Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

GIStorage: um serviço de informação para grades com suporte a algoritmos de predição de desempenho

Jean Paulo Sandri Orengo

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. César Augusto Fonticielha De Rose

Porto Alegre, janeiro de 2007

Dados Internacionais de Catalogação na Publicação (CIP)

O66g Orengo, Jean Paulo

GIStorage : um serviço de informação para grades com suporte a algoritmos de predição de desempenho / Jean Paulo Orengo. – Porto Alegre, 2007.
93 f.

Diss. (Mestrado em Ciência da Computação) – Fac. de Informática, PUCRS.
Orientação: Prof. Dr. César Augusto Fonticielha De Rose.

1. Informática. 2. Algoritmos. 3. Serviços de Informação. 4. Estrutura de Dados I. De Rose, César Augusto Fonticielha.

CDD 005.73

Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS

PUCRS
Campus Central
Av. Ipiranga, 6681 - prédio 16 - CEP 90619-900
Porto Alegre - RS - Brasil
Fone: +55 (51) 3320-3544 - Fax: +55 (51) 3320-3548
Email: bceadm@pucrs.br
www.pucrs.br/biblioteca

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***GIStorage: Um Serviço de Informação para Grades com Suporte a Algoritmos de Predição de Desempenho***", apresentada por Jean Paulo Sandri Orengo, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 18/01/2007 pela Comissão Examinadora:

Prof. Dr. César Augusto Fonticielha De Rose – Orientador (a) PPGCC/PUCRS

Prof. Dr. Luiz Gustavo Leão Fernandes – PPGCC/PUCRS

Prof. Dr. Cláudio Fernando Resin Geyer – UFRGS

Homologada em 04/06/2007, conforme Ata No. 013/07 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 16 – sala 106 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Este trabalho é dedicado ao meu filho Bruno.

Agradecimentos

Agradeço minha família, cujo apoio foi fundamental para a realização deste trabalho. Ao meu filho Bruno que pacientemente soube lidar com minha ausência. À minha querida noiva Jutami, que sempre esteve ao meu lado me incentivando e apoiando. Ao meu orientador, Prof. César De Rose, que mostrou ser acima de tudo um grande amigo.

Resumo

Para alocar recursos e submeter tarefas numa grade computacional serviços de descoberta, alocação e escalonamento precisam conhecer o desempenho dos recursos. Como as tarefas serão executadas num momento futuro, estes serviços podem empregar algoritmos de predição para prever o desempenho dos recursos, melhorando a qualidade de suas decisões. Além disso, algoritmos de predição baseados em séries temporais demandam informações históricas sobre o desempenho dos recursos para prever o comportamento futuro dos mesmos. Para dar suporte a algoritmos e serviços de predição é proposto o GIStorage, um serviço de informação para grades computacionais projetado para armazenar informações sobre recursos. O GIStorage é baseado no modelo GMA, sendo estruturado como uma árvore para obter bom desempenho e armazenar grande volume de dados.

Abstract

In order to allocate resources and submit jobs to a grid, resource discovery and scheduling services need to know in advance the performance of the candidate resources. Since jobs will be executed in a future time, these services may use prediction algorithms to forecast resources performance, improving the quality of their decisions. Additionally, prediction algorithms that use time series analysis demand historical performance information to predict future behavior. To support prediction algorithms and services we propose GIStorage, a grid information service designed for storing historical performance information about resources. GIStorage is GMA compliant and is structured as a tree in order both to achieve good response times and to store large amount of information.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API	*Application Programming Interface*	28
COW	*Cluster of Workstations*	41
D	*Directory Service*	50
FOSIS	*FOrest Structured Information Service*	36
GIS	*Grid Information Service*	21
GLUE^2	*GLUE schema Extended*	42
GLUE	*Grid Laboratory Uniform Environment*	41
GMA	*Grid Monitoring Architecture*	47
IP	*Internet Protocol*	42
LAN	*Local Area Network*	57
LDAP	*Lightweight Directory Access Protocol*	47
MDS	*Monitoring and Discovery System*	34
NOW	*Network of Workstations*	41
NWS	*Network Weather Service*	26
OASIS	*Organization for the Advancement of Structured Information Standards*	40
OGF	*Open Grid Forum*	40
OGSI	*Open Grid Services Infrastructure*	36
OSI	*Open System Interconnection*	42
OV	Organização Virtual	22
PACE	*Performance Analysis and Characterisation Environment*	26
QoS	*Quality of Service*	42
Remos	*Resource Monitoring System*	26
SE	*Storage Elements*	50
S	*Sensors*	50
SLA	*Service Level Agreement*	25
SNMP	*Simple Network Management Protocol*	43
TCP	*Transmission Control Protocol*	42

TGIS	*Tree-based Grid Information Service*	36
W3C	*World Wide Web Consortium*	40
WSDL	*Web Services Description Language*	40
WSRF	*Web Services Resource Framework*	34
XML	*eXtensible Markup Language*	41

Sumário

1 Introdução

Este trabalho é dedicado ao estudo de serviços de informação para grades computacionais. Com o propósito de dar suporte a algoritmos de predição é proposta uma nova arquitetura de serviço de informação.

1.1 Motivação

Avanços nas tecnologias de redes de computadores e infra-estrutura computacional tornaram possível a construção de ambientes computacionais distribuídos de alto desempenho em grande escala. Tais ambientes, conhecidos como grades computacionais [1], são formados por recursos heterogêneos, distribuídos geograficamente e conectados pela Internet.

Grades computacionais possibilitam o compartilhamento de recursos entre diversas organizações. É justamente essa característica que torna as grades tão atrativas, porém, essa mesma característica as tornam dependentes de serviços de monitoramento capazes de realizar tanto o monitoramento quanto a descoberta de recursos. Serviços de monitoramento são usados para encontrar recursos pertencentes à grade e verificar em que estado se encontram [2]. Uma série de serviços necessitam de mecanismos de monitoramento para realizar com êxito suas tarefas, tais como: descoberta de recursos, escalonamento de tarefas, *broker*, monitor de tempo real, entre outros.

Em especial, serviços de descoberta e alocação de recursos e escalonamento de tarefas têm por objetivo escolher os recursos que melhor atendem as necessidades de desempenho da aplicação e dividir as tarefas de acordo com a capacidade de cada recurso. Tais serviços podem empregar algoritmos de predição de desempenho em suas análises para melhorar os seus resultados. Exemplos de benefícios do uso de predição na descoberta e seleção de recursos podem ser encontrados nos trabalhos de Downey [3] e Smith [4], já os resultados obtidos por Berman *at al* [5] e Yang *at al* [6] demonstram os benefícios alcançados no escalonamento.

Diversos algoritmos de predição baseiam-se em séries temporais. Os modelos matemáticos empregados podem variar desde simples técnicas como média ou mediana até técnicas mais elaboradas como modelos de auto-regressão. Entretanto, todos estes modelos requerem informações históricas sobre o desempenho dos recursos.

Para dar suporte a algoritmos e serviços de predição este trabalho propõe um serviço de informação para grades computacionais (GIS) cujo objetivo principal é armazenar informações

históricas sobre o desempenho dos recursos. Tal serviço, chamado de GIStorage, permite que serviços de predição concentrem-se somente no processamento das informações coletadas ao invés de gerenciar sensores e as informações produzidas, liberando-os de problemas relacionados a grande quantidade de dados, escalabilidade, entre outros.

Usando um serviço de informação, usuários podem acessar informações históricas sobre o desempenho dos recursos independentemente do algoritmo ou serviço de predição empregado. Além disso, organizações virtuais (OV) [1] podem usar os sensores que melhor atendem as suas políticas internas, não sendo imposta a instalação de sensores compatíveis com determinados serviços de predição. Naturalmente, tanto os sensores quanto os serviços de predição devem compreender os protocolos definidos pelo GIStorage.

1.2 Objetivos

O objetivo deste trabalho é propor uma nova arquitetura para serviços de informação para grades com suporte a algoritmos de predição. O serviço deve armazenar informações históricas sobre o desempenho dos recursos.

Para o projeto de um serviço de informação uma série de questões devem ser tratadas como heterogeneidade de recursos e informações, grande volume de dados, desempenho, escalabilidade e padronizações [7].

Devido a diversidade de recursos existentes numa grade, as informações de desempenho variam. Deve-se utilizar um modelo de informação capaz de descrever recursos heterogêneos e suas relações com os demais.

Grades computacionais possuem numerosos recursos. Para cada um destes deve-se manter informações históricas, gerando um grande volume de dados para serem armazenados. Além disso, a arquitetura deve gerenciar tais informações de forma escalável, permitindo um elevado número de recursos.

Empregam-se algoritmos de predição com intuito de melhorar o desempenho de aplicações. Um serviço de informação deve ser estruturado para obter e entregar informações aos algoritmos de maneira otimizada, reduzindo o tempo despendido com a predição.

Para que o serviço seja amplamente utilizado é necessário basear sua arquitetura em padrões para grades computacionais. O serviço deve respeitar padrões como WSDL [8] e WSRF [9], assim como as recomendações provenientes de instituições como OASIS [10], OGF [11] e W3C [12].

1.3 Estrutura do texto

O trabalho está divido em seis capítulos. No capítulo introdutório (capítulo corrente), são apresentados os objetivos e motivação para a realização deste trabalho. No segundo capítulo a importância dos serviços de predição e os requisitos dos mesmos são descritos. O Capítulo 3 trata sobre os trabalhos relacionados. No Capítulo 4 é definida a arquitetura do novo serviço, sendo elucidadas as decisões de projeto e as suposições a cerca de suas qualidades. Os resultados obtidos por meio de simulações podem ser vistos no Capítulo 5. Por fim, o sexto capítulo discorre sobre as considerações finais deste trabalho.

2 Importância e requisitos de serviços de predição

O objetivo deste trabalho é propor uma nova arquitetura para serviços de informação para grades com suporte a algoritmos de predição. Para dar suporte adequado é necessário conhecer o funcionamento, utilidade e requisitos dos algoritmos e serviços de predição.

2.1 Utilização de algoritmos de predição

Os algoritmos de predição são empregados nas várias etapas que precedem a execução de aplicações. Para melhor compreender a sua importância, são descritos abaixo os principais usos de algoritmos de predição.

A execução de uma aplicação em grade abrange uma série de etapas que vão desde a descoberta dos recursos e atribuição de tarefas até o monitoramento da execução [13]. Durante as etapas de descoberta, alocação e escalonamento podem ser empregadas técnicas de predição visando um melhor desempenho da aplicação.

A etapa inicial é a descoberta de recursos, que se caracteriza pela verificação dos recursos que estão disponíveis para o usuário. Em um primeiro momento, deve-se verificar quais recursos que o usuário tem acesso, evitando assim, analisar detalhadamente elementos que o usuário não poderá utilizar. Nas próximas etapas deve-se realizar uma análise mais detalhada, excluindo-se os recursos que não possuem os requisitos mínimos exigidos pela aplicação.

Na etapa seguinte é realizada uma seleção dos recursos visando escolher aqueles que melhor se adequam as necessidades da aplicação. Nesta etapa, deve-se levar em consideração as características dos recursos como, carga do processador, memória disponível, latência da rede, entre outras, no momento em que a aplicação irá executar, pois: (i) aplicações paralelas freqüentemente têm um tempo de execução elevado e durante esse período as características dinâmicas dos recursos podem variar significativamente em relação aos valores encontrados no momento da escolha dos mesmos; (ii) aplicações que empregam vários recursos necessitam reservá-los para um momento futuro onde será possível usá-los simultaneamente [14], porém, naquele instante as características podem ter sofrido grandes modificações. Portanto, o emprego de técnicas de predição para realizar uma seleção baseada no comportamento futuro dos recursos pode ter impacto positivo no desempenho final da aplicação [3–5, 15–17].

As informações de predição de desempenho podem ser providas pelo administrador do recurso usando um *service level agreement* [18], ou obtidas através da análise do histórico de

desempenho do recurso. Pode-se inclusive empregar este último como método na geração do SLA [6].

Logo após a seleção, é necessário fazer o mapeamento das tarefas nos recursos alocados. O objetivo desta etapa é gerar um mapeamento de forma que otimize o desempenho segundo critérios do usuário. Para comparar diferentes tipos de mapeamentos e eleger aquele que atinge melhor desempenho pode-se modelar matematicamente a aplicação de acordo com sua necessidade. Por exemplo, o tempo de execução de uma tarefa pode ser representado pelo número de instruções que irá executar e pelo número de mensagens que irá enviar ou receber. Conhecendo-se a capacidade de processamento e a banda de rede disponível para cada recurso é possível calcular uma estimativa do tempo de execução de uma tarefa e assim eleger o mapeamento que finalizará a aplicação em um menor tempo. Como as tarefas executarão em um momento posterior a etapa de mapeamento, o uso de algoritmos de predição pode auxiliar na determinação das características dos recursos no instante em que a aplicação irá executar. Resultados obtidos por Berman *at al* [5] e Yang *at al* [6] demonstram alguns dos benefícios do uso de predição no auxílio ao escalonamento.

O uso de técnicas de predição não se restringe aos exemplos acima, incluí-se também previsão de tempo de execução de aplicações baseadas na carga do processador [16], previsão do tráfego do próximo ano de um *backbone* [19], previsão do tempo de espera na fila do escalonador [3], entre outros.

2.2 Algoritmos e serviços de predição

Diversos algoritmos e serviços de predição foram propostos ao longo dos anos, dentre os quais se destacam na literatura o Remos [20], NWS [21] e o PACE [15]. Além destes, o algoritmo proposto por Yang *at al* [6] merece atenção por adotar uma abordagem diferente dos anteriores.

2.2.1 *Network Weather Service* - NWS

O NWS foi projetado para prever o desempenho de recursos de uma grade [21], sendo usado pelo AppLes [5] para auxiliar na seleção e escalonamento de aplicações paralelas.

A arquitetura do NWS é composta por sensores, repositórios e algoritmos de predição. Os sensores são processos que monitoram os recursos e geram informações sobre o desempenho corrente dos mesmos. Tais informações são posteriormente armazenadas em um repositório distribuído que é acessado pelos algoritmos para realizar a predição de desempenho (ver Figura 1).

Cada recurso executa um sensor de máquina e de rede [22]. O sensor de máquina produz

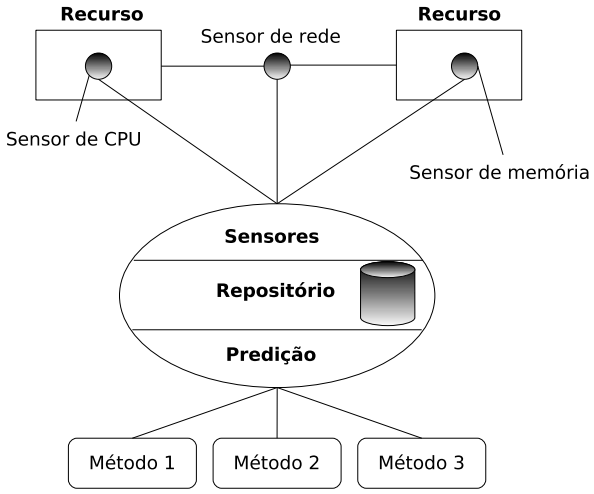

Figura 1 – Arquitetura do NWS.

informações sobre a porcentagem de CPU sendo usado, o número de processos de sistema e de usuário, espaço em disco disponível e tamanho de memória física livre. Já o sensor de rede informa a latência e taxa de transferência (*throughput*) da rede.

As informações coletadas são armazenadas em repositórios para posterior recuperação. Os repositórios são gerenciados por um processo que recebe pedidos de inserção e de recuperação. Podem existir mais de um repositório na grade e um único repositório pode conter informações sobre vários recursos. O repositório é composto por vários arquivos de texto comuns, organizados como uma fila circular, onde os dados mais antigos são retirados ao passo que novos são inseridos.

Baseados nas informações armazenadas nos repositórios os algoritmos de predição do NWS são capazes de prever o estado dos recursos uma unidade de tempo no futuro. Uma unidade de tempo refere-se ao intervalo entre duas medições dos sensores. A predição é calculada aplicando-se equações estatísticas como média ou mediana sobre séries temporais, obtendo valores aproximados para o comportamento futuro dos recursos.

2.2.2 *Resource Monitoring System* - Remos

O Remos foi projetado para prover informações sobre os recursos, abrangendo o estado do processador, rede, entre outros. O Remos possui ainda um serviço de predição capaz de prever o desempenho futuro dos recursos [20].

A arquitetura do Remos pode ser vista na Figura 2, sendo composta por coletores, serviço de predição, Modeler e aplicações. Os coletores são semelhantes aos sensores do NWS, responsáveis por coletar informações sobre o estado do processador (carga média) e rede (carga atual). O coletor Master mantém registro dos coletores e das porções da rede na qual eles atuam,

abstraindo da aplicação a existência e localização dos mesmos. Aplicações de usuário utilizam a API do Remos para coletar as informações. Tal API é chamada de Modeler.

O Remos inclui um serviço de predição que recebe um vetor de dados coletados e produz uma predição para um tempo futuro, determinado pelo usuário. O serviço permite também a geração contínua de predições, tão logo sejam coletadas mais informações. Outra característica do Remos é permitir o compartilhamento de predição entre vários usuários para não comprometer o desempenho do sistema, como aconteceria caso cada usuário solicitasse a mesma predição sob demanda.

Cada coletor mantém *buffers* das informações coletadas, gerando um histórico do desempenho do recurso que posteriormente é repassado para o serviço de predição.

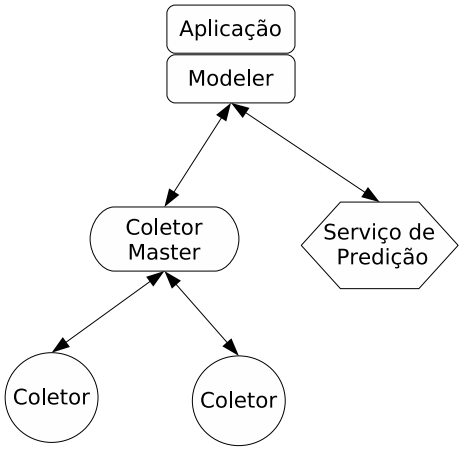

Figura 2 – Arquitetura do Remos.

O serviço de predição emprega diversos algoritmos baseados em modelos de séries temporais, como [23]: Box-Jenkins, AR, ARIMA, ARMA, MA, além de um modelo de média para solicitações *long term*.

2.2.3 *Conservative scheduling*

Em contraste com Remos e NWS que predizem o desempenho para um momento pontual no futuro, Yang *at al* [6] propuseram um algoritmo que prevê o desempenho de recursos para um intervalo de tempo futuro. O algoritmo é usado para auxiliar no escalonamento de *clusters*.

O escalonador gera uma série de mapeamentos e, empregando equações que levam em consideração o histórico da carga do processador e da rede, verifica qual alcançará melhor desempenho num intervalo de tempo futuro, condizente com o tempo de execução de uma tarefa.

Emprega-se um algoritmo que separa a série temporal em sub-séries, calcula a média destas sub-séries gerando uma nova série que por fim é usada para predizer o desempenho num intervalo de tempo determinado pelo usuário.

Os testes realizados utilizaram dados coletados por uma ferramenta que armazenou a carga do processador das máquinas durante um período de 240 minutos.

2.2.4 *Performance Analysis and Characterisation Environment* - PACE

O PACE é um sistema de predição capaz de prever o desempenho de aplicações paralelas executando sobre ambientes paralelos ou distribuídos [15]. O PACE permite a análise de diferentes mapeamentos, verificação de escalabilidade de uma aplicação, escalonamento e balanceamento de carga e predição de tempo de execução. Isso é possível combinando análise do código e do ambiente sobre o qual será executado.

A arquitetura do PACE, mostrada na Figura 3, é composta por ferramentas de aplicação, que analisam o código fonte e geram um *script* contendo os resultados da análise, e por ferramentas de recursos, que analisam características como capacidade de CPU, *cache* e rede e geram outro *script* que, em conjunto com o anterior, será usado para gerar as predições. A análise do código permite gerar um modelo da aplicação que descreve suas necessidades computacionais, já o modelo do recurso busca equacionar as suas capacidades. Ao efetuar um cruzamento de ambos os modelos pode-se predizer o tempo de execução da aplicação, encontrar melhores mapeamentos para as tarefas, entre outros.

As predições de desempenho dos recursos são gerados a partir de resultados de *benchmarks*, modelos estatísticos (baseados em histórico) e analíticos (baseados na arquitetura) [24]. Uma vez construído, o modelo do recurso é armazenado para posterior uso.

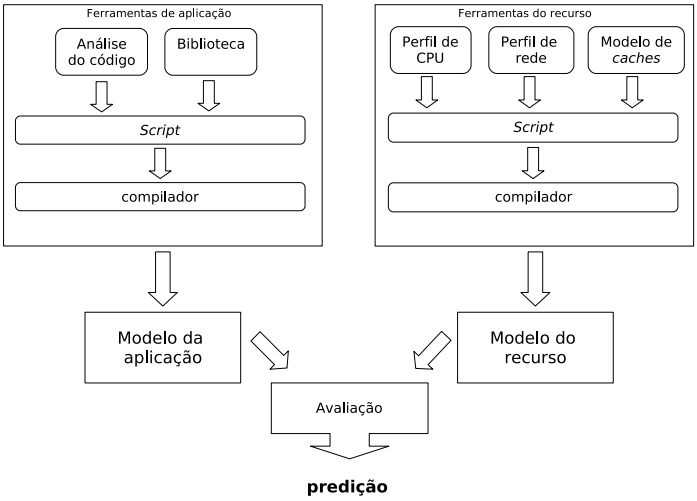

Figura 3 – Arquitetura do PACE.

2.2.5 Comparação entre os serviços e algoritmos

Os serviços e algoritmos descritos anteriormente são caracterizados por aplicar análises em séries de tempo, ou seja, a partir de uma série temporal de dados sobre determinada característica é aplicada uma técnica para predizer o próximo ou próximos valores da série. A técnica pode variar podendo ser empregada média aritmética, mediana, modelos de auto-regressão, entre outros. Entretanto, todas têm como ponto comum a análise de uma série de tempo.

O número de elementos da série pode variar, sendo difícil de definir *a priori* o número total que resulte em melhores predições [21]. Contudo, o total de elementos depende, em geral, de quanto tempo a frente se deseja realizar a predição. Para um tempo relativamente curto como um segundo no futuro, um total de 2000 entradas pode ser suficiente, porém, para uma predição meses a frente um ano de dados coletados pode ser necessário. Como exemplo do primeiro toma-se a predição da carga do processador usando-se a configuração padrão do NWS, já do segundo, a predição de um ano do tráfego de rede de um *backbone*, como exemplificado por Groshwitz *et al* [19].

Um aspecto convergente entre os algoritmos é a necessidade de dados sobre o recurso em termos de CPU e rede de interconexão. Serviços como o NWS ou Remos possuem uma série de sensores responsáveis por obter e distribuir dados relativos ao desempenho dos recursos. Tais dados se concentram na carga do processador, memória física disponível, número de processos, latência e carga da rede. PACE acrescenta ainda a necessidade de informações relativas a determinados *benchmarks*.

O uso desses algoritmos também varia de escalonamento de tarefas a seleção de recursos, testes de escalabilidade, entre outros. Nota-se que no caso de escalonamento as informações de histórico de desempenho devem ser obtidas com grande eficiência uma vez que esta etapa não pode se estender por muito tempo. Além disso, o intervalo de tempo histórico solicitado não é muito elevado pois o tempo de execução do algoritmo de predição é proporcional ao número de elementos da série. Por outro lado, predição para tempos muito a frente ou para se criar um modelo de desempenho do recurso necessitam de um intervalo de tempo histórico maior, não possuindo uma restrição de tempo tão expressiva quanto a do escalonador.

Tendo em vista as semelhanças entre as necessidades dos algoritmos de predição é proposta uma nova arquitetura de serviço de informação para grades computacionais cujo objetivo principal é armazenar informações históricas sobre desempenho de recursos. As informações que serão armazenadas devem caracterizar o desempenho passado dos recursos e serão organizadas de forma otimizada para dar suporte a algoritmos de predição. O emprego de tal serviço permite aos serviços de predição concentrarem-se no tratamento das informações históricas retirando-lhes a responsabilidade de ter que gerenciar sensores e as respectivas informações geradas. A Figura 4 ilustra de forma simplificada a arquitetura do serviço de informação, chamado de GIStorage. Este serviço trabalha como um mediador entre os diversos sensores de uma grade e os

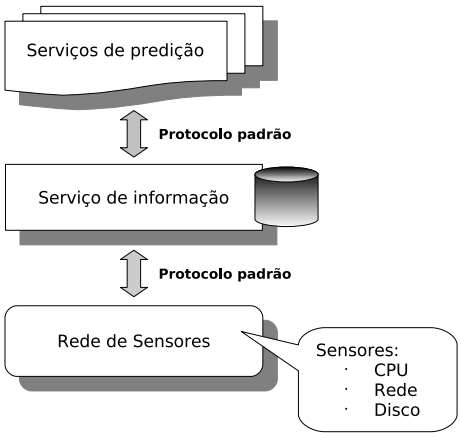

Figura 4 – Arquitetura simplificada do serviço proposto.

vários serviços de predição utilizando protocolos padrões para comunicação.

O GIStorage abstrai para os serviços de predição e sensores os problemas ligados ao grande volume de dados, a escalabilidade, resposta eficiente, distribuição das informações, etc., permitindo que ambos se concentrem nas tarefas de tratar e gerar as informações, respectivamente.

Para ilustrar alguns dos benefícios do emprego do GIStorage toma-se de exemplo o cenário descrito na Figura 5. Neste exemplo, um usuário, que pode ser um escalonador ou *broker*, deseja obter informações de predição sobre recursos de duas organizações virtuais [1] das quais tem acesso. Entretanto, cada OV possui um serviço de predição distinto, cada um com protocolos e sensores diferentes (OV A utiliza Remos e OV B emprega NWS). Além disso, o usuário conhece somente o protocolo de comunicação do NWS, portanto, não pode obter informações de predição da OV A (Fig. 5-a). O NWS poderia realizar a predição para os recursos da OV A, contudo, NWS não reconhece o protocolo e os próprios sensores desta organização e, mesmo que conhecesse, poderia ter restrições de segurança que o impedisse de acessá-los. Já com o uso

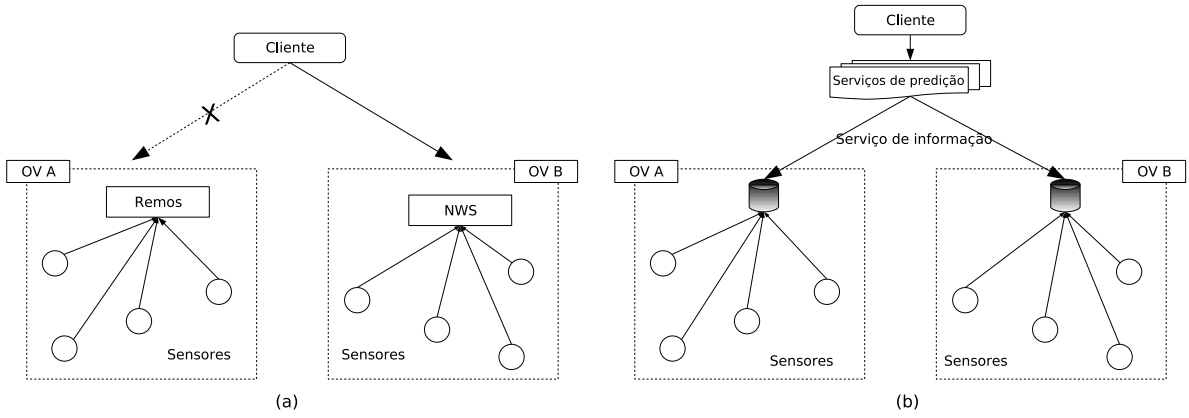

Figura 5 – Benefícios do uso de um serviço de informação para grades.

de um serviço de informação o usuário pode utilizar o serviço de predição que conhece e este poderá obter as informações sobre o histórico dos recursos, bastando para isso que reconheça o protocolo padrão do GIStorage (Fig. 5-b).

Um ponto que facilita a integração dos serviços de predição é o fato de ambos utilizarem informações sobre as mesmas características: carga do processador, latência da rede, espaço em memória e disco, resultados de *benchmarks*, etc. Existem diversas ferramentas que produzem este tipo de informação como *vmstat* para carga do processador em sistemas UNIX, Ganglia [25] para *clusters*, Nagios para redes de computadores [26], Hawkeye para redes Condor [27], entre outros. O uso de um serviço de informação permite que o administrador de uma organização escolha os sensores de acordo com as políticas internas não sendo necessário impor as OV a instalação de determinados sensores, muitas vezes de mesma função, devido aos serviços de predição.

3 Trabalhos relacionados

Como discutido no capítulo anterior, algoritmos de predição são usados para auxiliar serviços de descoberta e alocação de recursos e escalonamento de tarefas. Para prever o comportamento futuro dos recursos, tais algoritmos demandam informações históricas sobre o desempenho dos mesmos.

Tanto Remos quanto NWS são soluções completas tendo seus próprios sensores e elementos de armazenamento para manter o histórico dos recursos. Entretanto, ambos não são capazes de cooperação um com o outro, impondo a todas as OV [1] participantes a instalação de um destes ou ainda obrigando os programadores a criarem códigos para mais de um sistema.

Com o propósito de melhorar a cooperação entre os sensores e facilitar aos usuários a obtenção das informações coletadas foram propostos diversos serviços de informação para grades computacionais. Neste capítulo, serviços de informação já em uso assim como novas propostas serão descritas caracterizando o estado da arte nesta área.

3.1 Ganglia

O Ganglia é uma ferramenta de monitoramento focado em sistemas computacionais de alto desempenho como *clusters* e grades [25], podendo ser utilizado para prover informações coletadas de sensores para serviços de predição.

A arquitetura do Ganglia pode ser analisada na Figura 6. Cada nodo dentro de um *cluster* envia mensagens em *multicast* para identificar sua presença. O recebimento desta mensagem por parte dos demais participantes indica que o nodo em questão está disponível, por outro lado, o não recebimento indica que o nodo falhou. Cada nodo monitora seus recursos locais e envia mensagens aos demais a cada atualização significativa das informações. Desta maneira, cada nodo do *cluster* contém uma visão aproximada de todo o sistema, o que permite uma rápida reconstrução após a ocorrência de falhas. Ganglia permite ainda que aplicações usem o mesmo protocolo para publicar informações relativas ao seu andamento.

Na ilustração nota-se o sistema de monitoramento *gmond* sendo executado em cada nodo. Quando um usuário deseja obter informações sobre o sistema basta solicitar a qualquer nodo através de um protocolo de solicitação respondido pelo *gmond*.

Além de monitorar *clusters* individuais, o modelo hierárquico do Ganglia permite o monito-

ramento de uma federação de *clusters*[1]. Pode-se ver na Figura 6 entidades chamadas de *gmetad*. Cada uma dessas entidades é um agregador que mantém informações sobre a federação como um todo. O *gmetad* mais ao topo da ilustração combina informações sobre ambos os *clusters*, enquanto que os *gmetad* intermediários coletam informações sobre um *cluster* específico por meio de solicitações a alguns dos nodos do mesmo.

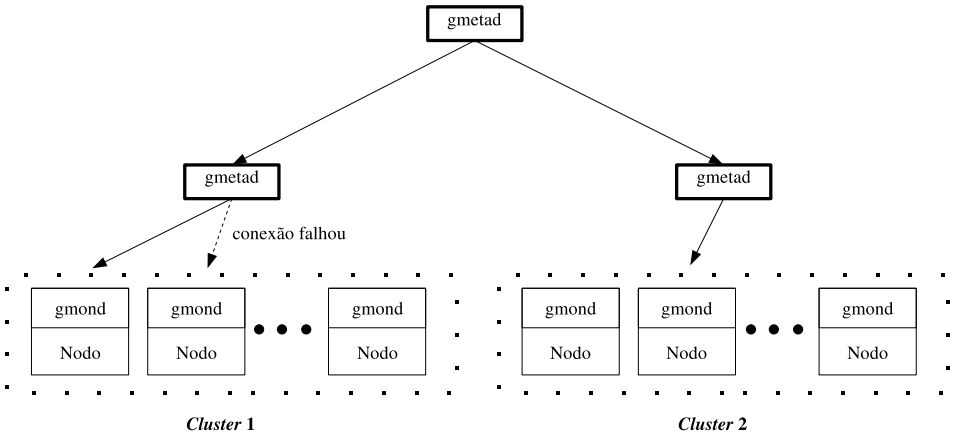

Figura 6 – Arquitetura do Ganglia.

Além das informações coletadas pelos seus próprios sensores como carga de cada processador e memória livre sendo usada, Ganglia permite que novas informações coletadas por sensores de terceiros sejam adicionadas, tornando-o adequado para servir algoritmos de predição.

3.2 MDS

Apesar de armazenar de forma escalável informações provenientes de diversas fontes, o Ganglia utiliza protocolos próprios que devem ser tratados pelas aplicações. Por outro lado, o MDS [2], seguindo os objetivos do Globus [28], visa dar suporte a serviços de alto nível provendo primitivas simples que possam ser estendidas para atender as necessidades de variados tipos de serviços como *brokers*, escalonadores e monitores de tempo real. Tais primitivas estão de acordo com os padrões atuais para grades computacionais, em especial com o WSRF [9].

A arquitetura do serviço de informação do Globus (MDS) é composta por duas entidades básicas: *information providers* e serviços de alto nível. Provedores de informação (do inglês *information providers*) formam um grande e distribuído conjunto de entidades que proporcionam acesso a informações detalhadas sobre recursos e serviços. Por exemplo, um provedor para um recurso computacional pode prover informações sobre o número de nodos, memória total, sistema operacional, carga do processador, etc. Cada provedor de informação publica suas

[1] Um conjunto de *clusters* é conhecido na literatura como uma federação de *clusters*.

informações em um ou mais serviços de alto nível que concentram informações sobre vários provedores e respondem a solicitações de usuários.

Os serviços de alto nível coletam, gerenciam, indexam, e disponibilizam informações sobre provedores de informação. Um tipo de serviço de alto nível muito importante é o serviço de agregação (do inglês *aggregate directory services*) que facilita a descoberta e monitoramento de recursos em OVs mantendo em um único local informações sobre os demais. O serviço de agregação chamado de *Index Service* é responsável por registrar os recursos e serviços existentes na organização.

A Figura 7 ilustra a arquitetura do MDS. Provedores de informação registram sua existência em um ou mais serviços de agregação. Usuários e outros serviços de busca e monitoramento entram em contato com agregadores para encontrar ou monitorar recursos da grade.

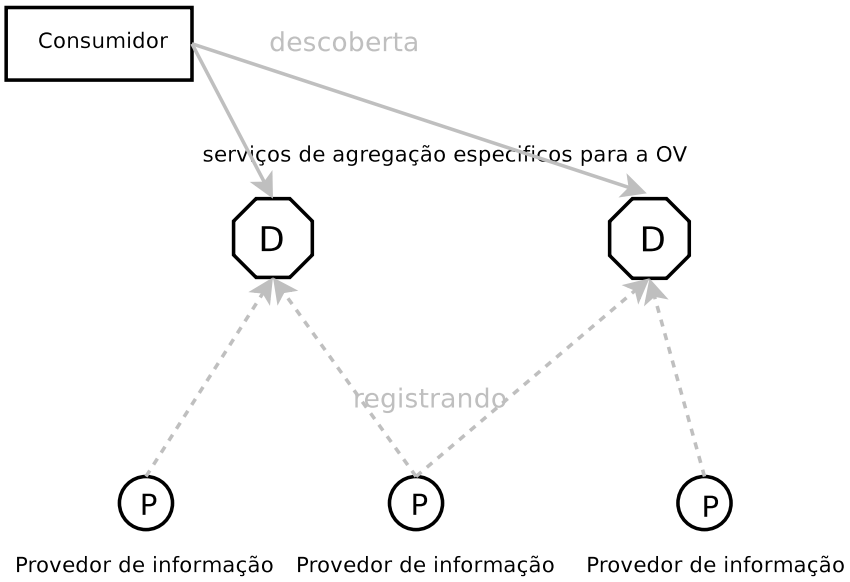

Figura 7 – Arquitetura do serviço de informação do Globus. Provedores de informação registram suas existências em um ou mais serviços de agregação específicos da OV. Após o registro, cada provedor envia informações para os agregadores. Um consumidor solicita aos agregadores informações sobre os recursos que eles gerenciam.

O MDS na sua versão atual (MDS 4 [29]) possui dois agregadores padrões: MDS-Trigger e MDS-Index. MDS-Index é um agregador padrão onde produtores (provedores de informação e serviços) se registram e onde consumidores (outros agregadores, *brokers*, usuários, etc) buscam informações sobre os recursos e serviços existentes. MDS-Trigger permite que uma determinada ação seja executada caso as informações alcancem determinados valores, por exemplo, enviar um *e-mail* para o administrador caso o número de recursos disponíveis esteja abaixo do normal.

3.3 TGIS

Estudos mostraram que o MDS não suporta um número elevado de provedores de informação, tornando-se um gargalo que impede um bom desempenho na resposta aos pedidos de usuários [30]. Buscando atingir uma melhor escalabilidade Chen *at al* propuseram uma extensão do MDS2[2] capaz de organizar os agregadores sob a forma de uma árvore hierárquica.

A principal contribuição deste trabalho é a geração de um protocolo que permite aos agregadores do MDS2 se organizarem hierarquicamente, característica não existente na arquitetura do anterior.

3.4 FOSIS

O FOSIS é uma proposta para um novo serviço de informação para grades que utiliza o padrão OGSI[3] [31] e é estruturado hierarquicamente como um conjunto de árvores [32].

A arquitetura do FOSIS é dividida em três camadas como ilustra a Figura 8. Na camada inferior (*Resource Layer*) encontram-se os recursos. Logo acima está a camada de provedores de informação (*Information Providers Layer*), composta por um conjunto de provedores responsáveis por coletar dados sobre os recursos. No topo encontra-se o *Information Service Layer*, que é constituído por diversos serviços de informação (ou agregadores) organizados numa estrutura baseada em árvore.

A camada superior agrega informação proveniente de numerosos recursos, sendo cada nível da árvore responsável por uma parte das informações. Usuários iniciam o processo de pesquisa no serviço de informação local (nível inferior da árvore) que por sua vez repassa o pedido para os níveis superiores caso não possua os dados desejados. Os serviços de informação raiz provenientes de todas as OV estão ligados uns aos outros. Caso uma solicitação não possa ser respondida localmente ela será repassada para as demais OV de forma transparente ao usuário.

3.5 OGSI-*based* GIS

O OGSI-*based* GIS é similar ao trabalho anterior por usar o padrão OGSI, ser divido em três camadas e por organizar os agregadores numa estrutura baseada em árvores [33]. Na Figura 9 estão ilustradas as camadas deste serviço de informação.

[2]Embora seja precedente ao MDS4 a segunda versão não possui diferenças substanciais com relação a versão mais atual. A principal diferença está na tecnologia empregada, a segunda versão utiliza protocolos próprios ao passo que a última se baseia no padrão WSRF [9].

[3]O OGSI foi o primeiro padrão para grades usando a tecnologia *Web Services* sendo posteriormente substituído pelo padrão WSRF, que herdou do primeiro muitas de suas características.

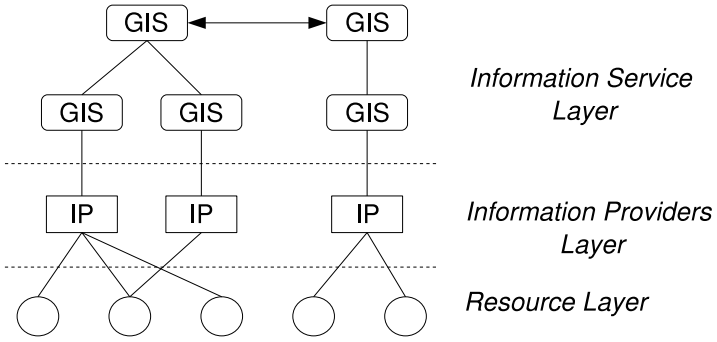

Figura 8 – Arquitetura do FOSIS.

A camada de recursos (*Resource Layer*) compreende os sensores que coletam informações dos recursos existentes. Mais acima se encontra a camada do *site* (*Site Collective Layer*), composta por vários agregadores que consolidam informações obtidas na camada inferior[4] . Por fim, a camada superior (*VO Collective Layer*) reúne as informações da camada anterior.

As informações sobre os recursos são classificadas em duas categorias: estáticas e dinâmicas. As informações estáticas compreendem dados que não mudam com freqüência como a configuração de *software* ou fabricante do *hardware*. Já as dinâmicas englobam os dados que são atualizados numerosas vezes como a carga do processador e espaço disponível em disco. Neste serviço de informação os dados estáticos são armazenados na camada superior enquanto que as informações dinâmicas são mantidas na camada de recursos. Com isso pretende-se responder rapidamente aos pedidos de informações estáticas, comuns durante a descoberta de recursos, e manter o serviço escalável.

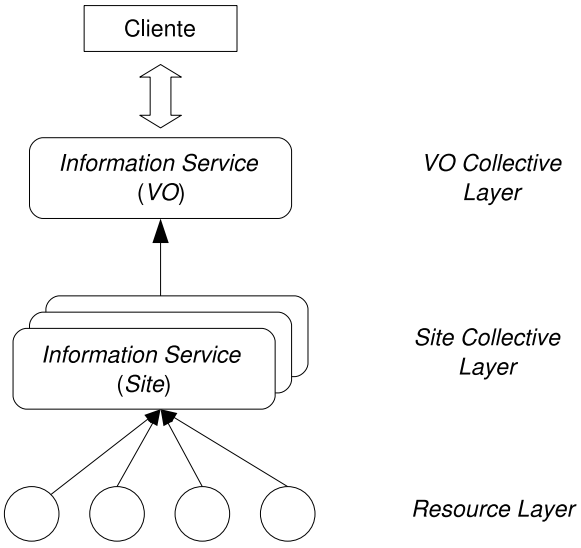

Figura 9 – Arquitetura do OGSI-*based* GIS.

[4]Na terminologia adotada pelos desenvolvedores do OGSI-*based* GIS uma OV pode possuir mais de um *site*, contudo, no trabalho corrente o termo *site* é sinônimo de OV (ver Capítulo 4, seção 4.1).

4 Serviço de informação para grades computacionais

Como visto no Capítulo 2, serviços de predição podem melhorar o desempenho das aplicações quando são empregados na descoberta e alocação de recursos e escalonamento de tarefas. Contudo, estes serviços requerem informações históricas sobre o desempenho dos recursos. Para prover informações a estes serviços é proposto o GIStorage, um serviço de informação cujo principal objetivo é armazenar informações históricas sobre recursos de uma grade computacional.

Com o emprego deste serviço, retira-se dos serviços de predição a responsabilidade de gerenciar sensores e as respectivas informações geradas, o que acarreta entre outras questões problemas relacionados ao grande volume de dados, escalabilidade, resposta eficiente, distribuição das informações, etc. Desta maneira os serviços de predição podem focar exclusivamente no tratamento das informações históricas. Ainda, por meio de protocolos padronizados para representar as informações, permite-se um melhor aproveitamento dos sensores existentes, não sendo necessário a instalação de ferramentas de mesma função devido a incompatibilidade entre os serviços de predição e os sensores a eles dedicados.

No projeto de um serviço de armazenamento e recuperação de informações para grades uma série de questões e desafios devem ser tratados [7]. Este trabalho é direcionado principalmente às questões descritas abaixo:

- **Heterogeneidade dos recursos:** Devido a grande diversidade de uma grade, as informações relativas ao desempenho podem variar sensivelmente de recursos a recurso. Por exemplo, para um microcomputador usualmente são levadas em consideração informações sobre a carga do processador, espaço livre de memória, arquitetura, entre outros, enquanto que para um *cluster* são adicionadas informações sobre tamanho da fila do escalonador, rede interna, etc. Portanto, deve-se projetar uma estrutura de dados que permita armazenar informações variadas, provenientes de recursos heterogêneos. Além disso, a estrutura deverá ser passível de extensão posterior uma vez que características inexistentes atualmente podem ser necessárias no futuro.

- **Conjunto de informações:** Apesar da variedade de informações coletadas apenas um subconjunto destas são relevantes aos algoritmos de predição de desempenho existentes. Deve-se pesquisar quais dados pertencem a este conjunto. As demais informações não serão arquivadas para permitir um melhor aproveitamento do espaço de armazenamento.

- **Grande volume de dados:** O serviço de informação precisa tratar um grande volume de

dados visto que existem centenas de recursos e é necessário manter informações históricas sobre cada um deles.

- **Solução distribuída:** Apesar de ser potencialmente mais simples, uma solução centralizada para o problema proposto não é viável pois implicaria em dois grandes problemas: (i) como toda a informação ficaria armazenada em um único ponto, a falha neste recurso dificultaria o uso da grade como um todo; (ii) uma vez que a etapa de descoberta de recurso de cada usuário depende deste serviço, este se tornaria um gargalo do sistema, comprometendo seu desempenho. Tendo isso posto, deve-se projetar a arquitetura do serviço de forma distribuída, com as informações e a responsabilidade por seu gerenciamento sendo divididas entre os recursos.

- **Escalabilidade:** Em uma grade o número de recursos varia ao longo do tempo. O serviço deve permitir que recursos sejam adicionados ou removidos da grade sem demandar um grande esforço por parte dos administradores dos mesmos. Deve ainda suportar um elevado número de recursos participantes.

- **Resposta eficiente:** Para o escalonamento de tarefas em grade é requerido informações históricas tão rápido quanto possível, pois esta etapa não deve se estender por um período maior que uma fração do tempo de execução da aplicação. Por essa razão, o serviço de informação precisa ser otimizado para responder a pedidos de escalonadores que, de modo geral, envolvem um intervalo de tempo não muito extenso.

- **Padronização**: Para que o serviço seja amplamente utilizado é necessário basear sua arquitetura em padrões para grades computacionais mundialmente aceitos. O serviço deve respeitar padrões como WSDL [8] e WSRF [9], assim como as recomendações provenientes de instituições como OASIS [10], OGF [11] e W3C [12].

O Capítulo 3 mostrou outros serviços de informação para grades. O GIStorage difere dos demais por dar suporte a algoritmos de predição, sendo otimizado para armazenar o histórico do desempenho dos recursos. Uma comparação detalhada entre o GIStorage e os demais serviços é apresentada no Capítulo 5.

Este trabalho além de definir uma arquitetura para serviços de informação define também um modelo de informação para descrever os recursos e as interações entre os mesmos. Nas próximas seções tanto o modelo quanto a arquitetura proposta serão descritos.

4.1 Modelo de informação

Antes de se definir a arquitetura do serviço deve-se conhecer o modelo de informação que será empregado. Será através deste que as relações entre os recursos serão descritas e a sua

estrutura deve ser levada em consideração na arquitetura do serviço para que possa mapear adequadamente as informações nos recursos físicos.

Num primeiro momento avaliou-se o modelo de informação GLUE *schema* [34] que visa padronizar a representação de informações sobre os recursos de uma grade computacional. Este modelo representa sistemas computacionais descrevendo seus elementos (computadores, *clusters*, etc) e as relações entre eles. O GLUE *schema* é um padrão que vem sendo usado por ferramentas como Ganglia [25] e Globus [28, 35].

O modelo hierárquico do GLUE *schema* é mostrado de forma resumida na Figura 10. No topo da hierarquia encontra-se o *site*, que representa um único domínio administrativo, ou seja, uma única organização virtual. Os demais componentes pertencem a um determinado *site* que possuí como atributos uma identificação única, nome, localização geográfica, entre outros. Logo abaixo na hierarquia encontra-se o *cluster*, que representa um conjunto de elementos computacionais cujo acesso é feito através gerenciadores de recursos. Qualquer conjunto de máquinas pode ser representado por um *cluster* sejam estes pertencentes a uma rede de computadores (máquinas NOW [36]) ou agregados (máquinas COW [36]). Esse componente do modelo é descrito usando-se atributos tais como identificação única, informações sobre o gerenciador de recursos (tipo, endereço IP de acesso), estado atual (fila do escalonador, número de processos executando, etc.), entre outros.

Um *cluster* é constituído de um ou mais *subclusters*, que são conjuntos de elementos computacionais de mesma arquitetura. Pertencem a descrição de *subclusters* atributos como identificação única, sistema operacional, modelo do processador, tamanho das memórias principal e *cache*, número de processadores, etc. Por fim, um *subcluster* é formado por vários *hosts*, componentes do modelo que expressam os atributos dinâmicos de cada máquina individualmente, como identificação única, quantidade de espaço livre na memória principal e disco rígido, porcentagem de uso do processador, entre outros.

O modelo de informação GLUE *schema* engloba características sobre os elementos computacionais comuns para serviços de predição como carga do processador e memória disponível,

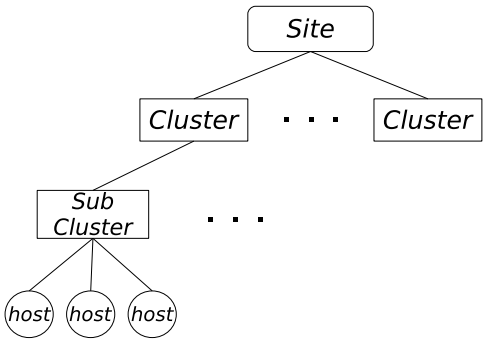

Figura 10 – Modelo de informação hierárquico do GLUE *schema*.

contudo, não contempla informações referentes a rede de interconexão tampouco é capaz de relacionar os eventos com o momento em que ocorreram.

Como parte deste trabalho criou-se o GLUE^2 (*GLUE schema Extended*), um modelo de informação baseado no GLUE *schema* capaz de descrever medições sobre a rede de interconexão e relacioná-las com uma marcação de tempo (*timestamp*). O *Open Grid Forum* (OGF) [11] realizou um importante trabalho buscando identificar quais características da rede são relevantes às aplicações, e posteriormente, elaborou uma terminologia padronizada para descrever medições de rede [37]. O modelo GLUE^2 combina tanto o GLUE *schema* quanto as recomendações do OGF.

O modelo proposto pelo OGF define duas entidades de rede: *nodes* e *paths*. *Nodes* podem ser qualquer entidade física tal como um computador, um roteador ou *switch* ou ainda um conjunto dos mesmos. Já *paths* são uma conexão unidirecional entre dois *nodes*. Desta forma, a rede é descrita por um grafo onde as arestas são representadas pelos *paths* e os vértices por *nodes*.

As entidades possuem atributos e características. Os atributos são empregados para indicar as condições sobre as quais as medições foram feitas, como protocolos, QoS, camada da rede, entre outros. Deste modo, os atributos permitem que várias topologias sobreposta sejam representadas. Por exemplo, uma conexão entre duas entidades pode ter como atributo o protocolo TCP, e uma outra conexão pode ter como atributo o protocolo IP, descrevendo deste modo duas topologias sobrepostas. Já as características são as propriedades relacionadas ao desempenho da rede tais como largura de banda ou *delay*.

A Figura 11 apresenta as características que foram consideradas importantes pelo OGF: largura de banda, *delay*, perda de pacotes, reordenação de pacotes, disponibilidade, políticas de encaminhamento (*forwarding*), fila do roteador, *hoplist* e proximidade.

Hoplist é a listagem das conexões entre os elementos que formam o *path*. Por exemplo, entre dois *nodes* existem vários *hops* entre os roteadores que os conectam. Já as políticas de encaminhamento tratam sobre o modo como os *nodes* encaminham o tráfego, o que ocorre nas camadas da rede 2 e 3[1] , em *switches* e roteadores, respectivamente.

Quanto a largura de banda, ou seja, o número de *bytes* por segundo de uma conexão, é importante medir sua capacidade máxima, utilização atual, largura disponível e a que poderá ser de fato utilizada considerando a carga atual, protocolo, sistema operacional e capacidade máxima.

A latência ou *delay* também é uma característica de grande importância, principalmente para aplicações de tempo real. Na definição empregada pelo OGF, *delay* é o tempo despendido entre o instante em que a primeira parte de um objeto passa por um ponto de observação até que sua última parte passe por um segundo ponto de observação.

A perda de pacotes durante uma conexão tem impacto sobre a qualidade do serviço provido por aplicações de rede, portanto, também deve ser medida. Além disso, a disponibilidade da

[1] Considerando o modelo de referência OSI [38]

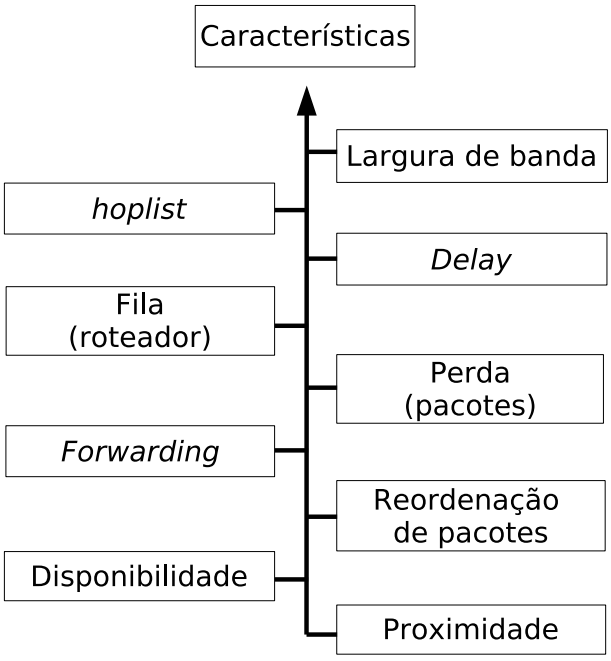

Figura 11 – Características de rede relevantes às aplicações.

rede, ou seja, o quão freqüente a rede está ativa ou inoperante é também objeto de atenção.

Em redes com cabos (como o Padrão IEEE 802.3), o tráfego congestionado se manifesta quando as filas dos roteadores estão sobrecarregadas. Para modelar este comportamento, modelos analíticos necessitam de informações sobre as características estáticas de dinâmicas das filas.

Apesar de não ocasionar impacto quando a freqüência é baixa, a reordenação de pacotes pode se tornar um problema se ocorrer repetidas vezes e por isso deve-se medí-la.

Por fim, uma noção de proximidade entre os *nodes* pode ser útil. Embora o valor mais adequado para o seu cálculo dependa da aplicação em questão, é possível medí-la balanceando características como largura de banda e latência.

Cabe ainda ressaltar que a terminologia adotada pelo OGF é independente da forma como as informações são coletas. Medições de rede podem ser realizadas usando-se técnicas passivas como SNMP [39] ou ativas, usando-se *benchmarks* como Gloperf [40]. O modelo GLUE^2 herda esta propriedade para permitir que uma grande variedade de sensores possam utilizá-lo, ficando a cargo dos atributos especificar qual técnica foi empregada. Uma discussão sobre os pontos positivos e negativos de cada técnica pode ser encontrada no trabalho de Roughan [41].

Com este modelo de dados os sensores podem representar as informações coletadas sobre o desempenho da rede de forma padronizada, contudo, medir a conexão todos-para-todos entre os elementos de uma grade não é escalável [22]. Diversos trabalhos [22,42,43] propuseram grupos hierárquicos de sensores para tornar a medição de rede mais escalável. Nesta hierarquia, elementos pertencentes a um determinado grupo medem as características da conexão entre todos

os pares possíveis. Para medir a conexão entre os grupos um representante de cada é escolhido e passa a realizar medições com os demais representantes, resultando em uma estimativa do desempenho das conexões todos-para-todos entre os recursos de ambos os grupos.

O administrador dos recursos é responsável por definir os grupos de acordo com sua localização e semelhança. Essa abordagem se justifica se considerarmos que os participantes de cada grupo possuem em comum a mesma arquitetura e compartilham um mesmo ramo de conexão ao enviar mensagens aos demais grupos, embora pequenas diferenças possam ocorrer devido a diferenças no *hardware* das placas de rede, nas configurações dos sistemas operacionais, pilhas TCP/IP empregadas, entre outras.

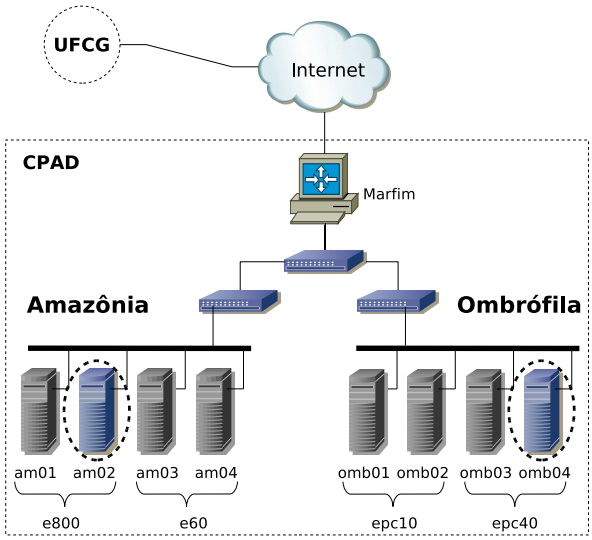

Figura 12 – Exemplo de uso dos grupos de rede.

Para ilustrar a abordagem de grupos toma-se a Figura 12, que mostra duas organizações virtuais participantes da grade Ourgrid [44], CPAD em detalhes[2] e UFCG. No CPAD existem dois *clusters*, Amazônia e Ombrófila, sendo o primeiro composto por dois *subclusters* (e800 e e60) e o segundo por outros dois (epc10 e epc40). Caso sejam empregados técnicas de medição ativa, os *hosts* de cada *subcluster* realizam medições todos-para-todos entre si. Como os *subclusters* de um determinado *cluster* compartilham o mesmo segmento de rede o resultado de uma medição entre dois representantes deve ser semelhante ao coletado em conexões todos-para-todos. De forma análoga, os *hosts* amb02 e omb04 foram escolhidos para medir a comunicação entre os dois *clusters* (ver figura). Por fim, é eleito um representante para todo o CPAD que irá medir a conexão entre os *sites* CPAD e UFCG.

É importante frisar que por meio de técnicas passivas como SNMP obtém-se a mesma informação por meio dos *switches* empregados, entretanto, a medição entre os dois *sites* pode ser difícil de ser obtida uma vez que os sensores de cada organização possivelmente não tem

[2]O Centro de Pesquisa em Alto Desempenho (CPAD), cujo endereço eletrônico é http://www.cpad.pucrs.br, localiza-se na Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre-RS.

permissão para coletar dados dos equipamentos da rede que os conectam.

A abordagem de grupos hierárquicos vai de encontro com o modelo de entidades hierárquicas do GLUE *schema*. Os *hosts* que formam um *subcluster* possuem a mesma arquitetura e compartilham um mesmo segmento de rede. As características da conexão entre *subclusters* de um *cluster* pode ser representada pela conexão entre um representante de cada *subcluster* ou grupo. De forma análoga, o desempenho da conexão entre dois *clusters* e entre dois *sites* também pode ser expressa pela conexão de um represente de cada grupo.

No modelo GLUE^2 inseriram-se as informações de rede respeitando a hierarquia do próprio modelo, definindo como grupos cada um dos seus componentes. A Figura 13 esclarece o novo modelo a partir do exemplo real citado anteriormente. Adicionou-se ainda um novo componente, chamado de *grid*, no topo da hierarquia para representar a grade, permitindo que conexões entre *sites* também possam ser descritas. O componente *grid* pode ser visto topo da Figura 13.

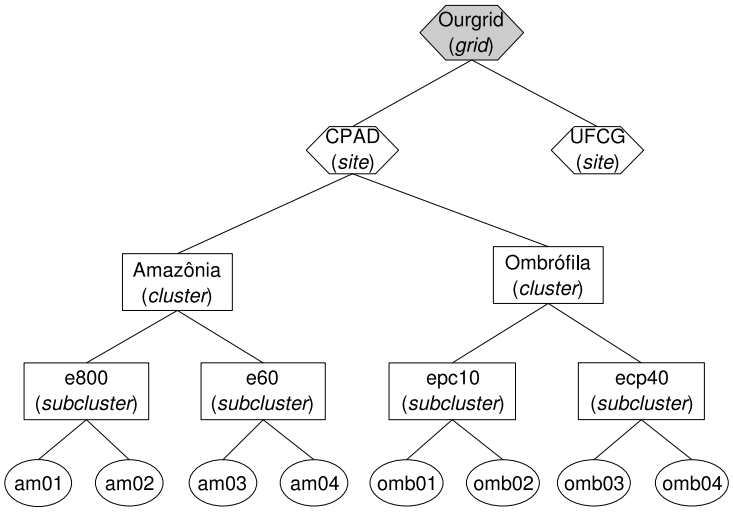

Figura 13 – Exemplo do modelo de informação GLUE^2.

Através destas modificações o modelo GLUE^2 pode ser usado para representar as informações sobre os recursos e as redes de interconexão. Tanto os sensores quanto os serviços de predição devem compreendê-lo a fim de publicar e obter dados do serviço de informações, respectivamente.

O modelo GLUE^2 foi mapeado em um *schema* XML como forma de implementar tal representação computacionalmente. Neste mapeamento foi adicionado um atributo de *timestamp* que especifica o momento em que a informação foi gerada pelos sensores. O serviço de informação, portanto, recebe dos sensores informações representadas através de documentos XML e as armazena. Os serviços de predição por sua vez, solicitam ao GIStorage informações sobre determinadas características como carga do processador ou largura de banda entre dois recursos durante um intervalo de tempo. A resposta à solicitação também é descrita através de um documento XML, no mesmo formato que o gerado pelo sensor. De posse dessas informações e

```
<?xml version="1.0"?>
<Grid Name="Ourgrid" UniqueID="ourgrid.org.br" xmlns="www.cpad.pucrs.br/archiveService-v0.2">
<Site  UniqueID="cpad.pucrs.br"          Name="CPAD"       Web="www.cpad.pucrs.br">
       <Cluster UniqueID="amazonia.cpad.pucrs.br" Name="Amazonia">
              <SubCluster UniqueID="sub1.amazonia.cpad.pucrs.br" Name="Sub1">
                     <PhysicalCPUs>2</PhysicalCPUs>
                     <LogicalCPUs>4</LogicalCPUs>
                     <Processor Vendor="Intel" Model="Pentium IV" ClockSpeed="1500"
                            InstructionSet="x86"/>
                     <MainMemory RAMSize="256" VirtualSize="0"/>
                     <Host UniqueID="am02.amazonia.cpad.pucrs.br" Name="Am02"
                                          IPAddress="192.168.1.2" UpTime="1162990487">
                            <MainMemory>
                                   <RAMAvailable>
                                          <Value Timestamp="1162991867">128</Value>
                                          <Value Timestamp="1162991927">128</Value>
                                   </RAMAvailable>
                            </MainMemory>
                            <Load>
                                   <Last1Min>
                                          <Value Timestamp="1162991867">25</Value>
                                          <Value Timestamp="1162991927">15</Value>
                                   </Last1Min>
                                   <Last5Min>
                                          <Value Timestamp="1162991867">5</Value>
                                          <Value Timestamp="1162991927">15</Value>
                                   </Last5Min>
                                   <Last15Min>
                                          <Value Timestamp="1162991867">1</Value>
                                          <Value Timestamp="1162991927">5</Value>
                                   </Last15Min>
                            </Load>
                     </Host>
                     ...
                     <Path Source="am02.amazonia.cpad.pucrs.br"
                                          Destination="am01.amazonia.cpad.pucrs.br">
                            <Bandwidth Protocol="UDP" Capacity="100">
                                   <Achievable>
                                          <Value Timestamp="1162991627">80</Value>
                                          <Value Timestamp="1162991927">80</Value>
                                   </Achievable>
                            </Bandwidth>
                            <Delay Methodology="One-way" Protocol="ICMP">
                                   <Value Timestamp="1162991627">0.10</Value>
                                   <Value Timestamp="1162991927">0.12</Value>
                            </Delay>
                            <Loss Methodology="Round-trip" Protocol="UDP">
                                   <LossPattern Pattern="Average">
                                          <Value Timestamp="1162991627">1</Value>
                                          <Value Timestamp="1162991927">0</Value>
                                   </LossPattern>
                            </Loss>
                     </Path>
                     ...
</Grid>
```

Figura 14 – Fragmento de um documento XML do modelo GLUE^2.

do momento em que foram coletadas os serviços de predição podem então predizer o comportamento dos recursos.

Um fragmento de documento XML[3] pode ser visto na Figura 14. No começo do documento são descritas informações sobre a grade (*grid*) e a OV (*site*) pertencente a mesma, como identificação única e página da Internet. Devido a restrições de espaço a figura mostra apenas as informações sobre o *host am02* e algumas informações sobre as entidades hierarquicamente superiores. Pode-se notar que as informações são marcadas com a hora em que foram medidas (*Timestamp*) e as informações de rede possuem atributos para indicar os protocolos e metodologias empregadas na medição (*Protocol*, *Methodology*). Informações estáticas como fabricante dos microprocessadores e número de processadores físicos também são descritos.

[3]Detalhes sobre o *schema* XML do modelo de informação GLUE^2 encontram-se no Apêndice A.

4.1.1 Outros modelos de informação

Poder-se-ia utilizar os modelos de informação empregados no NWS [45] ou Remos [46], contudo, faz parte dos objetivos deste trabalho utilizar padrões atuais de grades computacionais. Dentre estes padrões, destacam-se os modelos de informação hierárquicos como LDAP [47] ou GLUE *schema* e os relacionais como R-GMA [48].

É sabido que o modelo relacional tem maior expressividade em relação ao hierárquico [49], entretanto, uma solução distribuída com bom desempenho para o primeiro não é facilmente realizável [50]. Por outro lado, o modelo hierárquico pode ser distribuído de maneira simples, bastando para isso repartir os ramos da hierarquia entre os recursos. Além disso, serviços de predição requerem consultas com expressões simples, geralmente sob a forma de uma tupla (recurso, intervalo de tempo). Devido a facilidade de distribuição das informações e da expressividade adequada decidiu-se utilizar modelos hierárquicos. Em particular, o GLUE *schema* foi escolhido como base por ser uma padronização já em uso e suportada por diversos projetos como EU-DataTAG, US-iVDGL, EGEE, LCG, Grid3/OSG, Globus e NorduGrid [51].

O GLUE *schema*, assim como o GLUE^2, não especifica um mapeamento em particular, existindo implementações para LDAP, XML e ClassAd [51]. Apesar disso, implementou-se o GLUE^2 em um *schema* XML com o intuito de validar a solução e servir de base para o cálculo do tamanho das mensagens trocadas no estudo de caso (ver Capítulo 5).

4.2 Arquitetura do GIStorage

Tendo determinado o modelo de informação que será usado a arquitetura do serviço pode ser definida. A arquitetura do GIStorage leva em consideração a estrutura do GLUE^2 para distribuir de maneira adequada os dados entre os recursos.

O modelo GLUE^2 apresenta uma possível solução aos problemas relacionados com a heterogeneidade de recursos e conjunto de informações. Entretanto, um único documento contendo informações relativas a todos os recursos de uma grade não seria viável devido a grande quantidade de participantes de uma grade real.

Além do problema relacionado ao volume de dados, uma solução centralizada reduziria o desempenho de toda a grade ao criar um gargalo no sistema e, caso este gargalo falhasse, poderia inviabilizar o uso de todo o ambiente. Portanto, o GIStorage deve descentralizar as informações sobre os recursos e a responsabilidade por cada parte.

A arquitetura distribuída do GIStorage é baseada na arquitetura de monitoramento para grades (GMA) , um modelo conceitual proposto pelo OGF [52] para guiar projetos de novos serviços. Na seção seguinte será apresentado o modelo GMA e em seguida, é definida a arquitetura do GIStorage.

4.2.1 Arquitetura de monitoramento para grades - GMA

O modelo GMA surgiu da observação detalhada dos usuários deste tipo de serviço. Os possíveis usuários variam sensivelmente em termos dos tipos de informações que necessitam (características físicas, localização, etc) e o meio pelo qual são empregadas (descoberta, escalonamento, monitoramento, etc). Entretanto, em todos os casos são identificados dois principais elementos: um ou mais consumidores que obtêm informações sobre um ou mais produtores. Consumidores englobam diversos serviços tais como descoberta de recursos, escalonamento, agentes de adaptação, entre outros, cada um querendo consumir informações sobre recursos da grade. Já os produtores são elementos que geram informações a respeito dos recursos e as disponibilizam aos consumidores interessados.

Além desses componentes o modelo contempla um serviço de diretório responsável por manter informações sobre os consumidores e produtores existentes na grade. O modelo completo possui os seguintes elementos:

- Serviço de diretório: armazena informações sobre os produtores e consumidores. Tanto produtores quanto consumidores registram-se no diretório e publicam informações sobre os protocolos de comunicação aceitos, mecanismos de segurança usados, informações que produzem ou consomem, entre outros dados que podem ser relevantes a quem os procuram. O diretório não é responsável por armazenar os dados de monitoramento.

- Produtores: são responsáveis por prover dados de monitoramento sobre um ou mais recursos. Os dados são obtidos de sensores de recursos e podem ser recuperados por consumidores.

- Consumidores: são serviços ou usuários que consomem informações sobre recursos da grade. Pode-se citar como exemplo escalonadores, monitores de tempo real, entre outros.

A Figura 15 ilustra a interação entre os componentes da arquitetura. No passo 1 consumidor e produtor registram-se no diretório. No momento em que o consumidor quer obter informações ele entra em contato com o diretório (passo 2). Este realiza uma busca sobre todos os produtores registrados e envia uma mensagem para o consumidor contendo o produtor que detém os dados de seu interesse. No passo 3 o consumidor entra em contato com o produtor escolhido e ambos fazem a comunicação sem intermediários.

Buscando propor um protocolo abrangente o modelo especifica três tipos de comunicação entre consumidores e produtores:

- *Query/Response*: neste tipo de comunicação o consumidor solicita ao produtor informações sobre o recurso e este envia uma resposta única.

- *Subscription*: caso o consumidor queira receber constantemente dados sobre um determinado recurso será empregada uma comunicação do tipo *subscription*. Neste caso, o

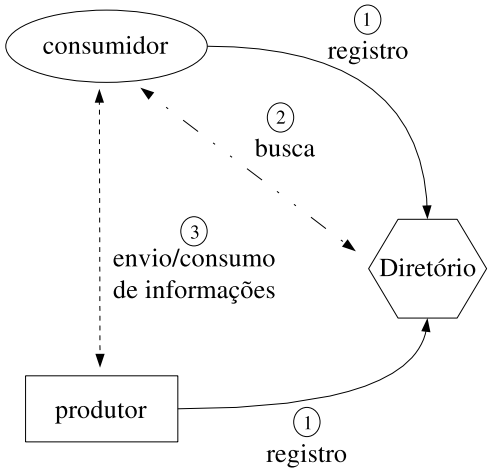

Figura 15 – Componentes do modelo de monitoramento proposto pelo OGF.

produtor enviará dados até que o consumidor explicitamente cancele o recebimento. Monitor de tempo real é um exemplo de consumidor que pode necessitar de uma comunicação desse tipo.

- *Notification*: a iniciativa de comunicação também pode vir de um produtor. Para isso, o produtor usa a comunicação do tipo *notification* enviando a um consumidor escolhido previamente uma única mensagem sobre o estado atual do recurso que é assistido. Por exemplo, um produtor pode procurar escalonadores registrados e enviar a eles informações sobre os recursos.

As informações que produtores disponibilizam são geradas por sensores. Existem diversos tipos de sensores, dependendo do objetivo em questão, mas pode-se citar como mais comuns os seguintes:

- Sensor da máquina: sensores que monitoram a carga do processador, memória livre, etc. Podem ser usados ainda para monitorar a configuração da máquina, como a versão e tipo de sistema operacional e outros *softwares* instalados.

- Sensor de rede: monitora a rede, verificando sua carga, largura de banda, perda de pacotes, etc.

- Sensor de processos: gera informações sobre o estado de processos, informando quando ocorre alguma mudança de estado, falha no processo, finalização, entre outros.

- Sensor de entrada/saída: esses sensores monitoram dispositivos de entrada e saída, como discos de armazenamento, obtendo informações sobre tamanho dos blocos, tempo de acesso, tempo de busca, sistema de arquivos, etc.

O modelo permite ainda a definição de componentes híbridos, tendo comportamento tanto de consumidor quanto de produtor. Tais componentes podem ser usados para criar serviços de filtros, *cache* de informações, repasse de dados, gatilhos para tomar determinada ação com base no estado do sistema, entre outros. Em especial, sistemas que coletam informações de vários produtores e disponibilizam para os interessados podem melhorar o desempenho da grade uma vez que podem ser instalados geograficamente próximos aos consumidores, reduzindo o tráfego da rede entre os produtores.

Cabe salientar que o modelo proposto pelo OGF não define as tecnologias que devem ser empregadas tanto na implementação dos componentes quanto nos protocolos envolvidos nas comunicações entre os mesmos. Devido a essa flexibilidade encontram-se implementações do modelo usando Java como jGMA [53], SQL como R-GMA [48] e até mesmo Python como em pyGMA [54].

4.2.2 Componentes do GIStorage

Os componentes da arquitetura do GIStorage se assemelham com os do modelo GMA. Serviços de predição podem ser vistos como consumidores e sensores como produtores. Já o GIStorage, que age como um intermediador, pode ser visto tanto como um consumidor quanto um produtor, consumindo informações de sensores e entregando-as para serviços de predição, respectivamente. Entretanto, como dito anteriormente, o serviço de informação deve ser formado por elementos distribuídos. Tais elementos são chamados de *storage elements* ou simplesmente *SE*.

A Figura 16 ilustra os elementos que compõem o GIStorage. Similar ao GMA, existe um serviço de diretório ou *D* (sigla para *directory service*) pelo qual os SEs registram qual informação armazenam e em qual intervalo de tempo. O diretório não armazena nenhuma informação sobre as medições, somente meta-dados descrevendo qual informação encontra-se em cada SE. Já os SEs armazenam informações provenientes de sensores ou *S* (sigla para *sensors*), componentes responsáveis por produzir informações sobre carga do processador, memória disponível, largura de banda, e assim por diante. Os SEs são organizados em uma árvore hierárquica, onde o SE raiz armazena informações recentes vindas dos SEs folhas, que por sua vez, armazenam um intervalo maior de informações.

Uma OV pode ter somente um serviço de diretório, um SE raiz e vários SEs folhas, um para cada recurso. Na verdade, o GIStorage não impõe nenhum número de SEs, os administradores de cada OV podem instalar qualquer estrutura em árvore que desejarem, de acordo com o número de recursos que possuem e com o desempenho que pretendem atingir. Por exemplo, considerando o *site* CPAD, poder-se-ia instalar um serviço de diretório e um SE raiz na Marfim (máquina *front-end* do *cluster*), duas SEs folhas para cada *cluster* e um SE folha para cada *host* existente.

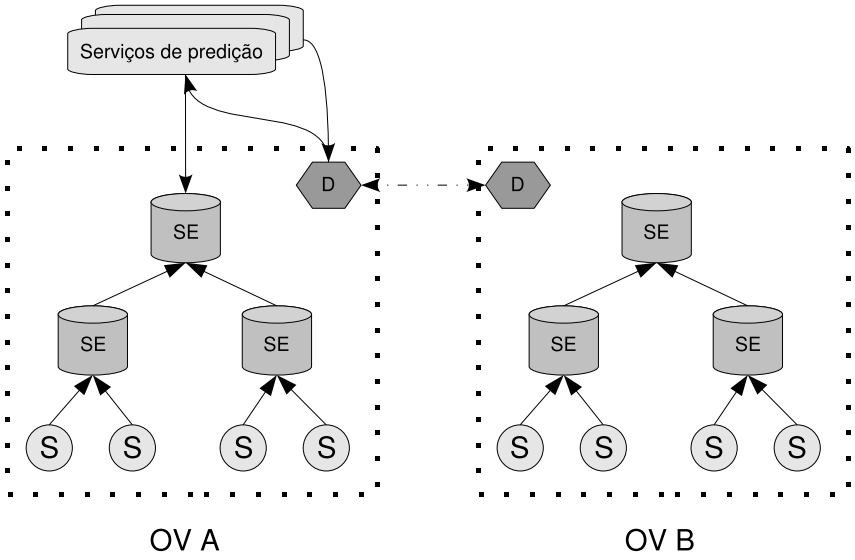

Figura 16 – Arquitetura distribuída do GIStorage.

É importante notar que o serviço de diretório não armazena informações relativas as demais OV que compõem a grade. Entretanto, um D armazena o endereço dos demais diretórios a fim de facilitar aos usuários e serviços de predição a obtenção de informações sobre os demais participantes da grade.

As interações entre os serviços de predição, GIStorage e sensores podem ser vistas nas Figuras 17 e 18. Em todos os casos, o sensor da máquina A envia com uma determinada freqüência informações sobre a carga do processador para um SE folha. Este último está configurado para armazenar esta informação e repassá-la para o SE raiz. Após receberem pela primeira vez os dados ambos registram-se no diretório, informando quais dados sobre qual máquina possuem e o intervalo de tempo que cada um armazena. O SE raiz e folha são previamente configurados para armazenar um determinado intervalo de tempo, sendo o do primeiro inferior ao do segundo.

Quando um serviço de predição quer obter uma série temporal sobre uma característica de um recurso, envia uma mensagem para o diretório especificando qual recurso, informação e intervalo de tempo que deseja. O D executa uma pesquisa em seus registros e envia para o serviço de predição o SE que armazena a informação desejada dentro do intervalo de tempo solicitado. Caso exista mais de um SE responsável, o diretório enviará aquele que armazenar o menor intervalo de tempo possível, ou seja, o SE que estiver numa posição mais alta na hierarquia.

O caso mais comum, onde as informações solicitadas encontram-se no SE raiz, está ilustrado na Figura 17. O serviço de predição solicita ao diretório qual SE tem armazenado os últimos cinco minutos da carga do processador da máquina A. Como essa informação é recente ela está contida no SE raiz assim como no SE folha, contudo, o diretório envia como resposta ao serviço

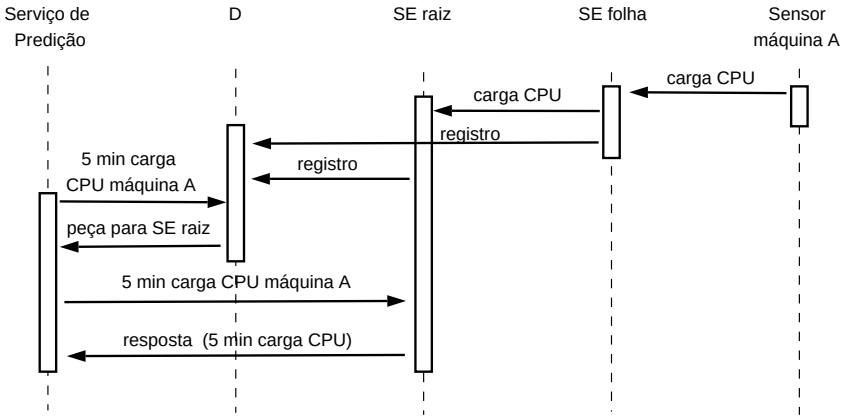

Figura 17 – Interação entre os componentes do GIStorage - Caso 1.

de predição o SE raiz, que se encontra mais acima na hierarquia. Por fim, o serviço de predição faz o pedido ao SE raiz e obtém a resposta.

Existirão casos em que o intervalo de tempo solicitado será elevado, como na Figura 18. Neste caso, o serviço de predição solicita a última hora de carga do processador. O diretório, após pesquisar nos seus registros, indica ao serviço de predição o SE folha pois este é o único que possui a informação dentro do intervalo de tempo requerido. O serviço de predição então entra em contato com o SE folha e obtém a informação desejada.

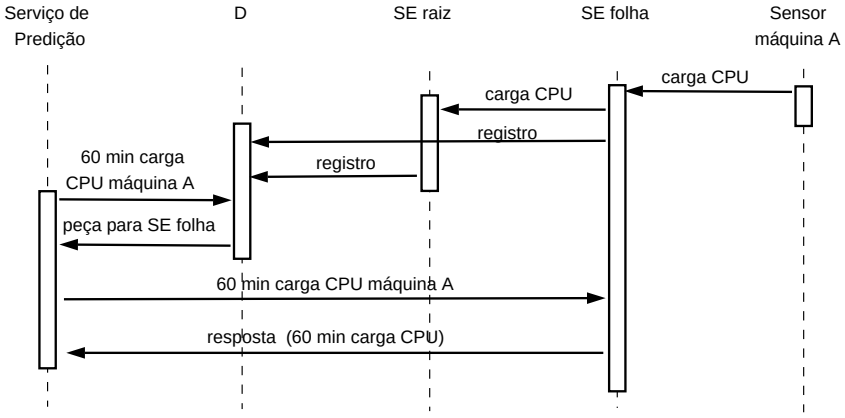

Figura 18 – Interação entre os componentes do GIStorage - Caso 2.

Um terceiro caso ocorre quando o serviço de predição (ou um usuário) deseja obter informações sobre recursos de outras OV. Quando isso ocorre, o serviço de predição solicita ao diretório conhecido que lhe envie o endereço dos demais diretórios existentes na grade. Essa operação é comum na etapa de descoberta de recursos, onde o usuário deseja conhecer os recursos que estão disponíveis e também quando uma informação está parcialmente armazenada em um par de OV. Como exemplo deste último cita-se informações sobre a largura de banda entre dois recursos provenientes de duas OV distintas. A largura de banda num determinado sentido pode estar sendo armazenada em uma OV e no sentido oposto na outra.

A arquitetura do GIStorage foi projetada para obter um melhor tempo de resposta às solicitações que requerem um intervalo de tempo pequeno, comuns quando se deseja predizer diferentes cenários de escalonamento. Devido a limitações de tempo a etapa de escalonamento não deve ser maior que uma fração do tempo total de execução da aplicação. Além disso, para predizer escalonamento geralmente é solicitado um pequeno intervalo de dados uma vez que o tempo de execução do algoritmo de predição é proporcional ao número de elementos da série.

Considerando-se a meta de obter menor tempo de resposta, supõe-se que um único SE seria mais rápido do que vários devido a um número menor de mensagens trocadas (entre o serviço de predição e o diretório e entre o primeiro e o SE). Por outro lado, um único SE não seria capaz de armazenar muitas informações por causa de restrições no espaço de armazenamento disponível. Muitos SEs poderiam armazenar um volume maior de informações, apesar de reduzirem o tempo de resposta devido a um número maior de mensagens trocadas (entre o serviço de predição e o diretório e entre o primeiro e cada um dos SEs existentes). Embora não tenha sido comprovada, uma suposição semelhante foi levantada por Smith *at al* [55].

Acredita-se que a arquitetura baseada em árvores do GIStorage consegue obter um tempo de resposta adequado às solicitações de intervalo de tempo reduzido e ainda armazenar um grande volume de informações. Solicitações com intervalo de tempo reduzido podem ser obtidas no SE raiz com um tempo de resposta menor, enquanto que solicitações com intervalos maiores podem ser obtidas através dos SEs folhas, embora com tempo de resposta mais elevado.

Outros trabalhos propuseram arquiteturas para serviços de informação baseados em árvores hierárquicas [30,32,33], contudo, GIStorage se diferencia por empregar um serviço de diretório e por ter como meta reduzir o tempo de resposta e ainda armazenar uma quantidade significativa de informações[4] . Acredita-se que o emprego de um serviço de diretório alcance tempos de resposta menores do que utilizar o SE raiz como um procurador, obtendo as informações dos SEs folhas sob demanda. Quando se utiliza um D, serviços de predição podem se conectar diretamente com o SE que possui a informação desejada. Sem utilizá-lo a informação tem que ser movida pela hierarquia a partir do SE folha, passando pelo SE raiz até finalmente ser entregue ao serviço de predição.

Em oposição a arquitetura proposta, resultados obtidos por Zang *at al* [56] indicam que um elemento central, que armazena informações provenientes de vários sensores não é escalável uma vez que o número de inserções é mais elevado que o número de pesquisas, que acabam tendo um tempo de resposta demasiadamente elevado. Este cenário é semelhante ao uso de um SE raiz, entretanto, supõe-se que ao dividir a responsabilidade por agregar informações entre vários SEs raízes possa tornar a arquitetura do GIStorage mais escalável. Por exemplo, pode-se empregar um SE raiz para cada *cluster* de uma OV caso estes sejam compostos por numerosos nodos. O serviço de diretório mais uma vez seria útil para indicar com precisão qual SE contém informação sobre um determinado *cluster*.

A arquitetura do GIStorage definida anteriormente pode ser implementada com as tecnolo-

[4]Comparações entre o GIStorage e os demais serviços serão descritas no Capítulo 5.

gias atuais de *Web Services*, em especial, usando-se o padrão WSRF. Nota-se que tanto o SE quanto o D são serviços de fato. Contudo, serviços de predição e sensores são clientes dos anteriores e não são necessariamente implementados como *Web Services*.

5 Estudo de caso

Para avaliar a arquitetura do GIStorage descrita no capítulo anterior é realizado um estudo de caso. Neste estudo, implementou-se o GIStorage em ambiente simulado com intuito de verificar o desempenho e a escalabilidade do mesmo.

5.1 Simuladores de grades computacionais

Simuladores são usados para modelar e avaliar sistemas visando comprovar suas qualidades e apontar suas imperfeições antes de sua implementação. Em especial, para ambientes em grade, os simuladores permitem testar uma solução em cenários controlados, compostos por um grande número de recursos. Simuladores para grades como o MicroGrid [57], SimGrid [58] ou GridSim [59] podem ser usados para avaliar o GIStorage.

O MicroGrid é um simulador usado para testar aplicações implementadas no Globus. Com este simulador pode-se emular um ambiente de larga escala sobre um pequeno conjunto de máquinas. Embora seja útil para depurar aplicações, o MicroGrid exige que a aplicação já esteja implementada.

Por outro lado, tanto SimGrid quanto GridSim permitem que um algoritmo seja testado antes de sua implementação final. Ambos possuem abstrações que permitem modelar o comportamento de uma aplicação, criar cenários com numerosos recursos e simular trocas de mensagens entre os mesmos. GridSim possui como vantagem frente ao SimGrid o suporte a um rico conjunto de entidades de rede [60], permitindo a simulação de cenários com diversos roteadores e *switches*.

Para avaliar o GIStorage, implementou-se um protótipo no simulador GridSim. Através deste protótipo é possível verificar o comportamento do GIStorage.

5.2 Avaliação do GIStorage

O processo de avaliação do GIStorage visa medir o tempo de resposta para pedidos de pequeno intervalo de tempo usando um único SE (solução centralizada), um SE para cada recurso (solução totalmente distribuída) e usando uma árvore de SEs (solução em árvore). Requisições de um curto intervalo de tempo histórico são comuns quando se deseja predizer diferentes ce-

nários de escalonamento, etapa que possui significativa restrição de tempo. Como discutido anteriormente, uma solução centralizada pode ter desempenho melhor do que uma solução totalmente distribuída, entretanto, a última consegue armazenar uma quantidade maior de dados. A arquitetura proposta para o GIStorage busca balancear as qualidades de cada uma dessas soluções sendo esperado que a solução em árvore tenha um bom desempenho e ainda consiga armazenar um grande volume de dados.

Também deseja-se medir o impacto no desempenho usando ou não um serviço de diretório. Apesar de outros serviços de informação para grades não empregarem um serviço de diretório [30, 32], acredita-se que o emprego do mesmo pode evitar transferências desnecessárias, atingindo assim um melhor desempenho. O serviço de diretório permite que usuários conectem precisamente o SE no qual está contida a informação desejada. Sem usá-lo, o SE raiz exerce o papel de procurador solicitando aos SEs corretos a informação, armazenando-a temporariamente e posteriormente repassando-a para o serviço de predição, aumentando o número de transferências.

Embora a arquitetura do GIStorage possa ter melhor desempenho, trabalhos que avaliaram o desempenho do MDS [55,61] e outro serviço de informação com arquitetura semelhante [56], apontam que um elemento central (como o SE raiz) compromete a escalabilidade do serviço. É necessário, portanto, avaliar a escalabilidade da solução em árvore. Em contrapartida, é esperado que a solução em árvore quando empregado mais de um SE raiz tenha melhor escalabilidade.

As seguintes métricas são comumente usadas para medir a escalabilidade de serviços de informação [50,56,61]: tempo de resposta, *throughput*, carga e uso do processador. O tempo de resposta mede o intervalo de tempo transcorrido entre o pedido do usuário (serviço de predição) e o recebimento dos resultados pelo mesmo. O *throughput*, por outro lado, mede o número de pedidos de leituras (ou inserções) por segundo que o serviço é capaz de tratar. A carga do processador calcula o número de processos prontos para usar o mesmo. Já o uso do processador indica a porcentagem de ciclos sendo consumidos.

Tanto o *throughput* quanto a carga e uso do processador são métricas úteis para avaliar o desempenho de serviços já implementados uma vez que exprimem a capacidade que uma determinada implementação tem de tratar numerosos pedidos e o uso que a mesma faz do processador, respectivamente. Como a avaliação do GIStorage levará em conta sua arquitetura em ambiente simulado a métrica que melhor define seu desempenho é o tempo de resposta. Usando-se o tempo de resposta como métrica pode-se medir o desempenho de diversas soluções de arquitetura e também verificar a escalabilidade das mesmas.

5.2.1 Ambiente de teste

Para realizar as medições utilizou-se o ambiente ilustrado na Figura 19. Este ambiente representa uma OV (ou *site*) composta por um agregado de computadores (*cluster*). Nos experimentos o número de nodos do agregado pode variar de 1 até 100, existindo além destes um nodo *front-end* que é acessado pelo serviço de predição.

Os nodos estão conectados por um *switch*, compondo uma rede LAN padrão *ethernet* de 100 Mbps. Já o serviço de predição está conectado ao *front-end* do *cluster* por uma rede de 100 Mbps. Todas as comunicações entre o serviço de predição e os nodos do *cluster* são intermediadas pelo nodo *front-end*.

Cada nodo executa um sensor que coleta informações sobre a carga do processador a cada 60 segundos e as envia para um SE pré-configurado. Tanto o serviço de diretório quanto o SE raiz (quando usado) encontram-se no nodo *front-end*. Os SEs folhas, quando empregados, são executados nos nodos. Além disso, o tamanho das mensagens trocadas entre os sensores, SEs, diretório e serviços de predição é baseado numa implementação em XML do GLUE[2].

As conexões, computadores e usuários (serviços de predição) são modelados pelo GridSim como *threads* independentes. O próprio simulador já possui entidades para simular *switches* e conexões com largura de banda configurável. O simulador possui também um tempo de simulação por meio do qual é possível medir os tempos de respostas das comunicações de dados. Pelo GridSim é possível ainda estipular o desempenho dos processadores simulados, contudo, neste estudo o tempo de processamento das tarefas foi negligenciado uma vez que não é possível estimar precisamente o número de instruções necessárias para cada uma das tarefas.

Os processos de sensores, serviço de diretório e SEs foram gerados pela extensão da classe de objeto *GridResource*, provida pela biblioteca de programação do GridSim. Na extensão fo-

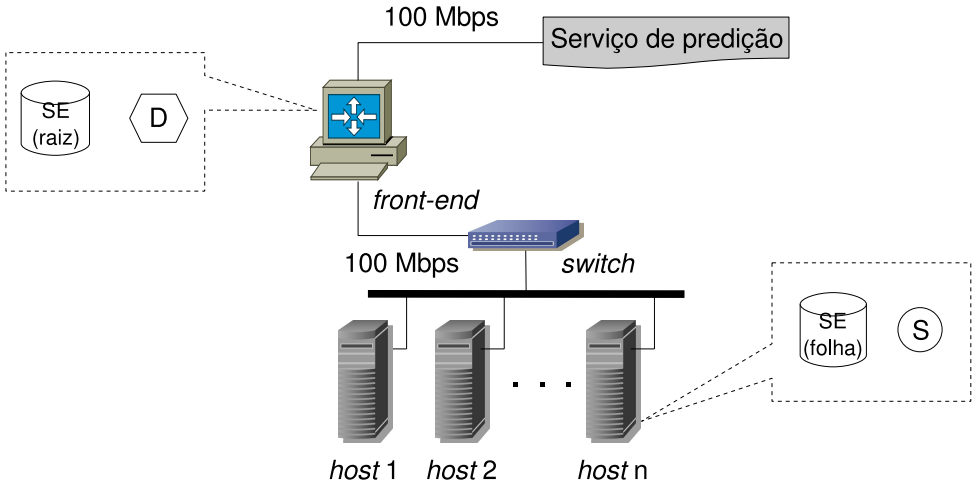

Figura 19 – Ambiente de teste.

ram incorporadas as interações entre as entidades do GIStorage e um protocolo de comunicação foi criado para identificar os tipos de mensagens sendo trocadas. Devido a própria estrutura do GridSim, cada entidade possui três *threads* associadas, uma representa o recurso em si e outras duas gerenciam a entrada e saída de pacotes de mensagens. Esta estrutura acaba gerando um número excessivo de *threads* tornando algumas simulações inviáveis.

5.2.2 Experimento 1: Desempenho das diferentes soluções de arquitetura

O primeiro experimento pretende avaliar o desempenho do GIStorage usando uma arquitetura centralizada, totalmente distribuída e em árvore. Entende-se por centralizada a arquitetura que possui somente um elemento de armazenamento localizado no nodo *front-end*. A solução totalmente distribuída emprega um SE para cada nodo e um serviço de diretório para facilitar a localização das informações. Já a solução em árvore consiste em executar um SE folha em cada nodo e ainda um SE raiz no nodo *front-end*. O SE raiz pode ter duas configurações: armazena apenas a última medição coletada ou armazena a última hora de medições (60 entradas).

Neste experimento um único serviço de predição requere ao GIStorage o histórico dos últimos 60 minutos da carga do processador de todos os nodos do agregado. O número de nodos varia de 1 até 100 e o tempo de resposta à solicitação do serviço de predição é medida. O gráfico da Figura 20 descreve o tempo de resposta com relação ao número de nodos do cenário.

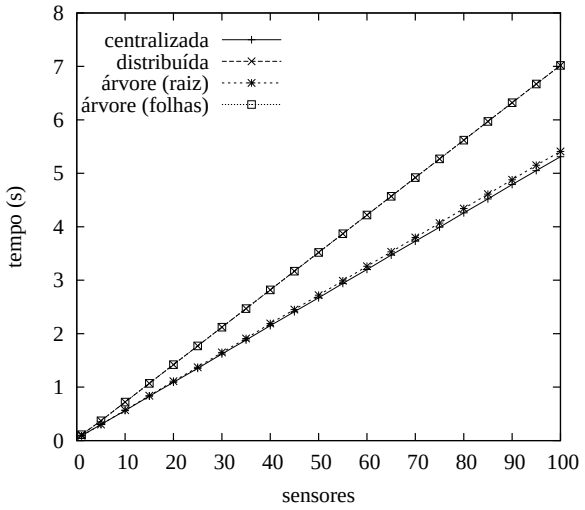

Figura 20 – Tempo de resposta usando uma arquitetura centralizada, totalmente distribuída e baseada em árvore.

Como indicado pelo gráfico, a solução centralizada (ver curva *centralizada*) de fato tem melhor desempenho que a totalmente distribuída (curva *distribuída*). Este resultado deve-se ao número de mensagens trocadas. No caso da solução centralizada são necessárias apenas quatro

mensagens para que o serviço de predição obtenha as informações desejadas (pedido e resposta entre serviço de predição e diretório, e entre o primeiro e o SE centralizado). Já no caso da solução totalmente distribuída são trocadas duas mensagens entre o serviço de predição e o diretório e várias outras entre o serviço de predição e cada um dos SEs folhas.

A solução em árvore obtém aproximadamente o mesmo tempo de resposta do que a solução centralizada (ver curva *árvore (raiz)*) quando o SE raiz armazena toda a informação desejada. Isso porque o número de mensagens trocadas entre o serviço de predição e o GIStorage é o mesmo em ambos os casos. No último caso, quando o SE raiz armazena apenas a última informação coletada, obrigando o serviço de predição a trocar mensagens como cada um dos SEs folhas, atinge-se um desempenho não pior do que na abordagem totalmente distribuída (curva *árvore (folhas)*).

Tanto a solução centralizada quanto a baseada em árvore são, em média, 30% mais rápidas que a totalmente distribuída. Esse resultado além de comprovar que a solução centralizada tem melhor desempenho, mostra que a arquitetura baseada em árvore alcança aproximadamente o mesmo desempenho que a anterior quando é solicitado um intervalo de tempo histórico reduzido. No caso em que a solicitação exige um intervalo não contido no SE raiz, a arquitetura baseada em árvore não tem desempenho significativamente pior do que a solução totalmente distribuída.

A arquitetura baseada em árvore é, portanto, capaz de armazenar tanta informação quanto uma solução totalmente distribuída e ter um desempenho muito próximo da solução centralizada quando o intervalo histórico está sendo armazenado no SE raiz.

5.2.3 Experimento 2: Desempenho usando um serviço de diretório

Acredita-se que o emprego de um serviço de diretório melhora o tempo de resposta ao permitir que o usuário (serviço de predição) acesse precisamente o SE que contém a informação requerida. O segundo experimento explora dois cenários, um onde existe serviço de diretório e outro onde o SE raiz atua como um procurador.

O resultado apresentado na Figura 21 mostra o tempo de resposta para a solicitação de um único serviço de predição em relação ao número de nodos. Tal serviço deseja obter o histórico da carga do processador de todos os nodos do agregado, considerando um intervalo dos últimos 60 minutos. Ambas as curvas foram obtidas usando-se a arquitetura baseada em árvore onde o SE raiz armazena somente o último valor coletado, isto é, o usuário deve acessar cada um dos SEs folhas.

A curva *usando diretório* descreve o resultado obtido quando é empregado um serviço de diretório. Neste caso o usuário primeiro solicita ao diretório qual SE armazena a informação desejada e obtém como resposta a lista de todos os SEs folhas, solicitando a cada um destes a informação desejada. Já a curva *sem usar diretório* ilustra os resultados obtidos quando não

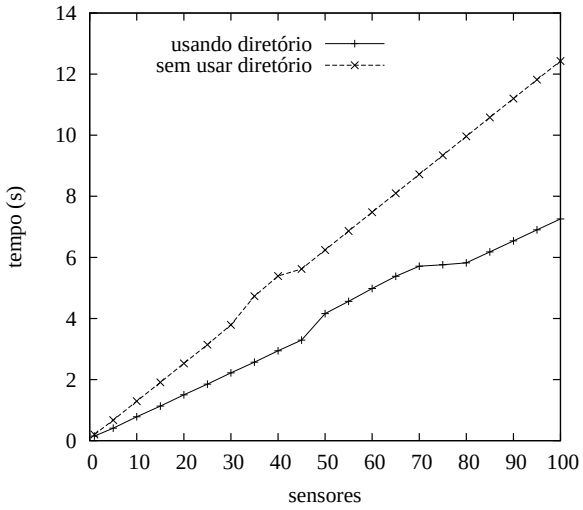

Figura 21 – Tempo de resposta usando e sem usar um serviço de diretório quando a informação encontra-se nos SEs folhas.

existe um serviço de diretório. Neste caso o usuário solicita ao SE raiz a informação e, como este não a armazena, repassa o pedido aos SEs folhas que por sua vez respondem ao SE raiz. Por fim, o SE raiz repassa a informação para o usuário.

A Figura 21 evidencia que o uso de um serviço de diretório de fato tem efeito positivo no tempo de resposta à solicitação do serviço de predição quando a informação não se encontra no SE raiz. O tempo de resposta usando o diretório foi, em média, 64% menor do que não o

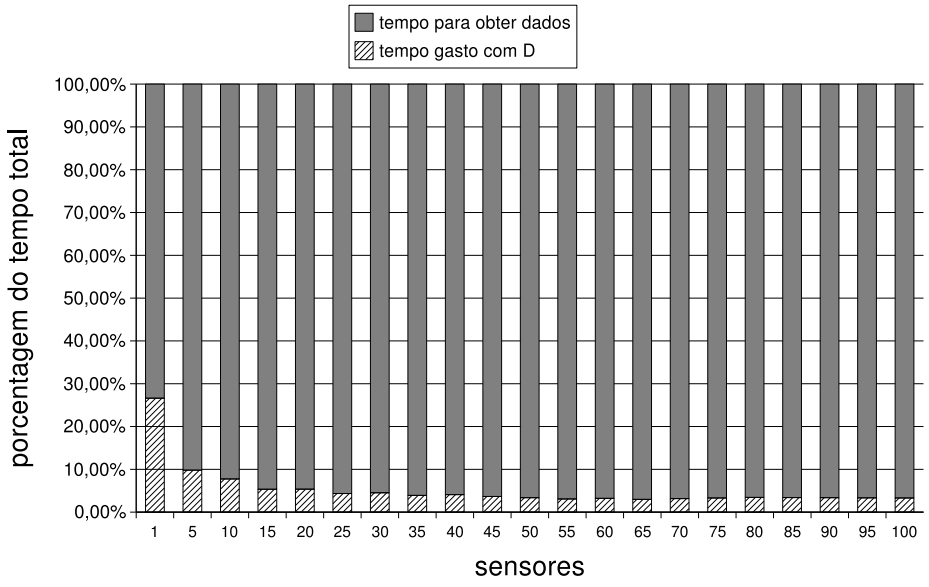

Figura 22 – Porcentagem do tempo de resposta sendo gasto na troca de informação com o diretório.

usando. Por outro lado, quando a informação solicitada encontra-se no SE raiz pelo menos 5% do tempo total é despendido trocando mensagens entre o serviço de predição e o diretório. O fato anterior é ilustrado na Figura 22 onde é mostrado que, neste caso, o uso de um serviço de diretório torna 5% mais lenta a resposta final.

A contribuição negativa que o serviço de diretório concede quando a informação está presente no SE raiz não é tão expressiva quanto a perda de desempenho ao não usá-lo no caso em que a informação encontra-se somente nos SEs folhas. Devido a esta constatação, a arquitetura do GIStorage mantém o serviço de diretório.

5.2.4 Experimento 3: Escalabilidade das diferentes soluções de arquitetura

Como foi indicado por outros trabalhos, um elemento central pode reduzir a escalabilidade de um serviço de informação. No experimento 3 deseja-se verificar se o SE raiz reduz de fato a escalabilidade do serviço.

Neste experimento o número de recursos é fixado em 100 nodos e varia-se o número de usuários (serviços de predição) de 1 até 250. Cada serviço de predição solicita a última hora de dados coletados sobre cada um dos recursos (60 entradas). O tempo médio de resposta para a solicitação de cada usuário é calculado e uma comparação entre a solução totalmente distribuída e baseada em árvore é feita. Como é esperado que a solução baseada em árvore utilizando mais de um SE raiz tenha uma melhor escalabilidade, também se compara esta última empregando dois SEs raízes. Quando é empregado somente um SE raiz este armazena o intervalo de tempo solicitado não sendo necessário obter informações dos SEs folhas. Para o caso em que existem dois SEs, cada um fica responsável por armazenar informações sobre metade dos recursos.

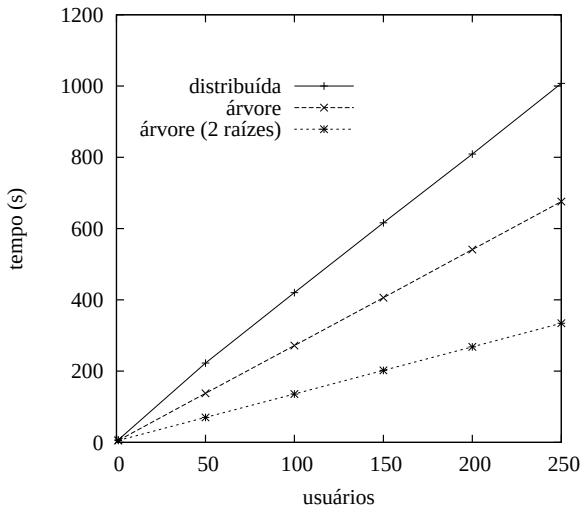

Figura 23 – Média do tempo de resposta para um número variável de usuários.

Ambos também armazenam todo o intervalo de tempo requerido pelos usuários.

O gráfico da Figura 23 exibe a média do tempo de resposta obtido pelos usuários. Como pode-se constatar o tempo de resposta no caso em que se utiliza mais de um SE raiz é o menor, indicando ser este o mais eficiênte em contraste com a solução totalmente distribuída. O que não fica claro neste gráfico é a tendência de reduzir a diferença entre as soluções baseadas em árvore e a totalmente distribuída.

Esta tendência de redução pode ser calculada pelo *speed up* entre a solução totalmente distribuída e cada uma das soluções baseadas em árvore. O resultado obtido é mostrado na Tabela 1 e ilustrado no gráfico da Figura 24. Os resultados apontam que, após um elevado *speed up* quando existem cinqüenta usuários, o bom desempenho das soluções baseadas em árvore começa a decair conforme aumenta o número de serviços de predição. Essa queda de desempenho indica que as soluções baseadas em árvore não escalam tão bem quanto a solução totalmente distribuída. Essa constatação vai de encontro com resultados obtidos por outros trabalhos [55, 56, 61], onde também é afirmado que uma solução totalmente distribuída tem melhor escalabilidade que soluções com elementos centralizados.

Em um cenário onde existe um grande número de usuários o SE raiz acaba se tornando um gargalo, reduzindo o desempenho. Como o SE raiz fica sobrecarregado, não consegue responder adequadamente às solicitações dos usuários. Já na solução totalmente distribuída, o tratamento dos pedidos é divido entre todos os SEs folhas acarretando num melhor desempenho. Espera-se que num cenário composto por numerosos usuários o tempo de resposta médio seja menor quando se usa a solução totalmente distribuída. Infelizmente, devido a limitações do simulador utilizado, não foi possível testar cenários com mais de 250 usuários mesmo confiando ao GridSim 2 GB de memória principal. Por outro lado, pôde-se verificar uma tendência de redução na diferença entre as soluções.

Nota-se que o *speed up* obtido pela solução que emprega dois SEs raízes é maior do que usando somente um SE, o que indica um melhor balanceamento da carga nesta solução. Como a tarefa de responder aos pedidos dos usuários é dividida entre mais de um SE raiz, esta abordagem consegue tratar um número maior de requisições, portanto, tem escalabilidade melhor.

Tabela 1 – *Speed up* da solução totalmente distribuída com relação as soluções em árvore.

Número de usuários	*Speed Up* (totalmente distribuído)	
	árvore	árvore (2 raízes)
1	1,3	1,3
50	1,62	3,19
100	1,55	3,09
150	1,52	3,05
200	1,5	3,02
250	1,49	3,01

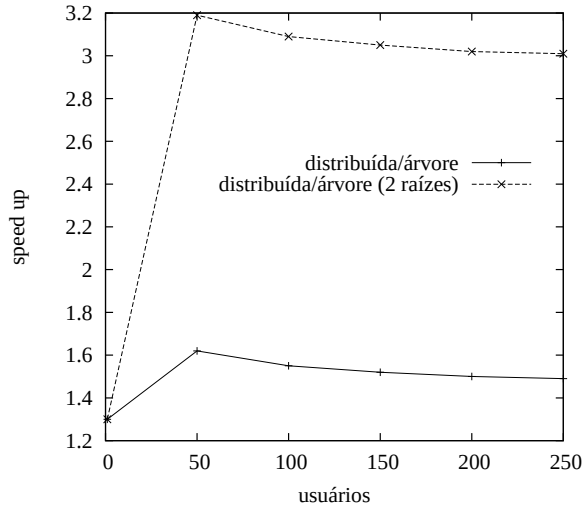

Figura 24 – *Speed Up* da solução totalmente distribuída com relação a baseada em árvore.

Embora, não seja tão escalável quanto a solução totalmente distribuída, uma solução em árvore que emprega mais de um SE raiz atinge um melhor desempenho. Não foi possível determinar o número máximo de usuários que conseguem obter informações mais rapidamente de uma arquitetura baseada em árvore, porém, pode se dizer que esta última ainda possui bom desempenho quando existem até 250 serviços de predição.

5.3 Comparações entre os serviços de informação existentes e o GIStorage

Tanto Remos [20] quanto NWS [22] são serviços de predição completos, possuindo seus próprios sensores, bases de dados e algoritmos de predição. Entretanto, um não consegue trabalhar em cooperação com o outro, impondo a todas as OV participantes da grade a instalação de um deles ou impondo aos programadores o desenvolvimento de código para mais de um serviço. O GIStorage, em contraste, permite as OV empregarem as ferramentas de monitoramento que melhor se adequam as suas necessidades. Programadores podem também desenvolver aplicações que obtêm informações sobre o desempenho passado dos recursos independentemente do fabricante e tipo de sensor. Além disso, novos serviços de predição podem ser construídos de maneira mais simples uma vez que não precisam lidar com problemas relacionados a produção e armazenamento de informações.

O Ganglia [25], uma ferramenta de monitoramento, poderia ser usado para prover informações coletadas de sensores para algoritmos de predição. De maneira similar ao GIStorage, o Ganglia tem sua arquitetura baseada em árvore, onde cada folha coleta informação de desem-

penho e, mais acima na árvore hierárquica, essa informação é armazenada em diferentes intervalos. Em oposição ao GIStorage, o Ganglia não emprega um serviço de diretório, tornando difícil a tarefa de encontrar onde a informação sobre determinado recurso num determinado intervalo de tempo está armazenada. Além disso, o Ganglia não é compatível com padrões para grades, em especial, com o WSRF [9].

Como parte do projeto Globus [28], um serviço de informação para grades conhecido como MDS [2] foi desenvolvido. O MDS se baseia em serviços de índices para agregar informações coletadas por provedores de informação, buscando facilitar a descoberta de serviços e recursos. Embora um serviço de armazenamento esteja previsto para o futuro, até o momento o MDS não tem a habilidade de armazenar informações históricas sobre o desempenho passado dos recursos. Apesar disso, o GIStorage e o MDS tem algumas semelhanças visto que provedores de informação e serviços de índices são análogos aos sensores e serviços de diretório, respectivamente. GIStorage pode ser implementado usando a API do Globus [62], empregando o serviço de índice como seu serviço de diretório e implementando um novo agregador para trabalhar como um SE.

O TGIS [30], outro serviço de informação, também possui algumas semelhanças com o GIStorage. O primeiro é estruturado como uma árvore, entretanto, não emprega um serviço de diretório tal qual faz o segundo. Como demonstrado anteriormente, o uso de um serviço de diretório melhora o tempo de resposta pois permite aos usuários conectarem diretamente os recursos responsáveis pela informação desejada, evitando transferências desnecessárias. Já o FOSIS [32], assim como o TGIS, é estruturado como uma árvore e também não emprega um serviço de diretório, sofrendo do mesmo problema que o anterior. Por outro lado, este último utiliza uma estrutura baseada em florestas, ou seja, um conjunto de árvores para melhor balancear a carga e melhorar a escalabilidade. Tal abordagem é usada pelo GIStorage para melhorar sua escalabilidade.

Outro serviço de informação para grades igualmente estruturado como uma árvore é o OGSI-*based* GIS [33]. Este serviço utiliza uma entidade conhecida como *VO Information Service*, semelhante ao serviço de diretório do GIStorage, para armazenar informações estáticas sobre os recursos e meta-dados indicando onde encontrar as informações dinâmicas. De forma oposta ao GIStorage, este serviço não contempla um elemento agregador de informações como o SE raiz, obrigando os usuários a buscarem informações dinâmicas diretamente nos recursos. Com isso seu desempenho deve se assemelhar ao da solução totalmente distribuída, sendo portanto, menos eficiente que o GIStorage.

O que motivou os projetistas do OGSI-*based* GIS a armazenar somente as informações estáticas e não as dinâmicas num agregador foi a constatação da baixa escalabilidade da segunda abordagem. Os experimentos demonstrados na Seção 5.2.4 indicam que uma solução totalmente distribuída realmente tende a escalar melhor. Por outro lado, ao utilizar mais de um SE raiz, o GIStorage aparenta melhorar sua escalabilidade e ainda ter um desempenho melhor do que a solução totalmente distribuída.

A arquitetura baseada em árvore do GIStorage é semelhante as arquiteturas propostas por outros trabalhos. Entretanto, como demonstrado anteriormente, o emprego do serviço de diretório tem impacto positivo no desempenho do serviço o que eleva a eficiência do GIStorage frente aos serviços TGIS e FOSIS. Por utilizar um serviço de diretório o OGSI-*based* GIS poderia ter um desempenho semelhante ao do GIStorage, contudo, como aquele não emprega um serviço de agregação, como o SE raiz, permanece com desempenho inferior ao deste. Por fim, a utilização de mais de um SE raiz melhora a escalabilidade do GIStorage e mantém seu bom desempenho para solicitações de informações históricas sobre um intervalo reduzido de tempo passado.

6 Considerações finais

Serviços de descoberta e alocação de recursos e escalonamento de tarefas podem empregar algoritmos de predição de desempenho em suas análises para melhorar os seus resultados [3–6]. Diversos algoritmos de predição [6, 21, 23] baseiam-se em séries temporais que, por sua vez, requerem informações históricas sobre o desempenho dos recursos.

Para dar suporte a algoritmos de predição este trabalho propôs a arquitetura de um novo serviço de informação para grades computacionais cujo objetivo principal é armazenar informações históricas sobre o desempenho dos recursos. Tal serviço, chamado de GIStorage, permite que serviços de predição concentrem-se somente no processamento das informações coletadas ao invés de gerenciar sensores e as informações produzidas, liberando-os de como problemas relacionados a grande quantidade de informação, escalabilidade, entre outros.

Através do GIStorage, usuários podem acessar informações históricas sobre o desempenho dos recursos independentemente do algoritmo ou serviço de predição empregado. Além disso, organizações virtuais podem usar os sensores que melhor atendem as suas políticas internas, não sendo imposta a instalação de sensores compatíveis com determinados serviços de predição.

Antes de ser projetada a arquitetura do GIStorage, foi necessário definir um modelo de informação para expressar o desempenho dos recursos. Com o intuito de empregar um modelo padronizado optou-se pelo GLUE *schema* [34]. Apesar de já estar sendo usado por ferramentas como o Globus [28], Ganglia [25] e pelo OGSI-*based* GIS [33], o modelo GLUE *schema* não contempla informações referentes a rede de interconexão tampouco é capaz de relacionar os eventos com o momento em que ocorreram.

Como parte deste trabalho criou-se o GLUE^2 (GLUE *schema Extended*), um modelo de informação baseado no GLUE *schema* capaz de descrever medições sobre a rede de interconexão e relacioná-las com uma marcação de tempo. O modelo GLUE^2 combina tanto o GLUE *schema* quanto as recomendações do OGF [11], que identificou quais características da rede são relevantes às aplicações e, posteriormente, elaborou uma terminologia padronizada para descrever medições de rede [37].

O modelo de informação GLUE^2 é usado pelo GIStorage para representar as informações sobre os recursos e as redes de interconexão. Tanto os sensores quanto os serviços de predição devem compreendê-lo a fim de publicar e obter dados do serviço de informações, respectivamente.

O modelo GLUE^2 apresenta uma possível solução aos problemas relacionados com a heterogeneidade de recursos e conjunto de informações. Entretanto, um único documento contendo informações relativas a todos os recursos de uma grade não seria viável devido a grande quan-

tidade de participantes de uma grade real. Além disso, uma solução centralizada reduziria o desempenho de toda a grade ao criar um gargalo no sistema e, caso este gargalo falhasse, poderia inviabilizar o uso de todo o ambiente. Devido a isso a arquitetura do GIStorage foi definida de forma distribuída.

Durante a definição de sua arquitetura buscou-se novamente seguir as recomendações do OGF para serviços de monitoramento para grades (GMA) [52]. Assim como na arquitetura do GMA, o GIStorage interage com consumidores (usuários e serviços de predição) e produtores (sensores). O GIStorage propriamente dito age tanto como um consumidor, obtendo informações de sensores, quanto como um produtor, disponibilizando-as para os serviços de predição.

O GIStorage é formado por uma série de elementos de armazenamento (SE) que são responsáveis por obter as informações de sensores e armazená-las, permitindo que serviços de predição possam recuperar informações sobre o histórico do desempenho dos recursos da grade. Os SEs estão organizados numa estrutura baseada em árvore para atingir um bom desempenho e ainda armazenar uma grande quantidade de dados. Adicionalmente, para encontrar facilmente o SE que armazena a informação desejada, o GIStorage contempla um serviços de diretório. Tal serviço registra qual SE é responsável por qual informação e qual o intervalo de tempo que o mesmo mantém em sua base de dados.

Outros serviços de informação propostos utilizam estruturas em árvore [30, 32, 33], entretanto, somente o GIStorage emprega um serviço de diretório e SEs raízes que armazenam informações recentemente coletadas. Como resultado dos experimentos descritos no capítulo anterior, foi possível comprovar o melhor desempenho obtido quando se emprega um elemento central para armazenar as informações recentes. Os resultados indicam também que a arquitetura do GIStorage tem uma boa escalabilidade ao se utilizar mais de um SE raiz quando a OV possui muitos recursos ou é acessado por um elevado número de usuários. No Capítulo 5 comprovou-se também que o uso de um serviço de diretório tem impacto positivo no desempenho do GIStorage.

Para os objetivos propostos o GIStorage possui qualidades frente aos demais serviços de informação. Por utilizar um serviço de diretório e um elemento que armazena as informações mais recentes o GIStorage apresenta um bom desempenho para solicitações que envolvem intervalos de tempo histórico reduzido, comuns quando se pretende predizer diferentes cenários de escalonamento. Por outro lado, quando o intervalo de tempo necessário é elevado, o GIStorage não tem desempenho pior do que uma solução totalmente distribuída, onde ocorrem numerosas trocas de mensagens entre o serviço de predição e os elementos de armazenamento.

A nova arquitetura proposta além de seguir os padrões atuais para grades computacionais cumpre a proposta inicial de obter um bom desempenho e conseguir armazenar uma grande quantidade de informações. O modelo de informação GLUE^2 consegue representar informações variadas condizentes com as necessidades de algoritmos de predição. Por fim, a solução distribuída do GIStorage tem boa escalabilidade.

Apesar de ser projetado para dar suporte a algoritmos de predição, o GIStorage tem ou-

tros usos. Pode-se empregar o GIStorage para administrar os recursos de uma organização, indicando gargalos ou apontando os recursos que estão fora de operação. É possível também empregar o GIStorage para verificar contratos de utilização de grades, como faz o Inca [63].

O GIStorage pode ser utilizado na depuração de aplicações paralelas, relacionando o desempenho dos recursos com a execução da aplicação, de maneira similar ao Netlogger [64]. Para isso é necessário adicionar ao mesmo a capacidade de armazenar informações provenientes das aplicações.

Uma questão importante que não foi contemplada neste trabalho é a segurança das informações. Entende-se por segurança a capacidade do serviço de autorizar os usuários a fazer leituras e os sensores a inserir informações. Mais uma vez deve-se seguir as recomendações do OGF [65] e utilizar uma infra-estrutura de chaves públicas (PKI) através da qual tanto os usuários quanto os sensores possuem certificados que os autenticam e permitem que os mesmos leiam ou insiram informações, respectivamente. Técnicas semelhantes são utilizadas pelo Globus [66].

Uma outra corrente de pesquisa em ascenção propõe o chamado *semantic grid* [67], grades onde a informação, recursos e serviços são descritos de forma padronizada, permitindo a um processo automatizado extrair o significado das informações. Pode-se adequar o GIStorage a este tipo de grade computacional. Em especial, já foi implementado por Niinimäki *at al* [68] um mapeamento do modelo de informação GLUE *schema* em ontologia. Um mapeamento do GLUE^2 também pode ser feito.

Referências

[1] FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, v. 15, n. 3, p. 200–222, 2001. ISSN 1094-3420.

[2] CZAJKOWSKI, K. et al. Grid information services for distributed resource sharing. In: *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*. Washington, DC, USA: IEEE Computer Society, 2001. p. 181.

[3] DOWNEY, A. B. Predicting queue times on space-sharing parallel computers. In: *11th International Parallel Processing Symposium*. [S.l.]: IEEE Computer Society, 1997. p. 209–218.

[4] SMITH, W. Improving resource selection and scheduling using predictions. In: NABRZYSKI, J.; SCHOPF, J.; WĘGLARZ, J. (Ed.). *Grid Resource Management, State of the art and future trends*. [S.l.]: Kluwer Academic Publishers, 2004. cap. 16, p. 237–253.

[5] BERMAN, F. D. et al. Application-level scheduling on distributed heterogeneous networks. In: *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 1996. p. 39–52. ISBN 0-89791-854-1.

[6] YANG, L.; SCHOPF, J. M.; FOSTER, I. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In: *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2003. p. 31–46.

[7] LASZEWSKI; SMITH, W.; TUECKE, S. A directory service for configuring high-performance distributed computations. In: *HPDC '97: Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*. [S.l.: s.n.], 1997.

[8] CHRISTENSEN, E. et al. *Web Services Description Language*. World Wide Web Consortium (W3C), 2006. Relatório Técnico. Acessado em novembro de 2006. Disponível em: <http://www.w3.org/TR/wsdl>.

[9] SNELLING, D.; ROBINSON, I.; BANKS, T. *Web Services Resource Framework*. Organization for the Advancement of Structured Information Standards (OASIS), 2006. Relatório Técnico. Acessado em novembro de 2006. Disponível em: <http://www.oasis-open.org>.

[10] OASIS. *Organization for the Advancement of Structured Information Standards (OASIS)*. Acessado em novembro de 2006. Disponível em: <http://www.oasis-open.org>.

[11] OPEN GRID FORUM. *Open Grid Forum Home Page*. Acessado em julho de 2006. Disponível em: <http://www.ogf.org/>.

[12] W3C. *World Wide Web Consortium (W3C)*. Acessado em novembro de 2006. Disponível em: <http://www.w3.org/>.

[13] SCHOPF, J. M. Ten actions when grid scheduling. In: NABRZYSKI, J.; SCHOPF, J.; WĘGLARZ, J. (Ed.). *Grid Resource Management, State of the art and future trends*. [S.l.]: Kluwer Academic Publishers, 2004. cap. 2, p. 15–24.

[14] FOSTER, I. et al. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In: *Proceedings of the International Workshop on Quality of Service*. London, UK: IEEE Computer Society, 1999. p. 27–36.

[15] JARVIS, S. et al. Performance prediction and its use in parallel and distributed computing systems. In: *Parallel and Distributed Processing Symposium*. [S.l.]: IEEE Computer Society, 2003. p. 8–15.

[16] DINDA, P. A. Online prediction of the running time of tasks. In: *High Performance Distributed Computing. 10th IEEE International Symposium on Performance*. [S.l.]: IEEE Computer Society, 2001. p. 336–337.

[17] SMITH, W.; FOSTER, I.; TAYLOR, V. Predicting application run times using historical information. *Lecture Notes in Computer Science*, v. 1459, p. 122–141, 1998.

[18] CZAJKOWSKI I. FOSTER, C. K. k. Agreement-based resource management. In: *Proceedings of the IEEE*. [S.l.]: IEEE Computer Society, 2005. v. 93, p. 631–643.

[19] GROSHWITZ, N.; POLYZOS, G. A time series model of long-term nsfnet backbone traffic. In: *Conference Record, Serving Humanity Through Communications*. [S.l.]: IEEE Computer Society, 1994. v. 3, p. 1400–1404.

[20] DINDA, P. A. et al. The architecture of the remos system. In: *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*. Washington, DC, USA: IEEE Computer Society, 2001. p. 252–268.

[21] WOLSKI, R. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 119–132, 1998. ISSN 1386-7857.

[22] WOLSKI, R.; SPRING, N. T.; HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, Elsevier, v. 15, n. 5–6, p. 757–768, 1999.

[23] DINDA, P. A. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, v. 17, n. 2, p. 160–173, fev. 2006.

[24] KERBYSON, D. et al. Is predictive tracing too late for hpc users. In: *High Performance Computing*. United Kingdon: Plenum Press, 1998. p. 57–67.

[25] MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, Elsevier, v. 30, n. 7, p. 817–840, jul 2004.

[26] NAGIOS. *The Nagios Project Home page*. Acessado em maio de 2006. Disponível em: <http://www.nagios.org/>.

[27] HAWKEYE. *The Hawkeye Project Home page*. Acessado em maio de 2006. Disponível em: <http://www.cs.wisc.edu/condor/hawkeye>.

[28] FOSTER, I.; KESSELMAN, C. The globus toolkit. In: FOSTER, I.; KESSELMAN, C. (Ed.). *The Grid: Blueprint for a New Computing Infraestruture*. [S.l.]: Morgan Kaufmann Publishers, 1999. cap. 11, p. 259–278.

[29] FOSTER, I. T. Globus toolkit version 4: Software for service-oriented systems. In: JIN, H.; REED, D. A.; JIANG, W. (Ed.). *Proceedings of Network and Parallel Computing, IFIP International Conference, NPC 2005*. [S.l.]: Springer-Verlag, 2005. v. 3779, p. 2–13.

[30] CHEN, Y. et al. A framework of a tree-based grid information service. In: *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*. [S.l.: s.n.], 2005.

[31] FOSTER, I. et al. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002. 1-15 p.

[32] SUN, H. et al. A new architecture for ogsa-based grid information service. In: *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*. [S.l.: s.n.], 2004.

[33] ZANG, T. et al. The design and implementation of an ogsa-based grid information service. In: *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*. [S.l.: s.n.], 2004.

[34] ANDREOZZI, S. et al. *Glue Schema Specification version 1.2*. Glue schema project, 2005. Relatório Técnico. Acessado em julho de 2006. Disponível em: <http://infnforge.cnaf.infn.it/glueinfomodel/>.

[35] GLOBUS ALLIANCE. *How-to configure GLUE in Globus*. Acessado em julho de 2006. Disponível em: <http://www.globus.org/toolkit/mds/glueschemalink.html>.

[36] ROSE, C. D.; NAVAUX, P. Fundamentos de processamento de alto desempenho. In: YAMIN, A.; BARBOSA, J. L. V. (Ed.). *Anais da Escola Regional de Alto Desempenho (ERAD 2004)*. [S.l.]: Sociedade Brasileira de Computação, 2004. cap. 2, p. 15–24.

[37] LOWEKAMP, B. et al. *A hierarchy of Network Performance Characteristics for Grid Applications and Services*. Open Grid Forum, 2003. Relatório Técnico. Acessado em julho de 2006. Disponível em: <http://www.ogf.org/>.

[38] TANENBAUM, A. *Computer Networks*. San Francisco, CA, USA: Prentice Hall Professional Technical Reference, 2002.

[39] SUBRAMANYAN, R.; MIGUEL-ALONSO, J.; FORTES, J. A. B. A scalable snmp-based distibuted monitoring system for heterogeneous network computing. In: *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. [S.l.: s.n.], 2000.

[40] LEE, C. A. et al. A network performance tool for grid environments. In: *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*. [S.l.: s.n.], 1999.

[41] ROUGHAN, M. Fundamental bounds on the accuracy of network performance measurements. In: *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. [S.l.: s.n.], 2005.

[42] OBRACZKA, K.; GHEORGHIU, G. The performance of a service for network-aware applications. In: *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*. [S.l.: s.n.], 1998.

[43] SESHAN, S.; STEMM, M.; KATZ, R. H. *SPAND: Shared Passive Network Performance Discovery*. Berkeley, CA, USA: University of California at Berkeley, 1997. Relatório Técnico. Acessado em novembro de 2006. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/1997/5839.html>.

[44] OURGRID. *Ourgrid Project Home Page*. Acessado em novembro de 2006. Disponível em: <http://www.ourgrid.org/>.

[45] SWANY, M.; WOLSKI, R. Representing dynamic performance information in grid environments with the network weather service. In: *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. [S.l.: s.n.], 2002.

[46] MILLER, N.; STEENKISTE, P. Collecting network status information for network-aware applications. In: *INFOCOM*. [S.l.: s.n.], 2000. p. 641–650.

[47] HODGES, J.; MORGAN, R. *Lightweight Directory Access Protocol (v3): Technical Specification*. , United States: RFC Editor, 2002. Relatório Técnico.

[48] COOKE, A. W. et al. The relational grid monitoring architecture: Mediating information about the grid. *Journal of Grid Computing*, v. 2, n. 4, p. 323–339, 2004.

[49] PLALE, B.; DINDA, P.; LASZEWSK, G. von. Key concepts and services of a grid information service. In: *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002)*. Louisville, KY: MIT Cambridge, 2002.

[50] ZHANG, X.; FRESCHL, J. L.; SCHOPF, J. M. A performance study of monitoring and information services for distributed systems. In: *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*. [S.l.: s.n.], 2003.

[51] KONYA, B.; FIELD, L.; ANDREOZZI, S. *GLUE Schema Home Page*. Acessado em novembro de 2006. Disponível em: <http://glueschema.forge.cnaf.infn.it/>.

[52] TIERNEY, B. et al. *A Grid Monitoring Architecture*. Open Grid Forum, 2002. Relatório Técnico. Acessado em julho de 2006. Disponível em: <http://www.ogf.org/>.

[53] GROVE, M. B. jgma: A lightweight implementation of the grid monitoring architecture. In: *Proceedings of the UK e-Science All Hands Meeting*. [S.l.: s.n.], 2004.

[54] PYTHON GMA PROJECT. *Python GMA Project Home Page*. Acessado em novembro de 2006. Disponível em: <http://py-gma.sourceforge.net/>.

[55] SMITH, W. et al. An evaluation of alternative designs for a grid information service. *Cluster Computing*, v. 4, n. 1, 2001.

[56] ZANG, T. et al. An ogsi-compliant grid information service - its architecture and performance study. In: *HPCASIA '04: Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference on (HPCAsia'04)*. [S.l.: s.n.], 2004.

[57] SONG, H. J. et al. The microgrid: a scientific tool for modeling computational grids. In: *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. [S.l.: s.n.].

[58] LEGRAND, A.; MARCHAL, L.; CASANOVA, H. Scheduling distributed applications: the simgrid simulation framework. In: *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. [S.l.: s.n.], 2003.

[59] BUYYA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Journal of Concurrency and Computation: Practice and Experience (CCPE)*, Wiley Press, v. 14.

[60] SULISTIO, A. et al. Constructing a grid simulation with differentiated network service using gridsim. In: *Proc. of the 6th International Conference on Internet Computing (ICOMP'05)*. [S.l.: s.n.], 2005. p. 20–30.

[61] ZHANG, X.; SCHOPF, J. Performance analysis of the globus toolkit monitoring and discovery service, mds2. In: *IEEE International Conference on Performance, Computing, and Communications*. [S.l.: s.n.], 2004. p. 843.

[62] SOTOMAYOR, B.; CHILDERS, L. *Globus Toolkit 4, First Edition: Programming Java Services (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[63] SMALLEN, S. et al. The inca test harness and reporting framework. In: *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. [S.l.: s.n.], 2004.

[64] LEE, J. et al. Monitoring data archives for grid environments. In: *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002. p. 1–10.

[65] HUMPHREY, M.; THOMPSON, M. R. Security implications of typical grid computing usage scenarios. In: *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*. Washington, DC, USA: IEEE Computer Society, 2001. p. 95.

[66] FOSTER, I. et al. A security architecture for computational grids. In: *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*. [S.l.: s.n.], 1998.

[67] ROURE, N. J. D. D.; SHADBOLT, N. The semantic grid: Past, present, and future. In: *Proceedings of the IEEE*. Washington, DC, USA: IEEE Computer Society, 2005. v. 93, p. 669–681.

[68] NIINIMäKI, M.; TOIVONEN, S.; NIEMI, T. Modelling, searching, and utilising grid resources. In: *Proceedings of the 15th European - Japanese Conference on Information Modelling and Knowledge Bases*. [S.l.: s.n.], 2005.

Apêndice A - Modelo de informação GLUE^2

Schema XML do modelo de informação GLUE^2.

```
<?xml version="1.0"?>
<xs:schema
    sxmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.cpad.pucrs.br/archiveService-v0.2"
    xmlns="www.cpad.pucrs.br/archiveService-v0.2"
    elementFormDefault="qualified"
>

<xs:complexType name="characValueType">
    <xs:simpleContent>
        <xs:extension base="xs:decimal">
            <xs:attribute name="Timestamp"
                type="xs:nonNegativeInteger"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="memoryType">
    <xs:sequence>
        <xs:element name="RAMAvailable" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="VirtualAvailable" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                           type="characValueType"
                           maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="loadType">
    <xs:sequence>
        <xs:element name="Last1Min">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Last5Min">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Last15Min">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="benchmarkType">
    <xs:sequence>
        <xs:element name="Value"
            type="characValueType"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string"/>
</xs:complexType>

<xs:complexType name="StorageDeviceType">
    <xs:attribute name="Name" type="xs:string" use="optional"/>
    <xs:attribute name="Type" type="xs:string" use="optional"/>
    <xs:attribute name="TransferRate" type="xs:integer"
        use="optional"/>
```

```
    <xs:attribute name="Size"
        type="xs:integer"
        use="optional"/>
    <xs:sequence>
        <xs:element name="StoragePartition"
            type="StoragePartitionType"
            minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="StoragePartitionType">
    <xs:attribute name="Name" type="xs:string" use="optional"/>
    <xs:attribute name="Size" type="xs:string" use="optional"/>
    <xs:attribute name="ReadRate" type="xs:integer"
        use="optional"/>
    <xs:attribute name="WriteRate" type="xs:integer"
        use="optional"/>
    <xs:sequence>
        <xs:element name="FileSystem"
            type="FileSystemType"
            minOccurs="0"
            maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="FileSystemType">
    <xs:attribute name="Name" type="xs:string" use="optional"/>
    <xs:attribute name="Root" type="xs:string" use="optional"/>
    <xs:attribute name="Size" type="xs:integer" use="optional"/>
    <xs:attribute name="ReadOnly" type="xs:boolean"
        use="optional"/>
    <xs:attribute name="Type" type="xs:string" use="optional"/>
    <xs:attribute name="Remote"
        type="xs:boolean"
        use="optional"/>
    <xs:sequence>
        <xs:element name="AvailableSpace">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

```
<xs:simpleType name="CEStatusEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Production"/>
        <xs:enumeration value="Queueing"/>
        <xs:enumeration value="Draining"/>
        <xs:enumeration value="Closed"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LRMSTypeEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="OpenPBS"/>
        <xs:enumeration value="LSF"/>
        <xs:enumeration value="Condor"/>
        <xs:enumeration value="BQS"/>
        <xs:enumeration value="CondorG"/>
        <xs:enumeration value="FBSNG"/>
        <xs:enumeration value="Torque"/>
        <xs:enumeration value="PBSPro"/>
        <xs:enumeration value="SGE"/>
        <xs:enumeration value="NQE"/>
        <xs:enumeration value="fork"/>
        <xs:enumeration value="other"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LRMSTypeOpenEnum">
    <xs:union memberTypes="LRMSTypeEnum xs:string"/>
</xs:simpleType>

<xs:simpleType name="UniqueIDType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="LocalIDType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="DirType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:complexType name="CEVOViewType">
    <xs:sequence>
        <xs:element name="ACL" type="ACLType" minOccurs="0"/>
        <xs:element name="RunningJobs"
            type="xs:integer"
```

```
            minOccurs="0"/>
        <xs:element name="WaitingJobs"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="TotalJobs"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="FreeJobSlots"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="EstimatedResponseTime"
            type="xs:integer" minOccurs="0"/>
        <xs:element name="WorstResponseTime"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="ApplicationDir"
            type="xs:string"
            minOccurs="0"/>
        <xs:element name="DataDir"
            type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="LocalID"
            type="LocalIDType"
            use="required"/>
</xs:complexType>

<xs:complexType name="ACLType">
    <xs:sequence>
        <xs:element name="Rule"
            type="xs:string"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="computingElementType">
    <xs:sequence>
        <xs:element name="LRMSType"
               type="LRMSTypeOpenEnum"
               minOccurs="0"/>
        <xs:element name="LRMSVersion"
               type="xs:string"
               minOccurs="0"/>
        <xs:element name="GRAMVersion"
               type="xs:string"
               minOccurs="0"/>
        <xs:element name="HostName"
               type="xs:string"
```

```
        minOccurs="0"/>
    <xs:element name="GateKeeperPort"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="JobManager"
        type="xs:string"
        minOccurs="0"/>
    <xs:element name="ContactString"
        type="xs:string"
        minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="ApplicationDir"
        type="DirType"
        minOccurs="0"/>
    <xs:element name="DataDir"
        type="DirType"
        minOccurs="0"/>
    <xs:element name="Status"
        type="CEStatusEnum"
        minOccurs="0"/>
    <xs:element name="RunningJobs"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="WaitingJobs"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="TotalJobs"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="EstimatedResponseTime"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="WorstResponseTime"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="FreeJobSlots"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="MaxWallClockTime"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="MaxCPUTime"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="MaxTotalJobs"
        type="xs:integer"
        minOccurs="0"/>
    <xs:element name="MaxRunningJobs"
```

```
                type="xs:integer"
                minOccurs="0"/>
            <xs:element name="Priority"
                type="xs:integer"
                minOccurs="0"/>
            <xs:element name="AssignedJobSlots"
                type="xs:integer"
                minOccurs="0"/>
            <xs:element name="ACL"
                type="ACLType"
                minOccurs="0"/>
            <xs:element name="VOView"
                type="CEVOViewType"
                minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="UniqueID"
            type="UniqueIDType"
            use="required"/>
        <xs:attribute name="Name"
            type="xs:string"
            use="optional"/>
        <xs:attribute name="InformationServiceURL"
            type="xs:anyURI"
            use="optional"/>
    </xs:complexType>

    <xs:complexType name="OperatingSystemType">
        <xs:attribute name="Name" type="xs:string"/>
        <xs:attribute name="Release" type="xs:string"/>
        <xs:attribute name="Version" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="ProcessorType">
        <xs:attribute name="Vendor" type="xs:string"/>
        <xs:attribute name="Model" type="xs:string"/>
        <xs:attribute name="ClockSpeed" type="xs:integer"/>
        <xs:attribute name="InstructionSet" type="xs:string"/>
        <xs:attribute name="OtherDescription" type="xs:string"/>
        <xs:attribute name="CacheL1"
            type="xs:integer"
            use="optional"/>
        <xs:attribute name="CacheL1I"
            type="xs:integer"
            use="optional"/>
        <xs:attribute name="CacheL1D"
            type="xs:integer"
            use="optional"/>
```

```
    <xs:attribute name="CacheL2"
        type="xs:integer"
        use="optional"/>
</xs:complexType>

<xs:complexType name="NetworkAdapterType">
    <xs:attribute name="InboundIP"
        type="xs:boolean"
        use="optional"/>
    <xs:attribute name="OutboundIP"
        type="xs:boolean"
        use="optional"/>
    <xs:attribute name="MTU"
        type="xs:integer"
        use="optional"/>
</xs:complexType>

<xs:complexType name="MainMemoryType">
    <xs:attribute name="RAMSize" type="xs:integer"/>
    <xs:attribute name="VirtualSize" type="xs:integer"/>
</xs:complexType>

<xs:complexType name="HostArchitectureType">
    <xs:attribute name="PlatformType"
       type="xs:string"
       use="optional"/>
    <xs:attribute name="SMPSize"
       type="xs:integer"
       use="optional"/>
    <xs:attribute name="SMTSize"
       type="xs:integer"
       use="optional"/>
</xs:complexType>

<xs:complexType name="BenchmarkType">
    <xs:attribute name="SI00" type="xs:integer" use="optional"/>
    <xs:attribute name="SF00" type="xs:integer" use="optional"/>
</xs:complexType>

<xs:complexType name="RunTimeEnvType">
    <xs:sequence>
        <xs:element name="Variable"
            type="xs:string"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="hostType">
```

```
    <xs:sequence>
        <xs:element name="MainMemory"
            type="memoryType"
            minOccurs="0"/>
        <xs:element name="Load"
            type="loadType"
            minOccurs="0"/>
        <xs:element name="Benchmark"
            type="benchmarkType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:choice>
          <xs:element name="FileSystem"
            type="FileSystemType"
            minOccurs="0"
            maxOccurs="unbounded"/>
          <xs:element name="StorageDevice"
            type="StorageDeviceType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="UniqueID"
        type="UniqueIDType"
        use="required"/>
    <xs:attribute name="Name"
        type="xs:string"
        use="optional"/>
    <xs:attribute name="UpTime"
        type="xs:integer"
        use="optional"/>
    <xs:attribute name="IPAddress"
        type="xs:string"
        use="optional"/>
</xs:complexType>

<xs:complexType name="subClusterType">
    <xs:sequence>
        <xs:element name="PhysicalCPUs"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="LogicalCPUs"
            type="xs:integer"
            minOccurs="0"/>
        <xs:element name="TmpDir"
            type="DirType"
            minOccurs="0"/>
        <xs:element name="WNTmpDir"
```

```
            type="DirType"
            minOccurs="0"/>
        <xs:element name="OperatingSystem"
            type="OperatingSystemType"
            minOccurs="0"/>
        <xs:element name="Processor"
            type="ProcessorType"
            minOccurs="0"/>
        <xs:element name="NetworkAdapter"
            type="NetworkAdapterType"
            minOccurs="0"/>
        <xs:element name="MainMemory"
            type="MainMemoryType"
            minOccurs="0"/>
        <xs:element name="Architecture"
            type="HostArchitectureType"
            minOccurs="0"/>
        <xs:element name="Benchmark"
            type="BenchmarkType"
            minOccurs="0"/>
        <xs:element name="RunTimeEnv"
            type="RunTimeEnvType"
            minOccurs="0"/>
        <xs:element name="Host"
            type="hostType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <!-- Network between hosts-->
        <xs:element name="Network"
            type="networkType"
            minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="UniqueID"
        type="UniqueIDType"
        use="required"/>
    <xs:attribute name="Name"
        type="xs:string"
        use="optional"/>
</xs:complexType>

<xs:complexType name="clusterType">
    <xs:sequence>
        <xs:element name="TmpDir" type="DirType" minOccurs="0"/>
        <xs:element name="WNTmpDir" type="DirType" minOccurs="0"/>
        <xs:element name="ComputingElement"
            type="computingElementType"
            maxOccurs="unbounded"
            minOccurs="0"/>
```

```
        <xs:element name="SubCluster"
            type="subClusterType"
            maxOccurs="unbounded"
            minOccurs="1"/>
        <!-- Network between subclusters -->
        <xs:element name="Network"
            type="networkType"
            minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="UniqueID"
        type="UniqueIDType"
        use="required"/>
    <xs:attribute name="Name"
        type="xs:string"
        use="optional"/>
</xs:complexType>

<xs:element name="Site">
    <xs:complexType>
        <xs:attribute name="UniqueID"
            type="xs:string"
            use="required"/>
        <xs:attribute name="Name"
            type="xs:string"
            use="optional"/>
        <xs:attribute name="Web"
            type="xs:anyURI"
            use="optional" />
        <xs:sequence>
            <xs:element name="Cluster"
                type="clusterType"
                minOccurs="0"
                maxOccurs="unbounded"/>
            <!-- Network between clusters -->
            <xs:element name="Network"
                type="networkType"
                minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Grid">
    <xs:complexType>
        <xs:attribute name="UniqueID"
            type="xs:string"
            use="required"/>
        <xs:attribute name="Name"
            type="xs:string"
```

```
            use="optional"/>
        <xs:sequence>
            <xs:element ref="Site"
                minOccurs="0"
                maxOccurs="unbounded"/>
            <!-- Network between sites -->
            <xs:element name="Network"
                type="networkType"
                minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:complexType name ="bandwidthType">
    <xs:attribute name="Protocol"
        type="xs:string"
        use="optional"/>
    <xs:attribute name="Layer"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:attribute name="Capacity"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:sequence>
        <xs:element name="Utilization" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Available" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                    type="characValueType"
                    maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Achievable" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
```

```
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="methEnumType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="One-way"/>
    <xs:enumeration value="Round-trip"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="delayType">
    <xs:attribute name="Protocol"
        type="xs:string"
        use="optional"/>
    <xs:attribute name="Layer"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:attribute name="Methodology"
        type="methEnumType"
        use="optional"/>
    <xs:sequence>
        <xs:element name="Value"
            type="characValueType"
            maxOccurs="unbounded"
            minOccurs="0"/>
        <xs:element name="Jitter" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="patternEnumType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Average"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="lossType">
    <xs:attribute name="Protocol"
```

```
        type="xs:string"
        use="optional"/>
    <xs:attribute name="Layer"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:attribute name="Methodology"
        type="methEnumType"
        use="optional"/>
    <xs:sequence>
        <xs:element name="Value"
            type="characValueType"
            maxOccurs="unbounded"
            minOccurs="0"/>
        <xs:element name="LossPattern"
            minOccurs="0"
            maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Pattern"
                   type="patternEnumType"/>
                <xs:sequence>
                   <xs:element name="Value"
                       type="characValueType"
                       maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="availabilityType">
    <xs:sequence>
        <xs:element name="Value"
            type="characValueType"
            maxOccurs="unbounded"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="reorderingType">
    <xs:attribute name="Protocol" type="xs:string" use="optional"/>
    <xs:attribute name="Layer"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:sequence>
        <xs:element name="ReorderingPattern"
            minOccurs="1"
            maxOccurs="unbounded">
            <xs:complexType>
```

```
                <xs:attribute name="Pattern"
                    type="patternEnumType"/>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="disciplineEnumType">
  <xs:restriction base="xs:string">
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="queueType">
    <xs:attribute name="Discipline"
        type="disciplineEnumType"
        use="optional"/>
    <xs:attribute name="Capacity"
        type="xs:positiveInteger"
        use="optional"/>
    <xs:sequence>
        <xs:element name="CurrentLength">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Value"
                        type="characValueType"
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="routerType">
    <xs:complexContent>
        <xs:extension base="hostType">
            <xs:sequence>
                <xs:element name="Queue"
                    type="queueType"
                    minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="hopListType">
    <xs:sequence>
        <xs:element name="Hop" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:string"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="pathType">
    <xs:attribute name="Source" type="xs:string"/>
    <xs:attribute name="Destination" type="xs:string"/>
    <xs:sequence>
        <xs:element name="Bandwidth"
            type="bandwidthType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="Delay"
            type="delayType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="Loss"
            type="lossType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="Availability"
            type="availabilityType"
            minOccurs="0"/>
        <xs:element name="Reordering"
            type="reorderingType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="HopList"
            type="hopListType"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="networkType">
    <xs:sequence>
        <xs:element name="Router"
            type="routerType"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="Path"
            type="pathType"
```

```
            minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

</xs:schema>
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo