 10
Saudade	1	512	1666	AMD Athlon 2000+ mod 8
Tapir	2	512	2400	Intel(R) Pentium(R) 4 CPU 2.40GHz
Zebra	2	512	2400	Intel(R) Pentium(R) 4 CPU 2.40GHz

As restrições de acesso foram definidas utilizando o padrão XACML para cada uma das máquinas do experimento. Em cada recurso há um agente que representa o servidor. As requisições do usuário para submissão de *jobs* são enviadas pelo *user_agent* e transformadas pelo *server_agent* em requisições em formato XACML de forma dinâmica utilizando um construtor Java de requisições conforme a Figura 5-3.

Nos experimentos desta seção, o número máximo de *jobs* simultâneos em cada recurso é um, ou seja, o *job* roda de forma exclusiva no recurso. Esta opção de exclusividade na execução do *job* se deve às condições do nosso ambiente de teste e para obtenção de melhor acurácia do modelo de predição conforme analisado no capítulo 4.

Tabela 5-2: Restrições de acesso utilizadas no experimento de validação do POLAR

Máquina	Preço mínimo	Garantia (desconto)	Usuário	Horário permitido
Alegria	80	40	maria@dcc.ufmg.br	0 a 24
Amizade	50	10	maria@dcc.ufmg.br	9 as 15
Caridade	90	30	maria@dcc.ufmg.br	18 a 24
Confiança	100	50	maria@dcc.ufmg.br	00 a 24
Ibituruna	70	30	maria@pbh.gov.br	10 a 22
Itapeva	50	50	maria@dcc.ufmg.br	10 a 22
Jumento	50	30	maria@dcc.ufmg.br	0 a 24
Montblanc	80	20	maria@dcc.ufmg.br	1 a 24
Saudade	70	40	maria@pbh.gov.br	18 a 24
Tapir	60	50	maria@pbh.gov.br	00 a 24
Zebra	80	20	maria@dcc.ufmg.br	01 a 24

Tabela 5-3: Requisições utilizadas no experimento de validação do POLAR

Requisição	Preço máximo	Horário	Garantia mínima (desconto)	Usuário	CPU mínima (MHz)	RAM mínima (MB)
1	80	8	20	maria@dcc.ufmg.br	1600	512
2	90	10	30	maria@dcc.ufmg.br	1600	512
3	70	15	40	maria@dcc.ufmg.br	1600	512
4	60	12	10	maria@pbh.gov.br	1600	512
5	50	9	10	maria@dcc.ufmg.br	1600	512
6	100	22	50	maria@dcc.ufmg.br	1600	512
7	60	21	50	maria@pbh.gov.br	1600	512
8	30	10	30	maria@pbh.gov.br	2800	512
9	100	8	40	maria@dcc.ufmg.br	1600	512
10	70	18	20	maria@dcc.ufmg.br	1600	512
11	100	20	40	maria@pbh.gov.br	1600	512
12	80	3	30	maria@pbh.gov.br	1600	512
13	50	2	10	maria@pbh.gov.br	2000	512
14	60	15	20	maria@pbh.gov.br	1600	512
15	60	7	30	maria@dcc.ufmg.br	1600	512
16	90	13	40	maria@dcc.ufmg.br	1600	512
17	90	11	50	maria@pbh.gov.br	2000	512
18	60	5	10	maria@pbh.gov.br	2000	512
19	70	20	20	maria@dcc.ufmg.br	1600	512

A configuração do ambiente aqui apresentado é adequada para realizar as verificações funcionais e de desempenho de POLAR, pois apresenta a replicação dos algoritmos de

verificação de restrições de acesso em cada recurso, utiliza um conjunto de requisições variadas e utiliza um conjunto heterogêneo de recursos e de políticas de granularidade fina distribuídas.

5.4.2 Verificação funcional

O objetivo desta seção é realizar a validação funcional do POLAR para um conjunto misto de requisições e restrições de acesso. A validação funcional é feita considerando a autorização do usuário, a carga de trabalho e os parâmetros de negociação. O resultado fornecido pelo mecanismo de combinação de restrições de acesso (Figura 5-5) é enviado para o agente do usuário por meio de um processo de negociação (Figura 5-3).

Tabela 5-4: Teste funcional das requisições para a máquina Ibituruna

Requisições	Autorização do usuário	Carga de trabalho do recurso	Parâmetros negociáveis de SLA	Resultado
1	Não ok	-	-	Rejeita proposta
2	ok	ok	ok	Aceita proposta
3	ok	ok	Garantia < 40	Contraproposta Garantia=30
4	ok	ok	Preço > 60	Contraproposta Preço=70
5	Não ok	-	-	Rejeita proposta
6	Não ok	-	-	Rejeita proposta
7	ok	ok	Preço > 60 Garantia < 50	Contraproposta Preço = 70 Garantia = 30
8	-	-	-	-
9	Não ok	-	-	Rejeita proposta
10	ok	ok	ok	Aceita proposta
11	ok	Não ok	-	Rejeita proposta
12	ok	ok	ok	Aceita proposta
13	-	-	-	-
14	ok	Não ok		Rejeita proposta
15	Não ok	-	-	Rejeita proposta
16	ok	Não ok	-	Rejeita proposta
17	-	-	-	-
18	-	-	-	-
19	ok	Não ok	-	Rejeita proposta

Um mesmo *job*, contendo a mesma aplicação e atributos, está associado às requisições apresentadas na Tabela 5-3. A duração da execução desse *job* no recurso considerado é de

2h. Todas as requisições são submetidas para serem executadas no dia seguinte e são enviadas de forma seqüencial (a cada 15 segundos) conforme o número da requisição (Tabela 5-3).

Normalmente, após a reserva de execução do *job* para o horário solicitado, o *server_agent* aguarda um período de 10s para receber a confirmação da reserva pelo *user_agent*. Esse tempo de espera é previamente configurado no MASK e é superior ao término de todo o processo de seleção do recurso e, portanto da confirmação de execução do *job* para o horário solicitado ao recurso. No presente experimento, o *server_agent* foi alterado para não liberar as reservas feitas pelas requisições. Tal ajuste visou apresentar um mapa temporal (Figura 5-8) para todas as requisições feitas e aceitas na máquina Ibituruna, independentemente da escolha desa máquina para processar as requisições solicitadas.

A Tabela 5-4 apresenta o comportamento funcional do POLAR na máquina Ibituruna para cada requisição da Tabela 5-3. As verificações da funcionalidade estão associadas aos processos do POLAR, denominados na Tabela 5-4 como "Autorização do usuário", que corresponde aos fluxos 1, 2 e 3 da Figura 5-3; "Carga de trabalho do recurso", que corresponde ao processo de "Restrição de carga de trabalho" da Figura 5-3; e "Parâmetros negociáveis de SLA", que corresponde aos fluxos 7n, 8n e 9n da Figura 5-3. A coluna "Resultado" da Tabela 5-4 apresenta o resultado da verificação das restrições de acesso, que corresponde aos fluxos 4, 7, 10 e 12 da Figura 5-3, para cada requisição da Tabela 5-4. As requisições 8, 13, 17 e 18 não foram avaliadas, pois a máquina Ibituruna não atendia aos requisitos solicitados.

A Figura 5-8 apresenta o mapa temporal de reserva de horários para execução de *jobs* que é utilizado pelo processo de "Restrições de carga de trabalho" (Figura 5-3) na máquina Ibituruna.

Tendo em vista os dados apresentados na Tabela 5-4 e na Figura 5-8, foi possível verificar a funcionalidade do POLAR para uma das máquinas do experimento, certificando um correto e detalhado funcionamento do modelo e apresentado aqui como exemplo de seu comportamento para uma dada configuração de ambiente.

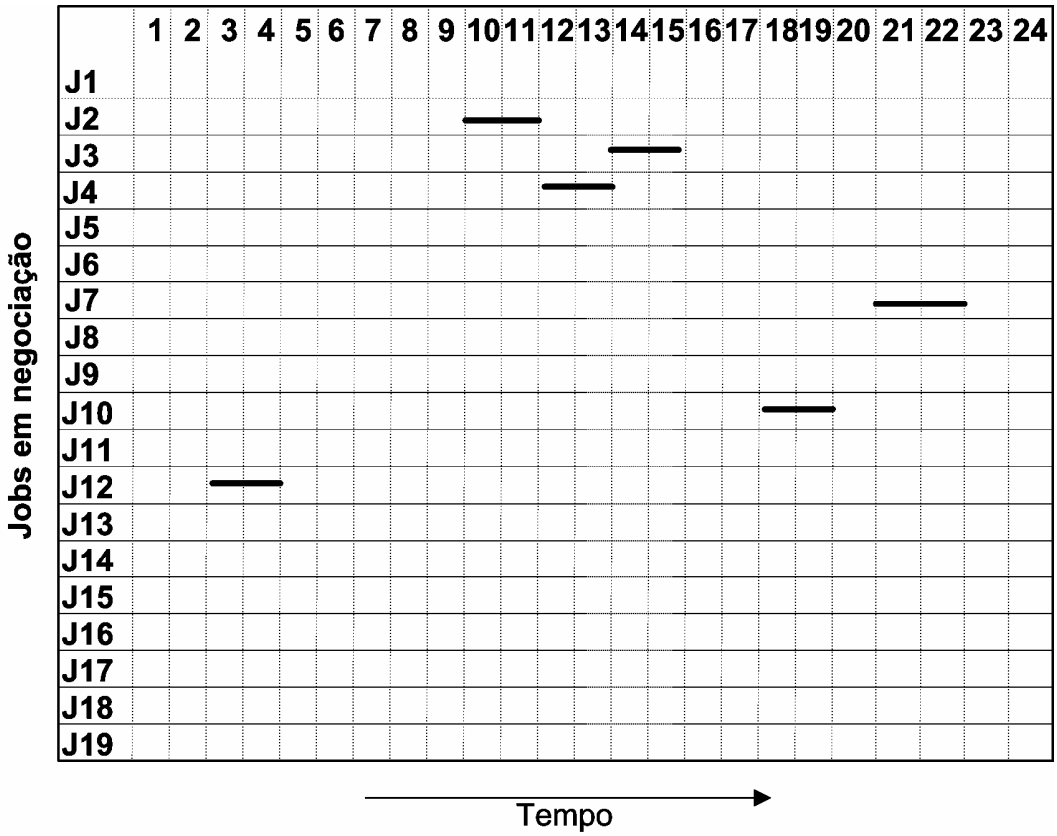

Figura 5-8: Mapa temporal para o recurso Ibituruna para as negociações das requisições da Tabela 5-3

5.4.3 Verificação de desempenho

Nesta seção, o nosso objetivo é analisar o desempenho do POLAR. Para isso definimos algumas métricas e apresentamos os resultados de experimentos em um ambiente real. Os experimentos destacam os problemas advindos de soluções que não estão preparadas para trabalhar com uma infra-estrutura de grade computacional que faz uso de políticas de granularidade fina, e demonstram as vantagens e desvantagens de abordagens distribuídas e centralizadas para a verificação de restrições de acesso.

Para esse fim, comparamos a nossa abordagem com um algoritmo de seleção randômica (que não realiza verificação de restrição de acesso antes da submissão) e com o Condor (que verifica as restrições de acesso de forma centralizada).

O POLAR verifica as requisições para o conjunto de máquinas que atendem aos requisitos, de acordo com as restrições descritas em cada recurso candidato. Esse conjunto de máquinas que atendem aos requisitos foi previamente verificado pelo mecanismo de descoberta de recursos do MASK (descrito em 3.5.2).

Considerando apenas o modelo de verificação de restrição de acesso, relatamos abaixo a seleção de recurso nos experimentos desta seção:

1) Cada recurso possui sua própria definição de políticas de restrição de acesso descritas em XACML;

2) O *user_agent* envia requisições para *n server_agents*;

3) O *user_agent* aguarda o resultado da computação do POLAR em cada recurso candidato, mas seleciona, neste experimento, a resposta afirmativa que chegou primeiro. A seleção do recurso é tratada no capítulo 6, onde outras métricas, além das restrições de acesso, são consideradas.

O algoritmo randômico seleciona um recurso aleatoriamente enviando uma requisição (Tabela 5-3). Na nossa implementação do algoritmo randômico, quando a requisição chega à máquina destino, essa verifica os requisitos e as restrições de acesso utilizando políticas locais (a mesma realizada por POLAR). Caso a máquina inicialmente selecionada pelo algoritmo randômico não atenda às requisições e/ou não permita o acesso, o algoritmo randômico re-submete a requisição para outra máquina do experimento. A requisição é re-submetida até que seja encontrada uma máquina que permita o acesso.

O Condor, ao receber uma requisição, executa o algoritmo *matchmaking* que então processa a combinação dos anúncios das requisições com os anúncios dos requisitos e restrições de todas as máquinas de forma centralizada.

Para medir o desempenho dessas abordagens utilizamos três métricas diferentes: Tempo de seleção de recurso para uma requisição (TSR), *Overhead* de verificação de restrições de acesso (OVRA) e *Overhead* de comunicação na rede (OCR). A métrica TSR mede o período de tempo entre a submissão da requisição e a seleção do recurso para cada abordagem. A métrica OVRA mede o número de máquinas que processaram a verificação de restrições de acesso. A métrica OCR mede o número de máquinas consultadas via rede para processar a verificação de restrições de acesso.

A configuração de cada máquina, o perfil de restrição de cada recurso e o perfil de cada requisição de acesso são os mesmos apresentados nas tabelas 5-1, 5-2 e 5-3 respectivamente.

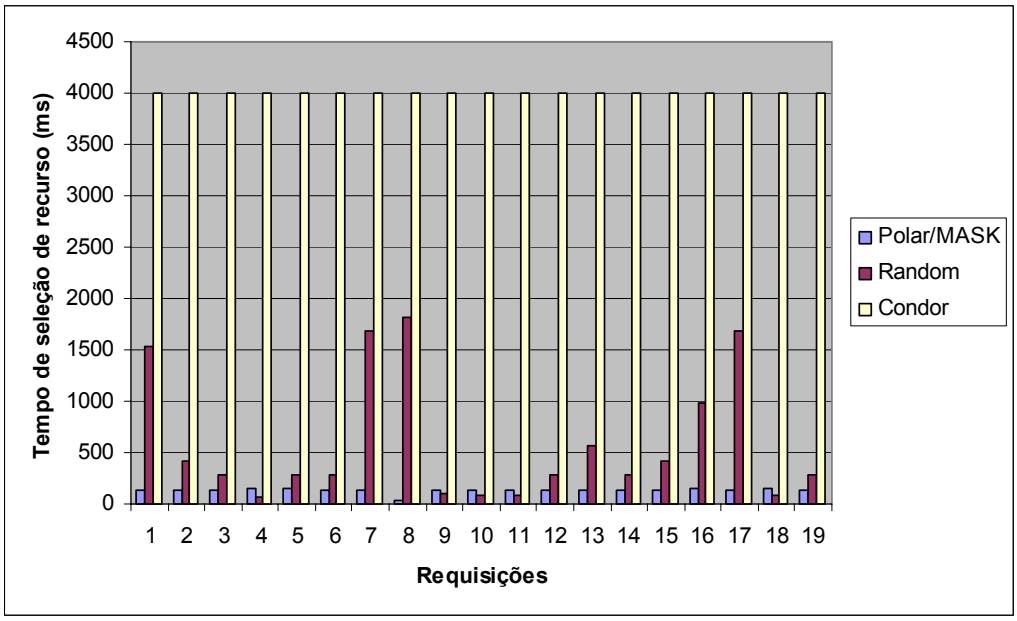

Figura 5-9: Tempo de seleção de recurso (TSR) para cada requisição

A Figura 5-9 apresenta a métrica TSR, o tempo de seleção de recurso para cada requisição e cada abordagem. As máquinas do experimento possuem diferentes tempos de verificação de políticas, já que é um ambiente heterogêneo. Como o MASK negocia com as máquinas que atendem aos requisitos, de forma paralela, o tempo de seleção de recurso é dado pelo tempo de processamento do POLAR na máquina mais lenta do experimento. O TSR no algoritmo randômico varia de acordo com o número de re-submissões necessárias, ou seja, agrega-se o tempo de processamento de cada uma das submissões. Para Condor, o TSR foi verificado por consulta ao arquivo de *log* das requisições submetidas. Nesse arquivo, subtraímos o tempo registrado para "*job submitted*" do tempo registrado para "*job executing*[11]". Para o conjunto de restrições, requisitos e máquinas deste experimento, o Condor apresentou sempre um valor de 4s para o TSR.

Quando o algoritmo randômico seleciona a máquina com somente uma submissão, como ocorre nas requisições 4, 9, 10, 11 e 18 da Figura 5-9, ele tem melhor desempenho que o POLAR. No entanto, o tempo de seleção (TSR) na abordagem randômica é pior nas demais requisições da Figura 5-9 e chega a ser bastante elevado quando há várias re-

[11] No arquivo de *log* de um *job* submetido por Condor, sempre estão presentes o registro do horário em que o *job* foi submetido e o horário em que o *job* iniciou sua execução.

submissões, como mostra o tempo associado à requisição 17. Se, no entanto, uma máquina muito lenta for incluída no experimento, ela irá elevar o TSR do POLAR.

Considerando as abordagens comparadas, identificamos modelos diferentes que podem influenciar o valor de TSR dadas pelas equações (1), (2) e (3) para a abordagem POLAR, Randômica e Condor, respectivamente.

$$TSR(POLAR/MASK)=S(A)+\{[C(B)+OC(N) \mid MaxTE\} \quad (1)$$

$$TSR(RANDOM)=\sum_{1}^{N}[D(B)+OC(N)] \quad (2)$$

$$TSR(Condor)=TC(N)+TM(A+B)*N \quad (3)$$

Onde:

A representa a quantidade de requisitos;

S(A) representa o tempo para descobrir os recursos que atendem a A requisitos;

B representa a quantidade de restrições;

C(B) representa o tempo de verificação de B restrições para o recurso mais lento;

N representa o número de recursos consultados;

OC(N) representa o *overhead* de comunicação para N recursos;

MaxTE representa o tempo máximo que o *user_agent* espera resposta dos *server_agents* para as propostas enviadas;

D(B) representa o tempo de verificação de B restrições para cada recurso;

TC(N) representa o tempo de coleta de informações de N recursos no Condor;

TM (X) representa o tempo de *matchmaking* para X variáveis no Condor;

Na equação (1), o *user_agent* em MASK pode aguardar pela verificação de restrições da máquina mais lenta até um limite de tempo máximo definido por MaxTE. Dessa forma, o *user_agent* considera somente as opções que chegaram antes de expirar esse limite de tempo.

Pelo fato de não termos o código fonte do Condor[12], não temos os dados precisos das variáveis envolvidas para generalizar um modelo do tempo de seleção para ele. Um modelo provável, com as devidas ressalvas, seria a computação de A+B restrições para N servidores, conforme apresentado na equação (3). Em baixa escala (no nosso ambiente

[12] No momento do presente experimento, não estava disponível o código fonte do Condor no *site* desse projeto: http://www.cs.wisc.edu/condor/

disponível), o Condor não mostrou-se sensível ao TSR para pequenas variações dos parâmetros A,B e N.

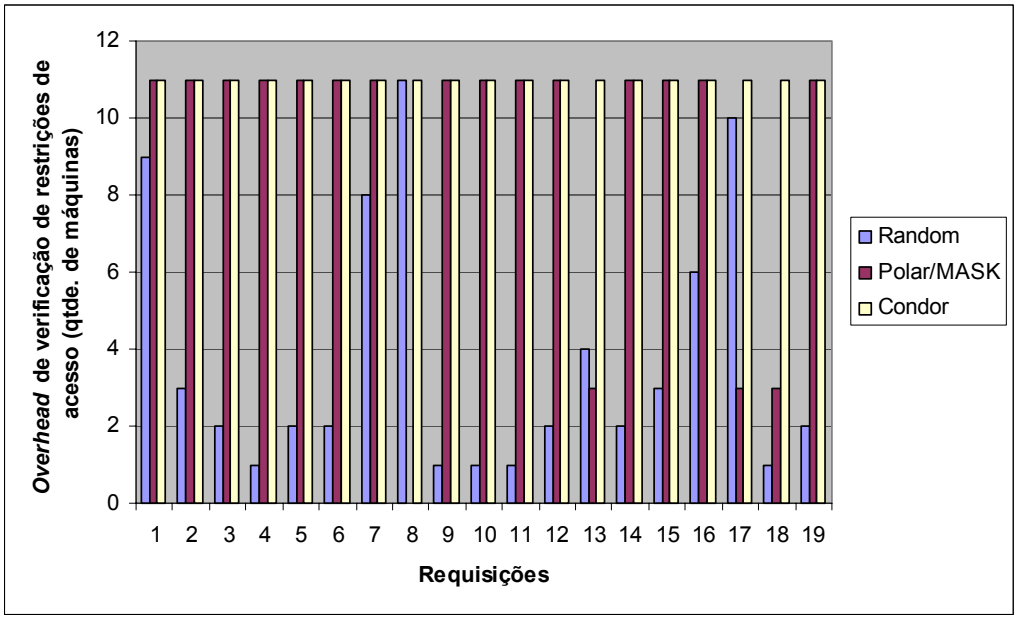

Figura 5-10: *Overhead* de verificação de restrições de acesso (OVRA)

A Figura 5-10 apresenta a métrica OVRA, a quantidade de verificações de restrições de acesso realizadas para cada abordagem. O Condor possui o maior OVRA uma vez que executa o algoritmo *matchmaking* para todas as máquinas. O OVRA para o MASK varia de acordo com a descoberta de recursos que atendem aos requisitos e a abordagem randômica varia de acordo com o número de re-submissões. Na Figura 5-10, o POLAR deixa de verificar as restrições para a requisição 8, pois nenhuma máquina do ambiente atende aos requisitos mínimos solicitados por essa requisição. Para a grande maioria das requisições analisadas, o OVRA da abordagem randômica foi menor (melhor) que o Condor e o MASK.

A Figura 5-11 apresenta a métrica OCR, a quantidade de máquinas consultadas via rede para realizar a verificação de restrições de acesso para cada abordagem. O Condor não tem registro de OCR para as requisições, pois possui as informações centralizadas sobre os requisitos e restrições de cada máquina e, portanto não utiliza a rede para fazer verificar as restrições de acesso. O *overhead* de comunicação no Condor só ocorre quando é alterada alguma configuração de restrição na máquina, que então comunica-se com a máquina central repassando os novos valores. O OCR do POLAR é o maior (pior) do conjunto das

abordagens analisadas. O que era de se esperar, por ser o POLAR uma abordagem distribuída e, portanto fazer mais uso da rede para a comunicação entre os processos da solução. A abordagem randômica utiliza a rede de forma seqüencial, conforme o número de re-submissões.

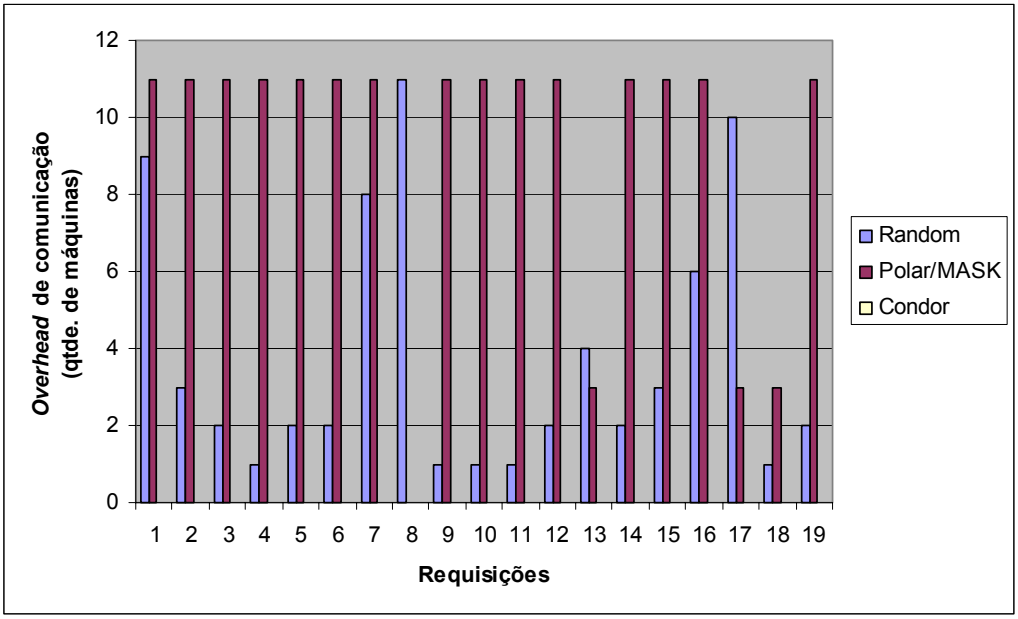

Figura 5-11: *Overhead* de comunicação na rede (OCR) para verificação de restrições de acesso

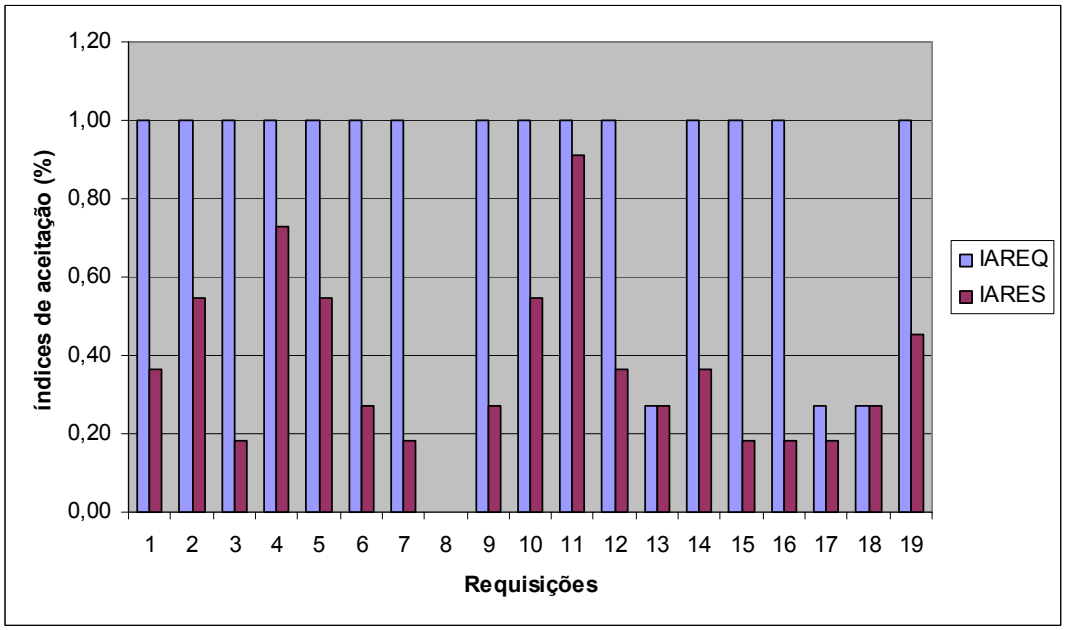

Figura 5-12: Índices de aceitação de requisitos e de restrições para cada requisição

O que pode variar nos resultados obtidos nos gráficos 5-9, 5-10 e 5-11 depende da variação do perfil das requisições, da configuração das máquinas e do perfil de restrições de acesso conforme tabelas 5-1, 5-2 e 5-3. Para analisarmos a composição desses parâmetros em conjunto e seu impacto no comportamento de TSR, OVRA e OCR, definimos dois índices: Índice de Aceitação de Requisitos (IAREQ) e Índice de Aceitação de Restrições (IARES). O IAREQ representa o percentual de máquinas no experimento capaz de atender aos requisitos (CPU, memória, número de processadores) das requisições. O IARES representa o percentual de máquinas no experimento capaz de atender às restrições (preço, garantia e carga de trabalho) das requisições.

Observa-se pela Figura 5-12, que quando várias máquinas atendem às restrições (IARES > 40%), o algoritmo randômico pode encontrar uma máquina adequada com menos re-submissões (até 3 re-submissões), por exemplo, para as requisições 2, 4, 5, 10, 11 e 19 da Figura 5-11. No entanto, pela sua característica aleatória o algoritmo randômico escolheu fortuitamente a máquina correta para a requisição 3, em apenas duas re-submissões, mesmo com um baixo índice de IARES (abaixo de 20%).

Conforme a Figura 5-11 e considerando os índices da Figura 5-12, o melhor caso na análise da métrica OVRA para o MASK (baixo OVRA) é quando nenhuma máquina atende aos requisitos (IAREQ=0). O pior caso para o MASK (alto OVRA) é quando todas as máquinas atendem aos requisitos (IARES=N) e nesse caso o OVRA do MASK é igual ao OVRA do Condor.

Ao compararmos as Figuras 5-9 a 5-11, percebemos que quanto menor o índice IAREQ, menores os *overheads* (OVRA e OCR) do MASK. Espera-se que quanto menor o índice IARES, maior o tempo (TSR) da abordagem randômica. Os índices IAREQ e IARES não influenciam no tempo de espera (TSR) e nem nos *overheads* (OVRA e OCR) do Condor, pois este verifica todos os requisitos e restrições para todas as máquinas e não utiliza comunicação via rede no momento do *matchmaking*.

5.5 Conclusões

O uso de políticas de granularidade fina para declarar as restrições de acesso permite que os proprietários dos recursos descrevam as restrições de acesso individualmente para cada usuário e ação associada. Essa flexibilidade avança as possibilidades do

compartilhamento de recursos em ambientes de grades computacionais passando das regras gerais definidas para a organização virtual para regras específicas ao nível de recursos e usuários.

A verificação das restrições de acesso deve anteceder a submissão de um *job* evitando o cancelamento do mesmo. Para essa função, o uso de um *broker* é imprescindível. Embora existam *brokers* na literatura que realizam essa verificação de restrições, ainda existem desafios a serem superados. Um dos *brokers* mais populares da área de grades computacionais, o Condor, utiliza uma linguagem própria e semi-estruturada para declarar as políticas locais de um recurso. Tal solução faz com que o ponto de verificação real das limitações do acesso via um padrão PEP tenha que ser administrado separadamente. Essa dupla configuração de políticas locais, uma para o *broker* e outra para o PEP, gera redundância no ambiente.

Destacamos neste capítulo a construção de um modelo denominado POLAR para a verificação de restrições de acesso baseado em políticas de granularidade fina. O POLAR é parte integrante do sistema multiagente MASK, que seleciona o melhor recurso para executar um *job* no ambiente de grade computacional em tempo de execução.

No POLAR, a descrição e a verificação das políticas integram as funções de controle de acesso, restrição quanto ao uso de parâmetros de SLA e controle da carga de trabalho permitida para o recurso no ambiente compartilhado. A estrutura local de definição de políticas utilizada pelo nosso *broker* segue os padrões de PEP. Dessa forma, a mesma estrutura para declarar políticas de uso para o ambiente compartilhado pode ser utilizada para declarar restrições de acesso para os usuários locais que não participam da grade computacional.

O Condor realiza a combinação dos requisitos de forma centralizada. Tal processamento tem suas vantagens e desvantagens. Nos experimentos realizados, analisamos o desempenho de três abordagens: MASK, Condor e uma abordagem ingênua que corresponde à submissão de um *job* na grade por um usuário que não utiliza *brokers*.

Por ser um *middleware* que realiza a combinação de restrições de forma distribuída, o MASK apresenta *overhead* de comunicação na rede para computar a verificação de restrições de acesso. No entanto, por ser distribuído, o MASK é capaz de oferecer o resultado dessa verificação mais rapidamente que Condor.

Capítulo 6

Modelo de decisão para seleção de recursos em grades computacionais

6.1 Introdução

Em um ambiente de grade computacional, organizações compartilham dinamicamente um conjunto de equipamentos heterogêneos e distribuídos [Foster & Kesselman 2004]. Devido à essa diversidade, selecionar recursos em tempo de execução para executar aplicações com perfis distintos é uma tarefa complicada de ser realizada pelo usuário sem o auxílio de um *broker* de recursos.

A seleção de um recurso sem critérios definidos não explora a potencialidade e os benefícios do ambiente compartilhado. O usuário não tem como escolher uma opção que melhor atenda às suas expectativas de custo e desempenho. Por exemplo, os proprietários de recursos não têm como influenciar a participação de seu recurso no ambiente compartilhado de forma justa e vantajosa; os administradores do ambiente não têm como distribuir a carga do sistema ou equilibrar o custo de uso.

Um conjunto de fatores está envolvido no momento de seleção do recurso, dentre eles os requisitos mínimos solicitados pela aplicação, o tempo de execução da aplicação e as restrições de acesso ao recurso. Além disso, a seleção de recursos deve considerar as incertezas associadas a cada recurso candidato à execução do *job* e responder, por exemplo,

às seguintes questões: Qual a confiabilidade do servidor? Qual a probabilidade de haver erro na predição do tempo de execução? Qual a probabilidade de mudança do custo final do provimento do serviço?

Esse problema de seleção de recursos em grade computacional é tratado atualmente na literatura sob várias formas. O capítulo 2 faz uma revisão bibliográfica que aponta abordagens diferentes para a solução desse problema. No entanto, geralmente a seleção dos recursos nos trabalhos relacionados é orientada somente a um fator relevante, que pode ser o desempenho da máquina, o preço de uso do recurso, a confiabilidade do servidor ou a permissão do acesso. Não há, até o momento da nossa revisão bibliográfica, um modelo que conjugue adequadamente todos os fatores importantes para a melhor seleção do recurso em grade computacional e que permita ao decisor ter preferências em proporções diferentes entre os fatores.

Desenvolvemos e apresentamos, neste capítulo, um modelo de decisão para a seleção de recursos em grade computacional que utiliza a teoria da decisão através do *middleware* MASK, apresentado no capítulo 3. O modelo considera a preferência do usuário entre preço e tempo de execução, os requisitos da aplicação, a avaliação de acesso ao recurso e a confiabilidade do servidor. A teoria da decisão é atrativa para problemas de seleção, pois provê uma metodologia para lidar com riscos e a natureza multi-objetiva das decisões. Para cada objetivo há um atributo e uma probabilidade associada. A teoria da decisão fornece uma forma sistemática para considerar relações entre atributos, permitindo a tomada de decisão [Butler 2001]. A seleção do recurso no sistema multiagente MASK utiliza técnicas da teoria da decisão ainda não exploradas para esse fim no ambiente de grades computacionais, e executa o modelo de seleção de forma paralela e distribuída, lidando com graus de risco. Os resultados mostram a influência das variáveis de risco e da preferência do usuário na seleção do recurso.

Este capítulo está organizado como se segue. A seção 6.2 apresenta conceitos relacionados com a teoria da decisão. A seção 6.3 apresenta a formalização do problema de seleção do melhor recurso para grade computacional. A seção 6.4 apresenta o modelo de seleção utilizando teoria da decisão. A seção 6.5 apresenta resultados experimentais e finalmente a seção 6.6 conclui o capítulo.

6.2 Teoria da decisão

Para apresentarmos a teoria da decisão, inicialmente definimos na seção 6.2.1 alguns termos que estamos adotando neste capítulo e que irão embasar a definição e uso da teoria da decisão apresentada na seção 6.2.2.

6.2.1 Algumas definições

Loteria: Uma loteria é uma distribuição de probabilidades sobre um conjunto real de conseqüências. [A,p;C,1-p] representa uma loteria entre as conseqüências A e C, onde p é a probabilidade de obter a conseqüência A e 1-p é a probabilidade de obter a conseqüência C [Gomes *et al.* 2002].

Estado da natureza - θ: É uma determinada representação da realidade [Russel & Norvig 2003];

Função valor esperado - E(X): O valor esperado de uma variável aleatória é a soma dos produtos das probabilidades de cada conseqüência e seu valor. Para uma variável aleatória discreta X com valores possíveis $x_1,x_2,...,x_n$ e com as suas probabilidades representadas pela função $Pr(x_i)$, o valor esperado para X é calculado por:

$$E(X)=\sum_{i=1}^{n} x_i \, Pr(x_i)$$

Função utilidade - U(S): Designa um número para expressar o desejo por um estado. Utilidade é uma função que mapeia estados em números. A notação U(S) denota a utilidade do estado S para o agente que está tomando decisões. A função utilidade para uma determinada variável pode ser definida utilizando-se expressões analíticas, tais como função utilidade linear, exponencial ou logarítmica [Bekman & Costa Neto 2002].

Comportamento do decisor diante do risco: O comportamento do decisor define seu perfil em relação ao risco [Gomes *et al.* 2002], podendo ser avesso, propenso ou neutro ao risco como descrito a seguir:

Avesso ao risco: O decisor é avesso ao risco quando ele prefere permanecer com o valor esperado de sua riqueza a apostar (função de utilidade côncava – função exponencial).

Propenso ao risco: O decisor é propenso ao risco quando ele prefere apostar a permanecer com o valor esperado de sua riqueza (função de utilidade convexa – função logarítmica).

Neutro ao risco: O decisor não se preocupa com os riscos a que sua riqueza esteja sujeita, preocupa-se somente com o valor esperado dela (função linear).

Máxima utilidade esperada - (MUE): O princípio da máxima utilidade esperada (MUE) diz que um agente racional deve escolher uma ação que maximize a utilidade esperada do agente.

Teoria da utilidade multiatributo - (MAUT): É referenciada por MAUT (*Multi-Attribute Utility Theory*) e incorpora à teoria da utilidade a questão do tratamento de problemas com múltiplos objetivos [Gomes *et al.* 2002]. Uma representação abrangente dos diversos aspectos que influenciam a escolha de alternativas de um problema é feita por meio da modelagem de preferências que envolvem mais de um atributo.

Como exemplo da aplicação de uma função multiatributo, considere o cenário para instalar um novo aeroporto, no qual é preciso analisar diversos problemas, tais como o custo do terreno, a distância dos centros populacionais, o barulho das operações de vôo, a topografia local e as condições climáticas. Problemas como esses onde os resultados são caracterizados por um ou mais atributos podem ser abordados pela teoria da utilidade multiatributo (MAUT).

Uma forma típica para a função utilidade multiatributo é a função utilidade aditiva dada a seguir [Almeida 2005]:

$$U(X,Y)=k_1U(X)+k_2U(Y)$$

Onde: X e Y são duas conseqüências (atributos) para uma alternativa de decisão;

k_1 e k_2 são constantes de escala (preferências); $k_1 + k_2=1$.

Enquanto a função utilidade aditiva pode ser aplicada para casos de dois atributos, à medida que o número de atributos aumenta, maior é a complexidade das funções que devem ser aplicadas. Para um número relativamente grande de atributos, o uso de estruturas hierárquicas para os atributos pode facilitar o processo de avaliação [Gomes *et al.* 2002]. A

função multiplicativa também pode ser utilizada e cuja forma geral é vista com relativa simplicidade para o caso de três atributos como dada a seguir. Para facilitar a fórmula usaremos U_i para significar $U_i(x_i)$ [Russel & Norvig 2003]:

$$U=k_1U_1+k_2U_2+k_3U_3+k_1k_2U_1U_2+k_2k_3U_2U_3+k_3k_1U_3U_1+k_1k_2k_3U_1U_2U_3$$

Notações sobre preferências: Utilizamos as seguintes notações para descrever as preferências dos agentes pelas alternativas A e B:

$A \succ B$: A é preferível a B;

$A \sim B$: O agente é indiferente entre A e B;

$A \succsim B$: O agente prefere A a B ou é indiferente entre eles.

Princípio da utilidade: Se um agente possui preferências entre A e B, então existe uma função de valor real U que opera nos estados tal que $\mathbf{U(A) \succ U(B)}$ se, e somente se, A é preferível a B e $\mathbf{U(A) = U(B)}$, se e somente se, o agente é indiferente entre A e B.

Risco e incerteza: A diferença básica entre os conceitos de risco e de incerteza é que no primeiro a distribuição do resultado num grupo de casos é conhecida (por meio de cálculo prévio ou de estatísticas de experiências passadas), enquanto no segundo isso não ocorre, em geral devido ao fato de que é impossível formar um grupo de casos, porque a situação que se enfrenta é, em alto grau, singular [Knight 1921].

6.2.2 Conceito e uso da teoria da decisão

A teoria da decisão é um arcabouço matemático para determinar a melhor ação de uma descrição de um problema de decisão, fornecendo recomendações ótimas [Holtzman 1989]. A teoria da decisão é normativa, ou seja, descreve como um agente racional deve agir [Russel & Norvig 2003]. A idéia fundamental da teoria da decisão é que um agente é racional se ele escolhe a ação que leva à mais alta utilidade esperada dentre os vários resultados possíveis da ação.

A teoria da decisão combina as preferências, expressas por utilidades, com probabilidades. A teoria da utilidade diz que cada alternativa tem um grau de utilidade para um agente e que o agente irá preferir alternativas com maior utilidade. A teoria da probabilidade designa para cada alternativa um valor numérico de crença entre 0 e 1.

A teoria da decisão auxilia tomadores de decisão a escolher entre um conjunto de alternativas à luz de suas possíveis conseqüências e pode aplicar condições de certeza, risco ou incerteza [Gomes *et al.* 2002].

Na decisão sob certeza, cada alternativa leva a uma e somente uma conseqüência. Uma escolha entre alternativas é equivalente a uma escolha entre conseqüências.

Na decisão sob risco, cada alternativa tem uma de várias possíveis conseqüências e a probabilidade de ocorrência para cada conseqüência é conhecida. Além disso, cada alternativa é associada a uma distribuição de probabilidade. Quando as distribuições de probabilidade são desconhecidas, diz-se que a decisão é tomada sob incerteza. A teoria da decisão para condições de risco é baseada no conceito de utilidade. As preferências dos tomadores de decisão para conseqüências mutuamente exclusivas de uma alternativa são descritas por uma função utilidade, que permite o cálculo da utilidade esperada para cada alternativa. A alternativa com a utilidade esperada mais alta é considerada a alternativa preferida.

Em uma situação de decisão sob incerteza, as preferências dos tomadores de decisão são simuladas por um único atributo ou por uma função de valor multiatributo que introduz ordenação no conjunto de conseqüências e, portanto, ordena as alternativas. Para o caso da incerteza, a teoria da decisão oferece duas abordagens principais. A primeira explora critérios de escolha desenvolvida sob o contexto da teoria dos jogos, como por exemplo, a regra max-min, onde se escolhe a alternativa cuja pior conseqüência possível seja melhor que (ou igual) a melhor conseqüência possível de qualquer outra alternativa. A segunda abordagem é reduzir o caso incerto para o caso de risco ao usar probabilidades subjetivas, baseadas nas designações de um especialista ou em uma análise de decisões prévias feitas em circunstâncias similares.

O modelo aqui apresentado corresponde à decisão sob risco, pois usamos probabilidades construídas pelas experiências passadas dos agentes em selecionar recursos no ambiente. Uma vez que as probabilidades e utilidades das possíveis conseqüências são especificadas, a utilidade de uma loteria envolvendo essas conseqüências é completamente determinada.

6.3 Formalização do problema

Esta seção formaliza o problema tratado na tese agrupando todas as variáveis apresentadas nos capítulos anteriores que influenciam a seleção do recurso. O Problema de Seleção do Melhor Recurso em Grade computacional (PSMRG) tem a seguinte formalização geral: seja um modelo de sistema de escolha de recurso composto por *n* agentes que representam recursos, $sa \in$ **SA,** um agente, $ua \in$ **UA,** que representa um usuário e onde PSMRG é definido pela tupla ($U,X,Y,$**R**,**J**,tj,**H**,$C,v,pa,$**N**,$p(s)_p$,**P**,z_p,q_p,**Pref**)[13], onde: U representa o conjunto de usuários da grade computacional; X representa o conjunto de domínios pertencentes a uma organização virtual; Y representa o conjunto de recursos; **R** representa requisitos, **R**={CPU, RAM, quantidade de processadores}; **J** representa um *job* com descrição **J**={nome da aplicação, atributos, dado, **R**}; tj representa o horário solicitado pelo usuário para executar o *job* **J**; **H** representa o intervalo de tempo com início em a e término em b permitido para $u \in$ U executar em $y \in$ Y; C representa o conjunto de casos passados de execução de *jobs* **J** em um recurso $y \in$ Y; v representa a carga máxima aceita pelo recurso $y \in$ Y; pa representa a predição de tempo de execução de **J** em y; **N** representa os parâmetros negociáveis entre u e y para provimento de **J**, **N**={pr,g,m}onde pr representa o preço mínimo aceitável para executar **J** em y, g representa a garantia aplicada através de desconto sobre o preço pr no caso de erro da predição pa e m representa a métrica utilizada como referência para aplicar a garantia g; p(s) representa a taxa de erro na predição para um grau de similaridade s entre **J** e casos passados $\in C$; **P** representa restrições de acesso para autenticação do usuário no recurso y, **P**={$u \in U$, $x \in X$, **H**}; z representa o grau de confiabilidade da máquina y para execução de j no tempo pa; q representa a probabilidade de alteração na predição pa; **Pref** representa graus de preferência do usuário para pr e pa, **Pref**={δ_1, δ_2}. O problema consiste em, considerando os dados desse modelo, escolher a melhor máquina y para executar o *job* **J** de acordo com a preferência multiatributo do usuário.

[13] Letras maiúsculas em itálico denotam conjunto de elementos (e.g., u $\in U$). Letras maiúsculas em negrito representam tuplas, (e.g., **N**={ pr,g,m}). Letras minúsculas em itálico representam variáveis. Funções de probabilidade são subscritadas com $_p$.

6.4 Modelo de decisão para seleção de recursos

Esta seção apresenta um modelo para seleção de recursos baseado na teoria da decisão utilizando os dados advindos dos modelos de predição [Nassif *et al.* 2005b], descritos no Capítulo 4, e de restrições de acesso, descritos no Capítulo 5. O modelo de predição compara a descrição do *job* submetido ao *middleware* MASK com casos passados e calcula a predição do novo *job* baseando-se no tempo de execução real dos casos mais similares. O modelo de verificação de restrições de acesso verifica a autorização do usuário, a carga de trabalho permitida e as condições de provimento do serviço. Os dados calculados e negociados nesses modelos serão utilizados como entrada para o modelo de decisão descrito nas próximas seções.

6.4.1 Integração do modelo de decisão ao sistema multiagente

As variáveis do problema PSMRG estão representadas na Figura 6-1. Tal figura apresenta a dinâmica do fluxo de informações entre os agentes que representam usuários e recursos através do processo de negociação envolvendo os modelos de predição, políticas e seleção. O modelo de seleção é descrito em detalhes nas seções subseqüentes.

Conforme a formalização do problema apresentada na seção 6.3, descreveremos a seguir a dinâmica dos fluxos associados à Figura 6-1 para o cenário onde um usuário deseja submeter um *job* no ambiente de grade computacional por meio do *middleware* MASK que então seleciona o melhor recurso para executar o *job*. Inicialmente o usuário $u \in \mathbf{U}$ configura o agente $ua \in \mathbf{UA}$ fornecendo as seguintes informações para a interface do *middleware* (fluxo 1): os dados sobre o acesso ao recurso (contendo $u \in \mathbf{U}$, $x \in \mathbf{X}$, tj**);** a descrição do *job* **J;** os requisitos **R;** os parâmetros negociáveis para o provimento do serviço, *pr* e *g*; e as preferências do usuário δ_1 e $\delta_2 \in$ **Pref.** O agente *ua* utiliza **R** para fazer um filtro de máquinas que atendem aos requisitos mínimos (fluxo 2). Esse procedimento de filtragem inicial é descrito no capítulo 3. Dessa forma, irão participar da negociação $Y - k$ máquinas (fluxo 3a), sendo k o número de máquinas que não atendem aos requisitos **R.**

No processo de negociação, o agente *ua* envia para cada agente *sa* que atende aos requisitos **R**, uma proposta contendo $<u \in \mathbf{U}, x \in \mathbf{X},$ *tj*, *pr, g*, **J>** (fluxo 3b). O agente *sa* divide a proposta em dois sub-grupos de informações. Um sub-grupo de informação

contendo **J** é enviado para o módulo de predição (fluxo 5a) e outro sub-grupo de informação contendo *u, x, tj , pr* e *g* é enviado para o módulo de verificação de políticas (fluxo 5b).

No módulo de predição, **J** é considerado um novo caso de execução, denominado NC. NC é comparado com casos passados existentes na base de casos do recurso y, $\mathbf{C_y}$. Os casos $\mathbf{C_y}$ com maior similaridade em relação a NC são escolhidos para servir de referência para o cálculo de *pa* (fluxo 6a).

No módulo de verificação de políticas, *u, x, tj, pr* e *g* são confrontados contra as políticas locais de *y*, podendo resultar em uma permissão (com eventual elaboração de contraproposta para *pr* e *g)* ou uma rejeição do acesso (fluxo 6b).

Os valores de *pa*, *pr* e *g* são repassados para o módulo de decisão do agente do servidor *sa* e também retornam para o agente do usuário *ua* dentro do processo de negociação (fluxo 7). O agente *ua* verifica se a contraproposta atende aos limites definidos pelo usuário e processa o modelo de decisão com o conjunto de máquinas M= *y* - *k* – *w* (fluxo 8), sendo *w* o conjunto de máquinas cujas restrições impedem o acesso à requisição enviada pelo *ua*.

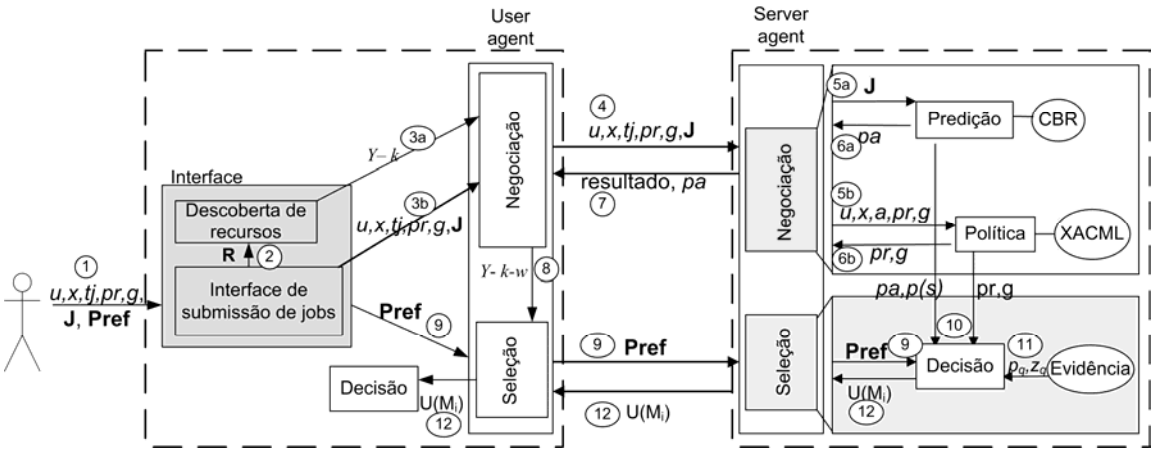

Figura 6-1: Estrutura de interação no MASK conforme formulação do PSMRG

No processo de seleção, o agente *ua* envia para cada agente *sa* as preferências do usuário, δ_1 e δ_2, com relação a preço e tempo de execução, respectivamente (fluxo 9). Evidências de execuções passadas refletem a confiabilidade do servidor através da probabilidade *z* e constroem uma distribuição de probabilidade *p(s)* que reflete a taxa de erro na predição para um grau de similaridade *s* com casos passados. O módulo de decisão,

baseando-se nas preferências δ_1 e δ_2 (fluxo 9), nas informações *pa*, *pr* e *g* recebidas dos módulos de predição e política (fluxo10) e nas probabilidades q_p, z_p e $p(s)_p$ (fluxo11), calcula a utilidade $U(M_i)$ do recurso *y*. O agente *sa* retorna essa informação para o agente *ua* (fluxo 12). O agente *ua* escolhe a máquina que maximiza a utilidade do usuário através da seleção da máquina com maior valor de $U(M_i)$.

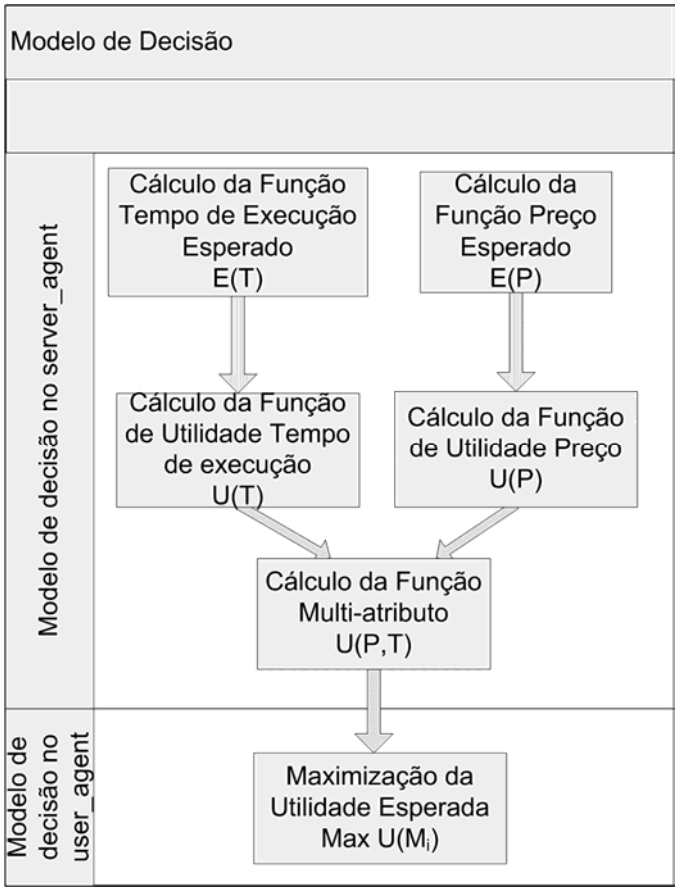

Figura 6-2: Fluxograma multifuncional do modelo de decisão

O fluxo dos cálculos envolvendo as variáveis de entrada (*pa*, *pr*, *g*, *p*, *z* e **Pref**) e de saída ($U(M_i)$) do modelo de decisão é apresentado na Figura 6-2. O modelo de decisão é implementado nos *server_agents* e no *user_agent* de forma complementar para um conjunto de M máquinas candidatas para executar um *job* **J**. Para cada máquina candidata M_i, existem funções que precisam ser calculadas pelos agentes *sa* que as representam. Conforme fluxograma da Figura 6-2, os cálculos das funções são realizados em três etapas. A primeira etapa calcula as funções esperadas para P (preço) e T (tempo de execução). A segunda etapa calcula as utilidades unidimensionais relativas à P e T, respectivamente,

U(P) e U(T). Posteriormente é aplicada a terceira etapa a fim de obter a função utilidade multiatributo U(P,T). Todos os cálculos do fluxograma da Figura 6-2 são apresentados nas seções 6.4.2 a 6.4.6.

6.4.2 Maximização da utilidade esperada

O decisor para o PSMRG é um agente *ua* que representa um usuário *u* para selecionar um recursos *y* para executar um *job* **J**. Dada uma determinada representação da realidade, chamada de estado da natureza, θ, e a alternativa de execução do *job* na máquina M_i, a utilidade $U(M_i, \theta)$ deverá ser encontrada. Se os parâmetros relativos ao estado da natureza são conhecidos, o decisor deverá escolher a alternativa M_i que dê valor máximo para a função utilidade $U(M_i, \theta)$. O problema é resolvido pela maximização da utilidade esperada, dada pela equação abaixo.

$$\max_{M_i} \; EU(M_i)|\theta \qquad (1)$$

O agente *ua* recebe *i* funções utilidade $U(M_i)$ calculadas de forma distribuída por *i* agentes *sa* e enviadas por cada máquina proponente. A função utilidade esperada para cada máquina M_i é dada por:

$$EU(M_i,\theta)=\sum_P \sum_T U(P,T)Pr(P,T|M_i,\theta) \qquad (2)$$

6.4.3 Função multiatributo

Cada agente *sa* calcula uma função multiatributo que associa a preferência do usuário para preço ou para tempo de execução, onde $U(P) \succ U(T), U(P) \prec U(T)$ ou $U(P) \sim U(T)$, e considerando-se a proporção entre tais preferências dadas por δ_1 e δ_2. A função utilidade multiatributo U(P,T) de cada máquina é calculada pela equação a seguir.

$$U(P,T)=\delta_1 * U(P)+\delta_2 * U(T) \qquad (3)$$

Onde:

δ_1 é a preferência dada pelo usuário para P, tal que $\delta_1 \in [0,1] \wedge \delta_1 + \delta_2 = 1$;

δ_2 é a preferência dada pelo usuário para T, tal que $\delta_2 \in [0,1] \wedge \delta_1 + \delta_2 = 1$;

U(P) é o valor resultante da computação da função de utilidade de P para a máquina M_i;

U(T) é o valor resultante da computação da função de utilidade de T para a máquina M_i;

Com esse modelo é possível atender às expectativas do usuário para a seguinte situação: "prefiro executar o *job* em 10s ao custo de 30 a executar o *job* em 20s ao custo de 20".

A função de utilidade associada ao preço, U(P), e ao tempo de execução, U(T), bem como a função utilidade multiatributo que representa a preferência entre P e T, U(P,T), são calculadas em cada *sa* participante da negociação cuja contraproposta foi considerada aceitável pelo *ua* durante o processo de negociação. Apresentamos a seguir os cálculos de U(P), U(T), P e T.

6.4.4 Função de utilidade preço

Na função U(P), quanto maiores os valores de P, mais indesejável se torna essa alternativa. A função exponencial é comumente utilizada para situações como essa.

$$U(P)=e^{-\sigma P} \qquad (4)$$

Onde: P é o valor da função preço (seção 6.4.6) e

σ é o valor máximo considerado na escala (na implementação *pr*=[0..100], logo σ=1/100).

6.4.5 Função de utilidade tempo de execução

Na função U(T), quanto maiores os valores de T, mais indesejável se torna essa alternativa. A função exponencial é comumente utilizada para situações como essa.

$$U(T)=e^{-\tau T} \qquad (5)$$

Onde: T é o valor da função tempo de execução (seção 6.4.7) representando a predição em milissegundos e

τ é o valor máximo considerado na escala (para predições com duração até 14h, τ =1/50.400.000)

6.4.6 Função preço esperado

O preço é negociado entre *ua* e *sa* no modelo de negociação e informado ao modelo de decisão através da integração já anteriormente explicitada na seção 6.4.1.

Dois possíveis valores existem para P, sendo eles p_1 e p_2, relacionados com as situações de sucesso ou de insucesso na predição respectivamente, com probabilidade z_p associada ao índice de confiabilidade do recurso.

A função P equivale ao valor esperado de P, E(P), conforme descrito nas equações (6),(7) e (8) a seguir:

$$P=E(P)=\sum_i p_i \, Pr(p_i) \qquad (6)$$

$$P=(z_p * p_1)+(1-z_p)p_2 \qquad (7)$$

$$P=(z_p * p_1)+(1-z_p)(p_1-g*p_1) \qquad (8)$$

Onde:

p_i pode assumir os valores p_1 e p_2 descritos a seguir;

p_1 = preço aplicado no caso de sucesso da predição;

$Pr(p_i)$ = probabilidade de ocorrência de p_i ;

p_2 = preço aplicado no caso de insucesso da predição, onde $p_2= p_1 -g*p_1$;

g = desconto no valor do preço p_1 no caso de insucesso da predição;

z_p = grau de confiança do provimento do serviço pela máquina M_i, onde z_p está associado às evidências coletadas pelos agentes *sa*. z_p representa a relação em M_i entre o total de execuções de *jobs* passados que executaram em tempo igual ou menor que o tempo previsto e o total de execuções passadas. A probabilidade z_p é aplicada ao preço negociado p_1.

As representações das evidências passadas ajudam a calcular as probabilidades usadas nessa função Preço. Cada servidor tem uma estrutura XML, representada na Figura 6-3, que armazena o número de vezes que o servidor foi selecionado (Num*Job*) e quantas negociações ele participou (NumNegot). Ao final de cada execução, o *sa* atualiza as informações dessa estrutura, informando se o tempo real de execução foi maior (PredMaior), menor (PredMenor), ou igual (PredIgual) ao tempo previsto. Os agentes utilizam esses dados como evidências.

```
<DataServer>
    <Negotiation>
        <PredMaior>3</PredMaior>
        <PredIgual>6</PredIgual>
        <PredMenor>4</PredMenor>
        <NumJob>13</NumJob>
        <NumNegot>32</NumNegot>
    </Negotiation>
</DataServer>
```

Figura 6-3: Registro do resultado das execuções dos *jobs* no recurso para representar a confiabilidade do recurso na predição.

6.4.7 Função tempo de execução esperado

O tempo de execução é calculado no modelo de predição conforme descrito no capítulo 4. O tempo de execução da máquina M_i é informado ao modelo de decisão através da integração já anteriormente explicitada na seção 6.4.1.

Dois possíveis valores existem para T, sendo eles t_1 e t_2, relacionados com as situações de sucesso ou de insucesso na predição respectivamente, com probabilidade q_p associada ao índice de acerto na predição dada as evidências coletadas por *sa*.

A função T equivale ao valor esperado de T, E(T), conforme descrito nas equações (9), (10) e (11) a seguir:

$$T=E(T)=\sum_i t_i \Pr(t_i) \qquad (9)$$

$$T=(t_1 * q_p)+t_2(1-q_p) \qquad (10)$$

$$T=(t_1 * q_p)+(t_1-p(s)*t_1)(1-q_p) \qquad (11)$$

Onde:

t_i pode assumir os valores t_1 e t_2 descritos a seguir;

t_1 = tempo de execução no caso de sucesso da predição;

$\Pr(t_i)$ = probabilidade de ocorrência de t_i ;

t_2 = tempo de execução no caso de insucesso da predição, onde $t_2 = t_1 - p(s) * t_1$;

$p(s)$ = é taxa de erro na predição para um grau de similaridade s;

q_p = grau de acerto na predição, onde q_p está associado a evidências coletadas pelos agentes *sa*. q_p representa a relação em M_i entre o total de execuções de *jobs* passados que executaram em tempo igual ao tempo previsto e o total de execuções passadas. A probabilidade q_p é utilizada para ajuste do tempo de predição t_1.

O grau de acerto na predição, q_p, é obtido a partir de dados estatísticos de execuções passadas conforme representação dada na Figura 6-4. A taxa de erro na predição, *p(s)* significa o quanto a predição do tempo de execução estava errada com relação ao tempo real de execução. *p(s)* é utilizada para o cálculo de t_2.

Representações de evidências passadas auxiliam o cálculo de probabilidades usadas nessa função tempo de execução. Cada servidor possui uma estrutura XML que armazena dados estatísticos e constrói uma distribuição de probabilidade empírica que associa o erro de predição ao grau de similaridade alcançado. Ao final de cada execução o *sa* atualiza

informações nessa estrutura no que se refere à distribuição de probabilidades. A Figura 6-4 apresenta essa distribuição de probabilidade definida em formato XML. Com tal representação é possível calcular a média de erros de predição com relação ao grau de similaridade alcançado. Por exemplo, conforme a Figura 6-4, nos casos com grau de similaridade global igual a 2,7 (90% de similaridade entre casos passados e casos novos), a soma de cada erro de predição (Total) dividido pelo número de casos (Count) resulta em uma média de erro de predição (AVG). Portanto, a média AVG representa o erro de predição esperado para casos que obtiveram mesmo grau de similaridade.

```
<Case>
   <SimilarityDegree>2,7</SimilarityDegree>
      <Total>0,9</Total>          (Soma de cada erro de predição)
      <Count>10</Count>           (No. total de casos)
      <AVG>0,09</AVG>             (Média de erro de predição)
</Case>
<Case>
   <SimilarityDegree>2,4</SimilarityDegree>
      <Total>1,76</Total>         (Soma de cada erro de predição)
      <Count>8</Count>            (No. total de casos)
      <AVG>0,22</AVG>             (Média de erro de predição)
</Case>
```

Figura 6-4: Registros de distribuição de probabilidade empírica para grau de similaridade com casos passados e erro na predição

6.4.8 O processo de seleção no MASK

Considerando a implementação completa do MASK, incluindo os modelos de predição e verificação de restrição de acesso envolvidos na seleção de recursos, relatamos abaixo o processo de seleção no MASK para diferentes situações da base de casos. Considerando que cada base de casos possui alguns casos previamente executados em cada recurso, tem-se;

1) Um novo *job* é submetido ao *broker* MASK. Caso não seja possível encontrar um caso similar em 90% (valor esse discutido na seção 4.5.4) com o novo caso, então, como nem todas as máquinas possuem predição, o *job* é submetido para a máquina com menor preço.

2) Enquanto existir alguma máquina que não possua predição para o *job*, a cada nova execução do *job*, o MASK irá escolher uma máquina que ainda não possui predição. Para acelerar o processo de aprendizado das máquinas, é possível, opcionalmente, que os

agentes dos servidores identifiquem que um novo *job* sem similaridade aceitável foi submetido ao conjunto de máquinas e o executem nas máquinas que eles representam para incorporá-lo em suas respectivas bases de casos. Essas execuções devem ser feitas em horários previamente estabelecidos na configuração do agente.

3) Após o processo de aprendizado, todas as máquinas passam a apresentar em sua base de casos, o novo *job* submetido.

4) Nas subseqüentes submissões do mesmo *job* ou um *job* similar em 90%, é possível calcular a predição de execução do mesmo em todas as máquinas do ambiente. Essa construção da base de casos garante que a melhor máquina do ambiente, considerando os critérios estabelecidos, seja selecionada.

6.5 Validação do modelo de decisão

Esta seção apresenta experimentos que demonstram a funcionalidade e o desempenho da nossa abordagem de seleção centrada na preferência multiatributo do usuário e implementada de forma paralela e distribuída[14].

Os objetivos dos experimentos para validação do modelo de decisão são: 1) a verificação funcional para (a) um objetivo único, (b) influência da preferência do usuário na seleção e (c) influência da confiabilidade do servidor na seleção; 2) a verificação de desempenho na seleção de recursos.

Para atender aos objetivos 1(a) e 2 utilizamos como estratégia a comparação da nossa solução com outras abordagens. A métrica utilizada para atender ao objetivo 1(a) é o tempo total que as abordagens demandam para computar um mesmo conjunto de *jobs*. Essa métrica indica que uma abordagem deve selecionar os melhores recursos que executem mais eficientemente quando o desempenho é o único critério preferencial na seleção.

A métrica utilizada para atender ao objetivo 2 é a média do tempo de seleção, ou seja, em quanto tempo cada abordagem seleciona um recurso. Essa métrica indica o desempenho da abordagem para realizar seleções, fator importante para seleções em tempo real.

Para contrapor o cenário de seleção para apenas um atributo preferencial (tal como apresentado no ambiente de validação do objetivo 1(a), mostramos que a preferência do

[14] O sistema multiagente MASK está completamente implementado, totalizando 60 programas escritos em Java utilizando Jade (ver detalhes da implementação no apêndice B1). Diferentes experimentos foram realizados para validar os seus modelos de predição e de políticas nos capítulos 4 e 5.

usuário por mais de um atributo pode alterar significativamente a seleção de recursos. Portanto, a estratégia utilizada para validar o objetivo 1(b) foi mostrar diferentes escalas de preferência entre atributos de custo e desempenho para um mesmo *job*.

A estratégia utilizada para validar o objetivo 1(c) foi apresentar situações onde a confiabilidade do servidor (interpretadas como as probabilidades da predição ser realizada no tempo estimado e do preço não ser alterado) têm influência na seleção do recurso. Para isso apresentamos uma aplicação numérica para as fórmulas apresentadas no modelo de decisão destacando a influência da utilização das probabilidades e mostramos as situações quando as probabilidades podem ser influentes em um processo de seleção.

6.5.1 Configuração do ambiente

A seleção no MASK foi comparada com a seleção no Condor, um sistema especializado de gerenciamento de carga de trabalho para *jobs* de computação intensiva. O Condor fornece um mecanismo de enfileiramento, política de escalonamento, esquema de prioridade, monitoração e gerenciamento de recursos. Os usuários submetem seus *jobs* paralelos ou seriais para o Condor, que então os enfileira e escolhe quando e onde rodá-los, monitorando o progresso da execução [Condor 2005].

Para validar a melhor seleção entre as duas abordagens, foram utilizadas cinco máquinas neste experimento. As máquinas possuem diferentes capacidades de desempenho, conforme mostra a Tabela 6-1.

Tabela 6-1: Características das máquinas no experimento do modelo de decisão

Máquina	Sistema Operacional	Processador	Memória (Mbytes)	Mips	Kflops
Itapeva	Linux version 2.4.22	AMD Athlon™ XP 2500+	512	2277	746377
Sajama	Linux version 2.6.14	AMD Athlon™ 64 Processor 3200+	1024	2846	762963
Ibituruna	Linux version 2.4.22	AMD Athlon™ XP 2000+	512	2189	666667
Palermo	Linux version 2.6.14	Intel® Pentium® 4 CPU 3.00GH	1024	2108	719023
Bonete	Linux version 2.6.11	AMD Sempron™ Processor 3000+	2048	2475	726631

Nos experimentos realizados, utilizamos um conjunto de *jobs* para três diferentes aplicações: BLAST, HMMER e programas genéricos escritos na linguagem C (maiores detalhes das aplicações no Apêndice B). O uso de diferentes aplicações no experimento tem

como objetivo mostrar a capacidade do MASK ser utilizado por diferentes aplicações no processo de seleção. Conforme definições anteriores, o nosso grupo de aplicações alvo são aplicações *batch* com um conjunto de argumentos.

O Condor requer que sejam executados dois processos *daemon*[15] em cada máquina cliente participante, denominados *startd* (que tem a função de executar *jobs*) *e schedd* (que tem a função de submeter *jobs*). Na máquina que exerce a função de Gerenciamento Central (CGC) do conjunto de máquinas denominado *pool* são executados quatro processos *daemon*: *startd, schedd, negotiator* (que tem a função de verificar requisitos de máquinas e *jobs*), e *collector* (que tem a função de coletar informações do nodo cliente). A máquina com função de Gerenciamento Central realiza o processo de combinação de requisitos (descrito no capítulo 3 como *matchmaking*) e funciona como repositório centralizado de informações.

O MASK requer que os agentes estejam associados a uma plataforma[16] (ver detalhes da plataforma de agentes no Apêndice B1). Para cada máquina participante é associado um *server_agent* à plataforma e para cada usuário participante é associado um *user_agent* à plataforma.

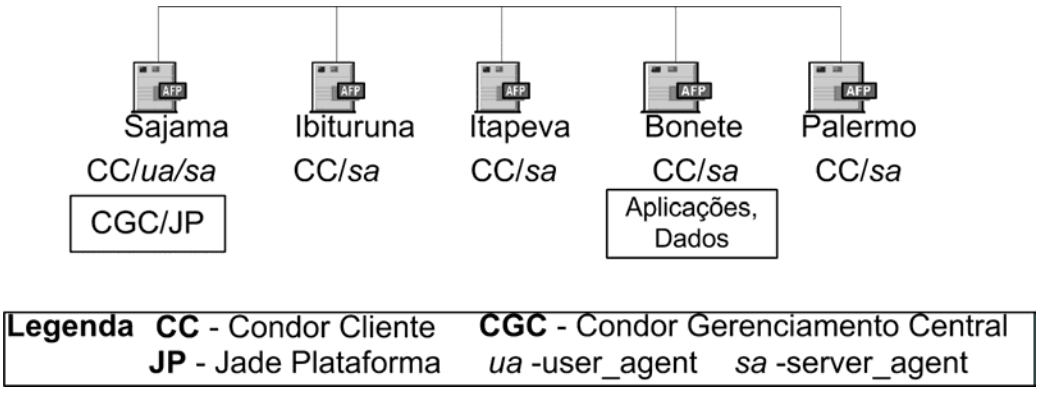

Figura 6-5: Configuração do ambiente de teste

[15] Um processo *daemon* é usado para denotar um processo do sistema que roda em segundo plano de forma ininterrupta.

[16] Cada instância do Jade *run-time* é chamado *container*. O conjunto de todos os *containers* é chamado de plataforma.

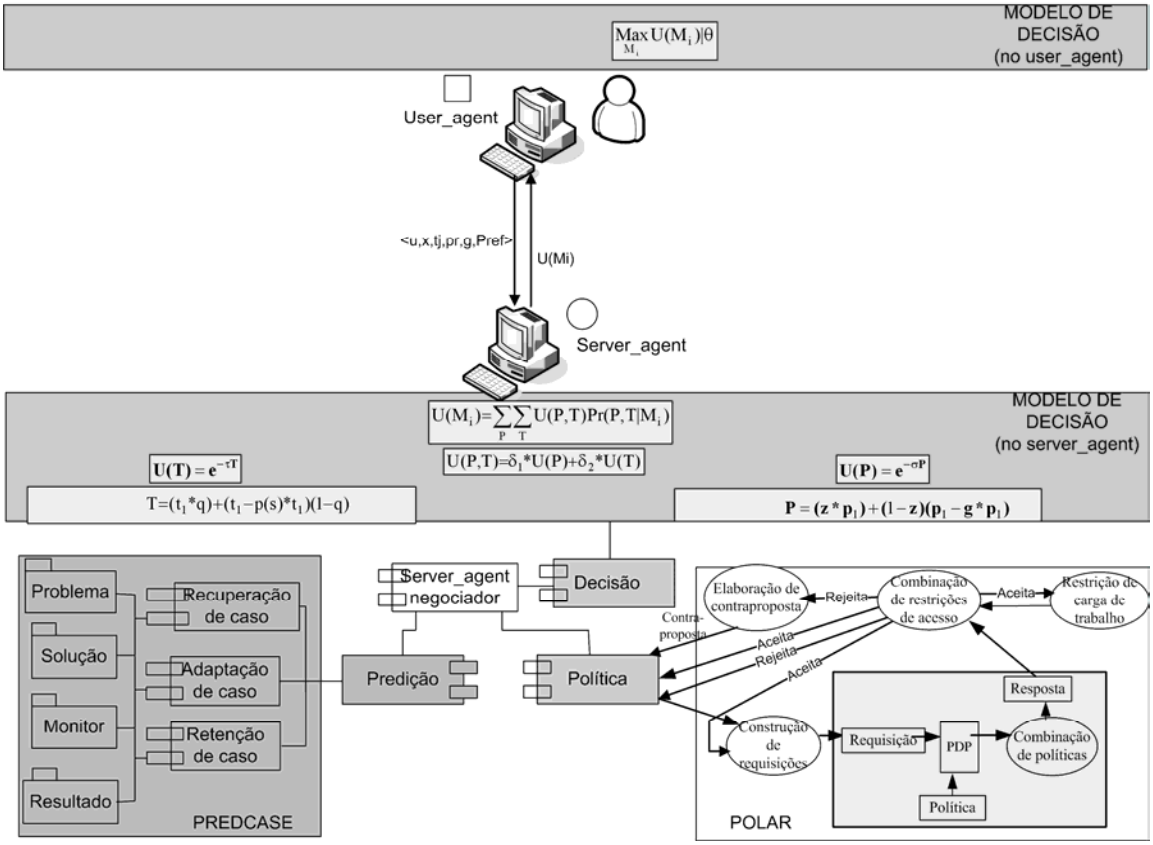

Figura 6-6: Módulos do MASK integrados e disponíveis no experimento

A Figura 6-5 apresenta a configuração das máquinas para executar o MASK e o Condor. A máquina Bonete foi utilizada como servidora de arquivos de aplicações e de dados para ambas as abordagens. A máquina Sajama foi definida como o Gerenciamento Central do conjunto de máquinas rodando Condor e também definida como a plataforma de agentes do MASK. Todas as submissões de *jobs* para Condor e para MASK foram feitas a partir da máquina Sajama.

Cada máquina do experimento tem um *server_agent* que a representa. Cada *server_agent* tem uma estrutura contendo os modelos de predição (PredCase), de verificação de restrições de acesso (POLAR) e de decisão. Cada usuário é representado por um agente usuário (*user_agent*). Um *user_agent* envia várias requisições de *jobs* para vários *server_agents*. Os *server_agents* fazem a predição do tempo de execução do *job*, verificam o acesso ao recurso e calculam a utilidade associada à máquina. O *user_agent* escolhe o recurso que maximiza a preferência do usuário. Essa organização está ilustrada na Figura 6-6.

6.5.2 Verificação funcional para um único atributo preferencial

Esta seção objetiva analisar a seleção realizada pelos sistemas Condor e MASK quando o usuário possui preferência por um determinado atributo. Neste experimento, o atributo preferencial é dado pelo desempenho da máquina, ou seja, o sistema deverá selecionar a máquina que ofereça o melhor desempenho na execução de um *job*.

Configuração do ambiente

O ambiente de teste para o experimento desta seção foi configurado conforme anteriormente detalhado na seção 6.5.1. Informações adicionais para o experimento específico desta seção são apresentadas a seguir.

Para expressar preferências, o Condor permite o uso de um atributo denominado "*rank*" dentro da configuração do *job*. Para o conjunto dos *jobs*, alteramos esse atributo para três configurações diferentes (denominadas aqui como Condor1, Condor2 e Condor3) dentro da especificação do *job* submetido ao Condor. A configuração Condor1 não faz uso do atributo *rank*; na configuração Condor2, *rank*=Mips+*memory size*; e na configuração Condor3, *rank*=Mips+Kflops+*memory size*. Vale ressaltar que pela instalação original do Condor, esse é o conjunto de variáveis associadas ao recurso que podem expressar o desempenho do processador[17]. Uma variável que apresente a velocidade do processador em GHz, por exemplo, só poderia ser obtida se o Condor fosse adequadamente instrumentalizado.

Para que o MASK expresse a preferência do usuário pela máquina mais rápida, deve-se atribuir na interface do sistema a preferência por tempo de execução, o que equivale a $\delta_2 = 1$ e $\delta_1 = 0$ no modelo de decisão (seção 6.4.3).

Todas as bases de casos do modelo PredCase das máquinas envolvidas nos experimentos possuíam um conjunto de *jobs* anteriormente executados. Dessa forma, o MASK era capaz de realizar predições para os novos *jobs* submetidos no experimento apresentado nesta seção.

Todas as máquinas do experimento possuem todos os requisitos necessários para o conjunto de *jobs* submetidos, evitando-se que a escolha seja em função de um desses parâmetros delimitadores, tais como o sistema operacional ou o número de processadores.

Execução e análise

Um conjunto de 49 *jobs* foi utilizado, construídos a partir da combinação de comandos das três aplicações aqui utilizadas (BLAST, HMMER e programas C genéricos)[18]. Um único usuário submeteu os *jobs* a partir da máquina Sajama. Os *jobs* foram submetidos sequencialmente, onde a submissão de um novo *job* se dava após o término do *job* anterior. Dessa forma a carga de trabalho do ambiente foi controlada e foi possível analisar a escolha nas mesmas condições para o Condor e MASK.

A Figura 6-7 mostra o tempo de computação total para o conjunto de *jobs* submetidos para o *broker* MASK e para o Condor em busca da máquina que apresenta o melhor desempenho na execução de cada *job*.

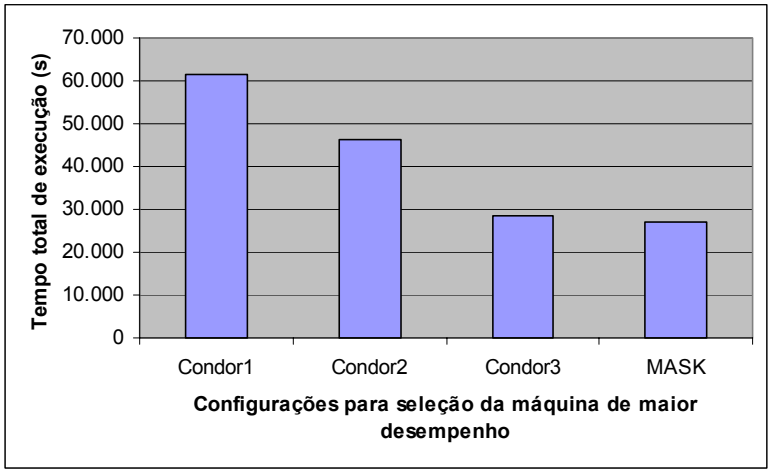

Figura 6-7: Tempo total de execução do conjunto de *jobs* submetidos para o MASK e o Condor

Para a configuração Condor1, o Condor escolhe sistematicamente a máquina Ibituruna; para a configuração Condor2, o Condor escolhe sempre a máquina Bonete e para a configuração Condor3, o sistema Condor sempre escolhe a máquina Sajama. O MASK escolheu em 95% dos casos a máquina Palermo, em 1% dos casos a máquina Sajama e em 4% dos casos a máquina Bonete. O MASK se baseou nos tempos de execução passados dos *jobs* nessas máquinas e observamos que a máquina Palermo foi a mais demandada por ter a maior velocidade do processador, que nesse experimento foi a métrica mais eficiente para analisar desempenho do que outras métricas tais como Mips e Kflops. A máquina Sajama

[17] O conjunto de variáveis de uma máquina no Condor pode ser obtido utilizando-se o comando "condor_status –long".

foi escolhida nos casos onde os valores dos tempos de execução de um caso na Sajama e na Palermo eram muito próximos, situação similar à verificada para os experimentos ilustrados na Tabela 4-8 para as máquinas Itapeva e Caridade. Observamos ainda que a máquina Bonete foi a escolhida nos casos de curta duração, pelo fato de ser ela a servidora de arquivos de aplicações e de dados.

Na situação mostrada acima, onde a preferência do usuário é totalmente voltada para apenas um atributo, neste caso, o desempenho, tanto o MASK quanto o Condor podem expressar adequadamente essa preferência. No entanto, Condor escolhe a máquina pelas suas características computacionais e MASK escolhe a máquina que executou casos similares mais rapidamente, de acordo com o modelo de predição descrito no capítulo 4. Em algumas situações, mesmo usando critérios diferentes, tanto MASK quanto Condor podem eventualmente escolher a mesma máquina. Conforme diversos trabalhos na literatura, a escolha da máquina mais rápida dada pelas suas características computacionais não resulta na melhor seleção para reduzir o tempo de execução [Jang *et al.* 2004] [Gibbons 1997]. O experimento aqui conduzido reafirma essa observação. Pela Figura 6-7, para o agrupamento de máquinas disponíveis, o MASK executou o conjunto de *jobs* em um tempo total que foi 8% mais rápido que a abordagem Condor, pois se baseou em predição para escolher a máquina mais rápida do ambiente.

Para a situação apresentada acima, o usuário tinha preferência por apenas um atributo, no caso, o desempenho. Em uma situação onde o usuário deseja expressar preferência por dois atributos distintos, o processo de seleção passa a ser mais complicado. O Condor não expressa adequadamente preferência por dois ou mais atributos. Os problemas para formular uma equação adequada para incluir no atributo *rank* do Condor são os seguintes: 1) os atributos podem possuir escalas diferentes e devem ser balanceados. Por exemplo, o preço varia entre 0 e 100 e o desempenho das máquinas entre 0 e 790.000 Kflops; 2) as características podem ter sentidos diferentes. Por exemplo, para o preço, quanto menor o valor, melhor para o usuário; e para o desempenho, quanto maior o valor em Kflops, melhor para o usuário; 3) não é totalmente garantido que a decisão realizada é a melhor e para isso é preciso decidir considerando riscos, que é um modelo que trabalha com probabilidades. Os exemplos das seções 6.5.3 e 6.5.4 mostram que o modelo de seleção

[18] No capítulo 4 ressaltamos também os motivos do quantitativo de comandos obtidos.

adotado pelo MASK supera essas dificuldades, pois permite a seleção do recurso com preferências por atributos diferentes utilizando função utilidade multiatributo e supera a incerteza utilizando probabilidades construídas.

6.5.3 Verificação funcional da influência da preferência multiatributo do usuário na seleção de recursos

Esta seção apresenta a influência da preferência multiatributo do usuário em um processo de seleção de recurso. O objetivo é mostrar a flexibilidade do nosso modelo de decisão para tratar escalas diferentes de preferência entre dois atributos.

Configuração do ambiente

O ambiente de teste para o experimento desta seção foi configurado conforme anteriormente detalhado na seção 6.5.1 no que se refere ao MASK. Informações adicionais para o experimento específico desta seção são apresentadas a seguir.

A preferência multiatributo de um usuário no MASK não foi comparada com outras abordagens pelo fato de não termos identificado na nossa revisão bibliográfica, outros modelos de decisão com preferências diferenciadas para atributos. Conforme anteriormente relatado, o Condor permite ao usuário utilizar livremente o atributo *rank* para expressar sua preferência. Uma possibilidade seria instrumentalizar o Condor com o nosso modelo de decisão, mas isso demandaria alteração do código do Condor (que não é código aberto) para os cálculos apresentados na seção 6.4.

A preferência entre dois atributos, preço e tempo de execução, é representada no modelo de decisão do MASK pelas variáveis δ_1 e δ_2 respectivamente, que orientam a seleção[19].

Execução e análise

Um mesmo *job* foi submetido ao MASK 11 vezes, número esse obtido pela variação da escala de preferência entre 0 e 1 para multiatributos com intervalos de 0,1 conforme mostra

[19] Embora a implementação do modelo atual tenha sido feita com o usuário fornecendo os valores para as variáveis δ1 e δ2, é possível, em trabalhos futuros, que o agente do usuário verifique a melhor relação entre essas variáveis segundo as opções do ambiente no momento e decida qual combinação de preferências é melhor para o usuário.

a Tabela 6-2. A cada submissão, o usuário alterou sua preferência para os atributos preço e desempenho para um valor pertencente a essa escala. As cinco máquinas descritas na Tabela 6-1 estavam disponíveis para execução. A Tabela 6-2 apresenta a utilidade $U(M_i)$ calculada para todas as submissões desse *job*.

Tabela 6-2: Utilidade $U(M_i)$ para diferentes preferências do usuário

Máquina	Tempo de execução (ms)	Preço	$U(M_i)$ para diferentes preferências do usuário (δ_2)										
			δ_2=**0**	δ_2=**0,1**	δ_2=**0,2**	δ_2=**0,3**	δ_2=**0,4**	δ_2=**0,5**	δ_2=**0,6**	δ_2=**0,7**	δ_2=**0,8**	δ_2=**0,9**	δ_2=**1**
Sajama	1365604	80	0,993	0,991	0,989	0,987	0,985	0,9829	0,980	0,978	0,976	0,974	0,972
Bonete	3197365	80	0,992	0,987	0,982	0,976	0,971	0,9656	0,945	0,937	0,929	0,921	0,913
Itapeva	5795727	50	0,995	0,984	0,974	0,963	0,952	0,9418	0,931	0,920	0,910	0,899	0,889
Ibituruna	4718492	70	0,993	0,985	0,977	0,968	0,960	0,9519	0,944	0,935	0,927	0,919	0,911
Palermo	1244007	90	0,991	0,989	0,988	0,986	0,984	0,9828	0,981	0,979	0,978	0,976	0,974

Na Tabela 6-2, a célula com cor diferente indica o maior valor da utilidade e a escolha feita. A máquina Itapeva é a escolhida quando o usuário tem total preferência pelo preço e nenhuma preferência pelo desempenho oferecido (δ_2=**0**). No entanto, quando o usuário passa a considerar o desempenho, mas ainda em proporção inferior à sua preferência por preço, ou seja enquanto $0,1 <= \delta_2 <= 0,5$, a máquina Sajama passa a ser a escolhida. A máquina Sajama oferece o segundo melhor desempenho (o melhor desempenho é oferecido pela máquina Palermo) a um preço inferior que a máquina Palermo. Quando a relação desempenho/custo aumenta a favor do desempenho, ou seja, quando $\delta_2 > 0,5$, a máquina Palermo passa a ser a escolhida.

O comportamento de uma seleção diante de variadas preferências do usuário, tal qual apresentado na Tabela 6-2, demonstra a flexibilidade do modelo de decisão para atender a escalas de preferência diferenciadas pelo usuário para atributos diferentes.

6.5.4 Verificação funcional da influência da confiabilidade do recurso na seleção de recursos

Esta seção apresenta a influência da confiabilidade do servidor no processo de seleção de recursos. O objetivo é mostrar que a confiabilidade do servidor e do serviço contratado pode alterar a seleção entre um conjunto de máquinas disponíveis. Dois experimentos são feitos para esse propósito. O objetivo do primeiro experimento é mostrar uma aplicação numérica para as fórmulas apresentadas no modelo de decisão, destacando a influência da

utilização das probabilidades que representam confiabilidade. O objetivo do segundo experimento é mostrar quando as probabilidades que representam confiabilidade podem ser úteis e influentes em um processo de seleção.

Configuração do ambiente

Conforme descrito na seção 6.4.6, ao final de cada seleção e execução de um *job* no MASK, o *server_agent* registra as evidências sobre o procedimento ocorrido. Tais evidências descrevem a confiabilidade do servidor e são utilizadas pelo modelo de decisão como variáveis de risco. Essas variáveis são extraídas da estrutura de dados descrita na Figura 6-3 e utilizadas para cálculo das probabilidades z_p e q_p desta seção.

Tabela 6-3: Preço e probabilidades z_q e q_q associadas a cada máquina do experimento

	preço	z_q	q_q
Sajama	80	0,67	0,05
Bonete	80	0,73	0,43
Itapeva	50	1,00	0,00
Ibituruna	70	1,00	0,00
Palermo	90	0,95	0,00

Para auxiliar o entendimento dos cálculos das funções de utilidade apresentadas nos experimentos desta seção, apresentamos na Tabela 6-3 as evidências registradas para as máquinas nos casos submetidos e o valor do preço acordado na negociação. Na Tabela 6-4, apresentamos para cada caso submetido, a sua correspondente preferência multiatributo e a predição de tempo de execução em cada máquina, calculada pelo PredCase.

Tabela 6-4: Preferência multiatributo e predição de tempo de execução para cada caso

casos	**Preferências**		**Tempo de execução (ms)**				
	$\delta_{1=}$**0**	$\delta_{2=}$**0**	**Sajama**	**Bonete**	**Itapeva**	**Ibituruna**	**Palermo**
caso1	0,3	0,7	231.977	278.341	491.643	457.625	198.639
caso2	0,2	0,8	24.255	29.114	50.209	46.377	16.784
caso3	0,9	0,1	36.287	38.818	71.008	82.547	32.996
caso4	0,8	0,2	109.476	144.127	221.221	268.333	114.477
caso5	0,5	0,5	1.365.604	3.197.365	5.795.727	4.718.492	1.244.007
caso6	0,1	0,9	76	61	114	94	76
caso7	0,7	0,3	111	22	28	165	63

Execução e análise – Aplicação numérica para fórmulas do modelo de decisão

Para o primeiro experimento, um mesmo *job* é enviado para o MASK que então consulta as cinco máquinas para execução. A Tabela 6-5 apresenta os valores das variáveis envolvidas nos cálculos das equações (8) e (11). Verifica-se que o preço esperado P é alterado em função da probabilidade da máquina executar em tempo maior que o negociado. As máquinas Bonete, Sajama e Palermo tiveram o preço esperado alterado de acordo com a probabilidade z_p. As máquinas Sajama, Palermo e Itapeva tiveram o tempo de execução esperado alterado de acordo com a distribuição de probabilidade empírica registrada em $p(s)_p$. Para facilitar o entendimento das informações da Tabela 6-5, hachuramos em tom de cinza escuro as variáveis que registram a confiabilidade do servidor e em tom de cinza claro as variáveis que alteram o valor original de t_1 e p_1. O cálculo de U(P,T) na Tabela 6-5 está considerando δ_1=0,5 e δ_2=0,5.

Tabela 6-5: Exemplo de algumas variáveis envolvidas no cálculo de U(P,T)

máquina	t1	p1	g	p(s)	z	q	T	P	U(P)	U(T)	U(P,T)
Sajama	1365604	80	0,2	-0,001	0,67	0,053	1366896,8	74,67	0,993	0,973	0,9829
Bonete	3197365	80	0,2	0	0,73	0,431	3197365	75,61	0,992	0,939	0,9656
Itapeva	5795727	50	0,2	-0,03	1,00	0	5969598,8	50	0,995	0,889	0,9418
Ibituruna	4718492	70	0,2	0	1,00	0	4718492	70	0,993	0,911	0,9519
Palermo	1244007	90	0,2	-0,049	0,95	0	1304963,3	89,18	0,991	0,974	0,9828

Execução e análise – Influência das variáveis de risco na seleção

Para o segundo experimento, cinco *jobs* são enviados duas vezes para o MASK, que então consulta as cinco máquinas para execução. Na primeira submissão de cada *job*, o código do MASK foi alterado de forma que o modelo de decisão desconsiderasse as variáveis g e $p(s)_p$, pois definiu-se os valores para z_p=1 e q_p=1 nas equações (7) e (10). Chamamos essa primeira variação de cálculo de $U(P,T)_1$. Na segunda submissão de cada *job*, o código de MASK foi restaurado para o modelo definido na seção 6.4, utilizando-se $0<z_p<1$ e $0<q_p<1$ conforme as evidências registradas e, portanto, considerando as variáveis g e $p(s)_p$ conforme equações (8) e (11). Chamamos essa segunda variação de cálculo de $U(P,T)_2$. A primeira variação de cálculo corresponde ao uso da teoria da utilidade, onde preferências são associadas a utilidades. A segunda variação de cálculo corresponde à

teoria da decisão, que além de associar preferência às utilidades, usa também probabilidades para o cálculo do valor esperado.

A Tabela 6-6 mostra um exemplo para o processo de avaliação em cinco máquinas candidatas para a submissão de *jobs* sujeitos a duas variações de cálculo, $U(P,T)_1$ e $U(P,T)_2$. Nessa tabela, os valores para preço seguem a informação disponível na Tabela 6-3. Para a primeira submissão de cada caso, a $U(P,T)_1$ é utilizada e a seleção está destacada em cinza claro na Tabela 6-6. Para a segunda submissão do caso, a $U(P,T)_2$ é utilizada e a seleção está destacada em cinza escuro na Tabela 6-6.

Tabela 6-6: Comparação de cálculo de utilidade com e sem risco e sua influência na seleção – para valores diferentes de preço

Casos	Máquinas									
	Palermo		Ibituruna		Itapeva		Bonete		Sajama	
	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2
caso1	0,9926	0,9926	0,9924	0,9923	0,9936	0,9935	0,9928	0,9930	0,9930	0,9934
caso2	0,9928	0,9928	0,9942	0,9942	0,9958	0,9958	0,9935	0,9939	0,9935	0,9939
caso3	0,9985	0,9985	0,9978	0,9978	0,9982	0,9982	0,9985	0,99855	0,9986	0,99860
caso4	0,9964	0,9964	0,9944	0,9942	0,9955	0,9953	0,9961	0,9961	0,9967	0,9967
caso5	0,9834	0,9834	0,9519	0,9528	0,9433	0,9443	0,9653	0,9659	0,9827	0,9832
caso6	0,9919	0,9920	0,9937	0,9937	0,9955	0,9955	0,9928	0,9932	0,9928	0,9933
caso7	0,9973	0,999999	0,9979	0,999997	0,9985	0,9999995	0,9976	0,9999996	0,9976	0,999998

A Tabela 6-6 não apresenta variação na seleção quando as variáveis de risco são utilizadas, uma vez que a mesma máquina é selecionada para as duas submissões. Isso porque a variação do valor esperado foi pequena e insuficiente para que uma máquina com tempo de execução e/ou preço maior, mas com maior probabilidade de cumprimento do serviço estabelecido superasse a utilidade calculada por outra máquina com tempo de execução e/ou preço menor e menor probabilidade de execução do serviço conforme estabelecido.

A Tabela 6-7 apresenta a submissão dos mesmos casos apresentados na Tabela 6-4, porém em uma situação onde todas as máquinas oferecem o mesmo preço. Nessa situação, como o preço negociado é o mesmo para todas as máquinas, a probabilidade de variação do preço irá depender da confiabilidade da máquina. A Tabela 6-7 apresenta as seleções realizadas para essa nova situação e mostra que diferentes seleções foram feitas usando-se $U(P,T)_1$ e $U(P,T)_2$.

Tabela 6-7: Comparação de cálculo de utilidade com e sem risco e sua influência na seleção – para valores iguais de preço

Casos	Máquinas									
	Palermo		Ibituruna		Itapeva		Bonete		Sajama	
	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2	U(P,T)1	U(P,T)2
caso1	0,9939	0,9939	0,9924	0,9923	0,9922	0,9921	0,9935	0,9937	0,9937	0,9940
caso2	0,9944	0,9944	0,9942	0,9942	0,9942	0,9942	0,9943	0,9946	0,9943	0,9947
caso3	0,9987	0,99872	0,9978	0,9978	0,9980	0,9980	0,9986	0,9986	0,9987	0,99869
caso4	0,9968	0,9967	0,9944	0,9942	0,9951	0,9949	0,9963	0,9963	0,9969	0,9969
caso5	0,9844	0,9846	0,9519	0,9528	0,9423	0,9433	0,9658	0,9664	0,9832	0,9837
caso6	0,9937	0,9938	0,9937	0,9937	0,9937	0,9937	0,9937	0,99407	0,9937	0,99414
caso7	0,9979	0,9973	0,9979	0,9979	0,9979	0,9979	0,9979	0,99802	0,9979	0,99804

Pelos resultados das Tabelas 6-6 e 6-7, pode-se notar que existe uma influência significativa das variáveis de risco na seleção apenas em algumas situações. O exemplo apresentado na Tabela 6-7 mostra que quando os valores das alternativas são iguais ou muito próximos, as probabilidades auxiliam a tomada de decisão pela tendência de alteração dos valores esperados, com base em evidências passadas e destacando o critério da confiabilidade da máquina para o provimento do serviço acordado.

6.5.5 Verificação de desempenho na seleção de recursos

Esta seção objetiva analisar o tempo decorrido para a realização da seleção de um recurso pelo Condor e MASK quando um usuário submete um *job* para esses sistemas.

Configuração do ambiente

O ambiente de teste para o experimento desta seção foi configurado conforme anteriormente detalhado na seção 6.5.1. Informações adicionais para o experimento específico desta seção são apresentadas a seguir.

Para cada *job* que submetemos ao MASK, geramos um *log* contendo as informações recebidas de cada máquina participante do processo de negociação. Nesse arquivo de *log*, incluímos ainda o tempo para executar os módulos de predição, verificação de políticas e seleção. Pelo fato do Condor não possuir um módulo de predição de tempo de execução, apresentamos os tempos decorridos para a execução de cada módulo do MASK para compararmos as abordagens.

Para cada *job* submetido ao Condor foi gerado um arquivo de *log*. Esse arquivo inclui, dentre outras informações, o tempo de chegada do *job* ao sistema e o horário de início de execução do *job* no recurso selecionado.

Resultados e análises

A estratégia utilizada para verificação do tempo decorrido na seleção de recursos consistiu da análise de todos os arquivos de *log* gerados pelo MASK nos experimentos realizados anteriormente na validação dos modelos PredCase, POLAR e do modelo de decisão. O conjunto de *logs* gerados durante essas validações passadas totalizou 268 arquivos de *log* do MASK. Capturamos as informações sobre a duração do processamento de cada módulo do MASK e geramos um resumo estatístico para esses valores. As máquinas do experimento possuem configurações diferentes e, portanto registraram tempos de execução distintos para cada um dos referidos módulos. Para o conjunto de máquinas do nosso experimento, o tempo médio para o MASK selecionar um recurso foi em torno de 700 ms, conforme mostra a Figura 6-8.

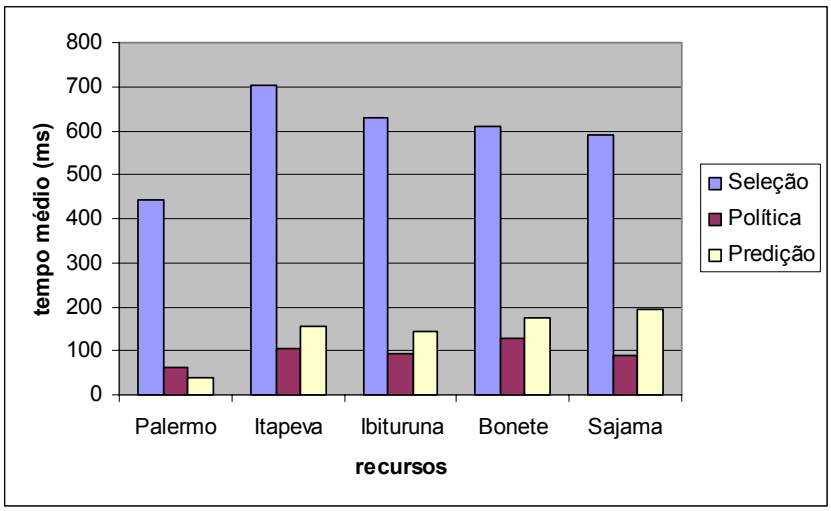

Figura 6-8: Tempo médio para realizar seleção utilizando o MASK

Os registros de tempo mínimos verificados para cada máquina são apresentados na Figura 6-9. O tempo mínimo registrado para o processo de predição ocorre quando não existe nenhuma aplicação com o mesmo nome da aplicação do novo *job*. Nesse caso o tempo para processar a recuperação de casos é mínino e não retorna nenhum caso similar. O tempo mínimo registrado para o processo de verificação de restrições de acesso ocorre

quando o usuário não tem autorização para executar um *job* no recurso e, nesse caso, não são negociados os parâmetros para provimento do serviço e não é verificada a carga de trabalho no recurso destino.

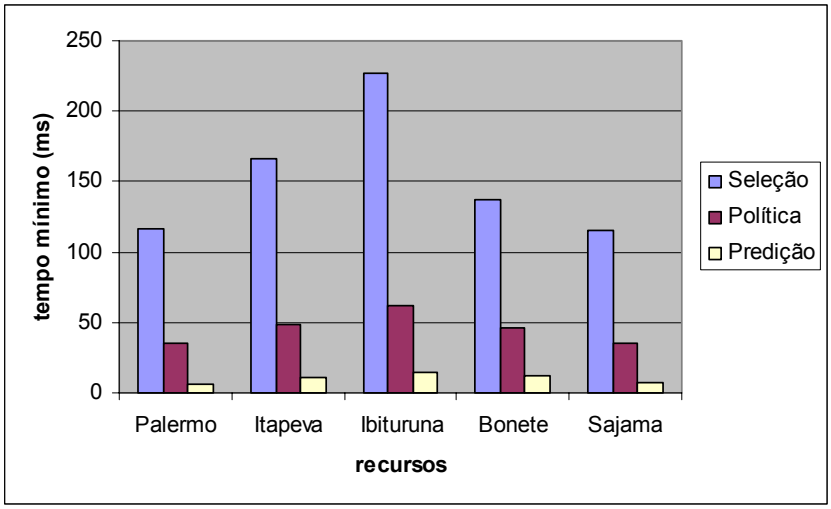

Figura 6-9: Tempo mínimo para realizar seleção utilizando o MASK

A Figura 6-10 apresenta o tempo máximo registrado por máquina para realizar uma seleção. Esse tempo máximo para realizar a seleção pode ainda ser estipulado no MASK como uma variável do vocabulário dos agentes. Essa variável representa o tempo que o *user_agent* deve aguardar por respostas das máquinas consultadas no processo de negociação. Nos experimentos efetuados esse tempo máximo foi de 3s.

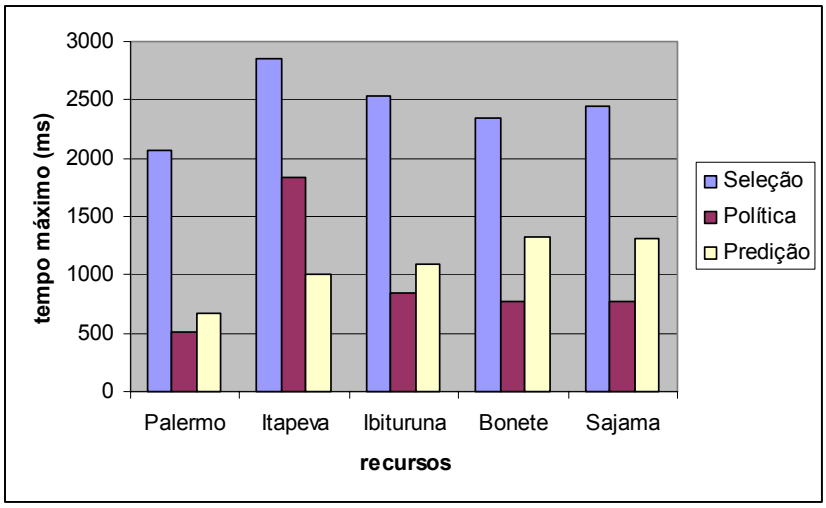

Figura 6-10: Tempo máximo para realizar seleção utilizando o MASK

A estratégia utilizada para a verificação do tempo decorrido na seleção de recursos do Condor consistiu na análise dos arquivos de *log* gerados pelo Condor. Para analisarmos esses arquivos, agendamos várias submissões de *jobs* ao Condor usando o processo *cron*[20] do Linux. As submissões foram seqüenciais, de forma que não houvesse espera na fila de escalonamento do Condor, situação essa que ocorre somente quando todos os recursos do *pool* já estão ocupados executando algum *job*. A periodicidade de re-submissão de um *job* variava de acordo com a previsão de sua execução, o intervalo total de tempo para re-submissões de um mesmo *job* foi aleatório. Seguindo essa simulação de submissões de *jobs*, foram gerados 630 arquivos de *log* do Condor. Nesses arquivos, foi registrado o tempo, que variava entre os valores de 4s e 5s para realizar a seleção de recurso.

Outros estudos sugerem que esse tempo de seleção do processo de *matchmaking* do Condor cresce significativamente com o aumento do número de máquinas [Liu & Foster 2004]. Embora não tenhamos realizado simulações para o nosso modelo para analisar a escalabilidade do mesmo, podemos compará-lo com Condor, para os experimentos conduzidos em ambiente real. Todo o processo de seleção do Condor é realizado em uma única máquina para todo o conjunto de máquinas disponíveis no ambiente e para todos os usuários. O nosso modelo, ao contrário, realiza as seleções de forma distribuída. A seleção é feita pelo agente que representa o usuário e que pode estar em qualquer recurso do ambiente.

6.6 Conclusões

A seleção de recursos em grades computacionais deve considerar diversos fatores e precisa estar direcionada para atender aos objetivos do decisor.

O capítulo aqui apresentado mostra que a decisão utiliza preferências do usuário e variáveis que trazem risco. O modelo é formulado utilizando a teoria da decisão que combina a teoria da probabilidade com a teoria da utilidade. Evidências de execuções de aplicações passadas são utilizadas como probabilidade de ocorrência dos fatores incertos, tais como a probabilidade de erro na predição e a mudança do preço estabelecido durante o processo de negociação. As utilidades são associadas às preferências do decisor, aqui

[20] O processo *cron* no Unix é um processo *daemon* com função de agendamento de tarefas que são executadas em determinados horários previamente descritos dentro de um arquivo denominada *crontab*.

tratado como o usuário. O objetivo do modelo é escolher a melhor máquina para executar um *job*.

Embora o Condor apresente uma opção que permita ordenar as máquinas para realizar a seleção através do atributo "*rank*", o modelo para essa seleção é aberto e definido pelo usuário. Não é possível obter o resultado esperado utilizando uma simples formulação quando multiatributos são considerados preferenciais. O MASK utiliza um modelo de decisão, que além das preferências definidas em forma de utilidades, também agrega probabilidades ao modelo de decisão.

Resultados experimentais mostraram que a seleção maximiza a utilidade do usuário. Diferentes máquinas podem ser escolhidas se forem alteradas a preferência do usuário e/ou as probabilidades resultantes de evidências passadas.

Pelo fato do Condor não possuir um modelo de predição e se basear nas características físicas da máquina quando a preferência é por desempenho, a execução total para um mesmo conjunto de *jobs* resultou em maior tempo computacional que os recursos escolhidos pelo MASK.

Embora seja interessante faze-lo no futuro, não realizamos um estudo experimental de escalabilidade da solução por termos feito toda a implementação em código para ambiente real de grades computacionais. Um estudo em escala demandaria a implementação de todo o código para um simulador. Mas, teoricamente, o MASK tende a ter os seguintes comportamentos com o aumento em escala de:

1) número de máquinas: A eficiência do MASK tende a ser maior comparada com as abordagens centralizadas, pois o processamento é distribuído e paralelo;

2) número de casos: A acurácia da predição do MASK tende a ser maior comparada com um número reduzido de casos. No entanto, o crescimento da base de casos deve ser controlado (conforme discutido no Capítulo 4) para não aumentar o tempo de seleção;

3) distribuição de máquinas no espaço: A eficiência do MASK passa a ser dependente da velocidade da rede, pois por ser uma solução distribuída depende do *overhead* de comunicação na rede;

4) heterogeneidade de máquinas: A eficiência do MASK pode ser dependente de máquinas mais lentas para a computação distribuída, no entanto MASK é altamente adaptável para a inclusão de máquinas heterogêneas ao ambiente compartilhado pelo fato de prover predição em casos passados e não em modelos de *hardware*.

Todas as variáveis que influenciam a seleção de máquinas candidatas são tratadas no Condor como requisitos e todas as informações são verificadas. Nossa abordagem, ao contrário, avança nos processamentos conforme verificações prévias. São analisados, em ordem, os requisitos de máquinas, as restrições de acesso e a carga de trabalho. Devido a essas diferenças de modelo entre MASK e Condor verificamos as seguintes características qualitativas: 1) a verificação de restrições no MASK é distribuída; 2) a seleção no MASK é distribuída; 3) O MASK permite a seleção baseada em preferências multiatributos; 4) O MASK decide baseando-se em utilidades e probabilidades. Tais avanços qualitativos fizeram com que o MASK apresentasse os seguintes avanços quantitativos nos experimentos realizados no que se refere à seleção de recursos: 1) O MASK realiza predição de tempo de execução e escolhe a máquina que roda mais rapidamente quando a preferência é por desempenho; 2) O MASK realiza a seleção de recurso em tempo inferior ao Condor (comportamento verificado para o conjunto de equipamentos do nosso experimento). Essa afirmação pode ser generalizada para escalas maiores, considerando configurações de ambiente de grades computacionais onde o *overhead* da rede seja inferior a 3s e a máquina mais lenta do ambiente tenha desempenho semelhante à máquina mais lenta do nosso experimento.

Os experimentos demonstraram o papel facilitador do *broker* na busca da seleção do melhor recurso no ambiente de grades computacionais para atender às preferências do usuário, conjugando as diversas variáveis definidas na formalização do problema PSMRG.

Capítulo 7

Conclusões e trabalhos futuros

7.1 Contribuição do trabalho desenvolvido

O trabalho aqui apresentado abordou o problema de seleção de recursos em grades computacionais. A seleção de recursos é uma das atividades iniciais do processo de execução de aplicativos em grades computacionais. É onde se pode prover a virtualização do uso do conjunto dos recursos compartilhados. Esse serviço ainda não é padronizado para grades computacionais e geralmente é oferecido por um *broker* integrado a um *middleware* para grade. Um *broker* oferece a um usuário uma forma mais simples de interação com a grade. Ao selecionar adequadamente um recurso para executar um *job* o *broker* deve considerar várias questões, tais como: Quais recursos estão disponíveis? Quais recursos atendem aos requisitos da aplicação? O usuário está autorizado a executar essa aplicação nesse recurso? Qual o tempo previsto para executar essa aplicação? Qual a garantia para que a aplicação rode nesse período? Qual a máquina que roda pelo menor preço? Qual a probabilidade da máquina não executar no tempo previsto? Qual a probabilidade da garantia ser aplicada caso a previsão não se confirme?

Os trabalhos relacionados na literatura abordam um conjunto diferente dentre essas questões para ser o aspecto principal para realizar a seleção do recurso. Propusemos uma

taxonomia para analisar os trabalhos relacionados onde distinguimos quem está tomando a decisão, como está sendo feita a tomada de decisão, quantos recursos estão sendo selecionados e quais objetivos estão sendo atendidos. Conduzimos a nossa solução centrada nos objetivos do usuário. Um modelo de decisão para atender diferentes aspectos foi elaborado baseando-se na teoria da decisão.

Um *middleware* com função de *broker*, denominado MASK, foi desenvolvido utilizando a tecnologia de sistemas multiagentes. O MASK possui agentes que representam usuários (*user_agent*) e agentes que representam servidores (*server_agent*). Através de um processo de negociação entre um *user_agent* e *n server_agents*, são apresentadas para o *user_agent*, as condições do provimento do serviço, contendo a predição da execução, obtida através de um modelo de predição; o preço de uso do recurso, a garantia e a confirmação do acesso do usuário a cada recurso candidato, obtidos através de um modelo de verificação de restrições de acesso. Diante dessas informações, o *user_agent*, considerando as preferências do usuário, seleciona a opção que maximiza a utilidade do usuário através de um modelo baseado na teoria da decisão.

Definimos, implementamos e testamos um modelo de predição de tempo de execução de *jobs* utilizando o paradigma de raciocínio baseado em casos onde a predição de execução de um novo *job* baseia-se em execuções passadas de *jobs* similares. Cada descrição de *job* é considerada um caso e a idéia chave é verificar se um servidor específico já executou um caso similar. O tempo gasto na execução de um *job* no passado auxilia no cálculo da predição de tempo para executar um novo *job*. Nesse modelo de predição, aqui denominado PredCase, é apresentado um novo algoritmo de recuperação de casos envolvendo distância geométrica e seqüência de relevância denominado algoritmo *refined nearest-neighbor*. O modelo foi validado com média de 9% de erro de predição.

Definimos, implementamos e testamos um modelo para verificação de restrições de acesso ao recurso baseado em políticas de granularidade fina de forma antecipada à submissão do *job* evitando submissões fracassadas. Nesse modelo os agentes verificam dinamicamente se a proposta enviada pelo usuário pode ser aceita, rejeitada ou há a possibilidade de elaboração de uma contraproposta. O modelo foi validado e comparado com uma abordagem ingênua, onde a seleção é feita de forma randômica e as políticas locais impedem a execução do *job,* levando a um conjunto de re-submissões aleatórias. O

modelo também foi comparado com Condor, que realiza o processo de verificação de restrições de acesso de forma centralizada.

Definimos, implementamos e testamos um modelo de decisão baseado na teoria da decisão. Valores probabilísticos decorrentes da aprendizagem do ambiente são associados a desempenho e preço entre os recursos candidatos. As preferências do usuário são expressas através de utilidades. Um modelo multiatributo é maximizado para selecionar a máquina com as melhores condições de executar o *job* no ambiente compartilhado.

Conforme discutido neste trabalho, as técnicas de predição de desempenho e de teoria da decisão associadas a um modelo de verificação de restrições de acesso baseado em políticas de granularidade fina agrupam os elementos essenciais para a seleção do melhor recurso em uma grade computacional. O conjunto de modelos e algoritmos foi validado com a execução de uma suíte de testes. O conjunto de aplicações e recursos utilizados nos experimentos de cada capítulo é apresentado de forma agrupada e com mais detalhes no Apêndice B. Uma lista de publicações dos modelos e resultados discutidos nesta tese é apresentada no Apêndice C.

Embora os modelos de predição, verificação de restrições de acesso e decisão possam ser implementados por outras abordagens que não a de sistemas multiagentes, esta apresenta grandes vantagens para a seleção de recursos em tempo real para submissões de *jobs* em grades computacionais. Primeiro, pelo fato de todos esses modelos terem sido implementados de forma distribuída, e pelo fato de um sistema multiagente ser um sistema inerentemente distribuído, e aplicado para um ambiente também distribuído que são as grades computacionais. Segundo, pela interpretação de que usuários e recursos estão negociando um acordo de nível de serviço e que possuem agentes para os representarem e buscarem seus diferentes objetivos. Finalmente, pelo fato dos sistemas multiagentes proverem métodos robustos para a coordenação e competição entre agentes que representam um mundo real com alto grau de dinamismo e complexidade.

O MASK responde a cada uma das questões formuladas no início desta seção, pois verifica a disponibilidade dos recursos pela análise de carga de trabalho (no POLAR); verifica quais recursos atendem aos requisitos da aplicação baseando-se em políticas de granularidade fina (no POLAR); verifica se o usuário está autorizado a executar uma aplicação em determinado recurso (no POLAR); calcula o tempo previsto para executar uma aplicação (no PredCase); negocia a garantia aplicada quando a aplicação não roda no

período previsto (no processo de negociação associado ao POLAR); é capaz de verificar qual a máquina que roda pelo menor preço (pela preferência informada no modelo de decisão); sabe a probabilidade da máquina não executar no tempo previsto e sabe qual a probabilidade da garantia ser aplicada caso a previsão não se confirme (pelos dados históricos registrados e uso das probabilidades no modelo de decisão). Dessa forma, o MASK mostra-se como uma abordagem original capaz de auxiliar a comunidade de usuários na interação com o ambiente de grades computacionais.

7.2 Direções futuras

A pesquisa aqui desenvolvida sugere caminhos em várias direções a partir do patamar de soluções alcançadas nesta tese e pela descoberta de novas possibilidades de pesquisa na área de grades computacionais.

Conforme nossas observações feitas no capítulo 4, que trata de um modelo de predição para execução de *jobs*, verificamos que a acurácia da predição é extremamente sensível à dedicação do recurso na tarefa de execução de um *job*. Um grande desafio é obter uma boa acurácia sem que a máquina esteja dedicada para a execução do *job*. Mais complicado ainda é desenvolver um modelo onde essa acurácia da predição esteja associada a um *job* escalonado para um tempo futuro e, portanto sendo executado conjuntamente com uma carga de trabalho local, gerada, por exemplo, pelo dono do recurso. Ainda relativamente aos mecanismos de predição, sugere-se uma extensão do trabalho para a fase de retenção da base de casos no modelo RBC. Essa fase pode demandar algoritmos originais para a análise de permanência dos casos em uma base de raciocínio baseado em casos. Um caminho pode ser o uso de algoritmos similares aos algoritmos utilizados para a análise de permanência de páginas em um servidor de *cache* para a Internet. Além dessa correlação, tais algoritmos devem revisar os casos da base considerando diversos fatores, tais como a influência positiva de um caso na predição, a quantidade de vezes que o caso é acessado e o tamanho máximo da base, garantindo um tempo aceitável de processamento. A fase de recuperação de casos do nosso modelo de predição pode ser investigada com alterações na organização da memória de casos e conseqüentemente no uso de novos algoritmos da área de recuperação de informações. Com tais melhorias, a acurácia da predição pode ser aumentada e o tempo de recuperação dos casos pode ser diminuído.

Pelo fato de termos desenvolvido um modelo de predição que permite um escalonamento dos *jobs* para um tempo futuro, surge um interessante e desafiador problema: em uma situação onde todas as máquinas estão ocupadas, o Condor enfileira os *jobs* excedentes e os aloca para a primeira máquina disponível. Essa situação não permite a melhor escolha e sim explicita a única escolha disponível do ambiente. Uma solução que apresenta predição pode escolher uma das máquinas do ambiente que possa otimizar a preferência do usuário, já que tem-se a estimativa de quando as máquinas do ambiente estarão liberadas e qual delas pode terminar a execução do novo *job* em um prazo de tempo mais curto ou oferecer o melhor custo, por exemplo.

Outra extensão do trabalho se refere à definição de uma arquitetura hierárquica entre o modelo de verificação de restrições de acessos locais, as políticas globais definidas para uma determinada organização virtual e o ponto de verificação de políticas conforme um modelo que utilize PEP e PDP. Essa integração permite definições e verificações de políticas para os recursos do ambiente de grades computacionais nos níveis macro e elementares de uma organização virtual.

Novos mecanismos de tolerância a falhas podem ser desenvolvidos para grades computacionais de acordo com as métricas definidas no SLA. Para isso é importante fazer uso de modelos de migração de *jobs* entre as máquinas que foram melhores classificadas pelo modelo de teoria da decisão.

A negociação encadeada aqui definida, porém não implementada por restrições do nosso ambiente de teste, também é uma extensão de pesquisa interessante. Ao considerar o tempo de transferência dos dados pela rede, uma nova variável deve ser incluída no modelo de seleção para considerar a movimentação de dados em caso de falhas. Dessa forma, seria possível considerar não somente as variáveis do ambiente computacional, mas também as características da rede. Tal implementação irá auxiliar nos mecanismos de tolerância a falhas que necessitem fazer migrações e, portanto, precisam considerar as condições de rede.

O nosso trabalho visou a seleção de um recurso ou *cluster* que atendesse à todos os requisitos de uma aplicação. Uma extensão interessante é buscar em nova pesquisa, a co-alocação, ou seja, a seleção de *n* recursos que, de forma complementar, atendam aos requisitos da aplicação. Novamente, a negociação encadeada passa a ser importante, pois será necessário coordernar vários recursos negociados conjuntamente para atender uma

determinada solicitação. Nesse caso, passa a ser importante também a existência de um agente com características de gerenciar um fluxo de alocação de recursos.

Referências bibliográficas

[Aamodt & Plaza 1994] Aamodt, A., plaza, E. (1994). *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. AI Communications.

[Abramson *et al.* 1995] Abramson, D., Sosic, R. Giddy, J. Hall, B. (1995).Nimrod: A tool for performing Parametised Simulations using distributed workstations. In Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing. Washington, DC, USA.

[AIAI 2003] Artificial Intelligence Applications Institute. http://www.aiai.ed.ac.uk/links/cbr.html. Visitado em fevereiro/2003.

[Alfieri *et al.* 2003] Alfieri, R., Cecchini, R., Ciaschini, V., dellAgnello, L., Frohner, A., Gianoli, A., Lõrentey, K., Spataro, F.(2003). VOMS, an Authorization System for Virtual Organizations. In Proceedings of 1st European Across Grids Conference, Santiago de Compostela, Spain.

[Almeida 2005] Almeida, Adiel T. (2005). Modelagem Multicritério Para Seleção de Intervalos de Manutençõ Preventiva Baseada na Teoria da Utilidade Multiatributo. Pesquisa Operacional, v.25,n.1, p69-81.Brasil.

[Alunkal *et al.* 2003] Alunkal, B., Valjkovic, I., Laszewski, G., Amin, K.(2003). Reputation-based Grid Resource Selection. In Proceedings of Workshop on Adaptive Grid Middleware (AGridM 2003), September 28, 2003, New Orleans LA, USA.

[Andrieux *et al.* 2003] Andrieux, A., Berry, D., Garibaldi, J., Jarvis, S., MacLaren, J., Ouelhadj, D., Sneling, D.(2003). Open issues in Grid Scheduling. Report of workshop at e-Science Institute. Edinburgh, Scotland.

[Azzedin & Maheswaran 2002] Azzedin, F., Maheswaran, M. (2002). Integrating Trust into grid Resource Management Systems. In Proceedings of International Conference on Parallel Processing (ICPP 2002), Vancouver, Canada.

[Bagrodia *et al.* 1999] Bagrodia, R., Deeljman, E.,.Docy, S. Phan, T. (1999). Performance prediction of large parallel applications using parallel simulations. In Proceedings of the seventh ACM SIGPLAN symposion on Principles and Practice of Parallel Programming. ACM Press: New York, USA.

[Bekman & Costa Neto 2002] Bekman, Otto R., Costa Neto, Pedro Luiz O. (2002). *Análise estatística da decisão*. Editora Edgard Blücher Ltda. São Paulo, Brasil.

[Beer *et al.* 1998] Beer, M., d'Inverno, M., Luck M., Jennings, N., Preist, C., Schroeder, M. (1998). Negotiation in Multi-Agent Systems. Knowledge Engineering Review 14 (3) 285-289.

[Bellifemine *et al.* 2003] Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. (2003). Jade a White Paper. Exp – Volume 3, no. 3, September.pp 6-19.

[Beowulf 2005] http://www.beowulf.org. Visitado em agosto/2005.

[Bertino et al. 2004] Bertino, E., Mazzoleni, P., Crispo, B., Sivasubramanian, S., Ferrari, E. (2004). Towards Supporting Fine-Grained Access Control for Grid Resources. In Proceedings of 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04). Suzhou, China.

[Bharadwaj & Baras 2003] Bharadwaj, V., Baras, J.(2003). Towards Automated Negotiation of Access Control Policies. 4th International Workshop on Policies for Distributed Systems and Networks (Policy'03). Lake Como, Italy.

[Bichindaritz 2005] Bichindaritz, I. (2005). Memory Organization as the Missing Link Between Case Based Reasoning and Information Retrieval in Biomedicine. Proceedings of the Workshop on CBR in the Health Sciences, Chicago, IL, USA, p.18-31.

[Bigus & Bigus 2001] Bigus, J., Bigus, J.(2001). *Constructing Intelligent Agents using Java.* Second Edition. Wiley. USA.

[Blast 2005] Blast.http://www.ncbi.nlm.nih.gov/BLAST/, visitado em Janeiro/2005.

[Borgeois & Spies 2000] Bourgeois, J. Spies, F. (2000). Performance Prediction of an NAS Benchmark Program with Chronosmix Environment. In Proceedings of the 6th International Euro-Par Conference on Parallel Processing (Lecture Notes in Computer Science 1900). Springer-Verlag: London, UK.

[Box *et al.* 2000] Box, D., Skonhard, A., Lam, J. (2000). *Essential XML: Beyond MarkUp*. Addison-Wesley. Boston, USA.

[Byde *et al.* 2002] Byde, A., Preist, C. Jennings, N. (2002). Decision procedures for multiple simultaneous auctions. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2, July 15-19, 2002, Bologna, Italy.

[Butler 2001] Butler, S. (2001). Improving Security Technology Selections with Decision Theory. In Proceedings of the Third Workshop on Economics-Driven Software Engineering Research (EDSER-3), affiliated with the 23rd International Conference on Software Engineering (ICSE 2001). Toronto, Canada.

[Buyya *et al.* 2002] Buyya, R., Abramson, D., Giddy, J., Stockinger, H. (2002).Economic models for resource management and scheduling in Grid computing. Journal Concurrency and Computation: Practice and Experience, vol. 14, pp. 1507--1542.

[Cactus 2005] Cactus Portal. High performance Parallel Scientifc Applications. http://portal.cct.lsu.edu. Visitado em novembro/2003.

[Campo 2002] Campo, C. (2002). Directory Facilitator and Service Discovery Agent. FIPA Ad-hoc. f-in-00070. Technical Committee.

[Cannon *et al.* 2003] Cannon, S., Chan, S., Olson, D. Tull, C. (2003). Using CAS to Manage Role-Based VO Sub-Groups. CHEP03. La Jolla, California, USA.

[Cao 2001] Cao, J. (2001). Agent-based Resource Management for Grid Computing. PhD Thesis, University of Warwick.

[Cao *et al.* 2002] Cao, J., Jarvis, S., Saini, S., kerbyson, D., Nudd, G. (2002). ARMS: An Agent-based Resource Management System for Grid Computing. Scientific Programming, Special issue on Grid Computing, IOS Press. 10(2): 135-148.

[Cardoso & Kon 2002] Cardoso, R., Kon, F. (2002). Mobile Agents: A Key for Effective Pervasive Computing. In Proceedings of Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). Anaheim, California, USA.

[Castellano *et al.* 2004] Castellano, M., Coviello, T., Piscitelli, G. (2004). An Intelligent Resource Selection System Based on Neural Network for Optimal Application Performance in a Grid Environment. In Proceedings of Computing in High Energy and Nuclear Physics (CHEP). Interlaken, Switzerland.

[Chan & Law 2002] Chan, A., Law, K. (2002). QoS Negotiations and Real-Time Renegotiations for Multimedia Communications. Proceedings of International Conference on Computer Communications and Networks (ICCCN). Miami, Florida, USA.

[Cheung *et al.* 2004] Cheung, W., Liu, J., Tsang, K., Wong, R. (2004). Dynamic Resource Selection for Service Composition in The Grid. In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04). Beijing, China.

[Chieng *et al.* 2001] Chieng, D., Marshall, A., Ho, I., Parr, G. (2001). Agent-Enhanced Dynamic Service level Agreement in Future Network Environments. In Proceedings of 4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS). Lecture Notes in Computer Science 2216. Chicago, IL, USA.

[Chunlin & Layuan 2005] Chunlin, L., Layuan, L. (2005). Pricing and Resource Allocation in Computational Grid with Utility Functions. In Proceedings of International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II pp. 175-180. Las Vegas, Nevada, USA.

[Claessens & Geuer-Pollman 2004] Claessens, J., Geuer-Pollman, C. (2004). Standardisation Roadmap V1.0. Trustcom Consortion 01945.

[Condor 2005] Condor Project. http://www.cs.wisc.edu/condor/, visitado em janeiro/2005.

[Coulouris *et al.* 2005] Coulouris, G. Dollimore, J., Kindberg, T. (2005). *Distributed Systems – Concepts and Design*. Addison Wesley. 4th edition.

[Cruz 2003] Cruz, L. (2003). BLAST – Teoria e Prática. Jornada de Bioinformática do Paraná (JOBIP). Curitiba, Brasil.

[Czajkowski *et al.* 2002] Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S. (2002). SNAP: A Protocol for negotiation of Service Level Agreements and Coodinated Resource Management. In Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'02), Edinburgh, Scotland.

[Dan *et al.* 2002] Dan, A., Franck, R., Keller, A., King, R., Ludwig, H., Kuebler, D., Polan, H., Spreitzer, M., Youssef, A. (2002). Web Services on demand: WSLA-driven Automated Management, IBM Systems Journal, Special Issue on Utility Computing, Volume 43, Number 1, pages 136-158, IBM Corporation.

[Dan *et al.* 2004] Dan, A., Ludwig, H., Rofrano, J. (2004) WS-Agreement Structure. Version 1.0. GRAAP-WG. Globol Grid Forum.

[Deelman al. 2004] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K., Livny, M. (2004). Pegasus: Mapping Scientific Workflows onto the Grid. In Proceedings of the 2nd European Accross Grids Conference. Nicosia, Cyprus.

[Dudani 1976] Dudani, S.(1976). The distance weighted k-nearest-neighbor rule. IEEE Transactions Systems, Man, Cybernetics. Vol 6, pp. 325-327.

[Eddy 2001] Eddy, S. (2001). HMMER User's Guide – Biological sequence analysis using profile hidden Markov models. Washington University School of Medicine. Technical Report. 2001.

[Elmroth & Tordsson 2004] Elmroth, E., Tordsson, J. (2004). A Grid Resource *Broker* Supporting Advance Reservations and Benchmark-Based Resource Selection. In Proceedings of Applied Parallel Computing. State-of-the-art in Scientific Computing. Springer-Verlag, Lecture Notes in Computer Science. Lyngby. Denmark. 2004.

[Ferber & Gasser 1991] Ferber, J., Gasser, L. Intelligence Artificielle Distribuée. (1991). Tutorial Notes of the 11th Conference on Expert Systems and their Applications Avignon, France.

[Ferber 1999] Ferber, J. (1999). *Multi-Agent Systems – An Introduction to Distributed Artificial Intelligence*. Addison-Wesley. Great Britain.

[FIPA 2002] Foundation for Intelligent Physical Agents. www.fipa.org. Visitado em janeiro 2002.

[Foster & Kelsselman 2004] Foster, I., Kelsselman, C. (2004). *The Grid2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers.

[Foster 2002] Foster, I. (2002).What is the Grid? A Three Point Checklist. GridToday, July 20, 2002.

[Foster *et al.* 2001] Foster, I., Kesselman, C., Tuecke, S. (2001) The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications 15 (2001).

[Foster *et al.* 2002] Foster, I., Kesselman, C., Nick, J., Tuecke, S. (2002). The physiology of the grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum.

[Foster 2003] Foster, I. (2003). The Grid: Computing without bounds. Scientific American. pp. 79-85. April.

[Franklin & Graesser 1996] Franklin, S., Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, Springer-Verlag (1997) 21—35. Budapest, Hungary.

[Gao 2004] Gao, G. (2004). Blast – A Practice-oriented Overview. Centre of BioInformatics at Peking University (CBI). Peking.

[García *et al.* 2005] García, J., Méndez, P., Valverde, J. (2005). GROCK: High-Throughput Docking using LCG Grid Tools. In Proceedings of The 6th IEEE/ACM International Workshop on Grid Computing (Grid 2005). Seattle, Washington, USA.

[Gibbons 1997] Gibbons, R. Historical Application Profiler for Use by Parallel Schedulers, Lecture Notes in Computer Science, pp. 58-75, 1997.

[Gilbert 1997] Gilbert, D. Intelligent agents: The right information at the right time. IBM Intelligent Agent White Paper (June 16, 1997), IBM.

[Globus 2005] Globus. http://www.globus.org. Visitado em janeiro/2005.

[Godik & Moses 2003] Godik, S., Moses, T. (2003). eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS Standard.

[Gomes *et al.* 2002] Gomes, L., Gomes, C., Almeida, A.(2002). *Tomada de Decisão Gerencial – Enfoque multicritério*. Editora Atlas S.A.São Paulo, Brasil.

[Green *et al.* 1997] Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F., Evans, R. (1997). Software Agents: A review. IAG Technical Report, Trinity College.

[GGF 2005] Global Grid Forum. http://www.gridforum.org/. Visitado em janeiro/2005.

[GRMS 2005] GRMS. http://www.gridlab.org/WorkPackages/wp-9. Visitado em fevereiro/2005.

[Gross *et al.* 2000] Gross, G., Gommans, L., Vollbrecht, J., Spence, D. (2000). Generic AAA Architecture. RFC 2903. IETF. 2000.

[Ho *et al.* 2002] Ho, S., Liu, C., Liu, S. (2002). Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm. Pattern Recognition Letters Journal (23), No. 13, November 2002, pp. 1495-1503.

[Holtzman 1989] Holtzman, S. (1989). *Intelligent decision systems*. Addison-Wesley.

[Jade 2005] Jade. http://jade.tilab.com. Visitado em janeiro 2005.

[Jang *et al.* 2004] Jang, S., Wu, X., Taylor, V. (2004). Using Performance Prediction to Allocate Grid Resources. GriPhyN Technical Report 2004-25.

[Jennings *et al.* 2000] Jennings, N., Faratin, P.,Lomuscio, A., Parsons, R., Sierra, C.,Wooldridge, M. (2000). Automated Negotiation: Prospects, Methods and Challenges. Journal of Group Decision and Negotiation. International Journal of Group Decision and Negotiation, 10 (2). pp. 199-215.

[Johnston *et al.* 1998] Johnston, W., Mudumbai, S., Thompson, M. (1998). Authorization and Attribute Certificates for Widely Distributed Access Control. In Proceedings of IEEE 7th International Workshops on Enabling Technologies: Infrastructures for Collaborataive Enterprises (WETICE). Palo Alto, CA. USA.

[Johnston *et al*. 1999] Johnston, W., Gannon, D., Nitberg, B. (1999).Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC). Redondo Beach, California. USA.

[Kay 2003] Kay, Russel. (2003). XACML. Computerworld. May 19, 2003.

[Kilkki *et al.* 1998] Kilkki, K., Ruutu, J., Strandberg, O. (1998). High Quality and High Utilization- Incompatible Objectives for Internet? In Proceedings of IEEE/IFIP 6 th Int. Workshop on Quality of Service, Napa Valley, CA, USA.

[Knapik & Johson 1998] Knapik, M., Johnson, J. (1998). *Developing Intelligent Agents for Distributed Systems*. McGraw-Hill.

[Kägström *et al.* 1995] Kägström, B., Ling, P., Loan, C. (1995). Gemm-based Level 3 BLAS: High-Performance Model Implementations and Performance Evaluation Benchmark. Report UMINF-95.19. Department of Computing Science, Umeä University, S-901 87. Umeä, Sweden.

[Knight 1921] Knight, Frank H.(1921) *Risk, Uncertainty and Profit*.Houghton Mifflin. New York. USA.

[Kolodner 1993] Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann Publishers: San Mateo, CA, USA.

[Koutsoupias & Papadimitriou 1999] Koutsoupias, E., Papadimitriou, C. (1999). Worst-case Equilibria. In Proceedings of Symposium in Theoretical Aspects of Computer Science (STACS). Lecture Notes in Computer Science 1563. pp-404-413. Trier, Germany.

[Kowalczyk *et al.* 2002] Kowalczyk, R., Ulieu, M., Unland, R. (2002).Integrating Mobile and Intelligent Agents in Advanced e-Commerce: A Survey. In the Proceedings of the International Symposium on Multi-Agent Systems, Large Scale Complex Systems and e-Businesses (MALCEB). Erfurt, Germany.

[Lee *et al.* 2004] Lee, H., Chin, S., Lee, J., Lee, D., Chung, K., Jung, S., Yu, H. (2004). A Resource Manager for Optimal Resource Selection and Fault Tolerance Service in Grids. In Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid. Chicago, USA.

[Lewis 1995] Lewis, L. (1995). *Managing Computer Networks: A Case-Based Reasoning Approach*. Artech House: London.

[Lin & Chiu 2002] Lin, F., Chiu, S. (2002). A Less Message Mechanism for Improving Cooperative Multi-Agent Negotiation. International Conference on Information Technology & Applications (ICITA). Bathurst, Australia.

[Liotta *et al.* 2002] Liotta, A., Pavlou, G., Knight, G. (2002). Exploiting Agent Mobility for large-scale Network Monitoring. IEEE Network, Special Issue on Applicability of Mobile Agents to Telecommunications, Vol. 16, No. 3, pp. 7-15, IEEE, May/June.

[Liu *et al.* 2002] Liu, C., Yang, L., Foster, I., Angulo, D.(2002). Design and Evaluation fo a Resource Selection Framework for Grid Applications. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11). Edinburgh, Scotland.

[Liu & Foster 2004] Liu, C., Foster, I. (2004). A Constraint Language Approach to Matchmaking. In Proceedings of the 14th International Workshop on Research Issues in Data Engineering. Boston, USA.

[Lomuscio *et al.* 2000] Lomuscio, A., Wooldridge, M., Jennings, N. (2000). A classification scheme for negotiation in electronic commerce: a European perspective. International Journal of Group Decision and Negotiation, 12(1):31—56.

[Lorch *et al.* 2003a] Lorch, M., Adams, D., Kafura, D., Koneni, M., Rathi, A., Shah, S. (2003). The PRIMA System for Privilege Management, Authorization and Enforcement

in Grid Environments. In Proceedings of the Fourth International Workshop on Grid Computing (Grid 2003). Phoenix, USA.

[Lorch *et al.* 2003b] Lorch, M., Proctor, S., Lepro, R., Kafura, D., Shah, S. (2003). First Experiences Using XACML for Access Control in Distributed Systems. ACM XML Security Workshop. Fairfax, VA, USA.

[Luce & Raiffa 1989] Luce, D., Raiffa, H. (1989). *Games and decisions: introduction and critical survey*. Dover Publications.

[Luck *et al.* 2005] Luck, M., McBurney, P., Shehory, O., Willnott, S. (2005). Agent technology roadmap – a roadmap for agent based computing. AgentLink Community.

[Ludwig *et al.* 2003] Ludwig, H. Keller, A., Dan, A, King, R., Franck, R. (2003). Web Service Level Agreement (WSLA) Language Specification. IBM Corporation.

[MacLaren *et al.* 2004] MacLaren, J., Sakellariou, R., Krishnakumar, K., Garibaldi, J., Ouelhadj, D. (2004) Towards Service Level Agreement Based Scheduling on the Grid. In the proceedings of the Workshop on Planning and Scheduling for Web and Grid Services, art of the 14th International Conference on Automated Planning & Scheduling (ICAPS), Whistler, British Columbia, Canada.

[Maes 1990] Maes, P. (1990). *Designing Autonomous Agents*. Cambridge. MA. MIT Press.

[Menascé *et al.* 1994] Menascé, D., Almeida, V., Dowdy, L. (1994). *Capacity Planning and Performance Modeling*. Ed. Prentice Hall.

[Montani & Portinale 2005] Montani, S., Portinale, L. (2005) Case Based Representation and Retrieval with Timed Dependent Features. In the proceedings of the International Conference on Case Based Reasoning (ICCBR). Lecture Notes in Computer Science 3620, H. Munoz-Avila, F. Ricci eds., Springer-Verlag, 353-367. Chicago, USA

[Moreno-Seco et. al 2003] Moreno-Seco, F., Mico, L., Oncina, J. (2004). A new classification rule based on nearest neighbour search. In Proceedings of the International Conference on Pattern Recognition (ICPR). (IV: 408-411). Cambridge, United Kingdom.

[Moore *et al.* 2001] Moore, S., Cronk, D., London, L. Dangarra, J. (2001). Review of Performance Analysis Tools for MPI Parallel Programs. Lecture Notes in Computer Science vol 2131. pp 241-248.

[Nassif *et al.* 2004] Nassif, L. N., Ahmed, M., Nogueira, J.M., Impey, R. (2004). Negotiation Process for Resource Allocation in Grid Using a Multi-Agent System. Proceedings of the International Workshop on Mobility Aware Technologies and Applications (MATA). (Lecture Notes in Computer Science 3284) Springer: Florianópolis, Brasil.

[Nassif *et al.* 2004b] Nassif, L. N., Ahmed, M., Nogueira, J.M., Karmouch, A., Impey, R. (2004). Agent-based negotiation for Resource Allocation in Grid. 2nd International Workshop on Middleware for Grid Computing (MGC). ACM International Conference Proceeding Series. Vol. 76. Middleware. Toronto, Canadá.

[Nassif *et al.* 2005a] Nassif, L., Nogueira, J., Karmouch, A., Ahmed, M., Andrade, V. (2005). Job Completion Prediction in Grid Using Distributed Case-based Reasoning. In Proceedings of the 14th IEEE International Workshops on Enabling Technologies:

Infrastructures for Collaborative Enterprises (WETICE). Workshop on Emerging Technologies for Next generation GRID (ETNGRID). Linköping, Sweden.

[Nassif *et al.* 2005b] Nassif, L., Nogueira, J., Ahmed, M., Karmouch, A., Impey, R. (2005). Agent-based negotiation for Resource Allocatin in grids. In Proceedings of the 3rd Workshop on Grid Computing and Applications (WGCA). Petrópolis, Brasil.

[Nassif *et al.* 2006] Nassif, L., Nogueira, J., Karmouch, A., Ahmed, M., de Andrade, F. (2006). A multi-agent based approach to distributed job completion prediction for grid computing environments. Journal Concurrency and Computation: Practice and Experience Special Issue. Ed. Wiley & Sons. (aceito para publicação).

[NGS 2005] National Grid Service Portal. http://www.ngs.ac.uk. Visitado em janeiro/2005.

[Nishandar 2004] Nishandar, A. (2004). Grid-fabric interface for job management in SAM_Grid, a distributed data handling and job management system for high energy physics experiments. Master's thesis. The University of Texas at Arlington.

[Nudd *et al.* 2000] Nudd, G., Kerbyson, D., Papefstathious, E., Perry, S., Harper, J., Wilcox, D.(2000). PACE – A toolset for the Performance Prediction of Parallel and Distributed Systems. The International Journal of High Performance Computing Applications, Volume 14, N0. 3, Fall 2000, pp.228-251. Sage Publications.

[Overeinder *et al.* 2002] Overeinder, B. Wijngaards, N., Steen, M.van, Brazier, F. (2002). Multi-agent Support for Internet-Scale Grid Management. Artificial Intelligence and Simulated Behaviour (AISB) Symposium on Artificial Intelligence and Grid Computing, London, United Kingdom.

[Paula Filho 2001] Paula Filho, Wilson de Pádua. (2001). Engenharia de Software – Fundamentos, Métodos e Padrões. Editora LTC.

[Payne *et al.* 2000a] Payne, L., Hahn, S., Lewis, M., Sycara, K. (2000). Agent-Based Team Aiding in a Time Critical Task. In Proceedings of the Thirty-Third Hawaii International Conference on System Sciences (HICSS33). Island of Maui, Hawaii.

[Payne *et al.* 2000b] Payne, T., Sycara, K., Lewis, M. (2000). Varying the User Interaction within Multi-Agent Systems. Autonomous Agent 2000. Barcelona, Spain.

[Pham & Karmouch 1998] Pham, V., Karmouch, A. (1998). Mobile Software Agents: An Overview. IEEE Communications Magazine. 36 (7), 26-37.

[Pinedo 2002] Pinedo, M. (2002). Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, 2nd Edition.

[Pirker *et al.* 2004] Pirker, M., Berger, M., Watzke, M. (2004). An Approach for FIPA Agent Service Dicovery in Mobile Ad Hoc Environments. In Proceedings of Workshop on Agents for Ubiquitous Computing (Ubiagents04). New York, USA.

[Permis 2005] Privilege and Role Management Infrastructure Standards Validation. www.permis.org. Visitado em Janeiro/ 2005.

[Prasad 2003] Prasad, B. (2003). Intelligent Techniques for e-commerce. Journal of Electronic Commerce Research. Vol 4. No, 2.

[Rahwan *et al.* 2003] Rahwan, I., Sonenberg, L., Dignum, F. (2003). Towards Interest-based Negotiation. In Proceedings of the Second International Conference on

Autonomous Agents and Multi-Agent Systems. ACM Press, pp 773-780. Melbourne, Australia.

[Raiffa 1982] Raiffa, H. (1982). *The Art and Science of Negotiation*. Harvard University Press. 1982.

[Raman *et al.* 1998] Raman, R., Livny, M., Solomon, M. (1998). Matchmaking: Distributed Resource Management for High Throughput Computing. In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, p.140, Chicago, USA.

[Raman *et al.* 2003] Raman, R., Livny M., Solomon, M. (2003). Policy Driven Heterogenenous Resource Co-Allocation with Gangmatching. In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing. Seattle, USA.

[Rezende 2003] Rezende, S. (2003). *Sistemas Inteligentes – Fundamentos e Aplicações*. Ed. Manole.

[Russel & Norvig 2003] Russel, S., Norvig, P. (2003). *Artificial Intelligence – A Modern Approach*. Second Edition. Prentice Hall.

[Romberg 2000] Rombert, M. (2000). The Unicore Grid Infrastructure. Scientific programming, 10 (2002), 2, p. 149.Special Issue on Grid Computing.

[Sahai *et al.* 2003] Sahai, A., Graupner, S., Machiraju, V., Moorsel, A. (2002). Specifying and Monitoring Guarantees in Commercial Grids through SLA. In Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid). IEEE Computer Society. Tokyo, Japan.

[Setten *et al.* 2002] Setten, M., Veenstra, M., Nijholt, A. (2002). Prediction Strategies: Combining Prediction Techniques to Optimize Personalization. In Proceedings of the TV-02: 2nd Workshop on Personalisation in Future TV. Malaga, Spain.

[Shaaf 1996] Schaaf, J. Fish and shrink. A next step towards efficient case retrieval in large scale case bases. Proceedings of the Third European Workshop on Advances in Case-Based Reasoning. Lausanne, Switzerland, 1996, in: Lecture Notes in Artificial Intelligence 1168, Springer Verlag, 1996, pp.362-376.

[Shen *et al.* 2002] Shen W., Yangsheng, L., Ghenniwa, H., Wang, C. (2002). Adaptive Negotiation for Agent-based Grid Computing. Proc. In Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS). Workshop on Agentcities: Challenges in Open Agent Environments, pp. 32-36, Bologna, Italy.

[Shintani *et al.* 2000] Shintatni, T., Ito, T., Sycara, K. (2000). Multiple Negotiations among Agents for a Distributed Meeting Scheduler. In the Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS). pp 435-436. Boston, MA, USA.

[Smith *et al.* 1998] Smith, W., Foster, I., Taylor, V. (1998). Predicting Application Run Times Using Historical Information. In Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science 1459, Orlando, USA.

[Song & Hwang 2004] Song, S., Hwang, K. (2004). Trusted Grid Computing with security assurance and resource optimization. In Proceedings of the 17th International Conference on Parallel and Distributed Computating Systems. San Francisco, USA.

[Sosa *et al.* 2002] Sosa, C., Tu, Z., Fast, P. (2002). Some Practical Suggestions for Performing NCBI Blast Benchmarks on a pSeries 690 System. IBM@serverPerformance Technical Report. IBM Redpapers.

[Stamoulis *et al.* 2000] Stamoulis, G., Kalopsikakis, D., Kyrikoglou, A., Courcoubetis, C. (2000). Efficient Agent-Based Negotiation for Telecommunications Services. In Proceedings of Globecom. Rio de Janeiro, Brasil.

[Stix 2001] Stix, G. (2001). The Triumph of the Light. American Scientific. Jan;284(1):80-86.

[Sun 2003] Sun's XACML Implementation. http://sunxacml.sourceforge.net/. Visitado em dezembro/2003.

[Sun & Wu 2003] Sun, X., Wu, M. (2003). Grid Harvest Service: A System for Long-term, Application-level Task Scheduling. In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS). IEEE Computer Society. Nice, France.

[Sycara 1998] Sycara, K. (1998). Multiagent Systems. AI Magazine 19(2): 79-92.

[Tanenbaum 2003] Tanenbaum, A. (2003). *Computer Networks*. 4th Ed. Prentice Hall PTR.

[Taylor *et al.* 2003] Taylor, V., Wu, X., Stevens, R. (2003). Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications. ACM SIGMETRICS Performance Evaluation Review, Volume 30, Issue 4, March 2003.

[Telescience 2005] Telescience: A Collaborative environment for Telemicroscopy and remote science. http://telescience.ucsd.edu. Visitado em janeiro/2005.

[TeraGrid 2005] http://www.teragrid.org/about/index.html. Visitado em janeiro/2005.

[Terplan & Morreale 2000] Terplan, K., Morreale, P. (2000). *The Telecommunications Handbook*. IEEE Press.

[Thompson *et al.* 2003] Thompson, M., Essiari, A., Keahey, K., Welch, V., Lang, S., Liu, B. (2003). Fine-Grained Authorization for Job and Resource Management Using Akenti and The Globus Toolkit. Journal Concurrency and Computation: Practice and Experience 16(5):477-488.

[Varian 1999] Varian, H. (1999). *Microeconomia – princípios básicos – Uma abordagem moderna*. Editora Campus. Rio de Janeiro.

[Walsh *et al.* 2002] Walsh, W., Das, R., Tesauro, G., Kephart, J.(2002). Analyzing Complex Strategic Interactions in Multi-Agent Systems. In Proceedings of American Association for Artificial Intelligence (AAAI) Workshop on Game Theoretic and Decision Theoretic Agents, pages 109--118. Edmonton, Alberta, Canada.

[Wang & Schulzrinne 2000] Wang, X., Schulzrinne, H. (2000). An Integrated Resource Negotiation, Pricing and QoS Adaptation Framework for Multimedia Applications. IEEE Journal on Selected Areas in Communications, 18(12), 2514-2529.

[Weiss 2001] Weiss, G. (2001). *Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence*. Massachusetts Institute of Technology. 3rd Edition.

[Wilde 1999] Wilde, P. (1999). How Soft Games Can be Played. In Proceedings of the 7th European Congress on Intelligent Techniques & Soft Computing. Aachen, Germany.

[Wolski 1998] Wolski, R.(1998). Dynamically forecasting network performance using the network weather service. Journal of Cluster Computing.119-132,1998.

[Wong *et al.* 2000] Wong, W., Zhang, D., Kara-Ali, M. (2000). Negotiating with Experience. In Proceedings of the Seventeenth National Conference on Artificial Intelligence. Workshop Knowledge-Based Electronic Markets (KBEM). Austin, TX, USA.

[Wooldridge & Jennings 1995] Wooldridge, M., Jennings, N. (1995). Agent Theories, Architectures, and Languages: a Survey. Intelligent Agents, 1-22. Berlin:Springer-Verlag.

[Wroclowski 1997] Wroclowski, J. (1997). Specification of the Controlled Load network Element Service. RFC2211.

[Wu & Sun 2004] Wu, M., Sun, X. (2004). A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing. In Proceedings of the IEEE International Conference Grid Computing. Hong Kong, China.

[Zhang & Lesser 2002] Zhang, X., Lesser, V. (2002). Multi-Linked Negotiation in Multi-Agent System. In Proceedings of International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS). p.1207-1214. Bologna, Italy.

[Zlotkim & Rosenschein 1989] Zlotkin, G., Rosenschein, J. (1989). Negotiation and Task Sharing among Autonomous Agents in Cooperative Domains. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI), Volume 2, 1518 - Pages 012-017. Detroit, Michigan, USA.

Apêndice A

Projetos, ferramentas, aplicações e ambientes de teste para grades computacionais

Neste apêndice, apresentamos o estado atual da pesquisa na área de grade computacional relacionando alguns exemplos de: fóruns abertos sobre grade computacional; *middleware* de grade computacional mais populares; ferramentas de escalonamento em grade computacional; ferramentas para desenvolvimento de portais para grade computacional; ambientes para programação em grade computacional; ambientes de teste de grade computacional; e aplicações para grades computacionais.

Tabela A- 1: Consórcios e fóruns abertos sobre grade computacional

Nome	Descrição
Global Grid Forum	O *Global Grid Forum* (GGF) é formado por uma comunidade de desenvolvedores, vendedores e usuários para definir padrões globais para a comunidade de grade computacional. O GGF engloba cerca de 400 organizações atuantes em aproximadamente 50 países. O *site* do GGF é www.ggh.org
OASIS	A *Organization for the Advancement of Structured Information Standards* (OASIS) é um consórcio sem fins lucrativos parar o desenvolvimento, convergência e adoção de padrões e-business. OASIS tem representações de mais de 600 organizações em aproximadamente 100 países. O *site* do OASIS é www.oasis-open.org.

Enterprise Grid Alliance	A *Enterprise Grid Alliance* (EGA) é uma organização aberta, sem fins lucrativos formada para desenvolver soluções de grade computacional, acelerando o desenvolvimento da computação em grade nas empresas. EGA está definindo requisitos para o desenvolvimento de aplicações comerciais nos ambientes de grade computacional. O *site* de EGA é www.gridalliance.org.

Tabela A- 2: *Middleware* para grade computacional

Nome	**Descrição**
Globus	A Aliance Globus é uma colaboração internacional que conduz pesquisa e desenvolvimento para criar tecnologias fundamentais de grade computacional. Globus é um *software* de código aberto usado para construir sistemas e aplicações de grade computacional. O *site* de Globus é www.globus.org.
Alchemi	Alchemi é um arcabouço para grade computacional que permite agregar facilmente o poder computacional de máquinas conectadas à Internet e a intranets em um supercomputador virtual e desenvolver aplicações para executar na grade computacional. O *site* de Alchemi é http://www.alchemi.net/.
Legion	Legion é um metasistema de *software* orientado a objetos projetado para um sistema de milhões de servidores e trilhões de objetos conectados com circuitos de comunicação de alta velocidade. Legion funciona como um *middleware* que inclui diferentes arquiteturas, sistemas operacionais e localizações físicas. Legion paraleliza problemas complexos e roda-os mais eficientemente superando problemas de conflito de plataformas, linguagens e falhas de *hardware*. Legion escalona e distribui processos em servidores apropriados e disponíveis e retorna resultados dando a ilusão de que o trabalho está sendo feito em uma única máquina virtual. O *site* de Legion é http://legion.virginia.edu/.
NorduGrid	O *middleware* NorduGrid (ou *Advanced Resource Connector, ARC*) é uma solução de código aberto que permite a produção de grades computacionais de qualidade. ARC implementa os serviços fundamentais de grade computacional tais como serviço de informação, descoberta de serviços, monitoração, submissão e gerenciamento de *job*, *brokering* e gerenciamento de recursos e dados. O *site* de NorduGrid é http://www.nordugrid.org.
Unicore	UNICORE (*Uniform Interface to Computing Resources*) oferece um sistema de grade computacional que inclui *software* cliente e servidor. UNICORE faz a distribuição de computação e recursos de dados disponíveis de forma segura na Internet e em intranets. O *site* do UNICORE é http://unicore.sourceforge.net/ e http://www.kfa-juelich.de/unicore/
InteGrade	InteGrade é um *middleware* para grade computacional que permite o uso do poder de computação ocioso das estações de trabalho. O InteGrade é estruturado pelo agrupamento de *clusters*. O Projeto InteGrade é uma iniciativa de várias universidades com o objetivo de construir um novo

	middleware de grade computacional orientado a objetos. O InteGrade provê suporte para aplicações paralelas, segurança e ambiente de desenvolvimento integrado. O *site* do InteGrade é http://integrade.incubadora.fapesp.br/.
OurGrid	OurGrid é um *middleware* para grade computacional que está em funcionamento desde dezembro de 2004. O poder computacional é fornecido por recursos ociosos de todos os participantes e é compartilhado de forma que aqueles que contribuem mais obtêm mais quando necessitam. Atualmente a plataforma executa aplicações que não se comunicam durante a execução, como por exemplo, simuladores, mineração e busca de dados. O *testbed* Ourgrid conta atualmente com mais de 300 máquinas em cerca de 20 diferentes locais. O *site* do OurGrid é http://www.ourgrid.org/.

Tabela A- 3: Escalonadores de *jobs*

Nome	**Descrição**
Grid Service *Broker* – GridBus	O projeto chamado GRIDBUS (**GRID** computing and **BUS**iness technologies) está engajado na criação de especificações de código aberto, arquiteturas e uma implementação de uma grade computacional orientado a serviço para aplicações eScience e eBusiness. O *site* de GridBus é http://www.gridbus.org/.
Nimrod/G	Nimrod é uma ferramenta que gerencia a execução de estudos paramétricos através de computadores distribuídos. Ele se responsabiliza pelo gerenciamento de um experimento assim como a distribuição de arquivos para sistemas remotos, executando a computação e obtendo os resultados. EnFuzion é uma versão comercial de Nimrod. Quando um usuário descreve um experimento para Nimrod, ele desenvolve um plano que descreve os parâmetros, os valores e os comandos necessários para executar o trabalho. O sistema usa essa informação para transportar os dados e escalonar o trabalho para a primeira máquina disponível [Abramson *et al.* 1995]. O *site* de Nimrod é http://www.csse.monash.edu.au/~davida/nimrod/.
Condor/G	O objetivo do projeto Condor é desenvolver, implementar e avaliar mecanismos e políticas que suportem computação de alto *throughput* em grandes coleções de recursos computacionais. Condor é um sistema de gerenciamento de carga de trabalho especializado para *jobs* de computação intensiva. Assim como outros sistemas com características totalmente *batch*, Condor fornece mecanismos de enfileiramento de *job*, política de escalonamento, esquema de prioridade, monitoração de recurso e gerenciamento de recurso. Usuários submetem seus *jobs* de forma serial ou paralela e Condor os coloca em uma fila, escolhe quando e onde rodá-los baseado em uma política, cuida do progresso da execução e informa ao usuário o término da execução. Condor, além de funcionar como um sistema de enfileiramento *batch* tradicional, apresenta uma arquitetura que o permite ter sucesso em áreas onde os sistemas tradicionais de escalonamento falham. Condor pode ser usado para gerenciar um *cluster* de nós computacionais dedicados. Condor também pode ser configurado

	para utilizar o poder de CPUs que estejam ociosas. Condor pode detectar que uma máquina não está mais ativa e migrar o *job* para outra máquina disponível. Condor pode transferir dados para o usuário e redirecionar as saídas de I/O para a máquina que submeteu o *job*. O *site* de Condor é http://www.cs.wisc.edu/condor/.
NetSolve	NetSolve é um sistema cliente-servidor que permite a usuários resolver problemas científicos complexos remotamente. O sistema permite aos usuários acessarem recursos de *hardware* e *software* remotos através da rede. NetSolve pesquisa pelo recurso computacional na rede, escolhe o melhor disponível e administra tolerância a falha, trazendo o resultado para o usuário. Uma política de balanceamento de carga é usada para o sistema NetSolve assegurar bom desempenho por permitir ao sistema usar os recursos disponíveis de forma eficiente. O *site* do netsolve é www.netsolve.com.
Silver MetaScheduler	SILVER é um escalonador para grade computacional baseado em prévia reserva otimizada que permite carga de trabalho distribuída através de clusters independentes. Pode ser usado com ou sem Globus e pode prover serviços de contabilização de dados permitindo aos usuários submeterem *jobs* distribuídos sem aprenderem novos comandos ou sintaxe de submissão de *jobs*. SILVER permite total autonomia por cluster respeitando as políticas e prioridades locais. O *site* de Silver é http://www.supercluster.org/projects/silver/index.html.
Grasp	O projeto *Grid-based Application Service Provision* (GRASP) objetiva estudar, projetar, desenvolver e validar uma nova infra-estrutura de sistemas avançados para adicionar valor e suportar modelos inovadores para *Application Service Provision* (ASP) baseado nos conceitos de Organizações Virtuais. GRASP maximiza os benefícios da infra-estrutura ASP clássica ao virtualizar os recursos e gerenciar Organizações Virtuais, ao potencializar aplicações em infra-estrutura de grade heterogênea, ao implementar facilmente a lógica de negócios de aplicações, ao oferecer um modelo customizado pago-pelo-uso (com monitoração de SLA e contabilização), ao oferecer serviços de segurança em um ambiente integrado e ao otimizar a utilização de recursos de clientes ASP. O *site* de Grasp é http://eu-grasp.net/english/default.htm.

Tabela A- 4: Ferramentas para desenvolvimento de portais de grade computacional

Nome	**Descrição**
GridScape	Gridscape é uma ferramenta que permite a rápida criação de portais de *testbeds* que sejam interativos e dinâmicos (sem qualquer esforço de programação). Gridscape objetiva fornecer uma solução para aqueles que precisam criar um portal *testbed* mas não necessariamente possuem tempo e recursos para construí-lo sozinho. O *site* de GridScape é http://www.gridbus.org/gridscape/.

IC2D	IC2D (*Interactive Control and Debugging of Distribution)* é um ambiente gráfico para monitoração remota de aplicações distribuídas e de metacomputação. IC2D possui visualização gráfica e permite a ação de migração de objetos remotos utilizando *drag and drop*. Como está sendo interfaceado com Jini e Globus, ele também pode servir como um bloco de construção para metacomputação e portais de computação. IC2D é construído sobre *Remote Method Invocation* (RMI) e ProActive que provê chamadas assíncronas e migração. O *site* de IC2D é http://www-sop.inria.fr/oasis/ProActive/IC2D/index.html.
Grid Desktop	Os atuais *middleware* de grade computacional mostram funcionalidade através de serviços, modelos de programação e ferramentas de linha de comandos, requerendo muito conhecimento técnico e de sistemas *middleware*. Os portais de grade computacional escondem muito dessas complexidades e permitem que usuários acessem a grade computacional facilmente. No entanto, o usuário está mais acostumado com interfaces tipo KDE (K *Desktop Environment*), Gnome e Windows. O Java CoG Kit Grid Desktop tenta fornecer à comunidade de usuários menos técnica, um workspace centrado no usuário que melhore o paradigma de desktop do sistema operacional com conceitos de grade computacional. O *site* do Grid desktop é http://www.cogkit.org/wiki/cog/moin.cgi/Projects/ANL/Desktop.
G-monitor	G-Monitor foi desenvolvido para interfacear com *Grid Resource Broker* (GRB) e fornecer para o usuário a habilidade de não somente monitorar e controlar a execução de aplicações *jobs* em Global Grids mas também fornecer funcionalidades tais como descoberta de recurso ao nível da grade computacional, economia em grade computacional, algoritmos de escalonamento baseado em economia de grade computacional. G-Monitor fornecer uma interface Web para o usuário permitindo que ele acesse o sistema a partir de qualquer localização. O usuário não precisa se preocupar em utilizar comandos de execução remota e customizar a interface gráfica do monitor para visualização remota, o que pode se tornar complexo caso hajam restrições de segurança. G-Monitor pode interagir com Gridbus, Nimrod e GRB. Tais *brokers* usam serviços de *middleware*s tais como Globus e Legion. O *site* de G-monitor é http://www.gridbus.org/gmonitor/.
GridPort	O Grid Portal Toolkit (GridPort) é uma coleção de tecnologias projetadas para auxiliar no desenvolvimento de portais científicos em grids computacionais. GridPort usa tecnologias portáveis e padronizadas para prover serviços de informações que os portais podem acessar e incorporar. O *site* de GridPort é https://gridport.npaci.edu/.

Tabela A- 5: Ambientes de programação para grade computacional

Nome	Descrição
Ninf	Ninf é um projeto japonês de desenvolvimento de um *middleware* de desenvolvimento de programação que permite a usuários acessar vários recursos tais como *hardware*, *software* e dados em

	grade computacional com uma interface fácil de usar. Ninf-G é um *software* de código aberto que suporta o desenvolvimento e execução de aplicações para grade computacional usando *Grid Remote Procedure Call* (GridRPC) em recursos de computação distribuída. O *site* de Ninf é http://ninf.apgrid.org/.
Cactus	Cactus é um ambiente de resolução de problemas, de código aberto permitindo computação paralela através de diferentes arquiteturas e desenvolvimento de código colaborativo entre diferentes grupos. O nome Cactus vem do projeto de um núcleo central (ou "*flesh*") que conecta módulos de aplicações (ou "*thorns*") através de uma interface extensiva. *Thorns* podem implementar o desenvolvimento científico customizado ou aplicações de engenharia tais como dinâmica dos fluidos computacionais. Cactus pode ser utilizado para as seguintes tarefas: 1) executar programação paralela nas linguagens F77, F90, C, C++; 2) executar em diversas arquiteturas e sistemas operacionais; 3) desenvolver código na máquina mais convenientemente disponível, por exemplo, estações de trabalho ou laptop; 4) mover código para um supercomputador (como Cray T3E ou Origin); 5) engajar em uma computação cluster de alta performance; 6) transformar um conjunto de PCs interligados em rede em um cluster; 6) trabalhar com colaboradores no mesmo código e evitar de ter programas fragmentados. O *site* do Cactus é http://www.cactuscode.org/
MetaMPICH	MetaMPICH (*Flexible coupling of Heterogeneous MPI Systems*) é uma biblioteca para aplicações distribuídas usando *Message Passing Interface* (MPI) para comunicação inter-processo. O *site* do MetaMPICH é http://www.lfbs.rwth-aachen.de/~martin/MetaMPICH/metaframe.html.
GrADS	O objetivo do projeto *Grid Application Development Software* (GrADS) é simplificar computação heterogênea distribuída explorando problemas científicos para fazer o desenvolvimento de aplicações grid uma prática simples. Destacam-se 3 sub-projetos dentro de GrADS: 1) GrADSoft – é uma arquitetura de *software* projetada para suportar monitoração de desempenho e adaptação; 2) MicroGrid – é um toolkit que emula uma variedade de configurações grids e comportamentos dinâmicos de recursos. Tais estudos de recursos controlados caracterizam a capacidade do *software* grid em adaptar às condições do ambiente computacional; e 3) O projeto VGrADS (*Virtual Grid Application Development Software*) desenvolve ferramentas que simplificam e aceleram o desenvolvimento de aplicações e serviços de grade computacional. O *site* do GrADS é http://www.hipersoft.rice.edu/grads/index.htm
CoG Kit	*Commodity Grid* (CoG) Kits permite a usuários, desenvolvedores e administradores de grade computacional, o uso, a programação e a administração de grades computacionai a partir de um arcabouço de alto nível. CoGs são mais do que uma interface para Globus. Eles permitem que programadores grid usem as tecnologias *Commodities* e as vantagens da grade computacional.

	Sem CoG a cada nova versão do Globus, que pode envolver mudança de protocolos, a revisão na programação é intensa. CoG facilita a transição entre versões Globus e a manutenção do desenvolvimento de aplicações Java que rodavam no Globus. Exemplo de CoG Kits são Java e Phyton. Tais CoG Kits estendem o uso de Globus ao fornecer acesso às características avançadas de Java tais como eventos e objetos para a programação para grade computacional. O *site* do CoG Kit é http://www.cogkit.org/.

Tabela A- 6: Ambientes de teste de grade computacional

Nome	Descrição
WWG (*World Wide Grid*)	Uso de aplicações, recursos e tecnologias de grupos e instituições de várias partes do mundo para demonstrar os desafios da computação de alto desempenho, principalmente os relacionados com aplicações de dados intensivos e distribuídos geograficamente. Nesse ambiente é demonstrada a análise de aplicações onde foi desenvolvido um diretório de dados e base de dados remota (por exemplo, banco de dados de proteínas) e mecanismos de acesso através de sistema de *brokering* (Gridbus Data-Grid scheduler). O *broker* realiza descoberta e extração de conjunto de dados da origem mais próxima e executa *jobs* para recursos ótimos. O *broker* analisa se processa o *job* em um recurso onde os dados estão disponíveis movendo o código da aplicação ou movendo dados para onde a aplicação está disponível ou movendo dados e aplicações para um recurso computacional adequado. O ambiente faz uso G-monitor, Gridbus data grid *broker*. O *testbed* é formado pelos seguintes participantes: IBM-BelleGrid, ApGrid, PRAGMA-Grid,Australian-Grid,Spain-IRIS-Grid, N*GridKorea, ThaiGrid, SingaporeGrid,IGrid-India,ItalianGrid,UK-Grid, NRC-CanadaGrid, BrazilianGrid, DutchGrid, SunGrid, JapanGrid, ANL, e UCSD. O *site* de WWG é http://gridbus.cs.mu.oz.au/sc2003/
NASA Information Power Grid (IPG)	NASA's *Information Power Grid* (IPG) é um grid de dados e computação de alto desempenho que integra computadores, base de dados e instrumentos geograficamente distribuídos. Assim como o fornecimento de eletricidade, IPG entrega poder computacional para quem precisa, não importando onde tal poder computacional esteja localizado. Os engenheiros e cientistas da NASA têm usado o IPG com sucesso em uma variedade de projetos. O *site* de IPG é http://www.ipg.nasa.gov/.
EuroGrid	O projeto EUROGRID foi financiado do ano 2000 a 2004 e demonstrou o uso de grades computacionais em comunidades industriais e científicas selecionadas para destacar os benefícios do uso de grades computacionais. O *software* utilizado no projeto EUROGRID foi o Unicore explorando temas tais como transferência de dados eficientes, *broker* de recurso, serviços ASP, acoplamento de aplicações e acesso interativo. O *site* do

	EuroGrid é http://www.eurogrid.org/.
TeraGrid	TeraGrid é um infra-estrutura científica aberta combinando recursos em oito locais de parcerias para criar um recurso computacional integrado e persistente. O desenvolvimento de TeraGrid foi completado em 2004 trazendo 40 teraflops de poder computacional e aproximadamente 2 petabytes de armazenamento rotativo além de análise e visualização de dados em produção, interconectando de 10-40 gigabits/segundo via uma rede nacional dedicada. O *site* de TeraGrid é http://www.teragrid.org/about/index.html.

Tabela A- 7: Aplicações para grade computacional

Nome	**Descrição**
NCGrid	O seqüenciamento de genomas inteiros a partir de plantas, ratos e humanos é o ponto de partida para a nova biologia do século 21. O objetivo da nova biologia é, no primeiro momento, alcançar a compreensão da vida e a partir disso criar predição e prevenção de doenças. A ciência da informação, a modelagem computacional e a mineração e análise de dados irá ter um papel fundamental nessa nova biologia. O projeto da Carolina do Norte Biogrid irá prover acesso para a infra-estrutura, armazenamento de dados e computação para pesquisadores de bioinformática. Tal projeto usa Globus e Avaki como *middleware* de Grid. O *site* de NCGrid é http://www.ncbiogrid.org/about/index.html.
NEES	NEES Consortium, (NEESinc) é um consórcio sem fins lucrativos para pesquisa sobre *Network for Earthquake Engineering Simulation* (NEES). A pesquisa no programa NEES visa avança a engenharia de terremotos através da integração de experimentos, teorias, dados e simulação baseadas em modelos. NEES inclui as 15 maiores instalações sobre engenharia de terremotos existentes. No Projeto NEES há um conjunto de aplicações que rodam usando um *software* de grade computacional chamado NEESgrid. Dentre as aplicações destacam-se NEESdaq (programas para aquisição de dados e formatação), FlexTPS (*software* projetado para permitir a visão remota e controle robótico de vídeo ao vivo pela Internet), NTCP para Matlab (fornecem acesso a programas para executar uma simulação em três sites em experimentos), NESStpm (facilita observação remota de vídeos e imagens estáticas via interface web), FEDEASLab (ferramenta Matlab para simulações de estruturas não-lineares sob condições estáticas e transientes), Opensees (um arcabouço para desenvolver aplicações para simular o desempenho de sistemas estruturais sujeitos a terremotos), Simcor (um Multi-Site de coordenação de simulação pseudo-dinâmica). O *site* do projeto NEES é http://it.nees.org/.
GeneGrid	GeneGrid é um projeto de bioinformática onde através de um portal, workflows podem ser gerados e submetidos para execução. É usado o Globus Toolkit 3 para permitir o acesso de aplicações às bases de dados específicas do projeto GeneGrid. O *site* do GeneGrid é

	http://www.qub.ac.uk/escience/projects/genegrid/.
Geodise	*Grid Enabled Optimisation and Design Search for Engineering* (Geodise) é um programa de pesquisa envolvendo equipes multi-disciplinares para desenvolver pesquisa de projeto de engenharia e otimização envolvendo dinâmica dos fluidos. A aplicação inicial irá focar no uso da dinâmica dos fluidos computacionais para pessoas envolvidas no projeto de superfícies aerodinâmicas como asas de aviões, carros de corrida, turbinas, etc. Geodise oferece uma nova dimensão nesses complicados processos tornando disponíveis as informações para projetistas. O *middleware* Globus é utilizado no ambiente do projeto. O *site* de Geodise é http://www.geodise.org/index.htm.
Aplicações diversas	*Basic Local Alignment Search Tool* (BLAST) é a ferramenta mais usada para calcular similaridade de seqüência. A comparação de seqüências de nucleotídeo ou proteína a partir de organismos iguais ou diferentes é uma ferramenta poderosa para a biologia molecular. Ao encontrar similaridades entre seqüências, os cientistas podem inferir a função dos genes seqüenciados, prevendo novos membros da família dos genes e explorando os relacionamentos evolucionários. O *site* do Blast é http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=handbook.chapter.610 - *Exonarate*. Uma ferramenta genérica para alinhamento de seqüência. O *site* do Exonarate é http://www.ebi.ac.uk/~guy/exonerate/. -SSAHA. *Sequence Search and Alignment by Hashing Algorithm*. Pesquisa a seqüência e alinhamento por algoritmo Hash. O *site* do SSAHA é http://www.sanger.ac.uk/Software/analysis/SSAHA/. - Wise2. Wise2 é um pacote focado na comparação de bio-polímeros, comumente seqüência de DNA e de proteína. O *site* do Wise2 é http://www.ebi.ac.uk/Wise2/documentation.html. - ClustalW - ClustalW é um programa de alinhamento de seqüência múltipla de propósito geral para DNA ou proteínas. ClustalW calcula a melhor combinação para as seqüências selecionada de forma que as similaridades e diferenças podem ser vistas. O *site* do ClustalW é http://www.ebi.ac.uk/clustalw . - QTL Cartographer é uma suíte de programas para mapear *quantitative trait loci* (QTLs) em um mapa genético. O *site* do QTL é http://statgen.ncsu.edu/qtlcart/manual/. - MapManager é um programa que auxilia a análise de resultados dos experimentos de mapeamento genético. O *site* do MapManager é http://iubio.bio.indiana.edu:7780/archive/00000064/. - MapDisto é um programa para mapear marcas genéticas em populações experimentais. O *site* do MapDisto é http://mapdisto.free.fr/.

Apêndice B

Ambiente de teste utilizado para validação dos modelos

Este apêndice apresenta informações sobre as ferramentas que utilizamos para o desenvolvimento do MASK, as aplicações para grade computacional utilizadas para validar os modelos desenvolvidos, o ambiente computacional utilizado nos experimentos e o pacote de programas e ferramentas que constituem o MASK.

B.1 Plataforma Jade de desenvolvimento do Sistema Multiagente MASK

O protótipo MASK foi desenvolvido em Jade [Jade 2005]. Jade é um *middleware* desenvolvido pelo TILAB (Telecom Itália Lab) para o desenvolvimento de aplicações multiagente distribuídas. O ambiente dinâmico envolve pares (*peers*), que em Jade são chamados de agentes, que aparecem e desaparecem no sistema de acordo com as necessidades e requisitos da aplicação. A comunicação entre os pares possibilita que cada agente tenha papéis distintos de iniciador ou respondedor na interação. Jade é completamente desenvolvido em Java e é guiado pelos seguintes princípios: 1) interoperabilidade, pois segue os padrões estabelecidos pela *Foundation for Intelligent Physical Agents* (FIPA) e, portanto pode se relacionar com qualquer outro agente que segue tal padrão [FIPA 2002]; 2) portabilidade, fornecendo um conjunto de *application program interfaces* (APIs) que são independentes da rede e da versão do Java; e 3) facilidade de uso,

dado o conjunto de APIs disponibilizado. Cada instância do Jade *run-time* é chamado *container*. O conjunto de todos os *containers* é chamado de plataforma. Jade fornece os serviços básicos necessários para aplicações ponto-a-ponto distribuídas. Cada agente descobre outros agentes dinamicamente e comunica entre si de acordo com o paradigma ponto-a-ponto. Cada agente é identificado por um nome único e fornece um conjunto de serviços. Jade é indicado para o desenvolvimento e execução de aplicações distribuídas, máquina-a-máquina, *multi-party*, inteligentes e pró-ativas. Há diversas implementações de Jade e testes nos mais variados ambientes tais como: *General Packet Radio Service* (GPRS), *Wireless Local Area Network* (WLAN), Bluetooth e Internet.

A implementação dos modelos apresentados nesta tese foi feita utilizando tais princípios disponibilizados no ambiente de desenvolvimento Jade. O protótipo MASK é composto de 60 programas, dentre eles 8 agentes, totalizando cerca de 400 linhas de código que, em conjunto, atinge 1,5MB. São trocados 8 tipos de mensagens entre os agentes (método *invocation*). A estrutura de dados é toda feita com XML, existindo 9 arquivos XML utilizados nos mecanismos de predição, política, monitoração e decisão.

B.2 Aplicações para grades computacionais

Existem diversas aplicações para grade computacional nos campos científicos, industriais e educacionais. As aplicações científicas que demandam características do ambiente de grade computacional se concentram principalmente nas seguintes áreas: 1) Física de alta energia; 2) Medicina e Biologia; 3) Meteorologia; 4) Astrofísica; 5) Química Molecular; 6) Fusão; e 7) Engenharia.

Foi realizada uma larga pesquisa sobre as aplicações científicas que poderiam ser utilizadas nos experimentos da presente tese. Constatou-se uma grande gama de aplicações na área de bioinformática sendo muitas delas de acesso livre e disponíveis para *download*, fazendo com que optássemos por esse grupo de aplicações para proceder com os testes. Inicialmente utilizamos as aplicações BLAST e HMMER, ambas do campo de aplicações científicas da bioinformática. Devido à necessidade de compreensão dos mecanismos de funcionamento de tais aplicações, os experimentos consideraram inicialmente somente essas duas aplicações. Os resultados obtidos são suficientes para provar o funcionamento e eficiência da nossa solução.

O *broker* aqui desenvolvido pode ser utilizado por diversas aplicações, por ser ele de propósito geral. Encaixam-se nesse escopo as aplicações que são submetidas para execução via linha de comandos em *background* podendo um dos argumentos ser um arquivo cujo tamanho tem influência significativa na execução do comando completo.

No campo da bioinformática, o MASK pode ser potencialmente utilizado pelas aplicações de seqüenciamento de genomas, pois possuem os requisitos de uso do MASK, ou seja, são aplicações seguidas de argumentos e arquivo de dados para execução em modo *batch*. As aplicações para seqüenciamento de genomas apresentam geralmente um conjunto de programas com finalidades distintas para alinhamento de seqüência, pesquisa rápida de base de dados, identificação da proteína, análise do domínio, análise do padrão de seqüência de nucleotídeo, rápida identificação de padrões de seqüência em conjuntos de seqüência de larga escala. Relacionamos na Tabela A-7 do apêndice A, um conjunto de aplicações de seqüenciamento de genomas que podem ser utilizados no MASK.

Para validar MASK, utilizamos os pacotes BLAST e HMMER como aplicações de grade computacional para compor o cenário do problema tratado nesta tese, ou seja, a seleção de recursos em grade computacional. A seguir descrevemos sucintamente as aplicações BLAST e HMMER.

B.2.1 BLAST

Os programas do pacote denominado *Basic Local Align Search Tool* (BLAST) são ferramentas largamente utilizadas para comparar uma seqüência fornecida pelo usuário, denominada *query*, com inúmeras outras seqüências contidas em um banco de dados, podendo tal banco ser constituído de seqüências de nucleotídeos ou proteínas. Esta comparação tem como finalidade reportar ao usuário quais as seqüências presentes no banco de dados que apresentam maior similaridade com a seqüência *query* [Cruz 2003].

BLAST é CPU-intensivo e I/O intensivo. Entretanto, o desempenho de Blast está mais fortemente relacionado com a variação do tamanho da memória do equipamento [Gao 2004]. O pacote BLAST é formado por cinco programas descritos na Tabela B-1.

Tabela B- 1: *Flavours* BLAST

Programa	Descrição
Blastp	Compara uma seqüência de aminoácidos com uma base de dados de proteína
Blastn	Compara uma seqüência de nucleotídeo com uma base de dados de nucleotídeo

Blastx	Compara os produtos de tradução conceitual de um nucleotídeo com uma base de dados de proteína
Tblastn	Compara uma seqüência de proteína com uma base de dados de nucleotídeo dinamicamente traduzida em seis quadros de leitura
Tblastx	Compara uma seqüência de um nucleotídeo com uma base de dados de um nucleotídeo de seis quadros de traduções

Há diversas bases de dados que os programas BLAST utilizam e que estão disponíveis através de banco de dados públicos de genomas tais como o *National Center for Biotechnology Information* (NCBI) – disponível em http://www.ncbi.nlm.nih.gov/. No entanto, devido aos tamanhos de tais bases na ordem de gigabytes e a nossa restrição de espaço em disco nos experimentos, selecionamos apenas algumas bases significativas e de tamanho médio conforme descrito na Tabela B-2.

Tabela B- 2: Bases de dados BLAST utilizadas nos experimentos

Nome	Tamanho
yeast.nt	3 MB
yeast.aa	12 MB
Nr	1.100MB
Pdbaa	20MB
swissprot	174MB
Nt	
Htgs	
Nr	

Tabela B- 3: *Queries* BLAST utilizadas nos experimentos

Nome	Tamanho (bytes)
aa.129295	406
aa.231729	844
aa.p38398	2160
asHs118.fasta.txt	178.749
asHs148.fasta.txt	150.668
humrep.fsa	50.078
nt.383410	20.387
nt47.5706771	204.866
nt.4883672	103.810
nt.5706771	197.867
nt.5764416	2.143

Os comandos blast são compostos pela seguinte sintaxe: blastall –p <program> -d <database> -i <query> -e <expected value> -o <output file>.

Nos experimentos foram utilizados comandos identificados em pacotes de teste, artigos e resultados de estudos sobre o desempenho de Blast tais como os disponíveis em [Kägström *et al.* 1995] e [Sosa *et al.* 2002].

Tabela B- 4: Alguns dos comandos BLAST utilizados nos experimentos

blastall -p blastx -d nr -i nt.5706771 -e 0.1 –o out.5706771.blastx
blastall -p blastx -d nr -i nt.5706771 -e 0.1 -o out.5706771.blastx
blastall -p blastp -d nr -i aa.129295 -o out.129295.blastp
blastall -p blastp -d nr -i aa.231729 -o out.231729.blastp'
blastall -p blastp -d nr -i aa.1177466 -o out.1177466.blastp'
blastall -p blastp -d nr -i aa.p38398 -o out.p38398.blastp'
blastall -p blastx -d nr -i nt.5764416 -e 0.1 -o out.5764416.blastx'
blastall -p blastx -d nr -i nt.3283410 -e 0.1 -o out.3283410.blastx'
blastall -p blastx -d nr -i nt.4883672 -e 0.1 -o out.4883672.blastx'
blastall -p blastx -d nr -i nt.5706771 -e 0.1 -o out.5706771.blastx'
blastall -p blastp -d swissprot -i seq.fasta
blastall -p blastp -d swissprot -i seq.fasta -q '-6' -m '8' -e '1.0' -F 'S 10 1.0 1.5'

B.2.2 HMMER

O HMMer é um seqüenciador de genoma. Tal aplicativo utiliza HMMs (*Hidden Markov Models*) para identificar semelhanças em estruturas genéticas sendo tal tarefa considerada crucial em áreas como seqüenciamento e análise de amostras de DNA e proteínas. Ao representar as propriedades de uma família de seqüências na forma de estatística, o HMM torna possível buscas altamente refinadas nos maiores bancos de dados de DNA e proteínas [Eddy 2001].

HMMER é um programa intensivo em CPU. O pacote HMMER é formado por nove programas conforme descrito na Tabela B-5.

Tabela B- 5: Programas disponíveis no pacote HMMER

Programa	Descrição
Hmmalign	Alinha seqüências para um modelo existente
Hmmbuild	Constrói um modelo a partir de múltiplas seqüências de alinhamento
Hmmcalibrate	Utiliza um HMM e empiricamente determina parâmetros que são usados para fazer pesquisas mais sensíveis, calculando valores mais acurados (E-values)
Hmmconvert	Converte um arquivo modelo em diferentes formatos, incluindo um formato HMMER 2 binário compacto e emulação de perfis GCG
Hmmemit	Emite seqüências probabilisticamente a partir de um perfil HMM
Hmmfetch	Obtém um modelo único a partir de um perfil HMM
Hmmindex	Indexa uma base de dados HMM
Hmmpfam	Pesquisa uma ou mais seqüências em uma base de dados HMM
Hmmsearch	Pesquisa combinações para um HMM em uma base de dados

Tabela B- 6: Bases de dados HMMER utilizadas nos experimentos

Nome	Tamanho (bytes)
pfam_ls	648363760
swissprot	75505630
artemia.fa	1489
globins630.fa	101046

Tabela B- 7: Seqüências de alinhamento HMMER utilizadas nos experimentos

Nome	Tamanho (bytes)
Tutorial/globins50.msf	25284
Tutorial/rrm.sto	18294
Tutorial/pkinase.sto	112640
Tutorial/fn3.sto	33550

Tabela B- 8: Arquivos HMM utilizados nos experimentos

Nome	Tamanho (bytes)
Tutorial/globin.hmm	53911
Testsuite/fn3.hmm	31903
Testsuite/trace_test.hmm	4313
Testsuite/weeviterbi_test.hmm	27073
Tutorial/pkin.hmm	110154
Testsuite/%OPT.HMM%	2249

Nos experimentos foram utilizados comandos do Hmmer providos por testes que acompanham o pacote de programas denominados *testsuite* e além de comandos disponibilizados no tutorial.

Tabela B- 9: Alguns comandos HMMER utilizados nos experimentos

```
hmmalign -m %OPT.HMM% Optiontests.fa
hmmbuild -F -n foo %OPT.HMM% Optiontests.nslx
hmmcalibrate --fixed 15 %OPT.HMM%
hmmconvert -b -F %OPT.HMM% %OPT.TMP%
hmmemit -o %OPT.OUT% %OPT.HMM%
hmmfetch %OPT.HMM% fn3
hmmpfam -A 0 %OPT.HMM% Optiontests.fa
hmmsearch %OPT.HMM% Optiontests.fa
hmmcalibrate globin.hmm
hmmsearch globin.hmm swissprot
hmmsearch globin.hmm yeast.aa
hmmsearch globin.hmm yeast.nt
hmmsearch pkin.hmm swissprot
```

B.3 Ambiente de teste

O ambiente de teste utilizou equipamentos do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais conforme mostra a Tabela B-10.

Tabela B- 10: Descrição das máquinas utilizadas nos experimentos

Máquina	Quantidade Processadores	RAM (MB)	CPU (MHz)	Modelo
Alegria	1	512	1666	AMD Athlon 2000+ mod 8
Amizade	1	512	1666	AMD Athlon 2000+ mod 8
Bonete	1	2048	1800	AMD Sempron™ Processor 3000+ mod 44
Caridade	1	512	1800	AMD Athlon 2500+ mod 10
Confiança	1	512	1666	AMD Athlon 2000+ mod 8
Hulk	1	1024	1700	Intel® Pentium® 4 CPU mod 1
Ibituruna	1	512	1666	AMD Athlon 2000+ mod 8
Itapeva	1	512	1800	AMD Athlon 2500+ mod 10
Jumento	2	512	2400	Intel® Pentium® 4 CPU 2.40GHz mod 2
Montblanc	1	512	1800	AMD Athlon 2500+ mod 10
Palermo	1	1024	3000	Intel® Pentium® 4 CPU 3.00GHz mod 4
Sajama	1	1024	2000	AMD Athlon™ 64 Processor 3200+
Saudade	1	512	1666	AMD Athlon 2000+ mod 8
Tapir	2	512	2400	Intel® Pentium® 4 CPU 2.40GHz mod 2
Zebra	2	512	2400	Intel® Pentium® 4 CPU 2.40GHz mod 2

B.4 Suíte de teste

Para conduzir os experimentos, foi desenvolvido um pacote para testes da solução, denominado *suitemask*. Tal pacote é composto de aplicações para grade computacional (BLAST e HMMER), do sistema multiagente MASK, do ambiente de desenvolvimento (Jade, Java, APIs), de exemplos de submissão de *jobs*. O tamanho total do pacote é de 4 GB.

Apêndice C

Publicações e Apresentações

Este apêndice apresenta as referências bibliográficas das publicações de partes desta tese e apresenta ainda publicações realizadas durante o doutorado que não estão diretamente relacionadas ao material aqui apresentado. Também relacionamos as apresentações feitas em conferências durante o doutorado.

C.1 Publicações diretamente relacionadas à tese

Revista

Nassif, Lilian, Nogueira, José Marcos, Karmouch, Ahmed, Mohamed, Ahmed, Andrade, Flávio. Job completion prediction using case-based reasoning for grid computing environments. *Journal Concurrency and Computation: Practice and Experiences*. Ed. Wiley & Sons. 2006 (artigo selecionado do WETICE/ETNGRID e aceito para publicação).

Anais de conferências

Nassif, Lilian N., Nogueira, José Marcos S., Ahmed, Mohamed, Impey, Roger. *Negotiation process for resource allocation in Grid using a multi-agent system". International Workshop on Mobility Aware Technologies and Applications* (MATA) 2004. Lecture Notes in Computer Science 3284, Florianópolis, Brasil. 2004.

Nassif, Lilian N., Nogueira, José Marcos S., Karmouch, Ahmed, Ahmed, Mohamed, Impey, Roger, de Andrade, Flávio V. *Job Completion Prediction in Grid Using Case-Based Reasoning. In Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises* (WETICE). ETNGRID. IEEE Computer Society, Linköping, Sweden, 2005.

Nassif, Lilian N., Nogueira, José Marcos S., Karmouch, Ahmed, Ahmed, Mohamed, Impey, Roger. *Agent-based Negotiation for Resource Allocation in Grid. Proceedings of ACM/IFIP/Usenix International Middleware Conference/ 2nd International Workshop on Middleware for Grid Computing (MGC)*. Toronto. Canadá, 2004.(resumo).

Nassif, Lilian N., Nogueira, José Marcos S., Karmouch, Ahmed, Ahmed, Mohamed, Impey, Roger, *Agent-based negotiation for resource allocation in Grid.* WGCA'05 proceedings, Petrópolis, Brasil. 2005.

C.2 Publicações realizadas no doutorado – sem relacionamento direto com a tese

Anais de conferências

Nassif, Lilian N., Correia, Luiz H., Cavalcanti, Carlos. F. M. C., Nogueira, José Marcos S. ; Loureiro, Antônio A.F., Mateus, Geraldo R. *InterQoS: Strategic Enterprise Game for price and QoS negociation on the Internet. In Proceedings of Internet Charging and QoS Technology. Lecture Notes in Computer Science Series* 2511, Springer Verlag. Zurich. Switzerland. 2002.

Nassif, Lilian N., Correia, Luiz H., Cavalcanti, Carlos. F. M. C., Nogueira, José Marcos S. ; Loureiro, Antônio A.F., Mateus, Geraldo R. Arquitetura e Modelo de interações na Internet através de um jogo de estratégia. Anais do 20º. Simpósio Brasileiro de Telecomunicações, Rio de Janeiro. Brasil. 2003.

Nassif, Lilian N., Nogueira, José Marcos S., Loureiro, Antônio A.F. Agentes inteligentes em jogo de estratégia usando interface geográfica. Anais do Workshop de Jogos 2003, Salvador. Brasil. 2003. (resumo).

C.3 Apresentações em conferências

- *InterQoS: Strategic Enterprise Game for price and QoS negociation on the Internet. Swiss Federal Institute of Technology ETH.* Zurich. Switzerland. 2002.
- Arquitetura e Modelo de interações na Internet através de um jogo de estratégia. 20º. Simpósio Brasileiro de Telecomunicações, Rio de Janeiro. Brasil. 2003.
- Agentes inteligentes em jogo de estratégia usando interface gráfica. Workshop de Jogos 2003, Salvador. Brasil. 2003.
- *Negotiation process for resource allocation in Grid using a multi-agent system. International Workshop on Mobility Aware Technologies and Applications (MATA) 2004.* Florianópolis, Brasil. 2004.
- *Agent-based negotiation for resource allocation in Grid. Workshop of Grid Computing and Applications* (WGCA), Petrópolis, Brasil. 2005.
- *Job Completion Prediction in Grid Using Case-Based Reasoning. 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises* (WETICE). ETNGRID. IEEE Computer Society, Linköping, Sweden, 2005.