UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

VXDL: UMA LINGUAGEM PARA DESCRIÇÃO DE INTERCONEXÕES E RECURSOS EM GRADES VIRTUAIS

DISSERTAÇÃO DE MESTRADO

Guilherme Piêgas Koslovski

Santa Maria, RS, Brasil

2008

VXDL: UMA LINGUAGEM PARA DESCRIÇÃO DE INTERCONEXÕES E RECURSOS EM GRADES VIRTUAIS

por

Guilherme Piêgas Koslovski

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Mestre em Informática

Orientador: Prof. Dr. Benhur de Oliveira Stein (UFSM)

Co-orientador: Profª Drª Andrea Schwertner Charão (UFSM)

Dissertação de Mestrado Nº 1
Santa Maria, RS, Brasil

2008

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

VXDL: UMA LINGUAGEM PARA DESCRIÇÃO DE INTERCONEXÕES E RECURSOS EM GRADES VIRTUAIS

elaborada por
Guilherme Piêgas Koslovski

como requisito parcial para obtenção do grau de
Mestre em Informática

COMISSÃO EXAMINADORA:

Profª Drª Andrea Schwertner Charão (UFSM)
(Presidente/Co-orientador)

Prof. Dr. Gerson Geraldo Homrich Cavalheiro (UFPel)

Profª Drª Iara Augustin (UFSM)

Santa Maria, 22 de Agosto de 2008.

“Não sabendo que era impossível, ele foi lá e fez.” — JEAN COCTEAU

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

VXDL: UMA LINGUAGEM PARA DESCRIÇÃO DE INTERCONEXÕES E RECURSOS EM GRADES VIRTUAIS

Autor: Guilherme Piêgas Koslovski
Orientador: Prof. Dr. Benhur de Oliveira Stein (UFSM)
Co-orientador: Prof[a] Dr[a] Andrea Schwertner Charão (UFSM)
Local e data da defesa: Santa Maria, 22 de Agosto de 2008.

Grades de computadores vêm se afirmando como uma infra-estrutura integradora de recursos distribuídos. Embora já seja utilizado em grande escala, este tipo de infra-estrutura computacional ainda constitui um campo de pesquisa ativo, com muitas questões em aberto. Pesquisas atuais investigam as chamadas grades virtuais, que tiram proveito de tecnologias de virtualização de recursos em sua composição. Essas grades podem ser definidas como uma abstração de alto nível dos recursos (computacionais e de comunicação), através da qual usuários têm uma visão de um amplo conjunto de computadores interligados, que podem ser selecionados e organizados virtualmente. Em grades virtuais, assim como em grades reais, usuários e *middlewares* devem dispor de ferramentas que permitam a composição e o gerenciamento das infra-estruturas. Dentre essas ferramentas, encontram-se as linguagens para descrição de recursos, que permitem a definição dos componentes que deverão ser utilizados na infra-estrutura. Em ambientes virtualizados, as linguagens descritivas devem oferecer atributos que interajam com peculiaridades, como a possibilidade de alocação de múltiplos recursos virtuais (computacionais e de rede) sobre um mesmo recurso físico. Neste contexto, este trabalho apresenta VXDL, uma linguagem projetada para a descrição de interconexões e recursos computacionais em grades virtuais. As inovações propostas em VXDL permitem a descrição, classificação e parametrização de todos os componentes desejáveis, bem como a definição de uma topologia de rede, inclusive informando a localização de roteadores virtuais. Permite-se também a definição de um cronograma de execução, que pode auxiliar *middlewares* nas tarefas de compartilhamento e escalonamento dos recursos. Para avaliar a linguagem proposta, apresenta-se I) um estudo comparativo entre VXDL e outras linguagens de descrição existentes, e II) uma análise de resultados obtidos com a execução de *benchmarks* sobre infra-estruturas compostas com diferentes descrições VXDL.

Palavras-chave: Virtualização, grids virtuais, clusters virtuais, linguagens para especificação de recursos.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

VXDL: A LANGUAGE FOR INTERCONNECTION AND RESOURCES SPECIFICATION IN VIRTUAL GRIDS

Author: Guilherme Piêgas Koslovski
Advisor: Prof. Dr. Benhur de Oliveira Stein (UFSM)
Coadvisor: Profª Drª Andrea Schwertner Charão (UFSM)

Grid computing has been defined as an infrastructure integrator of distributed resources. Although it is already used on a large scale in many areas, this type of computational infrastructure is still an area of active research, with many open questions. Today, new research works investigate the application of resources virtualization techniques to perform the composition of virtual grids. These grids can be defined as a high level abstraction of resources (computing and network), through which users have a view of a wide range of interconnected computers, that can be selected and virtually organized. In a virtual grid, as well in a real grid, users and middleware must have tools that allow the composition and management of the infrastructure. Among these tools, there are languages for resource description that allow the specification of components that will be used in the infrastructure. In a virtualized environment, the resources descriptions languages should offer attributes that interact with some peculiarities, such as the possibility of allocate multiple virtual resources (computing and network) on the same physical resource. In this context, this work presents VXDL, a language developed for the interconnections and resources description in virtual grids. The innovations proposed in VXDL allow the description, classification and parameter specification of all desirable components, including network topology and virtual routers. VXDL also allow the specification of a execution timeline, which can assist grid middleware in the tasks of resources sharing and scheduling. To evaluate the proposed language, this work presentes I) a comparative study between VXDL and other resources description languages and II) an analysis of results obtained with the benchmarks execution in virtual infrastructures composed using different VXDL descriptions.

Keywords: virtualization, virtual grids, virtual clusters, resources description language.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
GPL	General Public License
IP	Internet Protocol
MMV	Monitor de Máquinas Virtuais
NFS	Network File System
QoS	Quality of Service
SO	Sistema Operacional
SSH	Secure Shell
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e Motivação

Grades computacionais oferecem uma infra-estrutura de *software* e *hardware* composta por diversos computadores e *clusters* interligados, geralmente distribuídos geograficamente [37]. Nestes ambientes, os usuários têm a visão de um computador escalável, com memória distribuída, que oferece um amplo número de recursos para computação, comunicação e armazenamento. O conjunto de recursos disponíveis em uma grade pode ser compartilhado entre múltiplos usuários, com diferentes finalidades, como por exemplo, computação intensiva, armazenamento intensivo, computação sob demanda e processamento distribuído. Para permitir um compartilhamento eficiente em grades, alguns projetos atuais têm explorado o uso de virtualização de recursos computacionais [4, 3, 41, 16, 61, 57] e de rede [7, 64, 5] para criar e gerenciar ambientes isolados, compostos dinamicamente.

Aplicações que utilizam e compartilham os recursos de grades (reais ou virtuais) possuem diferentes requisitos de processamento, armazenamento e comunicação. Desta forma, usuários e *middlewares* devem dispor de mecanismos para especificar e reservar os componentes que serão utilizados. Na infra-estrutura de *software* para grades [37], o detalhamento da configuração desejável pode ser realizada utilizando linguagens de descrição dos recursos. Já a localização e reserva pode ser realizada por ferramentas capazes de interpretar e mapear a solicitação de acordo com os recursos existentes [55, 58]. O estudo e desenvolvimento de linguagens para grades reais já foi abordado em diversos trabalhos [54, 45, 19, 10, 50, 42, 38] que diferenciam-se basicamente pelo modelo de implementação, gramática proposta e disponibilidade ou não de alguns recursos.

As linguagens existentes para descrição de infra-estruturas virtuais apresentam algumas lacunas no detalhamento dos recursos, como a dificuldade de expressar parâmetros

para grupos de componentes (como *clusters*), que devem ser alocados e utilizados simultaneamente. Além disso, no contexto de infra-estruturas virtuais, os parâmetros devem contemplar a possibilidade de compartilhamento de um mesmo recurso entre diferentes *clusters* virtuais. Um mesmo computador pode alocar múltiplas máquinas virtuais, pertencentes a diferentes *clusters*. Desta forma, usuários devem ser capazes de especificar parâmetros que definam informações como o número máximo de máquinas virtuais alocadas em um componente.

Considerando a aplicação de virtualização nos canais de comunicação, deseja-se também detalhar a topologia de rede ideal para a composição do ambiente virtual. Analisando-se as soluções existentes, observa-se em algumas linguagens de especificações de recursos a possibilidade de uma parametrização básica, sem detalhar a existência de roteadores. Por outro lado, a utilização de uma linguagem específica para detalhar a topologia de rede [15, 63, 17], associada a outra linguagem para detalhar os recursos, dificulta a especificação de infra-estruturas complexas.

Esta problemática está presente nos projetos CARRIOCAS [3] e HIPCAL [4], aos quais o presente trabalho relaciona-se. Analisando-se a descrição dos *middlewares* em desenvolvimento nestes projetos, juntamente com a descrição de suas aplicações testes [14, 12], identificam-se requisitos básicos desejáveis para a especificação de infra-estruturas virtuais. Espera-se assim que uma linguagem de especificação permita:

- representar os recursos individualmente e em grupos;
- classificar os recursos em *funções elementares*, como solicitar um grupo de recursos para *computação* e *armazenamento*;
- descrever a composição da topologia de rede desejável, especificando os parâmetros para todos os componentes;
- especificar um cronograma de execução, especificando em qual momento os recursos serão necessários;
- especificação das configurações de segurança que serão utilizadas na infra-estrutura virtual.

1.2 Objetivos e Contribuição

A solução proposta neste trabalho tem por objetivo auxiliar usuários e *middlewares* na descrição dos recursos que devem compor uma infra-estrutura virtual, adaptando-se às requisições básicas identificadas. Aborda-se neste trabalho, o processo de descrição de recursos para grades virtuais, detalhando a composição de infra-estruturas para execução de aplicações, sem realizar a reserva e seleção dos recursos. Para este fim, desenvolveu-se a linguagem VXDL (*Virtual Resources and Interconnection Networks Description Language*), que apresenta como principais inovações a possibilidade de representação da topologia de rede desejável, cronograma de execução e a especificação da localização base, que auxiliam na composição e definição da forma de infra-estruturas virtuais. Basicamente, a gramática desta linguagem divide a descrição da infra-estrutura virtual em três partes principais:

Especificação dos recursos: VXDL permite a especificação dos componentes de uma infra-estrutura, detalhando suas funcionalidades e configurações desejáveis. É possível classificar recursos individualmente ou em grupos, solicitando por exemplo *um cluster contendo 20 nós bi-processados, que será utilizado para computação, no qual precisa-se: memória RAM mínima de 1GB; sistema operacional Debian; biblioteca de comunicação mpich; deve ser alocado no cluster físico PAPLE; e será utilizado por quatro horas*. Esta solicitação pode ser escrita em VXDL utilizando atributos como *function, parameters, software* e *anchor*, que permitem a classificação, parametrização, definição de *software* e localização base, respectivamente. Com a especificação de uma localização base, utilizando *anchor*, é possível solicitar a disponibilidade de um componente em um determinado *cluster* físico, como por exemplo a utilização de recursos para visualização alocados em um local específico, e realizar a composição da infra-estrutura virtual tendo como ponto base a localização informada.

Especificação da topologia de rede: Na especificação da topologia de rede, VXDL permite a descrição de todos os canais de comunicação que interligam os recursos detalhados anteriormente. Desta forma, pode-se ajustar a infra-estrutura de acordo com as necessidades das aplicações, como por exemplo aplicações sensíveis à latência ou comunicações que requerem uma largura de banda mínima. Nesse ponto

da especificação, pode-se ainda direcionar os canais de comunicação dos recursos classificados como roteadores virtuais, permitindo a definição da formato da infra-estrutura. Basicamente, os principais atributos nesta parte da especificação são *bandwidth, latency* e *direction*, que permitem a parametrização dos canais; e a definição das conexões identificando pares "(origem, destino)"utilizando o atributo *between*.

Especificação de um cronograma de execução: Esta parte da descrição visa auxiliar na especificação de infra-estruturas onde os recursos selecionados não são utilizados simultaneamente, e sim respeitando um cronograma de execução. A figura 1.1 mostra um possível cronograma de uma aplicação, apresentando a divisão em estágios distintos de utilização de recursos. Em um primeiro momento (*Etapa I*) utiliza-se uma largura de banda que permita a transmissão eficiente de um volume de dados para os *clusters* computacionais. Em um segundo momento (*Etapa II*), solicita-se por uma latência eficiente que permita a comunicação entre os *clusters*. No último estágio (*Etapa III*) solicita-se recursos para a visualização dos resultados e uma conexão de rede eficiente para a transmissão dos dados. Utilizando VXDL, usuários e *middlewares* podem representar esse cronograma em um *timeline* virtual, que define os marcos para inicialização de término de cada estágio, de acordo com o volume de dados a ser transferido ou o tempo total necessário para a computação.

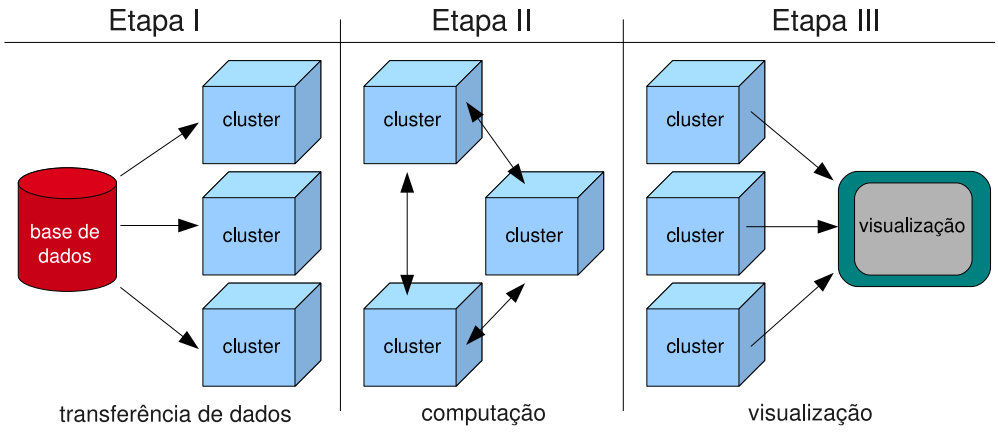


Figura 1.1: Cronograma contendo as etapas para uso de uma infra-estrutura virtual.

Na figura 1.2 exemplifica-se uma possível infra-estrutura virtual, apresentando-se um grafo que contém detalhes de uma especificação realizada usando VXDL. Observa-se a

existência de parâmetros para todos os componentes da infra-estrutura: recursos e conexões de rede. Os vértices B e H representam roteadores virtuais, cada um com quatro portas. A, C, I, J e F representam *clusters* de computadores, que possuem diferentes números de nós e diferentes classificações: computação, armazenamento e visualização. As arestas definem alguns canais de comunicação com largura de banda, latências e direções distintas.

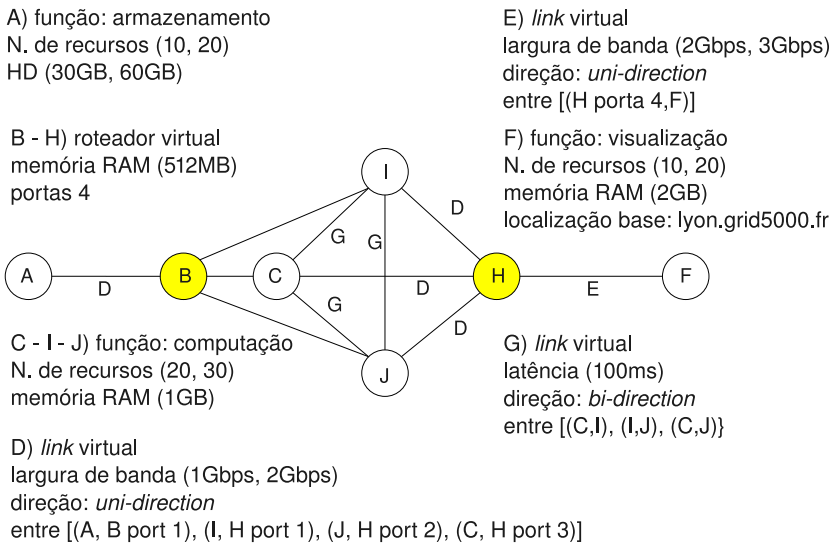


Figura 1.2: Grafo descrevendo uma infra-estrutura virtual.

1.3 Organização do Texto

Este trabalho está organizado da seguinte forma: inicialmente no capítulo 2 apresenta-se uma revisão de literatura sobre assuntos relacionados com o desenvolvimento da solução proposta. Busca-se destacar os principais desafios na organização de *middlewares* de gerenciamento em grades reais, ressaltando também a aplicação de virtualização de recursos computacionais para o desenvolvimento de grades virtuais. Observa-se que alguns tópicos de organização e gerenciamento em grades virtuais requerem uma atenção especial, como a descrição, seleção e reserva de recursos, para composição de infra-estruturas virtuais.

No capítulo 3 apresenta-se VXDL, uma linguagem para descrição de interconexões e recursos em grades virtuais. Destacam-se os pré-requisitos básicos definidos antes do desenvolvimento da gramática de VXDL, de acordo com as especificações do *middleware* HIPerNET. Apresentam-se também alguns cenários de utilização e exemplos de descrições de infra-estruturas virtuais e reais. Já no capítulo 4 é descrito o processo de avalia-

ção da linguagem desenvolvida, realizado por meio de uma comparação qualitativa com outras linguagens, e através da execução de *benchmarks* sobre infra-estruturas virtuais compostas com diferentes descrições VXDL. Por fim, no capítulo 5 apresentam-se as considerações finais referentes ao trabalho, e ressaltam-se algumas idéias para continuidade.

2 REVISÃO DE LITERATURA

Este capítulo apresenta uma revisão de literatura sobre conceitos relacionados com o desenvolvimento do trabalho, abordando em um primeiro momento as definições de grades computacionais e grades virtuais, ressaltando os desafios e as ferramentas necessárias para sua utilização. Neste contexto, discute-se a necessidade do detalhamento dos recursos computacionais utilizando linguagens de descrição, apresentando-se as principais particularidades necessárias para especificações de infra-estruturas virtuais.

2.1 Grades Computacionais

2.1.1 Definições e Principais Aplicações

A definição de *grades computacionais* foi originalmente utilizada na década de 90 [37], auxiliando na descrição e implementação de projetos como *SETI@home* [59] e *distributed.net* [24], que basicamente exploravam os recursos ociosos de computadores distribuídos geograficamente. Atualmente, define-se grades computacionais como uma infra-estrutura de *hardware* e *software* distribuída, constituída por computadores e *clusters*[1] de computadores interligados, geralmente distribuídos geograficamente [37, 22]. Nestes ambientes, usuários possuem a visão de um computador escalável com memória distribuída, que possui um elevado número de recursos para armazenamento, computação e comunicação. Considera-se ainda que nesses ambientes o acesso às informações e aos recursos pode ser realizada de forma segura e consistente.

Nos últimos anos, a utilização de computação em grades evoluiu, difundindo esse tipo de plataforma em diversas áreas como pesquisas científicas, ambientes governamentais, área da saúde, entre outros. Exemplifica-se essa evolução pela utilização de grades para

[1] O termo *cluster* é traduzido para o português como "aglomerado" ou "agregado". Neste documento, optou-se por utilizar geralmente o termo em inglês, devido à sua popularidade na área.

diferentes finalidades: as instituições governamentais podem explorar o poder computacional dessas infra-estruturas para processar informações administrativas, referentes ao controle de recursos e pessoas; na área da saúde, clínicas, laboratórios de pesquisa e hospitais podem compartilhar recursos e informações, explorando computadores ociosos em sua estrutura; áreas científicas utilizam os recursos de computação para projetar, testar e avaliar modelos e simulações. Um exemplo diário de computação em grade pode ser visto nas previsões climáticas.

Mais especificamente, pode-se classificar as aplicações de grades de acordo com suas características estruturais e número de recursos utilizados, organizando-as em cinco classes principais [37]:

- *Distributed Supercomputing*

  Esta classe compreende aplicações que possuem uma elevada complexidade computacional, e procuram em grades recursos em larga escala (processamento, armazenamento, memória principal, entre outros) para obter um tempo de execução eficiente.

- *High-throughput Computing*

  Engloba aplicações que possuem um elevado número de tarefas independentes e que normalmente realizam pouca comunicação entre os processos. Algumas vezes, sua alocação entre os componentes explora o aproveitamento de recursos ociosos.

- *On-demand Computing*

  Aplicações que requerem ambientes computacionais com elevado número de recursos, normalmente resultando em um alto custo financeiro e organizacional. Utilizam grades para obter recursos não disponíveis localmente, acessando-os remotamente por períodos determinados.

- *Data Intensive*

  Classe de aplicações que necessitam de um alto número de recursos para armazenamento de dados. Muitas vezes as informações devem ser transportadas entre diferentes componentes (resultado de processamentos que devem ser armazenados), necessitando uma largura de banda eficiente.

- *Collaborative* Estas aplicações permitem o compartilhamento de dados e informações entre múltiplos participantes, em ambientes onde as informações podem ser

geradas e distribuídas simultaneamente por diversos usuários.

Cada classe de aplicações descrita requer diferentes recursos computacionais em diferentes escalas. Na proporção que aumenta a escalabilidade dos recursos, aumenta a complexidade arquitetural na organização, seleção, compartilhamento e segurança [37]. Basicamente, pode-se classificar as estruturas de acordo com sua complexidade (em ordem crescente):

- *End system*

  Compreende aplicações com processamento local, usualmente *multi-threads*, que basicamente realizam acesso a recursos locais. Nessas estruturas a segurança das informações e o controle no acesso aos recursos é realizado pelo sistema operacional.

- *Cluster*

  Execução em computadores homogêneos, normalmente interconectados por uma rede de alta velocidade, onde utiliza-se o modelo de programação com memória distribuída. A privacidade das informações e as permissões de acesso são controladas por uma base de dados contendo as políticas de segurança utilizadas.

- *Intranet*

  Estas estruturas são compostas por computadores heterogêneos interligados, onde utiliza-se o modelo de programação Cliente-Servidor. É comum a utilização de sistemas para compartilhamento de arquivos e diretórios, e a segurança no acesso às informações é realizada no nível da rede.

- *Internet*

  Aplicações realizam acesso remoto às informações e recursos distribuídos geograficamente, sem a existência de um controle centralizado. Utilizam-se chaves públicas e políticas de acesso para prover uma comunicação segura.

Algumas aplicações podem utilizar combinações entre diferentes classes, como por exemplo processamento local (*end systems*) de informações armazenadas em *clusters* de armazenamento. Esta possibilidade aumenta a complexidade organizacional de grades dificultando o controle dos recursos e segurança das informações.

2.1.2 Desafios e Pesquisas Atuais

De acordo com a definição de grades computacionais, usuários têm a visão de um amplo conjunto de recursos com diferentes capacidades, distribuídos geograficamente e interconectados por redes com diferentes características e especificações. Considerando este cenário, pode-se ressaltar algumas dificuldades e desafios na composição, utilização e armazenamento, que têm sido abordados em diversas pesquisas atuais: *desempenho, segurança, escalabilidade e confiabilidade* [22]. O desempenho de uma infra-estrutura reflete diretamente nas aplicações em execução que utilizam seus recursos [67]. Considerando uma grade amplamente compartilhada entre múltiplos usuários, o compartilhamento e escalonamento eficiente dos recursos influencia diretamente no desempenho computacional da infra-estrutura.

A segurança e confiabilidade em grades são relacionadas com a utilização, permissões de acesso e confiabilidade na manutenção das informações administrativas e de usuários. Considera-se que *middlewares* de grades devem possuir mecanismos para controlar o acesso aos recursos e informações compartilhadas [29]. Diversas soluções implementam segurança e confiabilidade em grades computacionais utilizando abordagens centralizadas ou descentralizadas, estudando seus benefícios e aplicações em cada caso [60, 34]. Já a escalabilidade de uma grade refere-se à capacidade de alterar o conjunto de recursos disponíveis dinamicamente, garantindo a continuidade dos serviços em execução [33].

Estas características descritas são difíceis de serem implementadas e mantidas em grades computacionais, devido ao elevado número de recursos, distribuídos em diferentes centros administrativos e localidades, que possuem regras de acesso e utilização próprias. Considerando as conexões de rede que interligam esses recursos, observa-se que são compostas por canais de comunicações com diferentes configurações e capacidades. Neste contexto, diversos trabalhos investigam a organização e gerenciamento de grades computacionais, estudando tópicos como *monitoramento, escalonamento e descoberta de recursos* [23, 27, 43, 37].

O monitoramento de uma grade computacional é necessário para acompanhar a utilização, disponibilidade e correto funcionamento dos recursos, e para visualizar o andamento das aplicações em execução. Mais especificamente, administradores e usuários devem conhecer a situação atual, informações passadas e agendamentos futuros, relativos aos recursos computacionais e de rede, detalhando informações como percentual de uti-

lização e quais recursos estão sendo utilizados por determinado usuário, entre outros. A utilização de bases de informações aliadas com sistemas de monitoração permite que as informações sobre ocorrências passadas sejam analisadas e utilizadas para auxílio na tomada de decisões futuras, sobre manutenção de *hardware*, escalonamento, reconfiguração da plataforma, entre outras.

Já o escalonamento, a descoberta e a reserva dos recursos de uma grade computacional, compartilhada por múltiplos usuários e aplicações com diferentes pré-requisitos e infra-estruturas de utilização (seção 2.1.1), é uma tarefa complexa, atualmente abordada por trabalhos que estudam a implementação de soluções distribuídas [55, 36, 58]. As diversas aplicações com diferentes composições estruturais devem utilizar a grade simultaneamente, compartilhando os recursos de uma forma eficiente, isoladamente, onde os pré-requisitos de uma aplicação não interferem nas demais.

Uma das tecnologias utilizadas para auxiliar a composição de infra-estruturas para execução de aplicações em grades é a utilização de linguagens para descrição dos recursos necessários para execução [54, 45, 10, 50]. Pela interpretação dessas descrições, *middlewares* têm conhecimento da composição necessária, e podem realizar a reserva e o escalonamento dos recursos, baseado em informações definidas pelos utilizadores.

2.2 Grades Virtuais

Foster et al. [28] definiram a necessidade de representação e organização dos recursos de uma grade computacional em *organizações virtuais*, para permitir a utilização simultânea de múltiplas aplicações, com diferentes pré-requisitos de computação, armazenamento e comunicação. Atualmente, diversos projetos exploram a aplicação de tecnologias de virtualização de recursos [30] em grades computacionais, para permitir que usuários manipulem e organizem os recursos virtualmente, de acordo com as especificações de suas aplicações.

Projetos atuais, como HIPCAL [4], CARRIOCAS [3], VGrADS [41], Reservoir [52], VioCluster [57] e Virtual WorkSpace [61], definem grades virtuais como uma abstração de alto nível dos recursos computacionais e de comunicação existentes, com a qual usuários têm uma visão de um amplo conjunto de computadores interligados, que podem ser selecionados e organizados virtualmente. Este nível de separação, obtido com o uso da virtualização, permite uma independência entre as especificações das aplicações e os re-

cursos físicos. Desta forma, pode-se criar múltiplos *clusters* virtuais, alocados sobre o mesmo conjunto de *hardware*, permitindo o compartilhamento dos recursos em ambientes isolados e seguros.

Com a utilização de grades virtuais, objetiva-se ainda a organização virtual da comunicação entre os componentes utilizados. Normalmente, os recursos físicos estão distribuídos geograficamente, interligados por redes com diferentes capacidades e utilizações. Utiliza-se virtualização para realizar uma alocação eficiente, permitindo uma organização dinâmica de topologias, provendo garantias mínimas na qualidade do serviço e segurança na comunicação. Este tópico é abordado em projetos atuais como G-Lambda [64], Phosphorus [5] e UCLP [7].

2.2.1 Virtualização de Recursos Computacionais

As tecnologias de virtualização de recursos, atualmente aplicadas em grades computacionais, foram inicialmente abordadas na década de 60, quando serviam para apoiar o compartilhamento do poder computacional dos *mainframes* IBM [30]. Esta tecnologia tem sido explorada em diversos soluções acadêmicas e comerciais [66, 18, 25, 20, 53, 46, 35, 65], que exploram a aplicação em computadores modernos para diferentes finalidades, alocando múltiplos sistemas operacionais sobre uma mesma estrutura de *hardware* física.

Basicamente, a virtualização de recursos computacionais consiste na abstração da arquitetura real, por uma representação utilizando máquinas virtuais. Cada máquina virtual oferece uma representação próxima da arquitetura física existente, permitindo que sistemas executem isoladamente e concorrentemente sobre um mesmo *hardware*. Atualmente, existem três modelos básicos de implementação para virtualização de recursos: emulação, para-virtualização e virtualização total.

A emulação corresponde à técnica utilizada para executar *software* desenvolvido para uma arquitetura específica, sobre outras plataformas. Esta tecnologia é utilizada em algumas soluções computacionais como plataformas de testes, emulação de instruções de processadores não compatíveis (execução de 32 bits sobre arquiteturas de 64 bits, por exemplo), e para permitir a mobilidade de aplicações desenvolvidas em plataformas específicas (execução de aplicativos Windows sobre ambientes Linux, por exemplo).

Na virtualização total e na para-virtualização, utiliza-se a alocação de monitores de máquinas virtuais, diretamente sobre o *hardware* existente. Na primeira abordagem, mo-

nitores de máquinas virtuais possuem o mais alto privilégio de execução, controlando o acesso ao *hardware* realizado pelos sistemas operacionais virtualizados. Já a segunda abordagem é utilizada para a mesma finalidade nas famílias de processadores *IA-32* que não oferecem suporte nativo à virtualização. Este modelo requer a realização de alterações nos sistemas operacionais que serão virtualizados, já que a interface representada pelas máquinas virtuais difere em alguns pontos dos recursos reais disponíveis.

2.2.1.1 Monitores de Máquinas Virtuais

Dentre as soluções utilizadas pelos projetos atuais, destaca-se a utilização de monitores de máquinas virtuais (MMV) Xen [13] e VMWare [62]. Estes monitores oferecem uma camada de abstração localizada diretamente sobre o *hardware* virtualizado, permitindo a alocação de sistemas operacionais em máquinas virtuais, que representam o *hardware* real. Monitores possuem o mais alto nível na hierarquia dos privilégios de execução, controlando o acesso aos recursos.

Particularmente, o projeto HIPCAL optou pela utilização do monitor de máquinas virtuais Xen por permitir a virtualização completa nas famílias de processadores AMD64 e EM64T, e para-virtualização nas famílias IA-32. Além disso, Xen permite a manipulação das máquinas virtuais hospedadas utilizando um domínio administrativo, que oferece chamadas de sistema para administração local aliada a uma API [48] para manipulação remota. A figura 2.1 representa o monitor de máquinas virtuais Xen alocado diretamente sobre um *hardware* real. Sobre este monitor estão alocados sistemas operacionais virtualizados, em execução simultânea sobre a arquitetura.

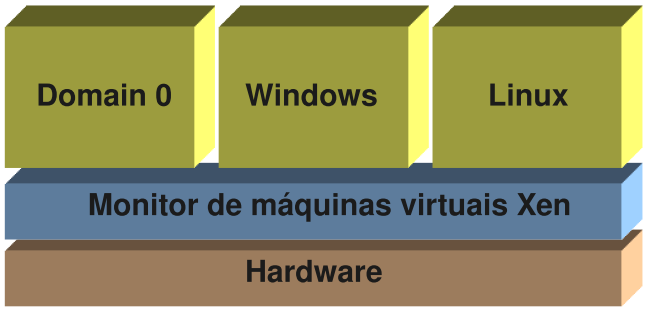


Figura 2.1: Arquitetura do monitor de máquinas virtuais Xen.

Observa-se na figura 2.1 que um sistema operacional virtualizado está isolado das demais máquinas virtuais e do *hardware* real. Esta definição permite que máquinas virtuais sejam migradas entre computadores hospedeiros (computadores executando Xen).

Desta forma, administradores de ambientes virtualizados com Xen podem migrar máquinas virtuais sem interromper sua execução [21], reorganizando a estrutura virtual dinamicamente. Este recurso auxilia na elaboração e manutenção de grades virtuais, onde pode ocorrer uma mobilidade de recursos durante a utilização de uma infra-estrutura.

2.2.1.2 Virtualização da Rede

A virtualização da rede em grades computacionais é relacionada com o compartilhamento dos canais de comunicação reais entre as diversas organizações virtuais elaboradas. Cada infra-estrutura virtual possui diferentes especificações de largura de banda, latência e topologias, que devem ser alocadas eficientemente sobre uma topologia física. Considera-se ainda nestas infra-estruturas a necessidade de controle na segurança das informações e mobilidade física dos recursos virtuais. Investigam-se atualmente [38, 44] a aplicação das soluções de VPN (*Virtual Private Network*), aplicadas no nível 3 do modelo OSI, criando e manipulando canais de comunicação diretos entre os componentes. Alguns projetos como G-Lambda [64], Phosphorus [5] e UCLP [7] abordam ainda a criação dinâmica de canais customizados para redes de alto desempenho.

A figura 2.2 representa um canal de comunicação virtual, interligando os pontos A e B, alocado sobre canais de comunicação reais. Desta forma, os pontos A e B realizam uma comunicação direta, desconhecendo o real caminho percorrido pelas mensagens.

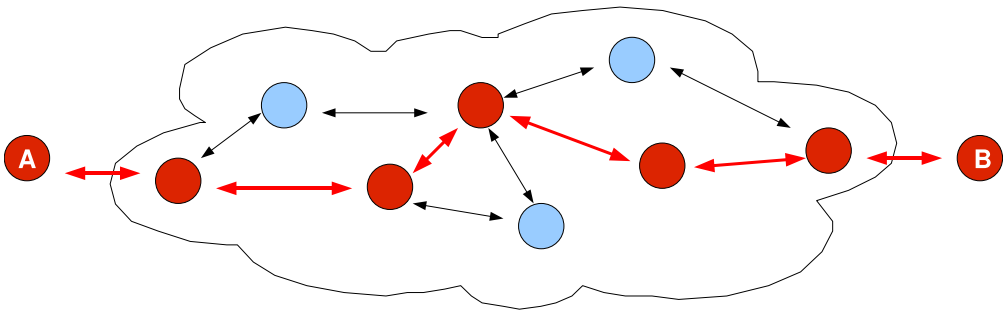


Figura 2.2: Representação de um canal de comunicação virtual.

2.2.2 Descrição de Infra-estruturas Virtuais

Assim como em grades reais, em grades virtuais alguns tópicos de gerenciamento e organização dos recursos devem ser abordados para uma utilização satisfatória. *Middlewares* devem oferecer mecanismos e ferramentas que permitam aos usuários detalhar e selecionar quais são os recursos necessários para a composição das infra-estruturas vir-

tuais. Para detalhar os recursos, utilizam-se linguagens específicas para descrição que permitem a parametrização dos componentes do ambiente.

A descrição de infra-estruturas virtuais pode ser considerada mais complexa que a descrição de uma estrutura física real. Ambientes virtuais possuem algumas peculiaridades que devem ser observadas durante a implementação de ferramentas para descrição e seleção de recursos. Um mesmo componente físico pode alocar múltiplas máquinas virtuais, pertencentes a infra-estruturas virtuais distintas, com parâmetros de configuração e especificações de capacidades específicos. Aplica-se esta mesma definição para canais de comunicação virtuais, que podem ser compartilhados entre múltiplos *clusters* virtuais com objetivos e capacidades distintas.

Segundo as definições dos projetos atuais, observa-se alguns pontos específicos, desejáveis para a especificação e detalhamento de uma infra-estrutura virtual. Considerando linguagens descritivas, deseja-se que usuários e *middlewares* possam descrever suas configurações especificando parâmetros relacionados com:

- detalhamento de todos os componentes, individualmente ou em grupos;
- definição de parâmetros computacionais necessários em cada recurso (ou grupo de recursos), descrevendo inclusive o conjunto de *software* desejável (sistemas operacionais, bibliotecas de comunicação, entre outros);
- classificação dos recursos de acordo com sua funcionalidade básica, como por exemplo a definição de *um* cluster *para processamento*, ou então de *um roteador virtual*;
- a definição de uma topologia de rede completa, compreendendo conexões entre todos os recursos descritos;
- descrição de um cronograma de execução, que possa detalhar o intervalo de utilização de cada recurso dentro do tempo reservado;
- especificação das configurações de segurança que serão utilizadas na infra-estrutura virtual.

2.3 Linguagens para Descrição de Recursos

A descrição de recursos computacionais e topologias de rede para composição de infra-estruturas já foi abordada em outros trabalhos e pesquisas [54, 50, 10, 45], resultando em uma lista de linguagens que diferenciam-se, basicamente, pelo modelo de implementação utilizado, gramática proposta e, principalmente, pela disponibilidade ou não de parâmetros para descrição. Esta seção apresenta uma revisão das principais linguagens para descrição de recursos e topologias de rede, relacionadas com grades computacionais.

2.3.1 Recursos Computacionais

Na lista de linguagens revisadas, selecionou-se, principalmente, as utilizadas para descrever recursos em grades computacionais reais e para descrever recursos genericamente, com um propósito representativo. As linguagens são apresentadas, abaixo, pela ordem em que foram desenvolvidas e publicadas. Desta forma, é possível observar uma necessidade constante de aperfeiçoamento no processo descritivo. Inicialmente, usuários apenas informavam o número de computadores necessários e submetiam sua tarefa para execução. À medida que a utilização de grades computacionais se estabeleceu para diversas finalidades, tornou-se necessário detalhar a composição desejável para execução; por exemplo, a realização de agrupamentos com as mesmas propriedades. Observa-se, também, que o passo seguinte na evolução refere-se à especificação de parâmetros de rede, detalhando quais as configurações necessárias.

- *ClassAd*

  ClassAd [54] oferece uma linguagem semi-estruturada, basicamente realizando um mapeamento entre nomes de atributos e expressões de valores. Atributos podem ser valores padrões (como inteiros, reais, etc) ou até mesmo solicitações (*queries*) com parâmetros de reserva de recursos. Seu desenvolvimento visava sua utilização junto ao *middleware* do projeto Condor [1] e permitia também a representação de tarefas para submissão, oferecendo ainda ao utilizador a possibilidade de criação de *ranks* para definição de pesquisas.

- *Redline*

  Redline [45] visava um aprimoramento da descrição oferecida por ClassAd, apresentando como maior diferencial a possibilidade de expressões utilizando conjuntos

de recursos e a utilização de uma mesma linguagem para descrever os recursos existentes e realizar as solicitações. Em sua gramática estruturada, possibilita a utilização de funções pré-definidas como *count(), max(), sum()*, e predicados como *forall x in <set>*, que permitem aplicar uma mesma regra para um conjunto de dados.

- *SWORD*

  SWORD [50] é um sistema para descoberta de recursos em grades que suporta a descrição em formato XML (SWORD XML) ou utilizando a linguagem ClassAd. O formato SWORD XML proposto, além de permitir a representação de conjuntos de nós e suas principais características, realiza pesquisas sobre recursos distintos, aglomerados e sobre a união de aglomerados distintos, aceitando parâmetros de pesquisa individuais ou sobre características do grupo. SWORD Aborda a descrição de alguns parâmetros de rede, como latência e largura de banda.

- *GLUE*

  GLUE [10] é uma linguagem de descrição de recursos computacionais para grades, modelada utilizando o paradigma de orientação a objetos. O conceito básico abordado em GLUE é a utilização de *Computing Elements (CE)*, uma representação que engloba as características, conjuntos de recursos e políticas de um grupo de tarefas. Permite ainda a representação de conjuntos, tais como *clusters* (recursos heterogêneos) e *subclusters* (recursos homogêneos). Além dos recursos físicos, GLUE permite a especificação dos requisitos de *software* na descrição de um ambiente.

- *BPDL*

  BPDL (*Broker Property Description Language*) [42] possui uma estrutura com formato XML que permite descrever uma grande quantidade de características de recursos, tarefas, *middlewares*, certificados e políticas de monitoração. Desenvolvida com o objetivo de representar todo o ambiente, possui ainda atributos que permitem descrever parâmetros de QoS, métricas de acompanhamento, tolerância a falhas e políticas de escalonamento.

- *vgDL*

  vgDL (*Virtual Grid Description Language*) [19] é uma linguagem para descrição de grades virtuais, desenvolvida para utilização no *middleware* do projeto VGrADS [41], que permite a descrição de recursos e a definição de tempo de utilização. Apresenta

em sua descrição a definição de *Associators* que permitem especificar características genéricas de um conjunto de recursos, como por exemplo *LooseBagOf, TightBagOf e ClusterOf*. Estes *associators* referem-se à especificação dos tipos de recursos (homogêneos ou heterogêneos) e da qualidade de seus recursos de interconexão. Já para a descrição de recursos de rede, vgDL possui operadores que indicam noções de proximidade entre os recursos, como *far, close, highBW e lowBW*.

Analisando a utilização das linguagens citadas para descrição de recursos em infra-estruturas virtuais, observa-se uma dificuldade na representação de grupos de recursos, com diferentes parametrizações, que devem ser utilizados simultaneamente, interagindo entre si durante a execução. Mais especificamente, não é possível representar uma infra-estrutura composta por diversos *clusters*, roteadores e outros componentes individuais. Algumas linguagens, como vgDL e SWORD oferecem recursos para a descrição parcial de topologias de rede, mas não permitem a definição de roteadores com canais de comunicação diferenciados.

2.3.2 Recursos de Rede

As linguagens de descrição dos recursos de rede revisadas foram selecionadas em diferentes áreas de utilização, sendo desenvolvidas com objetivos específicos: descrição de VPN, descrição genérica da rede e descrição para utilização em simulações. Justifica-se o estudo destas linguagens pelo seu potencial de representação, mas a aplicação em infra-estruturas virtuais torna-se complexa, pois exige a utilização de outra linguagem para descrição dos recursos.

- *VANDAL*:

  VANDAL [38] foi desenvolvida com o propósito de permitir múltiplas especificações de topologias de rede e representá-las utilizando *Virtual Private Networks*. Nas especificações, é possível informar características desejáveis de uma VPN, como por exemplo largura de banda mínima, capacidade máxima do canal e latência. O grafo de especificações resultante é utilizado para realizar um mapeamento no grafo de recursos existentes, que descrevem os canais reais de comunicação. Desta forma, VANDAL permite a representação de uma topologia virtual sobre uma arquitetura real.

- *NDL*:

  NDL (*Network Description Language*) [15] é parte do simulador de redes INSPIRE, onde permite a descrição e parametrização de todos os componentes da topologia que será simulada. A gramática de NDL aproxima-se da gramática da linguagem de programação C [56]. Usuários descrevem a configuração em um arquivo que será processado e anexado aos fontes originais, para que ocorra a compilação e simulação da arquitetura. NDL permite a descrição de recursos como nós, roteadores e *switches*, além da definição dos canais de comunicação individualmente. Devido à semelhança com a linguagem C, NDL utiliza a representação de conjuntos de componentes sob a forma de vetores, que devem ser declarados em uma seção especial para utilização posterior.

- *XML based Network Simulation Description Language*:

  Assim como NDL, *XML based Network Simulation Description Language* [17] é uma linguagem utilizada para especificação de configurações em simuladores de rede, que utiliza a definição de cenários de configuração. Basicamente, cenários são compostos pela topologia de rede, modelo de fluxo de dados, eventos padrões para casos normais e exceções, e saída de dados esperada. Esta linguagem foi desenvolvida com o propósito de utilização independente de simulador, podendo atender especificações de diferentes cenários, como redes *wireless*, computação móvel e recursos amplamente distribuídos.

2.4 *Middleware* HIPerNET e os projetos HIPCAL e CARRIOCAS

Este trabalho é diretamente relacionado com as especificações dos projetos HIPCAL [4] e CARRIOCAS [3], que basicamente exploram a aplicação de virtualização para composição dinâmica de infra-estruturas virtuais, compostas por recursos e interconexões de redes virtualizados. Em um primeiro momento, no projeto HIPCAL, está sendo desenvolvido um *middleware* de gerenciamento denominado *HIPerNET*, que após sua conclusão será aplicado também nas definições do projeto CARRIOCAS.

No projeto **HIPCAL** estuda-se a aplicação de tecnologias de virtualização de recursos computacionais para a organização de infra-estruturas em grades computacionais. Para a virtualização de recursos utiliza-se o monitor de máquinas virtuais Xen [13], permitindo que as máquinas virtuais, que deverão compor as infra-estruturas virtuais, sejam elabo-

radas de acordo com as definições de usuários. Já a virtualização dos recursos de rede é implementada utilizando o protocolo de comunicação HIP [49], que realiza um mapeamento entre os identificadores dos níveis de rede 3 e 4 (de acordo com o modelo OSI). O nível 3 (IP) utiliza o identificador *locator* para completar o roteamento de pacotes, enquanto o nível 4 (transporte), utiliza o identificador *identifier* para mapear as conexões. Assim, existe uma dependência entre os identificadores, que dificulta o redirecionamento de pacotes em infra-estruturas onde ocorre alteração dos endereços IP. O protocolo HIP propõe uma independência entre os identificadores, adicionando mais um nível intermediário, responsável pelo mapeamento das conexões. Atualmente, HIP é compatível com as versões IPv4 e IPv6, permitindo a comunicação utilizando referências para endereços IP ou para seus identificadores HIT (*Host Identity Tag*) para IPv4 e LSI (*Local Scope Identifiers*) para IPv6.

Já o projeto **CARRIOCAS** (*An Experimental High Bit Rate Optical Network for Computing Intensive Distributed Applications*), estuda-se a aplicação de virtualização para gerenciamento de redes com alta capacidade (com uma largura de banda de aproximadamente 40 Gb/s), utilizadas para interligar diversos computadores, servidores e *clusters* de computadores. Esta infra-estrutura será utilizada para execução de aplicativos para composição e visualização de imagens, em tempo real. Estuda-se, principalmente, a virtualização dos canais de comunicação, em nível físico (canais ópticos), definidos de acordo com as especificações dinâmicas dos aplicativos.

O ***middleware*** **HIPerNET** objetiva a composição e gerenciamento de infra-estruturas virtuais de acordo com especificações e necessidades das aplicações de usuários. Neste contexto, para permitir que usuários descrevam quais são os componentes necessários e qual a forma de sua infra-estrutura virtual, foi desenvolvido este trabalho, definindo uma linguagem para esta finalidade. O estágio de implementação atual de HIPerNET (primeiro protótipo para testes internos) ainda não oferece mecanismos capazes de interpretar especificações de usuários, realizando a seleção, reserva e inicialização de infra-estruturas virtuais manualmente.

Na definição de HIPerNET, propõe-se uma nomenclatura própria para descrever uma infra-estrutura virtual:

- HIPerSpace: conjunto de computadores virtualizados, executando o monitor de máquinas virtuais Xen, que realizam a comunicação utilizando o protocolo HIP,

incluindo recursos de segurança e autenticação de usuários;

- HIPI: alocação temporária de uma infra-estrutura virtual, composta de acordo com as definições de usuários. Múltiplas infra-estruturas virtuais podem ser definidas sobre um HIPerSpace, compartilhando os recursos simultaneamente;

- HIPerNET: *middleware* responsável por interpretar as descrições de usuários e realizar a reserva, alocação e gerenciamento das infra-estruturas virtuais (HIPI) e do ambiente virtualizado (HIPerSpace).

A figura 2.3 apresenta uma visão de alto nível de um ambiente virtualizado. Observa-se a representação de um HIPerSpace, onde todos os recursos (computacionais e de rede) estão virtualizados, representando a existência de um conjunto único de recursos interligados. No topo da figura, representa-se a composição de duas infra-estruturas virtuais, alocadas sobre os recursos do HIPerSpace. Estas infra-estruturas compartilham os recursos, desconhecendo a localização real dos componentes e os canais de redes físicos utilizados. Entre os HIPIs e o HIPerSpace, está localizado o *middleware* HIPerNET, que realiza a composição e gerenciamento dos recursos físicos e virtuais.

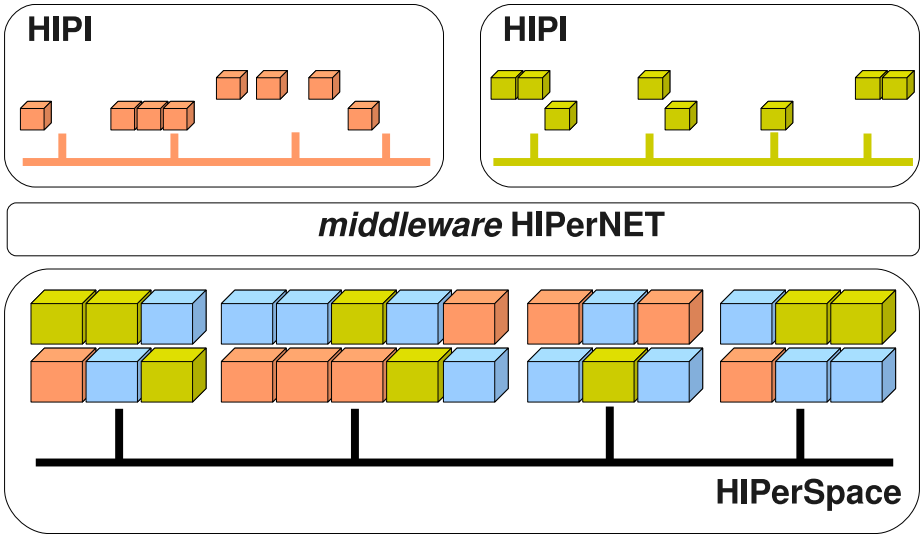


Figura 2.3: Representação da estrutura organizacional do *middleware* HIPerNET.

3 DESENVOLVIMENTO DA LINGUAGEM

Este capítulo apresenta o processo de desenvolvimento da solução proposta. Inicialmente é visto um cenário de utilização, contextualizando as funcionalidades da linguagem desenvolvida. Após, descreve-se a gramática, detalhando suas inovações e apresentando descrições testes de infra-estruturas virtuais e reais.

3.1 Cenário de Utilização

O objetivo principal desta linguagem é permitir a usuários e *middlewares* a descrição e parametrização dos componentes desejáveis (seção 2.2.2) em infra-estruturas virtuais. Embora os parâmetros desejáveis tenham sido identificados em uma revisão envolvendo os principais projetos atuais, o cenário de utilização da linguagem proposta adapta-se principalmente aos requisitos do *middleware* HIPerNET do projeto HIPCAL [4]. A figura 3.1 ilustra este cenário. Observa-se na borda inferior a existência de uma grade virtualizada, onde usuários têm a visão de um conjunto de computadores interligados. Sobre esta grade virtualizada deseja-se compor infra-estruturas virtuais de acordo com especificações de usuários. A linguagem proposta busca auxiliar a realização deste processo.

Aplica-se também nesta grade virtual a abstração e o compartilhamento dos canais de comunicação. Desta forma, usuários podem organizar suas infra-estruturas virtuais descrevendo sua topologia de rede virtual, solicitando por configurações mínimas de latência e largura de banda. Ainda na figura 3.1 apresentam-se dois grafos que descrevem possíveis topologias de rede aplicáveis à infra-estruturas virtuais. Na primeira topologia observa-se a existência de quatro canais de comunicação diretos, que podem ser descritos com diferentes características. Já na segunda topologia, adicionam-se recursos caracterizados como roteadores virtuais. Estes roteadores podem possuir diversos canais de comunicação, com propriedades distintas, interligando componentes individualmente ou

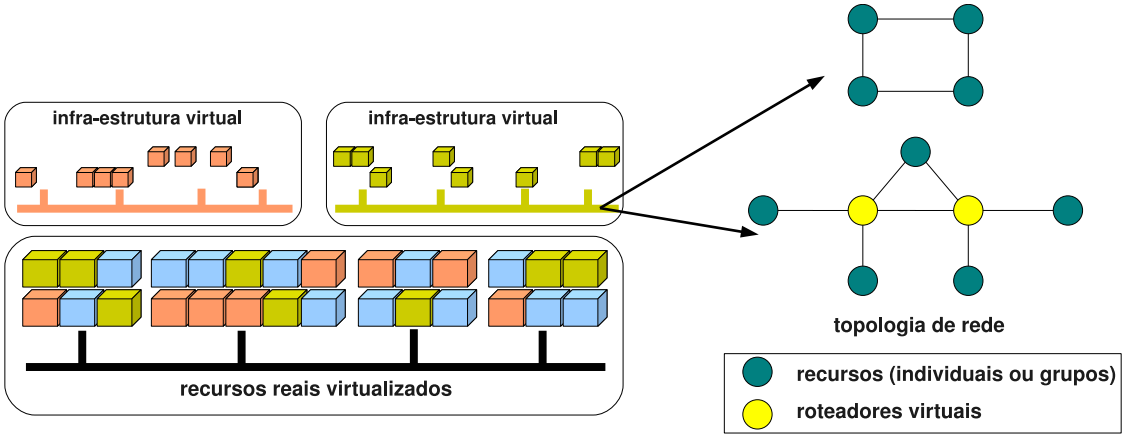


Figura 3.1: Representação da composição de uma infra-estrutura virtual.

em grupos.

Considerando a execução, em infra-estruturas virtuais, de aplicações que requerem alto desempenho computacional e realizam a transferência de um elevado número de dados, a composição da topologia de rede torna-se um ponto chave, já que influencia diretamente na execução satisfatória dessas aplicações. Neste contexto, esta linguagem visa auxiliar na descrição de ambientes complexos, com diferentes períodos de utilização de recursos. Este cenário é representado na figura 3.2 onde existe um grafo detalhando parâmetros desejáveis para uma infra-estrutura. Observa-se que os vértices A, B, C, F, H, I e J representam recursos virtuais com diferentes parametrizações, classificados como *clusters* ou roteadores virtuais. Enquanto isso, as arestas D, E e G representam canais de comunicação virtuais, que devem possuir um mínimo de capacidade de comunicação.

É importante ressaltar que, embora o desenvolvimento tenha sido orientado pelos requisitos do *middleware* HIPerNET, **nada impede** sua utilização por outros *middlewares*, capazes de interpretar as descrições realizadas com a linguagem.

3.2 Definições Semânticas

Para o desenvolvimento de uma sintaxe correta, é necessário definir a semântica esperada para a linguagem [9]. Neste processo é necessário definir quais são as expectativas por parte do *middleware* e da linguagem, para posteriormente definir uma semântica capaz de atender às necessidades descritivas.

O *middleware* espera a definição de uma linguagem capaz de informar parâmetros de configuração para os múltiplos componentes da infra-estrutura virtual. Parâmetros

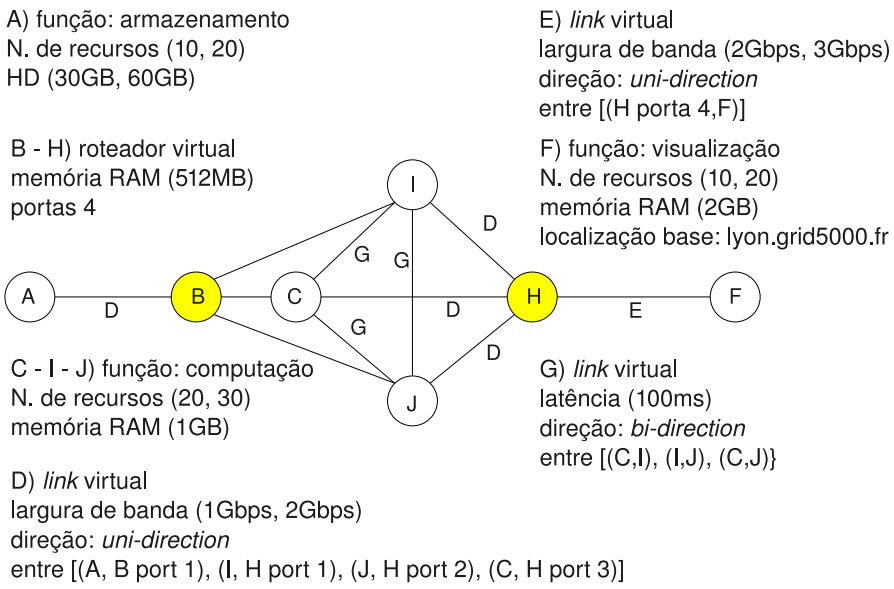


Figura 3.2: Grafo detalhando uma infra-estrutura virtual.

referentes à definição de atributos como memória RAM, freqüência de processadores, largura de banda, latência e espaço disponível para armazenamento devem ser descritos em intervalo de valores, ou então deve-se definir um máximo/mínimo tolerável. Espera-se ainda que a linguagem exija a definição correta de unidades de medidas para estes atributos. Também que é necessário que existam mecanismos no *middleware* para informar aos usuários os resultados e erros ocorridos em suas descrições. Este tratamento de erros deve abordar a definição incorreta de parâmetros, e a disponibilidade ou não dos recursos solicitados. Da mesma forma, devem existir mecanismos que interajam com a não realização de etapas definidas no cronograma de execução, permitindo que usuários definam o que deve ocorrer com o fluxo de execução de suas infra-estruturas.

Além disso, é necessário, por parte do *middleware*, que a linguagem permita a definição de diferentes caracterizações de funcionalidades para componentes pertencentes a um mesmo grupo. Desta forma, torna-se possível a representação de múltiplas formas de organizações, como por exemplo, os recursos computacionais (servidores e *clusters*) pertencentes a um determinado *site* de uma grade computacional.

A linguagem também necessita de algumas garantias por parte do *middleware*, como por exemplo a conferência de valores relativos aos atributos que definem localização e aplicativos necessários. Espera-se ainda que no desenvolvimento do *parser* interpretador do *middleware* garanta-se a referência a nomes de componentes previamente definidos, como por exemplo, permitir a descrição de canais de comunicação entre componentes

previamente informados.

3.3 Gramática de VXDL

De acordo com o cenário de utilização e as definições semânticas apresentadas, desenvolveu-se *VXDL: uma linguagem para descrição de interconexões e recursos em grades virtuais*. As figuras 3.3, 3.4 e 3.7 apresentam a gramática de VXDL, divida em descrição de recursos, descrição das topologias de rede e descrição de cronogramas de execução, respectivamente. Discute-se nessa seção cada uma dessas divisões organizacionais separadamente, relacionando suas funcionalidades com os atributos gramaticais.

3.3.1 Descrição de Recursos Computacionais

Os atributos desenvolvidos para descrição dos recursos computacionais (figura 3.3), permitem a composição e parametrização de todos os recursos, organizados individualmente ou em grupos. É possível classificar e parametrizar os componentes definindo sua utilização básica, como por exemplo: computação, armazenamento, visualização e até mesmo roteamento virtual. Esta definição auxilia na elaboração de *templates* que podem ser utilizados por *middlewares* para auxiliar usuários no processo descritivo. Na gramática, a classificação dos recursos é realizada utilizando o atributo *function* para definição da funcionalidade básica, aliado com o atributo *parameters*, que permite a parametrização detalhada dos recursos. É possível que ocorra a classificação de recursos individuais, pertencentes a um grupo, de forma distinta. Neste caso, o *middleware* deve descartar definições incorretas.

Mais especificamente, permite-se a descrição de ambientes como, por exemplo, *um conjunto de recursos homogêneos, compostos por computadores bi-processados que tenham entre 1GB e 2GB de memória RAM; executando o sistema operacional Ubuntu e bibliotecas de comunicação mpich e OpenMP*. Para expressar-se essa solicitação em VXDL, inicialmente informa-se o número de componentes desejáveis utilizando o parâmetro *size*, e posteriormente utilizam-se os parâmetros *cpu_processors* e *memory_ram* para caracterizar os componentes.

Além da parametrização básica apresentada, VXDL oferece atributos para interagir com algumas peculiaridades de grades virtuais, como: número de máquinas virtuais por nó (*vms_per_node*) e localização base (*anchor*). O primeiro refere-se à possibilidade de

```
<vxdl query> ::= "virtual grid" <name> [<time-to-use>] "{" <vg> "}"
<name> ::= <string>
<time-to-use> ::= "start" <date-time> "for" <total-time>
<date-time> ::= <date> " " <time>
<total-time> ::= <number>
<vg> ::= (<resource> | <group>)*
<resource> ::= "resource" "(" <name> ")" "{"
               ["function" <elementary-functions>]
               ["parameters" <resource-parameters>]
               ["software" <software-list>]
               ["anchor" <location>] "}"
<group> ::= "group" "(" <name> ")" "{"
        "size" <value-number>
        ["function" <elementary-functions>
        ["anchor" <location>]
        [<vg>] "}"

<value-number> ::= "(" <number> "," <number> ")"
        | "(" "min" <number> ")" | "(" "max" <number> ")"
<value-freq> ::= "(" <number> "GHz" "," <number> "GHz" ")"
        | "(" "min" <number> "GHz" ")" | "(" "max" <number> "GHz" ")"
<value-mem> ::= "(" <number> <men-unit> "," <number> <men-unit> ")"
        | "(" "min" <number> <men-unit> ")"
        | "(" "max" <number> <men-unit> ")"
<men-unit> ::= "MB" | "GB" | "TB"

<location> ::= <string>
<elementary-functions> ::= <function> ("," <function> )*
<function> ::= "endpoint" | "aquisition" | "storage" | "computing" |
        "visualization" | "network_sensor" |
        "router" "(" "ports" <ports> ")"
<ports> ::= <number>
<resource-parameters> ::= <parameters> ("," <parameters>)*
<parameters> ::= "cpu_frequency" <value-freq> |
    "cpu_mips" <value-number> |"cpu_processors" <value-number> |
    "memory_ram" <value-mem> | "hd_size" <value-mem> |
    "vms_per_node" <value-number>
<software-list> ::= <software> ("," <software>)*
<software> ::= <string>
```

Figura 3.3: Gramática de VXDL - Descrição de Recursos Computacionais.

compartilhamento de um mesmo *hardware* físico através da alocação de múltiplas máquinas virtuais. VXDL permite que, embora não utilizando todos os recursos disponíveis fisicamente, o usuário solicite uma alocação única, não compartilhando os recursos disponíveis. Já o segundo atributo refere-se à localização básica para composição da infra-estrutura. Em uma grade virtualizada, os recursos que atendem às especificações dos usuários podem ser alocados em qualquer localidade física. Algumas vezes não é desejável realizar este procedimento, podendo existir uma dependência local de recursos, como por exemplo: realizar a visualização dos resultados em uma cidade específica; ou então utilizar como dados de entrada para computação uma base da dados disponível localmente, que não pode ser transmitida. Utilizando-se o atributo *anchor*, define-se uma localização base para um alguns recursos virtuais, solicitando a composição da infra-estrutura próxima àquela localização.

3.3.2 Descrição de Topologias de Rede

O objetivo de descrever uma topologia virtual é permitir que usuários de aplicações sensíveis à latência, ou que necessitem de uma largura de banda específica, utilizem infra-estruturas virtuais de forma satisfatória, configurando-as de acordo com suas necessidades específicas. Além disso, a definição de roteadores e propriedades para todos os canais de comunicação auxilia na definição da forma da infra-estrutura. Pode-se assim, representar virtualmente um ambiente real e utilizá-lo para testes e validações de soluções.

A parte da gramática de VXDL apresentada na figura 3.4 permite a descrição de topologias, envolvendo todos os componentes descritos na seção anterior (individualmente ou em grupos). Observa-se que a descrição pode detalhar configurações internas em *clusters* bem como a configuração dos demais canais, ligando recursos individuais e grupos, através da identificação dos pares *(origem e destino)*. A figura 3.5 apresenta dois grafos (*G(V, a)*) que representam duas possíveis topologias de rede. A primeira (a) compreende somente recursos localizados nos vértices (grupos ou individuais), interligados usando quatro canais de comunicação que podem ter configurações distintas. Já a segunda topologia (b) apresenta uma organização utilizando roteadores virtuais. Nesse caso pode-se informar o caminho de comunicação entre componentes, com diferentes características, identificadas pelo número da porta no roteador.

Na gramática, a organização da topologia é realizada por meio da definição de *links*.

```
["virtual topology" <name> "{" <links> "}"]
<links> ::= (<link>)+
<link> ::= "link" "(" <name> ")" "{" <link-parameters> "}"
<link-parameters> ::= <link-parameter> ("," <link-parameter>)*
<link-parameter> ::= "bandwidth" <value-band> | "latency" <value-lat> |
                     "between" "[" <components-links> "]" |
                     "direction" <direction>
<direction> ::= "uni" | "bi"
<components-links> ::= <pair> ("," <pair>)*
<pair> ::= "(" <component> "," <component> ")"
<component> ::= <name> | <name> "port" <number>
<value-band> ::= "(" <number> <band-unit> "," <number> <band-unit> ")"
        | "(" "min" <number> <band-unit> ")"
        | "(" "max" <number> <band-unit> ")"
<value-lat> ::= "(" <number> <lat-unit> "," <number> <lat-unit> ")"
        | "(" "min" <number> <lat-unit> ")"
        | "(" "max" <number> <lat-unit> ")"
<band-unit> ::= "Kb/s" | "Mb/s" | "Gb/s"
<lat-unit> ::= "us" | "ms" | "s"
```

Figura 3.4: Gramática de VXDL - Descrição de Topologias de Rede.

Cada *link* contém a informação de atributos como *latency, bandwidth* e *direction*, para cada par *(origem, destino)*. Os dois primeiros atributos aceitam a informação de valores: (mínimo), (mínimo, máximo) ou (máximo), de acordo com a interpretação utilizada pelo *middleware*. Já origem e destino também podem informar a porta utilizada, em caso de roteadores virtuais.

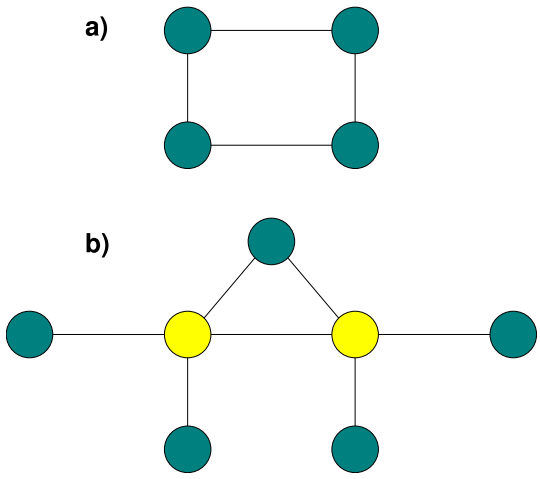


Figura 3.5: Grafos representando topologias de rede.

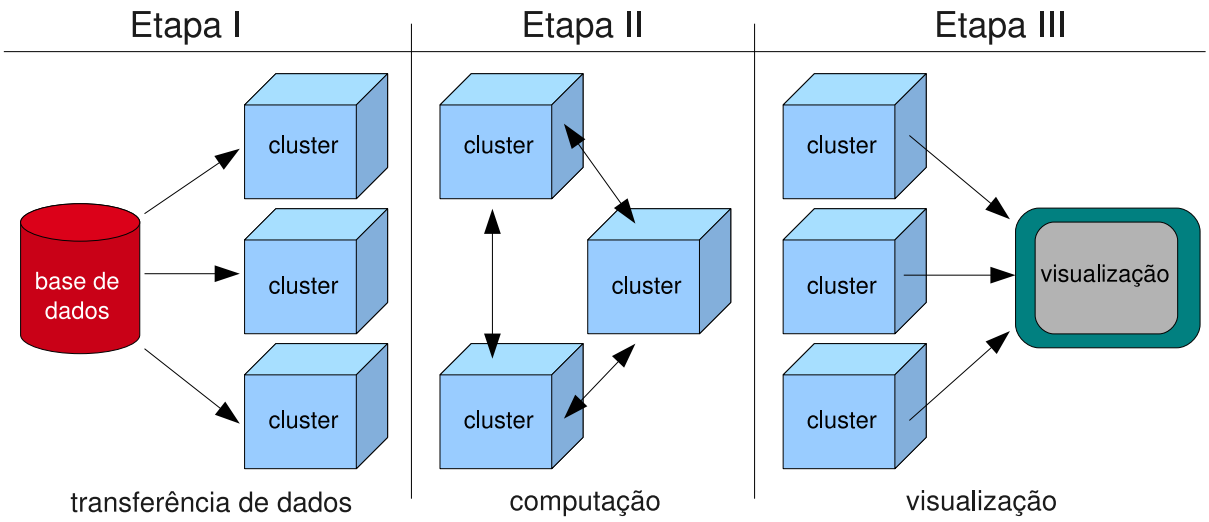


Figura 3.6: Exemplificação de um cronograma de execução.

3.3.3 Descrição de Cronogramas de Execução

Muitas vezes os recursos solicitados não são utilizados simultaneamente durante a execução das aplicações. Desta forma, a especificação de um cronograma de execução auxilia na definição dos períodos de utilização dos recursos em uma infra-estrutura, permitindo que o *middleware* realize um escalonamento eficiente, tendo conhecimento do cronograma interno utilizado pela aplicação. Definem-se períodos de utilização através da informação de um momento inicial e definição do marco de término: transferência de um determinado volume de dados ou tempo total de computação.

A figura 3.6 apresenta um exemplo de aplicação que pode ter sua execução dividida em períodos. No *estágio I* utiliza-se uma largura de banda que permita a transmissão eficiente de um volume de dados para os *clusters* computacionais. Já no *estágio II*, solicita-se por uma latência eficiente que permita a comunicação entre os *clusters*. No último estágio (*estágio III*) solicita-se recursos para a visualização dos resultados e uma conexão de rede eficiente para a transmissão dos dados

A figura 3.7 apresenta a parte da gramática de VXDL responsável pela descrição de cronogramas de execução. Basicamente, através da definição dos atributos *start, after* e *until*, cria-se uma linha do tempo, que permite informar quais recursos são necessários em cada período.

```
["virtual timeline" <name> "{" (<timeline>)+ "}"]
<timeline> ::= <time-name> "=" (<start> | <after>)
<time-name> ::= <string>
<start> ::= "start" "(" <components-list> ")" [<until>]
<after> ::= "after" "(" <time-name-list> ")" <start>
<components-list> ::= <component-name> ("," <component-name>)*
<component-name> ::= <name>
<time-name-list> ::= <time-name> ("," <time-name>)*
<until> ::= (<computation> | <transfer>)+
<computation> ::= "computation" "(" <total-time> ")"
<transfer> ::= "transfer" "(" <value-mem> ")"
```

Figura 3.7: Gramática de VXDL - Descrição de Cronogramas de Execução.

3.4 Desenvolvimento de um *Parser* Conversor para XML

Com o objetivo de testar a gramática da linguagem VXDL, desenvolveu-se um *parser* conversor que recebe como entrada um arquivo com uma descrição VXDL e realiza a análise e conversão para um arquivo no formato XML. Para isso, utilizou-se o gerador de *parsers* JavaCC (*Java Compiler Compiler*) [40]. Com JavaCC é possível desenvolver um analisador capaz de interpretar uma gramática, explorando os recursos disponíveis na API para leitura, interpretação e desenvolvimento de *parsers* com finalidades específicas. Uma das principais vantagens na utilização de JavaCC é a capacidade de identificação de erros gramaticais, exigindo que a gramática analisada não seja ambígua nem possua definições incorretas.

No desenvolvimento do *parser* pode-se verificar a definição de todas as recursões suportadas por VXDL, e identificar a facilidade na compreensão da descrição por parte dos *middlewares*. O uso do *parser* desenvolvido é simples, através da utilização de uma interface por linha de comando, como no exemplo seguinte: *$ java vxdl2xml [options]* <*input*> <*output*>. Em *options* pode-se definir uma opção de *debug*, que permite acompanhar os estágios de interpretação do arquivo. Já *input* e *output* referem-se aos arquivos de entrada, contendo a descrição em VXDL, e o arquivo de saída, gerado com conteúdo XML.

3.5 Exemplos de Descrições Utilizando VXDL

Para ilustrar a utilização de VXDL na descrição de infra-estruturas virtuais para aplicações reais, foram selecionadas três aplicações testes dos projetos HIPCAL e CARRIOCAS. Inicialmente, são descritas as necessidades das aplicações ARIA e PROTEOME,

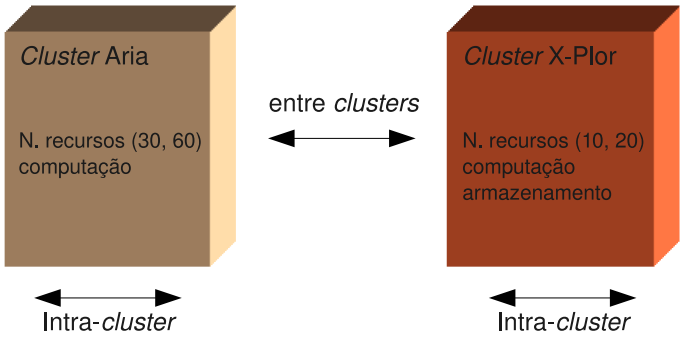


Figura 3.8: Infra-estrutura de execução da aplicação Aria.

oriundas do projeto HIPCAL. Basicamente, estas aplicações desenvolvem cálculos computacionais sobre seqüências de proteínas, usualmente necessitando de uma alta capacidade computacional, armazenamento eficiente e transmissão de grande volume de dados. Já a terceira aplicação, VISUPIPE, pertence ao conjunto das aplicações testes do projeto CARRIOCAS. Esta aplicação realiza um *pipeline* para visualização de imagens, exigindo a composição de uma infra-estrutura com diferentes *clusters*, cada um com uma propriedade e configuração. Além disso, a topologia de rede em VISUPIPE é extremamente importante, pois reflete diretamente no sucesso da execução da aplicação.

O último teste desenvolvimento aborda a utilização de VXDL para descrever uma infra-estrutura de grade computacional real. Para isso, selecionou-se o ambiente de testes Grid5000 [2], composto por aproximadamente 3500 computadores interligados.

3.5.1 Descrição de uma Infra-estrutura Virtual: Aria

Aria (*Ambiguous Restraints for Iterative Assignment*) é uma aplicação que efetua cálculos computacionais sobre as bases de dados experimentais NMR (*Nuclear Magnetic Resonance Spectroscopy*) [12] aplicando regras definidas pelo usuário para a criação de estruturas moleculares através do cálculo de novas seqüências de proteínas. A figura 3.8 apresenta uma possível infra-estrutura computacional para execução da aplicação. A computação ocorre na forma de um *pipeline*, sendo que os resultados de um processamento são utilizados como dados de entrada da etapa seguinte. Inicializa-se o cálculo utilizando o *software* alocado no *cluster Aria*, e a saída deste processamento inicial é direcionada para o *cluster X-Plor*. Esta recursão ocorre diversas vezes, sendo que o usuário define o número máximo necessário.

A figura 3.9 apresenta uma descrição VXDL para a infra-estrutura especificada na

```
virtual grid Aria start 20/06/2008 15:30 for 4000 {
    group (Cluster_Aria) {
        function computing
        size (30, 60)
        resource (Nodes_Cluster_Aria) {
            function computing
            parameters cpu_processors (1, 2),
                       cpu_frequency (1.60GHz, 3GHz)
        }
    }

    group (Cluster_XPlor) {
        function computing, storage
        size (10, 20)
        resource (Nodes_Cluster_XPlor) {
            function computing
            parameters hd_size (30GB, 60GB),
                cpu_frequency (2GHz, 3GHz)
        }
    }
}

virtual topology Aria_Network {
    link (Between_Clusters) {
        bandwidth (1Gb/s, 1Gb/s),
        between [(Cluster_Aria, Cluster_XPlor),
                    (Cluster_XPlor, Cluster_Aria)]
    }

    link (Intra_Cluster_Aria) {
        latency (0.2us, 0.8us),
        between [(Nodes_Cluster_Aria, Nodes_Cluster_Aria)]
    }
}
```

Figura 3.9: Descrição VXDL para a aplicação Aria.

figura 3.8. Basicamente, descrevem-se dois *clusters*, com diferentes propriedades como: número de componentes, classificação, requisitos de processamento e de memória RAM. Descreve-se ainda uma topologia de rede básica, solicitando uma baixa latência interna ao *cluster Aria* e uma largura de banda de 1Gbps ligando os dois *clusters*.

3.5.2 Descrição de uma Infra-estrutura Virtual: Proteome

A aplicação Proteome busca extrair o número máximo de informações em seqüências de proteínas recentemente identificadas, armazenadas e organizadas em base de dados biológicas [14]. Devido ao elevado número de informações e de bases de dados, Proteome utiliza diversas aplicações diferentes que efetuam processamento simultâneo sobre as informações. A figura 3.10 apresenta uma possível infra-estrutura para execução desta aplicação, contendo a descrição dos *clusters* necessários e da topologia de rede. Nesta descrição, existem três *clusters*, com diferentes classificações: um para armazenamento (*Cluster DB*) e dois para computação (*Cluster Clustaw* e *Cluster Search*). Inicialmente é realizado

uma pesquisa e indexação nos dados armazenados (computação realizada pelo *Cluster Search*). Após, inicializam-se tarefas individuais, que efetuam comunicação usando MPI (*Message Passing Interface*) [26], no *Cluster Clustaw*.

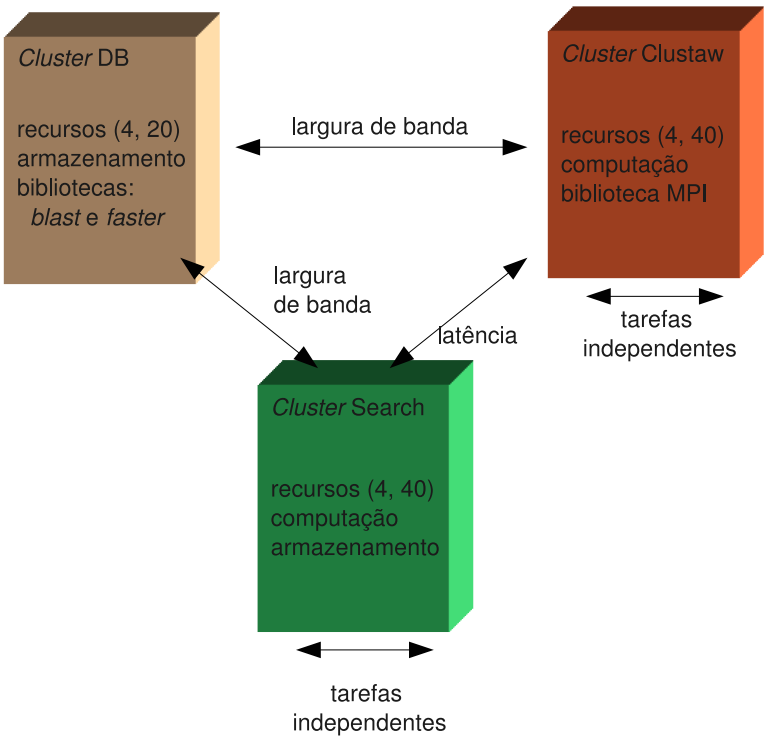


Figura 3.10: Infra-estrutura de execução da aplicação Proteome.

Uma possível descrição VXDL é apresentada na figura 3.11. Observa-se a descrição e parametrização dos três *clusters*, bem como a solicitação por uma alta largura de banda interna no *Cluster DB*, e uma comunicação eficiente entre todos os *clusters*. Os dados informados nessa descrição foram obtidos na documentação descritiva do projeto HIPCAL [4].

3.5.3 Descrição de uma Infra-estrutura Virtual: VISUPIPE

Visualization Haute Performance Collaborative Distant (VISUPIPE) é uma aplicação utilizada para visualização *on-line* de imagens. Basicamente, VISUPIPE representa um *pipeline*, onde as imagens armazenadas são processadas e compostas para a realização da visualização final. A figura 3.12 apresenta a infra-estrutura necessária para execução de VISUPIPE, composta por cinco *clusters* interconectados, com classificações para armazenamento, computação e visualização. No *pipeline* representado nesta figura, os canais de comunicação utilizados são extremamente importantes, pois influenciam diretamente no resultado da aplicação.

```
virtual grid Proteome {
    group (Cluster_DB) {
        function storage, aquisition
        size (4, 20)
        resource (Nodes_Cluster_DB) {
            function storage
            parameters hd_size (30GB, 100GB)
        }
    }
    group (Cluster_Clustaw) {
        function computing
        size (4, 40)
        resource (Nodes_Cluster_Clustaw) {
            function computing
            parameters cpu_frequency (2GHz, 3GHz)
        }
    }
    group (Cluster_Search) {
        function computing, storage
        size (4, 40)
        resource (Nodes_Cluster_Search) {
            function computing, storage
            parameters cpu_frequency (2GHz, 3GHz),
            hd_size (10GB, 20GB)
        }
    }
}
virtual topology Proteome_Network {
    link (High_Bandwidth_Between_Clusters) {
        bandwidth (min 2Gb/s),
        between [(Cluster_DB, Cluster_Clustaw),
                 (Cluster_DB, Cluster_Search)]
    }
    link (High_Bandwidth_Intra_Cluster) {
        bandwidth (min 1Gb/s),
        between [(Cluster_DB, Cluster_DB)]
    }
}
```

Figura 3.11: Descrição VXDL para a aplicação Proteome.

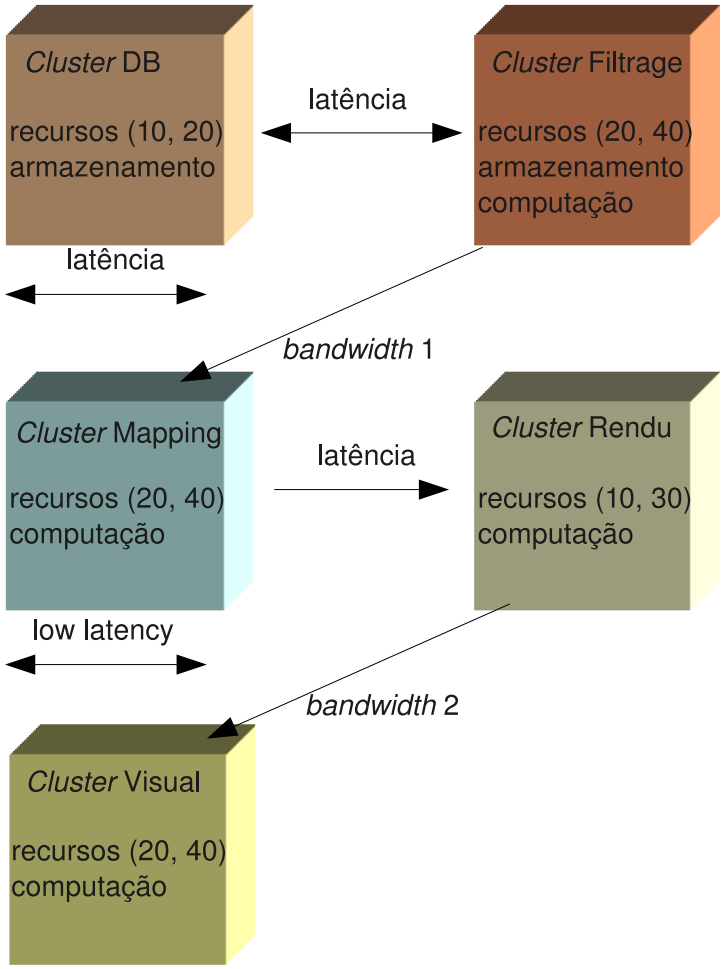


Figura 3.12: Infra-estrutura de execução da aplicação VISUPIPE.

As figuras 3.13, 3.14 e 3.15 apresentam uma descrição da infra-estrutura utilizando VXDL. Para exemplificação, separou-se a solicitação em: descrição de recursos, descrição da topologia de rede e descrição do cronograma de execução, respectivamente. A figura 3.13 apresenta a descrição dos cinco *clusters* e sua parametrização, identificando a classificação, data e hora em que deseja-se utilizar, tempo total de utilização e a definição de uma localização base para o recurso de visualização (figura 3.16 [1]): *anchor "lyon.grid5000.fr"*.

Na figura 3.14 é descrita a configuração da topologia de rede desejável para esta aplicação. Os principais pontos nessa descrição são a solicitação por altas larguras de banda entre os *clusters* computacionais e de visualização. Para o primeiro solicita-se entre 7Gbps e 13Gbps, e para o segundo entre 13Gbps e 26Gbps. Já na figura 3.15 apresenta-se um possível cronograma para execução de VISUPIPE. Identificam-se os períodos de exe-

[1]Fonte: documentação do projeto CARRIOCAS [3]

cução de cada trecho do *pipeline*, onde são inicializados os recursos necessários, como por exemplo no período *t0: start (Cluster_DB, Cluster_Filtrage, Low_Latency_Between_Cluster) until transfer (50GB)*. Neste trecho, inicializam-se os *clusters* de armazenamento e de aplicação do filtro inicial, juntamente com um canal de comunicação que possua baixa latência. O término deste período é caracterizado pela transferência de 50GB de dados.

3.5.4 Descrição de uma Infra-estrutura Real - Grid5000

VXDL foi desenvolvida apresentando inovações que permitam a descrição de infra-estruturas virtuais, mas também pode ser utilizada para descrever infra-estruturas reais. Para representação deste cenário, selecionou-se a grade computacional Grid5000 [2], composta por aproximadamente 3500 computadores interconectados. Localizada na França, Grid5000 é um ambiente de testes utilizado por diversos experimentos, possuindo recursos amplamente variados, inclusive com configurações de rede distintas. Basicamente, esta infra-estrutura é composta por *clusters* divididos em nove *sites*: Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia e Toulouse. As figuras 5.1, 5.2, 5.3, 5.4 e 5.5, localizadas no apêndice deste trabalho, apresentam a descrição de todos os componentes da infra-estrutura. Especificamente na figura 5.5 apresenta-se a topologia de rede básica que interliga os recursos da grade. As informações utilizadas foram obtidas na página da internet *www.grid5000.fr*.

```
virtual grid VISUPIPE start 19/02/2008 14:32 for 4 {
    group (Cluster_DB) {
        function storage, aquisition
        size (10, 20)
        resource (Node_Cluster_DB) {
            function storage
            parameters hd_size (30GB, 60GB)
        }
    }
    group (Cluster_Filtrage) {
        function computing
        size (20, 40)
        resource (Node_Cluster_Filtrage) {
            function computing
            parameters cpu_frequency (1.6GHz, 3GHz),
            memory_ram (512MB, 2GB)
        }
    }
    group (Cluster_Mapping) {
        function computing
        size (20, 40)
        resource (Node_Cluster_Mapping) {
            function computing
            parameters cpu_processors (min 2),
            cpu_frequency (1.6GHz, 3GHz)
        }
    }
    group (Cluster_Rendu) {
        function computing
        size (10, 30)
        resource (Node_Cluster_Rendu) {
            function computing
            parameters cpu_frequency (1.6GHz, 3GHz),
            memory_ram (512MB, 2GB)
        }
    }
    group (Cluster_Visual) {
        function computing, visualization, endpoint
        size (20, 40)
        resource (Node_Cluster_Rendu) {
            function computing, visualization
            parameters cpu_frequency (1.6GHz, 3GHz),
            memory_ram (2GB, 4GB)
            anchor lyon.grid5000.fr
        }
    }
}
```

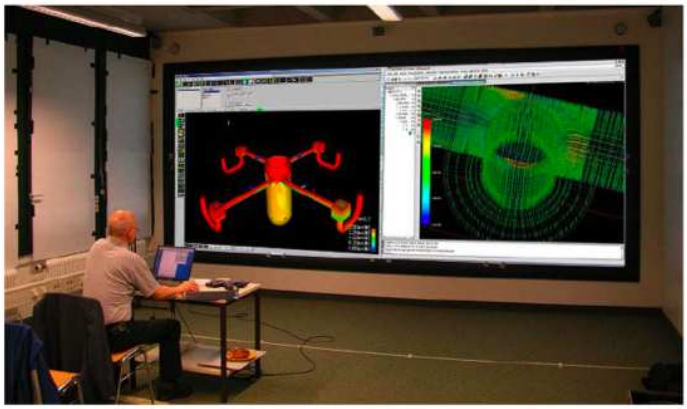
Figura 3.13: Descrição VXDL para a aplicação VISUPIPE - A.

```
virtual topology VISUPIPE_Network {
    link (Low_Latency_Intra_Cluster) {
        latency (max 0.01ms),
        between [ (Node_Cluster_DB, Node_Cluster_DB),
             (Node_Cluster_Mapping, Node_Cluster_Mapping),
             (Node_Cluster_Rendu, Node_Cluster_Rendu)]
    }
    link (Low_Latency_Between_Cluster) {
        latency (max 0.05ms),
        between [(Cluster_DB, Cluster_Filtrage),
        (Cluster_Filtrage, Cluster_DB),
        (Cluster_Mapping, Cluster_Rendu)]
    }
    link (Bandwidth1) {
        bandwidth (7Gb/s, 13Gb/s),
        between [(Cluster_Filtrage, Cluster_Mapping)]
    }
    link (Bandwidth2) {
        bandwidth (13Gb/s, 26Gb/s),
        between [(Cluster_Rendu, Cluster_Visual)]
    }
}
```

Figura 3.14: Descrição VXDL para a aplicação VISUPIPE - B.

```
virtual timeline VISUPIPE_Timeline {
    t0 = start (Cluster_DB, Cluster_Filtrage,
        Low_Latency_Between_Cluster) until transfer (50GB)
    t1 = after (t0) start (Cluster_Mapping, Bandwidth1)
        until computation (1hour)
    t2 = after (t0) start (Cluster_Rendu) until computation (45min)
    t3 = after (t2) start (Cluster_Visual, Bandwidth2)
}
```

Figura 3.15: Descrição VXDL para a aplicação VISUPIPE - C.



Exemple de mur d'images ayant une résolution de plusieurs millions de pixels.

Figura 3.16: Exemplo de recursos de visualização utilizados pela aplicação VISUPIPE.

4 AVALIAÇÃO

Este capítulo descreve o processo de avaliação de VXDL. Inicialmente, apresenta-se a metodologia utilizada, descrevendo as etapas e os objetivos principais. Após, apresenta-se uma comparação com outras linguagens para descrição de recursos, seguida de uma análise da execução de *benchmarks* sobre infra-estruturas virtuais elaboradas com diferentes descrições VXDL.

4.1 Metodologia

A análise da linguagem desenvolvida foi realizada em duas etapas [39]. Inicialmente, realizou-se uma comparação das descrições VXDL com outras linguagens existentes, utilizadas em grades computacionais reais. Em um segundo momento, desenvolveram-se infra-estruturas virtuais elaboradas com diferentes descrições VXDL, e executaram-se sobre essas infra-estruturas *benchmarks* para ambientes distribuídos.

Basicamente, a análise comparativa e a utilização de infra-estruturas descritas com VXDL buscam investigar os seguintes fatores:

- facilidade na descrição de infra-estruturas complexas;
- composição de diferentes infra-estruturas virtuais, utilizando os parâmetros oferecidos por VXDL;
- visualização do impacto na execução de aplicações sobre infra-estruturas; virtuais, variando em sua descrição parâmetros de rede, como latência e largura de banda;
- descrição e utilização de um ambiente em que utilizam-se roteadores virtuais;
- ressaltar a importância de interagir com as particularidades de infra-estruturas virtuais, como a alocação de múltiplas máquinas virtuais em um mesmo *hardware*.

É importante ressaltar que a análise de VXDL visa investigar a capacidade descritiva da linguagem, e não analisar o processo de seleção e reserva de recursos. De acordo com o projeto HIPCAL, esta funcionalidade será implementada no *middleware* HIPerNET, nas etapas futuras. Quando necessário, no presente trabalho, a seleção de recursos e a composição das infra-estruturas virtuais foi realizada manualmente, apenas com o objetivo de exemplificar a execução em uma arquitetura virtualizada.

4.2 Comparação de Descrições

Nesta seção, comparam-se descrições elaboradas utilizando VXDL com descrições utilizando outras linguagens para representação de infra-estruturas e recursos. Foram selecionadas as linguagens ClassAd [54], utilizada para realizar especificações no *middleware* do projeto Condor [1]; SWORD [50], uma linguagem genérica para descrição de recursos em grades computacionais; e vgDL [19] uma linguagem desenvolvida com o propósito de permitir especificações de infra-estruturas virtuais, utilizada no *middleware* do projeto VGrADS [41].

Para realizar a comparação entre as descrições, selecionou-se a aplicação VISUPIPE, por possuir um conjunto de requisitos computacionais distintos e por ser uma aplicação onde a composição da infra-estrutura de rede influencia diretamente no resultado da aplicação. O *pipeline* para visualização de imagens, VISUPIPE, foi descrito na seção 3.5.3 e sua estrutura computacional é detalhada na figura 3.12. As figuras 4.1, 4.2 e 4.3 apresentam as descrições utilizando as linguagens ClassAd, SWORD e vgDL, respectivamente. A descrição utilizando VXDL é apresentada nas figuras 3.13, 3.14 e 3.15, que descrevem os recursos, a topologia de rede e o cronograma de execução, respectivamente.

Analisando a descrição realizada com ClassAd (figura 4.1) observa-se uma dificuldade na representação de um grupos de recursos, com diferentes parâmetros, que devem ser utilizados simultaneamente. Para representar infra-estruturas computacionais como a de VISUPIPE, é necessário descrever cada *cluster* como um *job* independente, e utilizar o mesmo *owner* para permitir uma troca de informações. A utilização de *jobs* independentes não permite representar a interação e o cronograma de execução entre os *clusters*, e não garante a solicitação simultânea dos recursos, já que um *job* pode ser aceito pelo *middleware* e os outros negados. Além disso, ClassAd não oferece recursos para a descrição de canais de comunicação, impossibilitando a composição de topologias de rede.

```
{[type       = "job";
 qdate      = 886799469;
 owner      = "visupipe";
 cmd        = "cluster_db.sh";
 constraint = other.type == "machine"
              && os == "debian"
              && disk >= 30GB ]
 [type       = "job";
 qdate      = 886799469;
 owner      = "visupipe";
 cmd        = "cluster_filtrage.sh";
 constraint = other.type == "machine"
              && cpu >= 1.6GHz
              && memory >= 1GB ]
 [type       = "job";
 qdate      = 886799469;
 owner      = "visupipe";
 cmd        = "cluster_mapping.sh";
 constraint = other.type == "machine"
              && cpu >= 1.6GHz
              && memory >= 512MB ]
 [type       = "job";
 qdate      = 886799469;
 owner      = "visupipe";
 cmd        = "cluster_rendu.sh";
 constraint = other.type == "machine"
              && cpu >= 1.6GHz
              && memory >= 1GB ]
 [type       = "job";
 qdate      = 886799469;
 owner      = "visupipe";
 cmd        = "cluster_visual.sh";
 constraint = other.type == "machine"
              && cpu >= 2GHz
              && memory >= 2GB ]
};
```

Figura 4.1: Descrição utilizando ClassAd para a aplicação VISUPIPE.

Na descrição utilizando SWORD (figura 4.2) é possível representar mais detalhadamente todos os recursos computacionais da infra-estrutura. Esta linguagem permite a representação de grupos de recursos (*clusters*), mas não prevê a representação de grades computacionais. Embora SWORD não tenha sido desenvolvida com o propósito de representação de infra-estruturas virtuais, é possível especificar uma localização base de um recurso. Considerando parâmetros relacionados com a composição da rede, SWORD permite a informação de latência e largura de banda desejável, representando canais de comunicação únicos entre os recursos, não permitindo a definição de roteadores virtuais. Além disso, a parametrização da rede em SWORD é realizada juntamente com os recursos computacionais, dificultando a composição de infra-estruturas complexas.

Utilizando-se vgDL para descrever a infra-estrutura de VISUPIPE (figura 4.3) é possível detalhar todos os parâmetros para os recursos computacionais mas não explora-se a definição de uma localização base e de um número máximo de máquinas virtuais que devem ser alocadas em um mesmo computador. Inicialmente, observa-se uma facilidade na representação, devido à abstração da parametrização. Esta mesma facilidade torna complexa a definição de requisitos essenciais para uma aplicação, já que vgDL objetiva a abstração da especificação de topologias de rede através da utilização dos identificadores *Close, HighBW, ClusterOf* e *LooseBagOf*, que de acordo com a linguagem, definem a proximidade e a classificação dos recursos. Este tipo de parametrização não permite a definição da topologia de rede necessária para VISUPIPE, pois não permite especificar valores para os parâmetros desejados.

Analisando-se as descrições realizadas com as linguagens citadas e comentando a descrição VXDL, observa-se que é possível descrever a composição de toda infra-estrutura necessária. Ressaltam-se pontos como a especificação de parâmetros (latência, largura de banda e direção) para todos os canais de comunicação; definição do número de máquinas virtuais que devem compartilhar um recurso; e definição de uma localização base para a composição da infra-estrutura virtual.

Neste exemplo não se explora a utilização de roteadores virtuais, já que a infra-estrutura de VISUPIPE representa um *pipeline*, não sendo necessário definir canais de comunicação distintos para um mesmo recurso. Mas, considerando a utilização de grades virtualizadas, por aplicações que requerem a composição da topologia, as linguagens detalhadas não estão preparadas para representar o direcionamento dos canais de comu-

```
Group Cluster_DB
    NumMachines 20
    Required FreeDisk [30, 60] (GB)
    Required AllPairs Latency [0.01, 0.03] (ms)
    Required OS ["Debian"]

Group Cluster_Filtrage
    NumMachines 40
    Required CPUFrequency [1.6, 3] (GHz)
    Required Memory [1, 3] [GB]

Group Cluster_Mapping
    NumMachines 40
    Required CPUFrequency [1.6, 3] (GHz)
    Required AllPairs Latency [0.01, 0.03] (ms)
    Required Memory [1, 2] [GB]

Group Cluster_Rendu
    NumMachines 30
    Required CPUFrequency [2, 3] (GHz)
    Required AllPairs Latency [0.01, 0.03] (ms)
    Required Memory [1, 3] [GB]

Group Cluster_Visual
    NumMachines 40
    Required Memory [2, 4] [GB]
    Required Location ["lyon.grid5000.fr", 0.0, 10.0] (ms)

InterGroup
    Required OnePair BW
        Cluster_Filtrage Cluster_Mapping [7, MAX] (Gb/s)
    Required OnePair BW
        Cluster_Rendu Cluster_Visual [13, MAX] (Gb/s)
    Required OnePair Latency
        Cluster_DB Cluster_Filtrage [0.05, 0.10] (ms)
    Required OnePair Latency
        Cluster_Mapping, Cluster_Rendu [0.05, 0.10] (ms)
```

Figura 4.2: Descrição utilizando SWORD para a aplicação VISUPIPE.

```
VISUPIPE = ClusterOf<Nodes_DB>[10:20];
    Nodes_DB = {disk > 30GB}
Close
LooseBagOf<Nodes_Filtrage>[20:40];
    Nodes_Filtrage = {memory > 512MB, cpu > 1.6GHz}
HighBW
ClusterOf<Nodes_Mapping>[20:40];
    Nodes_Mapping = {cpu > 1.6GHz}
Close
LooseBagOf<Nodes_Rendu>[10:30];
    Nodes_Rendu = {cpu > 1.6GHz, memory > 512MB}
HighBW
ClusterOf<Nodes_Visual>[20:40];
    Nodes_Visual = {cpu > 3GHz, memory > 2GB}
```

Figura 4.3: Descrição utilizando vgDL para a aplicação VISUPIPE.

nicação virtuais. Muitas aplicações são representadas em *workflows* ou grafos acíclicos direcionados, que podem ser mapeados facilmente em VXDL, definindo os requisitos de comunicação e o cronograma de execução.

4.3 Comparação da Disponibilidade de Parâmetros

As tabelas 4.1 e 4.2 revisam a disponibilidade de parâmetros para descrição de recursos computacionais e de rede, respectivamente. Objetiva-se com essa comparação clarificar quais são as possibilidades de descrição das linguagens, quando aplicadas a infra-estruturas virtuais. As informações referentes aos recursos disponíveis em cada linguagem foram obtidas na documentação de cada projeto (artigos, páginas de projetos e relatórios técnicos). Todos os parâmetros selecionados para comparação estão disponíveis para descrição utilizando VXDL.

Inicialmente, na tabela 4.1, compara-se a possibilidade de detalhar parâmetros para descrição de: recursos individuais (representado por Rec. Indiv.), *clusters*, grades computacionais, ponto de localização base, número de máquinas virtuais alocados por computador (representado por MV/comp), classificação básica da funcionalidade do componente e descrição de *software*. De acordo com a tabela observa-se que, particularmente, a linguagem vgDL possui recursos para classificação de componentes de acordo com sua funcionalidade básica, podendo auxiliar *middlewares* na criação de *templates*. Já SWORD aceita a especificação de uma localização base para uma infra-estrutura, adaptando-se nesse caso, a um contexto virtual.

Tabela 4.1: Disponibilidade de parâmetros para descrição de recursos.

Linguagem	Rec. Indiv.	*Clusters*	Grades	Localização	MV/comp	Classificação	*Software*
ClassAd	Sim	Não	Não	Não	Não	Não	Não
Redline	Sim	Sim	Não	Não	Não	Não	Não
SWORD	Sim	Sim	Não	Sim	Não	Não	Não
GLUE	Sim	Sim	Sim	Não	Não	Não	Sim
BPDL	Sim	Sim	Sim	Não	Não	Não	Sim
vgDL	Sim	Sim	Sim	Não	Não	Sim	Não
VXDL	Sim	Sim	Sim	Sim	Sim	Sim	Sim

Na tabela 4.2, apresenta-se uma comparação envolvendo a disponibilidade de parâmetros para descrição da rede, considerando somente linguagens desenvolvidas com o objetivo de descrever recursos (infra-estruturas). Observa-se, neste caso, que somente SWORD, BPDL e GLUE permitem a especificação de alguns parâmetros (latência e largura de banda), mas não aceitam a contextualização utilizando roteadores virtuais.

Tabela 4.2: Disponibilidade de parâmetros para descrição da rede.

Linguagem	Latência	Largura de banda	Direção	Roteadores virtuais
ClassAd	Não	Não	Não	Não
Redline	Não	Não	Não	Não
SWORD	Sim	Sim	Não	Não
GLUE	Sim	Sim	Não	Não
BPDL	Sim	Sim	Não	Não
vgDL	Não	Não	Não	Não
VXDL	Sim	Sim	Sim	Sim

Além dos recursos citados, considera-se também a possibilidade de representação de um cronograma de execução, proposto e implementado em VXDL, que permite a definição dos períodos de execução de cada recurso (computacional e de rede) dentro do tempo total da reserva.

4.4 Análise de Infra-estruturas Descritas Utilizando VXDL

Nesta seção, apresentam-se os resultados da execução de *benchmarks* sobre infra-estruturas virtuais compostas com diferentes descrições VXDL. Inicialmente, descreve-se os *benchmarks* utilizados e o ambiente de experimentação (configurações reais e virtuais). Após, analisam-se os resultados da execução em cada uma das infra-estruturas virtuais.

4.4.1 *Benchmark* NAS

O *benchmark* para arquiteturas paralelas NAS (*Numeric Aerodynamic Simulation*) [11] é formado por um conjunto de aplicações desenvolvidas pela NASA, agrupadas e organizadas com o objetivo de criar um *benchmark* que explora diferentes requisitos de comunicação. As aplicações que compõe NAS foram desenvolvidas utilizando as linguagens de programação C [56] e Fortran [47], e são organizadas em classes, criadas de acordo com o tipo de comunicação utilizado e parâmetros iniciais para computação, que definem a ocupação de memória e tempo de processamento (classes: *sample code*, A e B).

As aplicações que compõem NAS são listadas abaixo, descrevendo sua comunicação básica:

- *Embarrassingly Parallel (EP)*: esta aplicação requer pouca comunicação, utilizando mensagens pequenas.
- *Multigrid (MG)*: utiliza comunicações em curta e longas distâncias.
- *Conjugate Gradient (CG)*: realiza muita comunicação utilizando mensagens pequenas e grandes.
- *3D FFT PDE (FT)*: utiliza comunicação coletiva.
- *Integer Sort (IS)*: comunicação utilizando mensagens grandes.
- *LU*: realiza muita comunicação utilizando mensagens de tamanho médio.

Para comunicação entre os processos, NAS utiliza MPI (*Message Passing Interface*) [26] uma biblioteca de comunicação que oferece recursos para troca de mensagens em arquiteturas com memória distribuída. Basicamente, as principais características de comunicação abordadas em NAS são *end-to-end, multicast* e sincronização.

Neste trabalho, utilizam-se apenas quatro aplicações do *benchmark* NAS (cg, is, lu e mg), classe B, que possuem diferentes características de comunicação, ocupação de memória e computação. Foi utilizado como base para esta seleção um estudo recente [32] que apresentou uma análise da comunicação destas aplicações em grades computacionais, desenvolvido sobre a mesma estrutura física utilizada neste trabalho.

4.4.2 *Benchmark* HPL

High Performance Linpack [51] é um *benchmark* para análise de desempenho em arquiteturas com memória distribuída, que realiza vários cálculos em um sistema linear aleatório, utilizando dupla precisão aritmética. Desenvolvido utilizando a linguagem de programação C [56], HPL também realiza a comunicação entre os processos distribuídos através da biblioteca MPI [26].

Umas das principais características de HPL é a possibilidade de parametrização através de um arquivo de configuração, onde pode-se informar o número de computadores que deverão ser utilizados e outros parâmetros que influenciam na ocupação de memória em cada componente, como o tamanho da matriz de cálculo. Durante a compilação de HPL é necessário informar qual a biblioteca matemática com suporte para cálculos algébricos será utilizada: *Basic Linear Algebra Subprograms (CBLAS)* ou *Vector Signal Image Processing Library (VSIPL)*. Neste trabalho selecionou-se CBLAS. Além disso, foram utilizadas duas configurações de dimensões para matrizes de dados: 3000 e 8000. Estas configurações foram selecionadas por resultarem em diferentes ocupações de memória e volume de dados para transmissão. Os resultados foram coletados diretamente das medidas internas do *benchmark*.

4.4.3 Infra-estruturas Físicas e Virtuais

Para realização desta avaliação utilizou-se a infra-estrutura física disponível no ambiente de testes Grid5000 [2]. Basicamente, o propósito principal desta plataforma, alocada na França, é servir como base experimental para pesquisas envolvendo grades computacionais. Para isso, oferece aproximadamente 3500 computadores interligados, distribuídos em nove *sites*, conectados com redes de comunicação de 1Gb/s e 10Gb/s. A figura 4.4 representa a distribuição geográfica dos *sites* e descreve quais conexões são utilizadas para interligação. Os *sites* destacados foram utilizados para composição da infra-estrutura virtual, por possuírem recursos que permitem a execução do monitor de máquinas virtuais Xen. Atualmente, as configurações utilizadas em outros *sites* não oferecem carregadores de inicialização capazes de inicializar múltiplos módulos, como o aplicativo *grub* por exemplo [1].

Mais especificamente, foram selecionadas três *clusters* nos *sites* descritos: Capricorne

[1] maiores informações podem ser obtidas em http://www.gnu.org/software/grub/ e http://kadeploy.imag.fr/

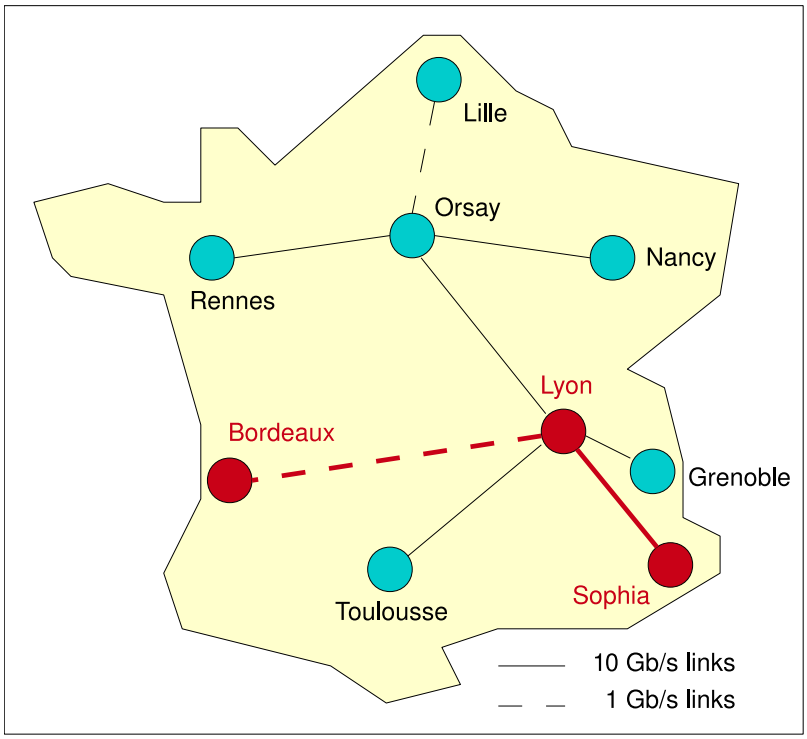


Figura 4.4: Representação organizacional dos recursos do Grid5000.

em Lyon, Azur em Sophia e Bordemer em Bordeaux. A tabela 4.3 oferece mais detalhes sobre as configurações computacionais de cada *cluster*. Para preparação da infra-estrutura virtual, utilizou-se o monitor de máquinas virtuais Xen, na versão 3.1. A versão MPI utilizada para comunicação foi *mpich* [31], e todos os resultados apresentados são o resultado da média de dez execuções para cada aplicação.

Tabela 4.3: Descrição dos *clusters* utilizados.

Componente	Bordemer	Azur	Capricorne
Modelo	IBM eServer 325	IBM eServer 325	IBM eServer 325
Nós	48 (2 cpus per node)	72 (2 cpus per node)	56 (2 cpus per node)
CPU	AMD 248 2.2GHz	AMD 246 2.0GHz	AMD 246 2.0GHz
Memória	2 GB	2GB	2 GB
Rede	Myrinet-2000	Myrinet-2000	Myrinet-2000
Armazenamento	80 GB	80 GB	80 GB

Para realizar os testes envolvendo latência entre troca de mensagens, foi inicialmente identificada a latência interna nos *clusters* e entre os *sites*. Internamente, a latência média é de 0.98ms. Entre os *sites* é de 9.4ms. A seleção dos recursos para composição das infra-estruturas virtuais foi realizada manualmente, sem auxílio de *middlewares*.

4.4.4 Descrições de Topologias de Rede: Latências

4.4.4.1 Benchmark *HPL*

Para execução do *benchmark* HPL foram desenvolvidas duas descrições VXDL, representadas por *Descrição sem latência* e *Descrição com latência*.

- *Descrição sem latência*

  Esta descrição, representada pela figura 4.5 apenas descreve o número de componentes necessários, solicitando por uma configuração de memória RAM mínima de 1GB e alocação de uma máquina virtual por computador.

- *Descrição com latência*

  Nesta segunda descrição, adiciona-se aos parâmetros da primeira, uma descrição básica da rede, solicitando por uma latência máxima de 0.100ms.

```
virtual grid Descrição_sem_latência {
    nodes (Cluster_HPL) {
        function computing, storage
        size (16, 20)
        node (Nodes_Cluster_HPL) {
            function node
            parameters memory_ram (1GB), vms_per_node (1)
        }
    }
}
```

Figura 4.5: Descrição VXDL para o *benchmark* HPL sem detalhar a topologia de rede.

As duas descrições resultaram na seleção de 16 máquinas virtuais. No primeiro caso, sem a informação da latência foram alocadas 8 máquinas virtuais em cada *cluster* físico. Já na segunda descrição, alocaram-se as 16 máquinas virtuais em um mesmo *cluster* físico, de acordo com a latência especificada.

Foram executadas sobre essas infra-estruturas virtuais as duas configurações do *benchmark* HPL, representadas por HPL 3000 e HPL 8000. Na tabela 4.4 e na figura 4.7 apresentam-se os resultados dessas execuções. Observa-se um tempo de execução 14 e 8 vezes maior para as configurações HPL 3000 e HPL 8000, quando não informado a latência desejada. Este fato deve-se a latência no canal de comunicação que interliga os *sites*.

```
virtual grid Descrição_com_latência {
    nodes (Cluster_HPL) {
        function computing, storage
        size (16, 20)
        node (Nodes_Cluster_HPL) {
            function node
            parameters memory_ram (1GB), vms_per_node (1)
        }
    }
}

virtual topology Descrição_com_latência {
    link (Intra_Cluster) {
        latency (0.155ms),
        between [(Nodes_Cluster_HPL, Nodes_Cluster_HPL)]
    }
}
```

Figura 4.6: Descrição para o *benchmark* HPL especificando latência máxima de 0.155ms.

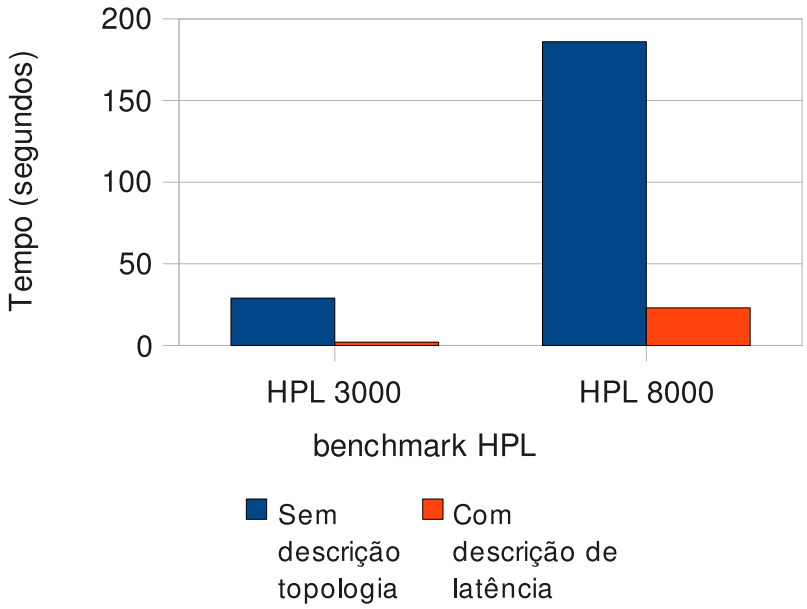


Figura 4.7: Resultados da execução do *benchmark* HPL com descrição de latências.

Tabela 4.4: Resultados da execução do *benchmark* HPL com descrição de latências.

Aplicação	Sem descrição da topologia	Latência de 0.155ms
HPL 3000	29s	2s
HPL 8000	186s	23s

4.4.4.2 Benchmark *NAS*

Para execução do teste de latência com o *benchmark* NAS, desenvolveram-se quatro descrições VXDL, identificadas por: *Descrição sem topologia (1 MV/computador), Latência de 0.100ms - 1 MV/computador, Latência de 0.100ms - 2 MV/computador* e *Latência de 10ms - 2 MV/computador*.

- *Descrição sem topologia - 1 MV/computador*:
  Esta descrição (figura 4.8) solicita um conjunto de recursos com no mínimo 1GB de memória RAM, e a alocação de 1 máquina virtual em cada computador físico.

- *Latência de 0.100ms - 1 MV/computador*:
  Neste caso (figura 4.9) solicita-se pelo mesmo conjunto de recursos da descrição anterior, adicionando uma especificação de latência máxima de 0.100ms.

- *Latência de 0.100ms - 2 MV/computador*:
  Nesta descrição (figura 4.10) é mantido os parâmetros de rede (latência máxima de 0.100ms) da descrição anterior, mas é alterado o número de máquinas virtuais alocados em cada computador.

- *Latência de 10ms - 2 MV/computador*:
  Por fim, esta descrição (figura 4.11) solicita por uma latência máxima de 10ms entre os recursos, e solicita a alocação de 2 máquinas virtuais em cada computador físico.

A figura 4.12 descreve as infra-estruturas virtuais resultantes das quatro descrições. Em todos os casos, são selecionadas 16 máquinas virtuais, necessárias para a execução do *benchmark* NAS. Quando solicitada uma infra-estrutura sem a descrição da latência máxima (1 MV/computador), as máquinas virtuais são distribuídas entre dois *clusters* físicos, alocados em *sites* distintos. Já a solicitação de uma composição com latência máxima de 10ms, resulta no mesmo cenário, mas atende-se a especificação de alocar 2 MV/computador.

```
virtual grid Descrição_sem_topologia {
    nodes (Cluster_NAS) {
        function computing
        size (16, 20)
        node (Nodes_Cluster_NAS) {
            function node
            parameters memory_ram (1GB), vms_per_node (1)
        }
    }
}
```

Figura 4.8: Descrição VXDL para o *benchmark* NAS sem especificação de topologia. Alocando 1 MV/computador.

```
virtual grid Latencia_100_1MV {
    nodes (Cluster_NAS) {
        function computing
        size (16, 20)
        node (Nodes_Cluster_NAS) {
            function node
            parameters memory_ram (1GB), vms_per_node (1)
        }
    }
}

virtual topology grid Latencia_100_1MV {
    link (Intra_Cluster) {
        latency (0.100ms),
        between [(Nodes_Cluster_NAS, Nodes_Cluster_NAS)]
    }
}
```

Figura 4.9: Descrição VXDL para o *benchmark* NAS com latência de 0.100ms. Alocando 1 MV/computador.

```
virtual grid grid Latencia_100_2MV {
    nodes (Cluster_NAS) {
        function computing
        size (16, 20)
        node (Nodes_Cluster_NAS) {
            function node
            parameters memory_ram (1GB), vms_per_node (2)
        }
    }
}

virtual topology grid Latencia_100_2MV {
    link (Intra_Cluster) {
        latency (0.100ms),
        between [(Nodes_Cluster_NAS, Nodes_Cluster_NAS)]
    }
}
```

Figura 4.10: Descrição VXDL para o *benchmark* NAS com latência de 0.100ms. Alocando 2 MV/computador.

```
virtual grid grid Latencia_10_2MV {
    nodes (Cluster_NAS) {
        function computing
        size (16, 20)
        node (Nodes_Cluster_NAS) {
            function node
            parameters memory_ram (1GB), cpu_frequency(1GHz),
                       vms_per_node (2)
        }
    }
}

virtual topology grid Latencia_10_2MV {
    link (Intra_Cluster) {
        latency (10ms),
        between [(Nodes_Cluster_NAS, Nodes_Cluster_NAS)]
    }
}
```

Figura 4.11: Descrição VXDL para o *benchmark* NAS com latência de 10ms. Alocando 2 MV/computador.

As descrições que solicitam por uma latência máxima de 0.100ms resultam na alocação das máquinas virtuais em um mesmo *cluster* físico, diferenciando apenas a organização, de acordo com a solicitação de alocar 1 ou 2 máquinas virtuais por computadores físicos.

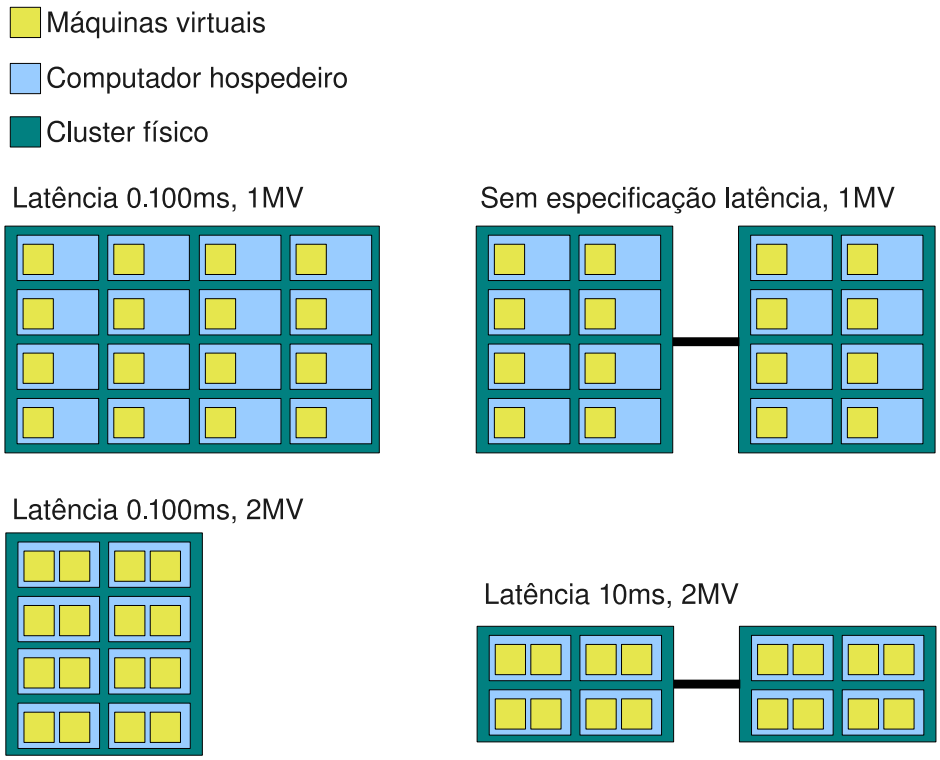


Figura 4.12: Representação das infra-estruturas resultantes das diferentes especificações de latência.

A tabela 4.5 e a figura 4.13 apresentam o resultado da execução das aplicações do *benchmark* NAS nas quatro infra-estruturas virtuais. É possível observar que as descrições que informam uma latência máxima de 0.100ms, e conseqüentemente resultaram na alocação em um mesmo *cluster* físico, apresentam um tempo menor de execução. Comparando-se os tempos de execução das descrições *Descrição sem topologia* e *Latência de 0.100ms, 1MV/computador* observa-se um tempo total de execução para as aplicações cg, is, lu e mg aproximadamente 6 vezes, 26 vezes, 3 vezes e 7 vezes menor, respectivamente. Quanto a informação do número de máquinas virtuais alocadas em cada computador físico, para algumas aplicações resultou em um *overhead*, mas para as aplicações cg e is, executando em dois *clusters*, resultou em um ganho de desempenho.

Tabela 4.5: Resultados da execução do *benchmark* NAS com descrições de latências.

Aplicação	Sem descrição	Latência 0.100ms	Latência 0.100ms	Latência 10ms
	1 MV/comp	1 MV/comp	2 MV/comp	2 MV/comp
cg	893s	137s	220s	860s
is	182s	7s	11s	171s
lu	191s	63s	93s	209s
mg	36s	5s	13s	38s

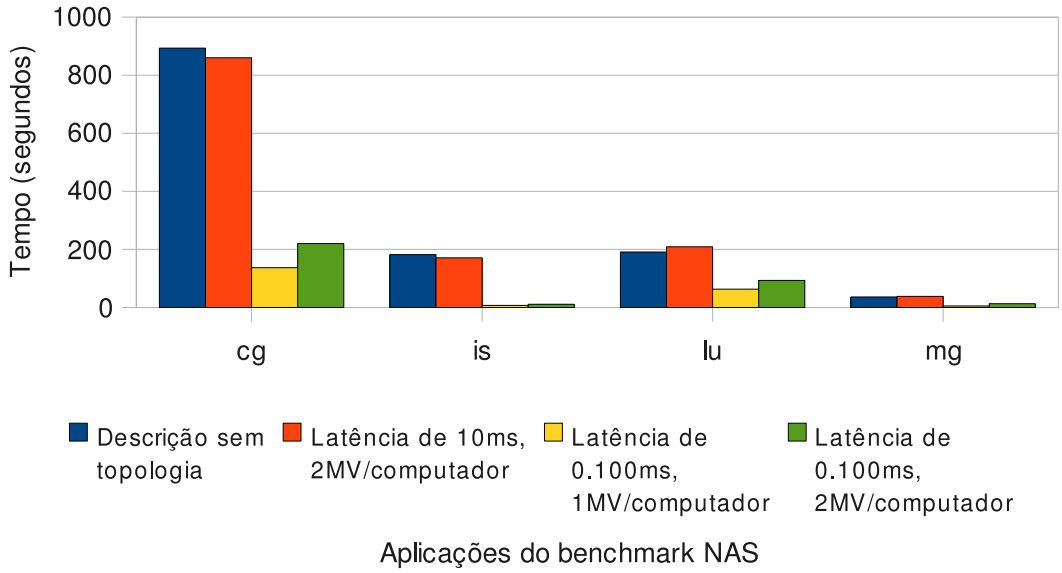


Figura 4.13: Resultados da execução do *benchmark* NAS com descrições de latência.

4.4.5 Descrições de Topologias de Rede: Larguras de Banda

Para realização dos testes envolvendo largura de banda e utilização de roteadores virtuais, foi desenvolvida a descrição VXDL apresentada na figura 4.15, que solicita por uma infra-estrutura virtual com a forma mostrada na figura 4.14. Solicita-se por uma infra-estrutura composta por 2 *clusters* virtuais, cada um contendo 8 máquinas virtuais (representados em 4.15 por *Cluster1* e *Cluster2*). Interligando estes *clusters*, solicita-se pela utilização de 2 roteadores virtuais (representados em 4.15 por *Router1* e *Router2*). Para elaboração deste experimento, alterou-se os limites de largura de banda entre os roteadores virtuais em 1Gb/s, 100Mb/s e 10Mb/s (representado em 4.15 pelo canal *Between_routers*). Para o controle de largura de banda entre os roteadores, utilizou-se a ferramenta *Traffic Control (tc)*, realizando-se configurações utilizando *tc qdisc htb*.

Em alguns momentos, para complemento da utilização do canal foi gerado *cross-traffic* utilizando a ferramenta *iperf*, visando representar o compartilhamento de um mesmo

canal real entre diferentes canais virtuais. Estes cenários são identificados nos resultados do *benchmark* NAS pela legenda *"cross-traffic"*.

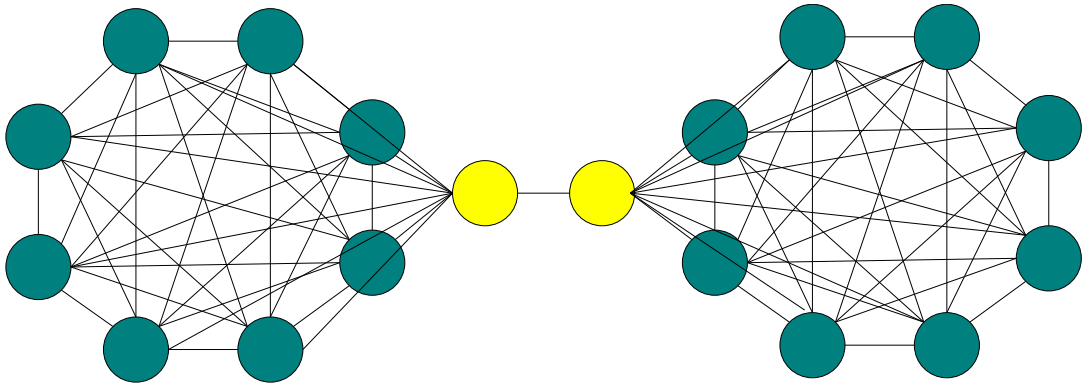


Figura 4.14: Infra-estrutura utilizada para testes com diferentes larguras de banda.

4.4.5.1 Benchmark HPL

Os resultados da execução do *benchmark* HPL nas infra-estruturas virtuais, variando a configuração de largura de banda entre os roteadores são apresentados na tabela 4.6 e na figura 4.16. Quando comparado o canal de comunicação com largura de banda de 10Mb/s, com a execução em utilizando 1Gb/s, observa-se um aumento no tempo de execução de aproximadamente 16 e 12 vezes para as configurações de matrizes 3000 e 8000, respectivamente. Já a proximidade no tempo de execução utilizando a largura de banda de 100Mb/s, quando comparado com o canal de 1Gb/s ressalta a possibilidade de compartilhamento de um canal físico entre múltiplos canais virtuais.

Tabela 4.6: Resultados da execução do *benchmark* HPL com diferentes larguras de banda.

Aplicação	1Gb/s	100Mb/s	10Mb/s
HPL 3000	3s	4s	48s
HPL 8000	26s	29s	325s

4.4.5.2 Benchmark NAS

Para analisar os resultados da execução do *benchmark* NAS sobre as infra-estruturas desenvolvidas (tabela 4.7 e 4.17) tomamos como base os tempos obtidos com a execução no canal limitado em 1Gb/s. Comparando estes resultados com o tempo de execução nos canais limitados em 100Mb/s com *cross-traffic* de 900Mb/s, um aumento no tempo total de execução das aplicações cg, is, lu e mg de 59%, 400%, 90% e 33%, respectivamente.

```
virtual grid Teste start 15/05/2008 14:00:00 for 48:00:00 {
    group (Cluster1) {
        function computing
        size (8)
        anchor capricorne.sophia.grid5000.fr
        resource (Nodes_Cluster1) {
            parameters memory_ram (512MB)
            softwares debian, mpi
        }
    }
    resource (Router1) {
        function router (ports 9)
        anchor capricorne.sophia.grid5000.fr
    }
    resource (Router2) {
        function router (ports 9)
        anchor capricorne.sophia.grid5000.fr
    }
    group (Cluster2) {
        function computing
        size (8)
        anchor capricorne.sophia.grid5000.fr
        resource (Nodes_Cluster2) {
            parameters memory_ram (512MB)
            softwares debian, mpi
        }
    }
}

virtual network topology Test {
    link (Intra_Cluster) {
        latency (0.200ms), bandwidth (1Gbps),
        between [(Nodes_Cluster1, Nodes_Cluster1),
            (Nodes_Cluster2, Nodes_Cluster2)]
    }
    link (Nodes_to_Routers) {
        bandwidth (1Gbps),
        between [(Nodes_Cluster1, Router1), (Nodes_Cluster2, Router2)]
    }
    link (Between_Routers) {
        bandwidth (100Mbps),
        between [(Router1 port 1, Router2 port 1)]
    }
}

virtual timeline Test {
    time0 = start (Cluster1, Cluster2, Router1, Router2,
            Intra_Cluster, Nodes_to_Routers, Between_Routers)
}
```

Figura 4.15: Descrição VXDL para testes com diferentes larguras de banda.

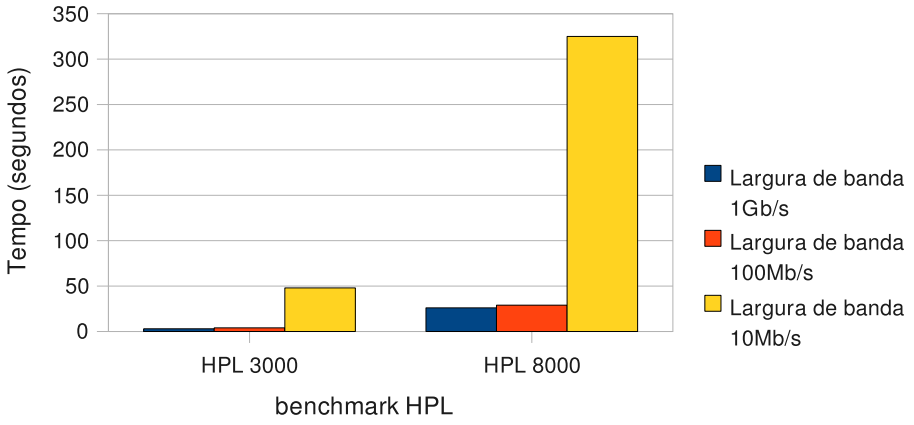


Figura 4.16: Resultados da execução do *benchmark* HPL com diferentes larguras de banda.

Já a comparação envolvendo a limitação da largura de banda em 100Mb/s sem *cross-traffic* e 100Mb/s com *cross-traffic* de 900Mb/s, representa o impacto com o compartilhamento de um canal de comunicação, tornando o tempo total de execução 1.60, 1.41, 1.79 e 1.90 vezes maior para as aplicações cg, is, lu e mg, respectivamente. A análise desses resultados ajuda a observar a possibilidade de compartilhamento de canais de comunicação, mas ressalta a importância da existência de ferramentas para atendar as descrições de usuários, garantindo a utilização dos recursos solicitados.

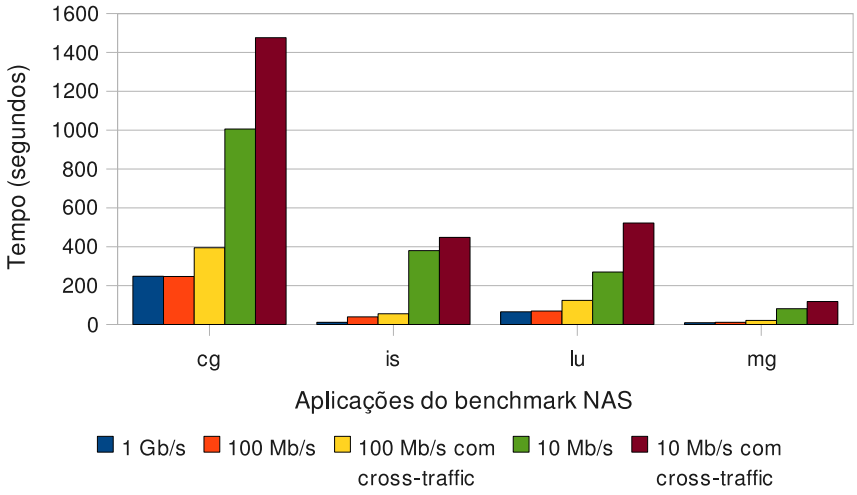


Figura 4.17: Resultados da execução do *benchmark* NAS com diferentes larguras de banda.

Tabela 4.7: Resultados da execução do *benchmark* NAS com diferentes larguras de banda.

NAS	1Gb/s	100Mb/s	100Mb/s	10Mb/s	10Mb/s
cross-traffic	0	0	900Mb/s	0	990Mb/s
cg	248s	247s	395s	1006s	1476s
is	11s	39s	55s	380s	448s
lu	65s	69s	124s	270s	522s
mg	9s	11s	21s	81s	118s

4.5 Conclusão da Avaliação

A avaliação da solução proposta, apresentada neste capítulo, ressaltou *a capacidade descritiva de VXDL e sua facilidade de utilização*, quando comparada com outras linguagens existentes. Um maior poder de expressividade, permite que o usuário descreva as necessidades de suas aplicações de uma forma mais objetiva, detalhando parâmetros para os componentes definidos. Nas tabelas comparativas apresentadas, destacam-se as inovações propostas em VXDL: *descrição de roteadores virtuais, número de máquinas virtuais alocadas em um computador, localização base para definição da infra-estrutura, e definição de um cronograma de execução*.

Além disso, a avaliação realizada com a execução de *benchmarks* em infra-estruturas virtuais, ressalta a importância de informar parâmetros referentes a configuração dos canais de comunicação, já que sua variação pode ser interpretada de diferentes maneiras pelos *middlewares*. Analisando os tempos de execução dos *benchmarks*, observa-se que a composição da infra-estrutura virtual (localização e configuração dos componentes) influencia diretamente no tempo de execução das aplicações, destacando-se a importância de parametrizar recursos computacionais e canais de comunicação. Analisando a utilização de roteadores virtuais, observa-se a possibilidade de definição do formato da infra-estrutura, permitindo a representação de um ambiente real em uma infra-estrutura virtual.

5 CONCLUSÃO

A aplicação das tecnologias de virtualização de recursos para composição de grades virtuais é um tópico recorrente em diversas pesquisas. Projetos atuais exploram sua aplicação para virtualizar recursos computacionais e redes de interconexões, oferecendo aos usuários a representação de um conjunto único de computadores interligados, abstraindo a localidade geográfica dos componentes. Em uma grade virtualizada, ferramentas e soluções devem ser implementadas para auxiliar *middlewares* e usuários na composição das infra-estruturas virtuais. Usuários devem ser capazes de detalhar e configurar a plataforma de execução de acordo com as necessidades de suas aplicações.

Dentre as soluções utilizadas em grades convencionais, destaca-se a utilização de linguagens para descrição de recursos, que permitem o detalhamento dos componentes que devem ser reservados. As linguagens para descrição de recursos devem possuir atributos capazes de atender as peculiaridades de especificação de contextos virtualizados, principalmente a possibilidade de compartilhamento de um mesmo recurso físico (computacional e de rede) entre diversas infra-estruturas virtuais.

Este trabalho apresentou o desenvolvimento de uma *linguagem para descrição de interconexões e recursos em grades virtuais (VXDL)*, desenvolvida de acordo com os pré-requisitos dos projetos HIPCAL [4] e CARRIOCAS [3]. O desenvolvimento de VXDL foi realizado durante a participação em um estágio "sanduíche"no grupo RESO (*Optimized protocols and software for high performance networks*), do laboratório LIP (*Laboratoire de l'Informatique du Parallélisme*), na *École Normale Supérieure* de Lyon.

As principais inovações apresentadas em VXDL permitem que usuários e *middlewares* descrevam todos os componentes desejáveis em uma infra-estrutura virtual, detalhando a classificação e os parâmetros de *clusters* e recursos individuais. Além disso, permite-se em VXDL a descrição da topologia de rede desejável, onde usuários podem

informar a configuração de todos os canais de comunicação, inclusive detalhando a forma da infra-estrutura através da utilização de roteadores virtuais.

Atendendo as especificações de ambientes virtualizados, VXDL permite a descrição do número de máquinas virtuais que devem ser alocadas sobre um mesmo recurso físico, e oferece um atributo para definição de uma localização base, utilizada como marco de partida para composição de uma infra-estrutura. Explora-se também a definição de cronogramas de execução, onde usuários definem qual o período de utilização dos recursos, dentro do tempo total da reserva. Este nível de detalhamento pode auxiliar *middlewares* na realização de um compartilhamento e escalonamento eficiente entre as infra-estruturas alocadas. Embora o desenvolvimento de VXDL tenha sido guiado pelos requisitos do *middleware* HIPerNET, nada impede sua utilização em outras ferramentas, capazes de interpretar descrições escritas utilizando a linguagem.

Para avaliar as funcionalidades de VXDL, foi desenvolvida uma metodologia que abordou a comparação de descrições com outras linguagens existentes, e a execução de *benchmarks* sobre infra-estruturas definidas por diferentes descrições VXDL. Esta análise identificou a facilidade de utilização de VXDL e a disponibilidade de recursos inovadores para essa descrição. VXDL compreende diversos atributos que permitem a representação de uma infra-estrutura, apresentando originalmente recursos para identificação de cronogramas de execução e roteadores virtuais. Além disso, através da execução dos *benchmarks*, visualizou-se o impacto causado no tempo total de execução das aplicações, quando executadas em diferentes infra-estruturas virtuais, compostas variando parâmetros da composição de rede (latência e largura de banda) e número de máquinas virtuais alocadas por computador físico.

Partindo da linguagem desenvolvida neste trabalho, uma possível continuidade nesta linha de pesquisa corresponde ao processo de interpretação das descrições VXDL, realizado pelos *middlewares*, para selecionar e reservar os recursos solicitados. Utilizando VXDL, usuários podem solicitar por infra-estruturas complexas, compostas por diversos componentes com diferentes parâmetros. A seleção e o compartilhamento destes recursos em grades virtuais é uma tarefa complexa, devido a possibilidade de compartilhamento de um mesmo recurso em diferentes infra-estruturas virtuais.

REFERÊNCIAS

[1] Condor - High Throughput Computing, 2000. http://www.cs.wisc.edu/condor/.

[2] Grid500:Home, 2005. https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[3] Calcul Réparti sur Réseau Internet Optique à Capacité Surmultiplée (CARRIOCAS), 2007. http://www.carriocas.org/.

[4] HIPCAL project, 2007. http://hipcal.lri.fr/wakka.php?wiki=PagePrincipale.

[5] Phosphorus project - lambda user controlled infrastructure for european research, 2007. http://www.ist-phosphorus.eu/.

[6] *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), 14-17 May 2007, Rio de Janeiro, Brazil*. IEEE Computer Society, 2007.

[7] UCLP user-controlled lightpaths, 2007. http://www.canarie.ca/canet4/uclp/.

[8] *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), 19-22 May 2008, Lyon, France*. IEEE Computer Society, 2008.

[9] A. Aho and J. D. Ullman. *Compiladores: conceitos, técnicas e implementaćão.* Makron Books, 1993.

[10] Sergio Andreozzi, Stephen Burke, Flavia Donno, Laurence Field, Steve Fisher ans Jens Jensen Ans Balazs Konya, Maarten Litmaath, Marco Manbelli, Jennifer Schopf, Matt Viljoen, Antony Wilson, and Riccardo Zappi. GLUE Schema Specification. Technical Report 1.3, 2007.

[11] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon,

V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[12] Benjamin Bardiaux, Aymeric Bernard, Franćois-Regis Chalaoux, Dinu Gherman, Michael Habeck, Jens Linge, Thérese Malliavin, Michael Nilges, Sean O'Donoghue, and Wolfgang Rieping. Aria - ambiguous restraints for iterative assignment, 1995.

[13] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM symposium on Operating systems principles (SOSP)*, pages 164–177, Bolton Landing, NY, USA, October 2003. ACM.

[14] Biobase. Proteome: Protein interaction biological databases, 1999.

[15] T. Boku, T. Harada, T. Sone, H. Nakamura, and K. Nakazawa. Inspire: A general purpose network simulator generating system for massively parallel processors, 1995.

[16] Erol Bozak, Franco Travostino, and Wolfgang Reichert. GridVirt-WG grid and virtualization working group, 2007.

[17] Roberto Canonico, Donato Emma, and Giorgio Ventre. An xml based network simulation description language, 2001.

[18] Samuel Chang and Peter Strazdins. A survey of how virtual machine and intelligent runtime environments can support cluster computing. Technical Report TR-CS-07-01, Australian National University, 2007.

[19] Andrew Chien, Henri Casanova, Yang suk Kee, and Richard Huang. The Virtual Grid Description Language: vgDL. Technical Report TR0.95, VGrADS Project, 2004.

[20] Stephen Childs, Brian Coghlan, David O'Callaghan, and Geoff Quigley and. A single-computer grid gateway using virtual machines. In *Proc.19th International Conference on Advanced Information Networking and Applications (AINA'05)*, 2005.

[21] Christopher Clark, Keir Fraser, Steven Hand, Jakob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, USA, May 2005. Usenix.

[22] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing, 2001.

[23] Mathias Dalheimer, Franz-Josef Pfreundt, and Peter Merz. Formal verification of a grid resource allocation protocol. In *CCGRID* [8], pages 332–339.

[24] distributed.net. distributed.net: Node Zero, 1997. http://www.distributed.net/.

[25] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[26] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[27] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[28] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–??, 2001.

[29] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[30] Robert Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.

[31] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[32] Ludovic Hablot, Olivier Gluck, Jean-Christophe, and Pascale Vicat-Blanc Primet. Etude d'implémentations mpi pour une grille de calcul. *RenPar'18 / SympA'2008 / CFSE'6, Fribourg, Suisse*, 11 2008.

[33] Michael Heidt, Tim Dörnemann, Kay Dörnemann, and Bernd Freisleben. Omnivore: Integration of grid meta-scheduling and peer-to-peer technologies. In *CCGRID* [8], pages 316–323.

[34] Vincent C. Hu, David F. Ferraiolo, and Karen Scarfone. Access control policy combinations for the grid using the policy machine. In *CCGRID* [6], pages 225–232.

[35] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar Panda. A case for high performance computing with virtual machines. *The 20th ACM International Conference on Supercomputing*, 2006.

[36] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Washington, DC, USA, 2002. IEEE Computer Society.

[37] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, 2004.

[38] R. Isaacs and I. Leslie. Support for Resource-Assured and Dynamic Virtual Private Networks, 2001.

[39] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, 1991.

[40] java.net The source for Java Technology Collaboration. javacc - java compiler compiler - the java parser generator - version 4.0, 2007.

[41] Key Kennedy, Francine Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, S. Lennart Johnsson, Carl Kesselman, John Mellor-Crummey, Daniel Reed, Linda Torczon, and Richar Wolski. The VGrADS Project, 2003. http://vgrads.rice.edu/.

[42] A. Kertész, I. Rodero, and F. Guim. BPDL: A Data Model for Grid Resource Broker Capabilities. Technical Report TR-0074, Institute on Resource Management and Scheduling - CoreGRID, 2007.

[43] Klaus Krauter, Rajkumar Buyya, and Mathucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2):135–164, 2002.

[44] Julien Laganier and Pascale Vicat-Blanc Primet. Hipernet: fully distributed security for grid environments. Technical Report RR-....-2005-..., INRIA/ENS-LIP, April 2005.

[45] Chuang Liu and Ian Foster. A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science - University of Chicago, 2003.

[46] Ross McIlroy and Joe Sventek. Resource virtualisation of network routers. In *Workshop on High Performance Switching and Routing (HPSR 06)*, June 2006.

[47] L. P. Meissner. FORTRAN 77. 12(1):93–94, January 1977.

[48] Ewan Mellor, Richard Sharp, David Scott, and Jon Harrop. Xen management API draft - revision 0.4.3. Technical Report 25-08-06, XenSource, Inc., 2006.

[49] R. Moskowitz and P. Nikander. Host Identity Protocol Architecture, 2004. http://www.ietf.org/html.charters/hip-charter.html.

[50] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery, 2005.

[51] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/, 2004.

[52] Reservoir Project. Reservoir: Resources and services virtualization without barriers, 2008.

[53] Benjamin Quetier, Vincent Neri, and Franck Cappello. Scalability comparison of 4 host virtualization tools. Technical Report 1433, INRIA/LRI, Universite Paris-Sud, 2006.

[54] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *HPDC*, pages 140–, 1998.

[55] Rajiv Ranjan, Lipo Chan, Aaron Harwood, Shanika Karunasekera, and Rajkumar Buyya. Decentralised resource discovery service for large scale federated grids. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 379–387, Washington, DC, USA, 2007. IEEE Computer Society.

[56] D. M. Ritchie, B. W. Kernighan, and M. E. Lesk. The C programming language. Comp. Sci. Tech. Rep. No. 31, Bell Laboratories, Murray Hill, New Jersey, October 1975. 31 Superseded by B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, N.J., 1988.

[57] Paul Ruth, P. McGachey, and Dongyan Xu. Viocluster: Virtualization for dynamic computational domains. In *CLUSTER*, pages 1–10. IEEE, 2005.

[58] Cristina Schmidt and Manish Parashar. Flexible information discovery in decentralized distributed systems. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 226, Washington, DC, USA, 2003. IEEE Computer Society.

[59] SETI. SETI@home: Search for Extraterrestrial Intelligence at home, 1999. http://setiathome.ssl.berkeley.edu/.

[60] Richard O. Sinnott, David W. Chadwick, Thomas Doherty, David Martin, Anthony Stell, Gordon Stewart, Linying Su, and J. P. Watt. Advanced security for virtual organizations: The pros and cons of centralized vs decentralized security models. In *CCGRID* [8], pages 106–113.

[61] Borja Sotomayor, Kate Keahey, and Ian Foster. Overhead matters: A model for virtual resource management. In *VTDC '06: Proceedings of the 2nd International*

Workshop on Virtualization Technology in Distributed Computing, page 5, Washington, DC, USA, 2006. IEEE Computer Society.

[62] J. Sugerman, G. Venkitachalam, and B-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. 2001 Usenix Annual Technical Conference*, pages 1–14. Usenix Assoc., 2001.

[63] Martin Swany and Paola Grosso. (NML-WG) network mark-up language working group, 2005.

[64] Atsuko Takefusa, Michiaki Hayashi, Naohide Nagatsu, Hidemoto Nakada, Tomohiro Kudoh, Takahiro Miyamoto, Tomohiro Otani, Hideaki Tanaka, Masatoshi Suzuki, Yasunori Sameshima, Wataru Imajuku, Masahiko Jinno, Yoshihiro Takigawa, Shuichi Okamoto, Yoshio Tanaka, and Satoshi Sekiguchi. G-lambda: coordination of a grid scheduler and lambda path service over gmpls. *Future Gener. Comput. Syst.*, 22(8):868–875, 2006.

[65] Masaki Tatezono, Naoya Maruyama, and Satoshi Matsuoka. Making wide-are, multi-site mpi feasible using xen VM. In *Proceedings of 2006 ISPA workshop on Xen in High-Performance Clusters and Grid Computing Environments (XHPC 06)*, Sorrento, Italy, December 2006.

[66] Weikuan Yu and Jeffrey S. Vetter. Xen-based hpc: A parallel i/o perspective. In *CCGRID* [8], pages 154–161.

[67] Yang Zhang, Charles Koelbel, and Ken Kennedy. Relative performance of scheduling algorithms in grid environments. In *CCGRID* [6], pages 521–528.

APÊNDICE A - DESCRIÇÃO DE UMA INFRA-ESTRUTURA REAL

Neste apêndice apresenta-se a descrição VXDL que descreve os componentes de infra-estrutura computacional Grid5000 [2]. As figuras 5.1, 5.2, 5.3, 5.4 e 5.5 apresentam a descrição escrita em VXDL para os aproximadamente 3500 componentes da grade. Particularmente na figura 5.5 apresenta-se a descrição das interconexões de rede.

```
virtual grid Grid5000 {
    group (Site_Bordeaux) {
        group (Cluster_Bordemer) {
            size (max 48)
            resource (Nodes_Cluster_Bordemer) {
                function computing
                parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Bordeplage) {
            size (max 51)
            resource (Nodes_Cluster_Bordeplage) {
                function computing
                parameters cpu_frequency (max 3GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 70GB)
            }
        }
        group (Cluster_Bordereau) {
            size (max 93)
            resource (Nodes_Cluster_Bordereaux) {
                function computing
                parameters cpu_frequency (max 2.6GHz), cpu_processors (max 2),
                           memory_ram (max 4GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Borderline) {
            size (max 10)
            resource (Nodes_Cluster_Borderline) {
                function computing
                parameters cpu_frequency (max 2.6GHz), cpu_processors (max 4),
                           memory_ram (max 32GB), hd_size (max 600GB)
            }
        }
    } //end Site_Bordeaux
    group (Site_Grenoble) {
        group (Cluster_IDPOT) {
            size (max 32)
            resource (Nodes_Cluster_IDPOT) {
                function computing
                parameters cpu_frequency (max 2.4GHz), cpu_processors (max 2),
                           memory_ram (max 1.5GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Icluster2) {
            size (max 103)
            resource (Nodes_Cluster_Icluster2) {
                function computing
                parameters cpu_frequency (max 900MHz), cpu_processors (max 2),
                           memory_ram (max 3GB), hd_size (max 72GB)
            }

        }
    } //end Site_Grenoble

    group (Site_Lille) {
        group (Cluster_Chuque) {
            size (max 53)
            resource (Nodes_Cluster_Chuque) {
                function computing
                parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                           memory_ram (max 4GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Chti) {
            size (max 20)
            resource (Nodes_Cluster_Chti) {
                function computing
                parameters cpu_frequency (max 2.6GHz), cpu_processors (max 2),
                           memory_ram (max 4GB), hd_size (max 80GB)
            }
        }
```

Figura 5.1: Descrição VXDL de uma infra-estrutura real: Grid5000 - A.

```
    group (Cluster_Chicon) {
        size (max 26)
        resource (Nodes_Cluster_Chicon) {
            parameters cpu_frequency (max 2.6GHz), cpu_processors (max 2),
                       memory_ram (max 4GB), hd_size (max 80GB)
        }
    }
    group (Cluster_ChinqChint) {
        size (max 46)
        resource (Nodes_Cluster_ChinqChint) {
            function computing
            parameters cpu_frequency (max 2.83GHz), cpu_processors (max 2),
                       memory_ram (max 8GB), hd_size (max 250GB)
        }
    }
} //end Site_Lille

group (Site_Lyon) {
    group (Cluster_Capricorne) {
        size (max 56)
        resource (Nodes_Cluster_Capricorne) {
            function computing
            parameters cpu_frequency (max 2.0GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 80GB)
        }
    }
    group (Cluster_Sagittaire) {
        size (max 70)
        resource (Nodes_Cluster_Sagittaire) {
            function computing
            parameters cpu_frequency (max 2.4GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 73GB)
        }
    }
    group (Cluster_Scorpion) {
        size (max 4)
        resource (Nodes_Cluster_Scorpion) {
            function computing
            parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                       memory_ram (max 4GB), hd_size (max 72.8GB)
        }
    }
} //end Site_Lyon

group (Site_Nancy) {
    group (Cluster_Grillon) {
        group (Cluster_Grillon_Compute) {
            size (max 47)
            function computing
            resource (Nodes_Cluster_Grillon) {
                parameters cpu_frequency (max 2.0GHz), memory_ram (max 2GB),
                    hd_size (max 80GB)
            }
        }
        resource (Service_Node_Grillon) {
            parameters cpu_frequency (max 2.4GHz), memory_ram (max 4GB),
                hd_size (max 300GB)
        }
        resource (Storage_Node_Grillon) {
            function storage
            parameters cpu_frequency (max 2.2GHz), memory_ram (max 2GB),
                hd_size (max 4.5TB)

        }
    } //end Cluster_Grillon
    group (Cluster_Grelon) {
        group (Cluster_Grelon_Compute) {
            size (max 120)
            function computing
            resource (Nodes_Cluster_Grelon) {
                parameters cpu_frequency (max 1.6GHz), memory_ram (max 2GB),
                    hd_size (max 80GB)
            }
        }
```

Figura 5.2: Descrição VXDL de uma infra-estrutura real: Grid5000 - B.

```
        resource (Service_Node_Grelon) {
            function computing
            parameters cpu_frequency (max 1.6GHz), memory_ram (max 4GB),
                hd_size (max 144GB)
        }
        resource (Storage_Node_Grelon) {
            function storage
            parameters cpu_frequency (max 1.6GHz), memory_ram (max 4GB),
                hd_size (max 784GB)
        }
    } //end Cluster_Grelon
} //end Site_Nancy

group (Site_Orsay) {
    group (Cluster_NetGDX) {
        size (max 30)
        resource (Nodes_Cluster_NetGDX) {
            function computing
            parameters cpu_frequency (max 2.0GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 80GB)
        }
    }
    group (Cluster_GDX_Former) {
        size (max 186)
        resource (Nodes_Cluster_GDX_Former) {
            function computing
            parameters cpu_frequency (max 2.0GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 80GB)
        }
    }
    group (Cluster_GDX) {
        size (max 126)
        resource (Nodes_Cluster_GDX) {
            function computing
            parameters cpu_frequency (max 2.4GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 80GB)
        }
    }
    group (Frontal_Orsay) {
        size (max 4)
        resource (Nodes_Frontal_Orsay) {
            function endpoint
            parameters cpu_frequency (max 2.0GHz), cpu_processors (max 1),
                       memory_ram (max 1GB), hd_size (max 80GB)
        }
    }
    group (Server_Orsay) {
        size (max 2)
        resource (Noder_Server_Orsay) {
            function computing, storage
            parameters cpu_frequency (max 3.0GHz), cpu_processors (max 2),
                       memory_ram (max 2GB), hd_size (max 1TB)
        }
    }
} //end Site_Orsay

group (Site_Rennes) {
    group (Cluster_Paramount) {
        size (max 33)
        resource (Nodes_Cluster_Paramount) {
            function computing
            parameters cpu_frequency (max 2.33GHz), cpu_processors (max 2),
                       memory_ram (max 8GB), hd_size (max 600GB)
        }
    }
    group (Cluster_Paraquad) {
        size (max 64)
        resource (Nodes_Cluster_Paraquad) {
            function computing
            parameters cpu_frequency (max 2.33GHz), cpu_processors (max 2),
                       memory_ram (max 4GB), hd_size (max 160GB)
        }
    }
```

Figura 5.3: Descrição VXDL de uma infra-estrutura real: Grid5000 - C.

```
        group (Cluster_Paravent) {
            size (max 99)
            resource (Nodes_Cluster_Paravent) {
                function computing
                parameters cpu_frequency (max 2.0GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Parasol) {
            size (max 64)
            resource (Nodes_Cluster_Parasol) {
                function computing
                parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 73GB)
            }
        }
    } //end Site_Rennes

    group (Site_Sophia) {
        group (Cluster_Azur) {
            size (max 72)
            resource (Nodes_Cluster_Azur) {
                function computing
                parameters cpu_frequency (max 2.0GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 80GB)
            }
        }
        group (Cluster_Helios) {
            size (max 56)
            resource (Nodes_Cluster_Helios) {
                function computing
                parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                           memory_ram (max 4GB), hd_size (max 146GB)
            }
        }
        group (Cluster_Sol) {
            size (max 50)
            resource (Nodes_Cluster_Sol) {
                function computing
                parameters cpu_frequency (max 2.6GHz), cpu_processors (max 2),
                           memory_ram (max 4GB), hd_size (max 250GB)
            }
        }
        resource (Storage_Sophia) {
            function storage
            parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                       memory_ram (max 4GB), hd_size (max 2TB)
        }
    } //end Site_Sophia

    group (Site_Toulouse) {
        group (Cluster_Violette) {
            size (max 57)
            resource (Nodes_Cluster_Violette) {
                function computing
                parameters cpu_frequency (max 2.2GHz), cpu_processors (max 2),
                           memory_ram (max 2GB), hd_size (max 73GB)
            }
        }
        group (Cluster_Pastel) {
            size (max 80)
            resource (Nodes_Cluster_Pastel) {
                function computing
                parameters cpu_frequency (max 2.6GHz), cpu_processors (max 2),
                           memory_ram (max 8GB), hd_size (max 250GB)
            }
        }
    } //end Site_Toulouse
} //end virtual grid
```

Figura 5.4: Descrição VXDL de uma infra-estrutura real: Grid5000 - D.

```
virtual topology Grid5000_Topology {
    link (Intra_Site) {
        latency (0.100ms, 0.200ms), bandwidth (max 1Gb/s), direction (bi),
        between [(Site_Bordeaux, Site_Bordeaux), (Site_Grenoble, Site_Grenoble),
                 (Site_Lille, Site_Lille), (Site_Lyon, Site_Lyon),
                 (Site_Nancy, Site_Nancy), (Site_Orsay, Site_Orsay),
                 (Site_Rennes, Site_Rennes), (Site_Sophia, Site_Sophia),
                 (Site_Toulouse, Site_Toulouse)]
    }
    link (Site_to_Site_1Gbps) {
        latency (10ms, 15ms), bandwidth (max 1Gb/s), direction (bi),
        between [(Site_Lille, Site_Orsay), (Site_Bordeaux, Site_Lyon)]
    }
    link (Site_to_Site_10Gbps) {
        latency (7ms, 10ms), bandwidth (max 10Gb/s), direction (bi),
        between [(Site_Rennes, Site_Orsay), (Site_Orsay, Site_Nancy),
                 (Site_Orsay, Site_Lyon), (Site_Lyon, Site_Grenoble),
                 (Site_Lyon, Site_Toulouse), (Site_Lyon, Site_Sophia)]
    }
} //end virtual topology
```

Figura 5.5: Descrição VXDL de uma infra-estrutura real: Grid5000 - E.

APÊNDICE B - PUBLICAÇÕES GERADAS A PARTIR DO TRABALHO

Durante a realização deste trabalho, foram desenvolvidos e submetidos dois artigos. O primeiro apresenta a linguagem VXDL e os resultados obtidos durante a etapa de avaliação. Já o segundo aborda o conceito de infra-estruturas virtuais, compostas por recursos computacionais interconectados, destacando a utilização de roteadores virtuais.

1. Título: *VXDL: Virtual Resources and Interconnection Networks Description Language*, submetido e aceito para publicação na conferência *GridNets 2008* (http://gridnets.org/).

2. Título: *Virtual Infrastructures as a service: abstractions, opportunities and issues*, submetido à conferência *ACM CoNEXT 2008* (http://sigcomm.org/co-next2008/) e aguardando avaliação no momento da conclusão desta dissertação.