Luigi Jacometti de Oliveira

Comparação de ferramentas de simulação de grades computacionais

São José do Rio Preto – SP

Novembro / 2007

Luigi Jacometti de Oliveira

Comparação de ferramentas de simulação de grades computacionais

Monografia apresentada à comissão examinadora como parte dos requisitos necessários para a aprovação na disciplina de Estudos Especiais do Programa de Pós-Graduação em Ciência da Computação - Unesp/Ibilce.

Orientador:
Prof. Dr. Aleardo Manacero Jr.

Departamento de Ciências da Computação e Estatística
Instituto de Biociências, Letras e Ciências Exatas
Universidade Estadual Paulista

São José do Rio Preto – SP

Novembro / 2007

Sumário

Lista de Figuras

1 Introdução geral

Avanços nas tecnologias de *hardware* e *software* tornaram possível a execução de aplicações paralelas em grandes conjuntos de recursos distribuídos. Tais avanços, traduzidos na forma de melhorias nas tecnologias de interconexão, permitiram a distribuição aplicações em conjuntos de recursos computacionais crescentes e dispersos: primeiramente dentro de um sistema simples, depois entre nós de um sistema de máquinas massivamente paralelas (MPP) ou em múltiplos *clusters*, e recentemente por meio *grids* computacionais [1].

O estudo de *grids* computacionais, assim como outras áreas da ciência, está baseado em um conjunto de metodologias e ferramentas [2]. A análise do comportamento deste tipo de sistema pode ser feita através de modelos - de simulação ou analíticos - ou por meio da experimentação de um sistema real. Em sistemas distribuídos como os *grids*, muitos parâmetros devem ser considerados, e complexas interações ocorrem, o que torna a sua modelagem analítica impraticável.

Experimentos em plataformas reais, embora resultem em dados mais confiáveis, apresentam uma série de limitações: sua escalabilidade, sua pequena possibilidade de reconfiguração de *software*, sua dificuldade em mimetizar diferentes componentes da infra-estrutura de *hardware* e topologia de rede, sua intrínseca dependência em relação a um conjunto de condições reais, entre outros fatores. Isso faz com que os resultados obtidos por uma plataforma real dificilmente sejam representativos de outras plataformas.

Os simuladores, por sua vez, são ferramentas de alto nível, que permitem abordar comportamentos ou mecanismos específicos de um sistema distribuído e abstraí-los do restante do sistema. Sua grande vantagem é a sua independência da plataforma de execução. Essa vantagem é possível, porque o simulador não executa um sistema distribuído real, mas um

modelo dele.

O uso de simuladores para *grids* computacionais é de especial importância para o estudo de algoritmos de escalonamento de tarefas, como *Workqueue with Replication* (WQR) [3], *Xsufferage* [4], *Dynamic-FPLTF* [3], *Storage Affinity* [5, 6], entre outros. Por meio das ferramentas de simulação, torna-se possível avaliar e comparar o desempenho de diferentes algoritmos em diferentes cenários. Diversas ferramentas foram concebidas com o propósito de investigar estratégias de escalonamento, como SimGrid [7], GridSim [8], GangSim [9], Bricks [10] e OptorSim [11].

Há situações, no entanto, em que o sistema distribuído não pode ser facilmente reduzido a um simulador. Além da dificuldade em modelar complexas interações e comportamentos do sistema, um outro obstáculo ao uso da simulação é o grau de realismo inferior ao dos experimentos reais. Uma possível forma de contornar tais problemas é utilizar emuladores. Os emuladores, assim como os simuladores, mimetizam o comportamento de sistemas reais. Na emulação, entretanto, *softwares* reais executam em máquinas reais, pois o *hardware* de um sistema é simulado em outro sistema. Apesar de superar algumas limitações apresentadas pela simulação, a emulação é pouco escalável, o que é uma grande deficiência quando se considera a modelagem de um sistema distribuído complexo.

O objetivo deste trabalho é, portanto, desenvolver um estudo que permita delimitar as características funcionais das mais representativas ferramentas de simulação de *grids*. Essas características servirão de base para identificar os pontos fortes e as deficiências de cada ferramenta, auxiliando no processo de especificação das funcionalidades que devem estar presentes em um ambiente de simulação completo.

1.1 Organização do estudo

O restante deste estudo encontra-se organizado da seguinte forma. No capítulo 2 são apresentadas as ferramentas de simulação e emulação abordadas e comparadas no estudo. No capítulo 3, as ferramentas previamente apresentadas são comparadas segundo determinados requisitos. Finalmente, no capítulo 4 são feitas as conclusões do trabalho.

2 *Ferramentas de simulação e emulação para* Grids

Neste capítulo serão apresentadas as principais ferramentas de simulação de *grids* existentes. Serão descritos os principais conceitos de cada simulador, bem como detalhes sobre a sua implementação, disponibilidade e aplicações. Além disso, para efeito de comparação, será abordada a ferramenta de emulação de grades MicroGrid.

2.1 SimGrid

A ferramenta SimGrid foi concebida em 1999, num projeto de pós-doutorado desenvolvido por Henri Casanova. A maior motivação para a construção da ferramenta foi a necessidade de se utilizar simulação, ao invés de experimentos reais, no estudo prático de algoritmos de escalonamento centralizados para aplicações científicas paralelas em plataformas computacionais distribuídas e heterogêneas [7].

A versão inicial do SimGrid consistia em uma ferramenta de simulação orientada a eventos. Ela fornecia um conjunto de abstrações e funcionalidades que permitiam construir com relativa facilidade simulações para domínios de aplicações específicos. Por meio de *traces*, ela possibilitava a simulação de flutuações de desempenho arbitrárias, tais como as observadas em sistemas distribuídos reais. A ferramenta basicamente implementava uma interface, ou API escrita em linguagem C, denominada *SG*. Com tal API, era possível especificar a simulação do escalonamento de tarefas em determinados recursos.

Esta primeira versão, no entanto, apresentava uma série de limitações. Não havia como representar ou modelar aspectos como roteamento e escalonamento descentralizado.

Diferentes componentes, portanto, foram adicionados à ferramenta, procurando contornar tais limitações e, ao mesmo tempo, ampliar as suas funcionalidades.

O SimGrid v3, atual versão da ferramenta, provê alguns ambientes de programação, construídos sobre um único núcleo de simulação. Cada ambiente é destinado a um usuário alvo e constitui um paradigma diferente. Na próxima seção, será discutida a arquitetura da atual versão do SimGrid.

2.1.1 Arquitetura do SimGrid

A arquitetura do SimGrid, ilustrada na figura 2.1, é descrita na forma de camadas, cada uma contendo um ou mais componentes.

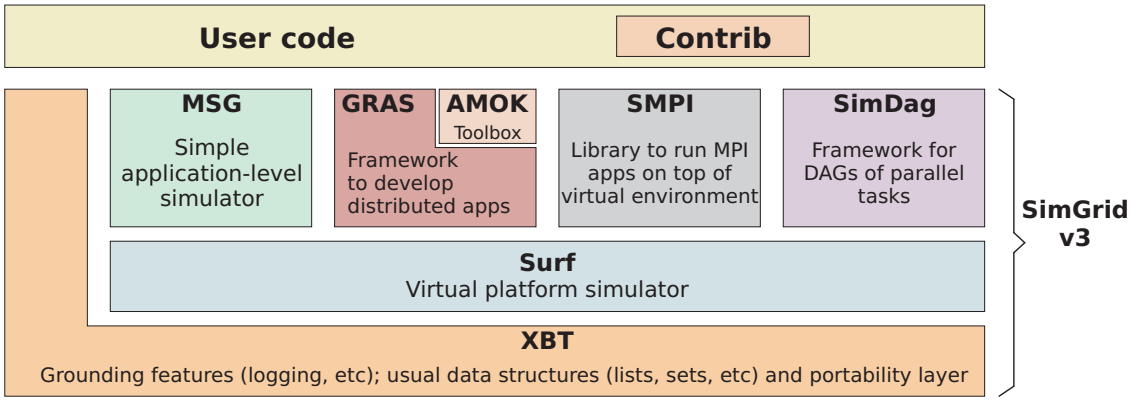


Figura 2.1: Componentes do Simgrid v3 e suas relações

O código da aplicação escrito pelo usuário, utilizando a biblioteca do SimGrid, faz uso de algum dos ambientes de programação situados na primeira camada. O projeto SimGrid disponibiliza também um conjunto de exemplos e aplicações desenvolvidos e compartilhados por usuários. Entre os projetos de contribuição, cita-se o GarSim, voltado para a simulação de escalonamento em sistemas *batch* [12], e o Simbatch, que simula o comportamento de um escalonador de processos *batch* trabalhando em um *cluster* [13].

2.1.2 Componentes do Simgrid

A escolha do ambiente correto dependerá do tipo de aplicação que se deseja avaliar. Nas próximas seções serão detalhados os ambientes de programação do SimGrid, bem como

as situações em que seu uso é indicado.

Ambiente MSG

O MSG foi o primeiro ambiente de programação disponibilizado pelo SimGrid [14] e é o de uso mais difundido. Construído a partir do módulo SG do SimGrid v1, o MSG visa a realização de simulações em termos de *agentes de comunicação*. É usualmente utilizado no estudo de heurísticas para um problema, possibilitando uma comparação entre elas. O realismo da simulação não é objetivo principal deste módulo, e muitos detalhes técnicos da plataforma do *grid* são omitidos.

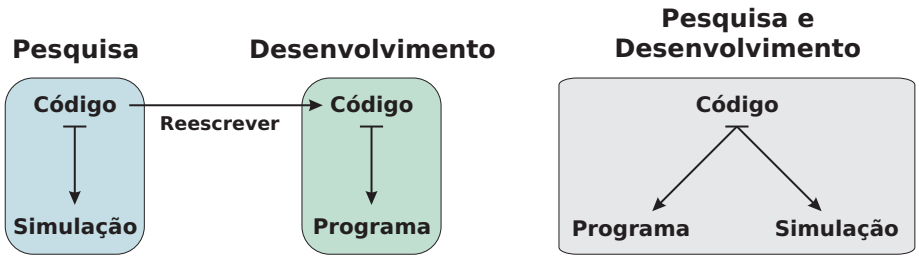
Um *agente* representa uma entidade que toma decisões de escalonamento, executa em uma dada *localidade*, e interage enviando, recebendo, e processando *tarefas*. Uma *localidade* é definida como um recurso computacional situado dentro da topologia do *grid*. Uma *tarefa* é uma atividade, a qual pode ser uma computação ou uma transferência de dados.

A construção de um programa de simulação usando o MSG envolve a codificação de cada *agente* (modelagem da aplicação), criação dos recursos (modelagem da plataforma física), na qual são especificados os *hosts*, *links* de comunicação e uma tabela de roteamento. Em seguida, cria-se e aloca-se os *agentes* nas *localidades* e realiza-se a simulação.

Ambiente GRAS

GRAS (*Grid Reality and Simulation*) [15] é um ambiente concebido como um *framework* que facilita o desenvolvimento de aplicações distribuídas orientadas a eventos. Conforme mostrado na figura 2.2, o GRAS possibilita a execução da aplicação juntamente com o simulador e em uma plataforma distribuída real (utilizando duas versões distintas de sua API). Com isso, procura-se explorar as vantagens dos dois métodos de análise: desenvolvedores beneficiam-se da facilidade de uso e de controle oferecidos pelo simulador durante os estágios do ciclo de desenvolvimento, enquanto um código para plataformas reais é automaticamente produzido. Este ambiente faz uso da API fornecida pelo MSG (para implementar a simulação) e de *sockets* (para implementar a aplicação real).

Acima da API do GRAS, há uma *toolkit* chamada AMOK (*Advanced Metacomputing*



(a) Simuladores Clássicos (b) GRAS

Figura 2.2: Comparação entre as abordagens do GRAS e dos simuladores convencionais

Overlay Kit) que implementa em alto nível diversos serviços necessários a várias aplicações distribuídas.

Ambiente SMPI

O ambiente SMPI, ainda em fase de desenvolvimento, é um *framework* que permitirá executar aplicações MPI não alteradas, tanto em modo de simulação, quanto em modo de aplicação real (atuando como um GRAS para MPI).

Ambiente SimDag

Este ambiente é dedicado à simulação de aplicações paralelas, por meio do modelo DAG (*Direct Acyclic Graphs*). Com este modelo é possível especificar relações de dependência entre tarefas de um programa paralelo. Em outras palavras, cada programa no código da simulação será descrito por um conjunto de tarefas paralelas inter-dependentes.

Camada SURF (núcleo do simulador)

Logo abaixo da camada formada pelos ambientes de programação, há o núcleo de todos os simuladores implementados pelo SimGrid, denominado SURF. Este núcleo provê um conjunto de funcionalidades para simular uma plataforma virtual. O SURF, introduzido na versão 3, substitui o SG, sendo mais mais rápido do que o módulo anterior. Uma vez que implementa aspectos específicos da simulação orientada a eventos, não é acessível aos usuários.

Camada XBT (*eXtended Bundle of Tools*)

Esta camada implementa diversas funcionalidades, como estruturas de dados usadas pelo SURF, serviço de *logging*, biblioteca de funções para representação e manipulação de grafos. Além disso, o XBT provê suporte à portabilidade da ferramenta.

2.1.3 Modelagem da plataforma do *grid* e das aplicações

Um importante recurso existente no simulador SimGrid é a especificação da plataforma computacional por meio de arquivos formatados em XML (*Extensible Markup Language*). O conjunto de recursos da plataforma do *grid*, bem como a estrutura de interconexão entre eles são especificados pelo usuário em um arquivo XML (`platform.xml`) . Os recursos computacionais que executam tarefas são modelados pelo seu poder computacional, enquanto que os *links* de comunicação são modelados pela sua largura de banda e latência (tempo para acessar o recurso). Entre dois recursos, pode haver mais de um *link*, e um dado *link* pode ser utilizado tanto para enviar, quanto para receber *tarefas*.

Em outro arquivo XML (`deployment.xml`) são especificados os *processos* que serão executados em cada recurso, e possivelmente argumentos que sejam necessários aos *processos*. Um *processo*, indicado pelo par *host*-função, indica uma atividade exercida por um determinado recurso. Um dado recurso pode executar "n" *processos*, cada um associado a uma porta diferente. A função especificada no arquivo XML deve ser implementada no código da aplicação.

Por meio de funções implementadas pelo SimGrid, é possível ler corretamente tais arquivos e recuperar as informações a respeito da plataforma do *grid*, e auxilia a especificação dos processos. Dessa forma, a modificação da plataforma computacional e de outros aspectos da aplicação torna-se uma tarefa muito mais simples, além de tornar o código da aplicação independente da plataforma do *grid* sob análise.

Outra aspecto importante abordado pelo SimGrid, e disponível para todos os ambientes da ferramenta, é a possibilidade de conferir dinamismo à plataforma, indicando no arquivo `platform.xml` mudanças na disponibilidade dos recursos com o uso de *traces*. Os *traces* compreendem arquivos nos quais é possível indicar variações na carga de trabalho dos *hosts*

e na disponibilidade dos *links* de comunicação, ou mesmo ocorrência de falha (temporária ou permanente) de algum recurso. Este mecanismo da ferramenta permite que o modelo de simulação seja mais fiel ao mundo real, no qual os recursos componentes do sistema podem ser usados localmente pelas organizações pertencentes ao *grid*, e a rede de comunicação (*internet*) é compartilhada com outros usuários.

2.1.4 Visualização dos resultados

Uma vez definidos a plataforma do *grid* e os processos que realizarão o escalonamento e a execução das tarefas, o SimGrid é executado passando ambos arquivos XML (`platform.xml` e `deployment.xml`) como parâmetros. O SimGrid não define nenhum padrão de exibição de resultados, mas oferece ferramentas que auxiliam na visualização do andamento da simulação. Por meio de funções existentes na biblioteca do SimGrid, é possível imprimir mensagens pré-formatadas, contendo uma *string* definida pelo usuário, o recurso e respectivo processo originador da mensagem e o valor corrente do relógio de simulação. Tais mensagens se mostram úteis na tarefa de depuração do programa de simulação. A seguir, é mostrado um exemplo de mensagens exibidas no SimGrid.

```
[   Tempo ] [ Recurso:processo    ] String
[   0.011 ] [  gspd:scheduler     ] Sending "task 1" to "spd02"
[   0.032 ] [ spd02:slave         ] Received "task 1"
```

2.1.5 Implementação e documentação

O SimGrid (incluindo sua suíte de testes) é implementado em linguagem C. Otimizações efetuadas no código melhoram o uso de memória e a velocidade de execução. Algumas das técnicas de otimização são empregadas na manipulação dos *traces*: eles podem ser compartilhados pelos recursos; conjuntos grandes de *traces* são carregados em memória apenas quando necessários, e descartados após usados. O esforço na otimização dos *traces* é justificado pelo fato de a incorporação de seus valores nos intervalos de tempo ocupar a maior parte do tempo de execução da simulação.

A ferramenta, que é *opensource*, funciona apenas em modo texto, e está disponível

para os ambientes Linux, Windows e MacOS. O SimGrid possui uma boa documentação, que pode ser obtida no *site* oficial do projeto [16]. Nele é possível obter o código fonte do simulador, uma lista de *bugs* conhecidos, descrições sobre o módulos da ferramenta e seu uso, publicações sobre o SimGrid ou que utilizam a ferramenta para obtenção de resultados.

2.1.6 Aplicações

Os principais trabalhos que fazem uso do SimGrid concentram-se na área de escalonamento de tarefas em ambientes distribuídos. Em [17] é realizada uma comparação entre alguns dos mais representativos algoritmos de escalonamento em *grids*, sob diferentes cenários para uma mesma plataforma de *grid*. Os algoritmos avaliados foram implementados no simulador SimGrid.

Em [5, 6] uma heurística de escalonamento para aplicações que processam grandes quantidades de dados foi proposta e seu desempenho foi avaliado e comparado com outras heurísticas por meio da sua implementação no SimGrid. Em [18], algoritmos são propostos com o objetivo de otimizar a redistribuição de dados em anéis de processadores (disposição das unidades de processamento em uma rede em anel) homogêneos e heterogêneos. Tal problema surge quando o mecanismo de balanceamento de carga é invocado. A avaliação das novas estratégias de redistribuição propostas é realizada com o SimGrid.

2.2 Bricks

Bricks [10] é uma ferramenta de avaliação de desempenho que permite a análise e comparação de diferentes estratégias de escalonamento de tarefas em sistemas computacionais de alto desempenho de escala global, como *grids*. A ferramenta é, portanto, tipicamente empregada para simular diversos comportamentos de um sistema computacional distribuído, como o de algoritmos de escalonamento, da topologia de sistemas cliente-servidor, e de estratégias de processamento para redes e servidores.

O Bricks também coleta informações sobre os recursos computacionais necessárias para os algoritmos de escalonamento, por meio de monitoração e predição da disponibilidade dos recursos no ambiente computacional. Para tanto, o simulador provê alguns componentes

que facilitam o monitoramento, predição e escalonamento de tarefas na rede simulada.

Internamente, o simulador emprega o conceito de redes de filas para executar a simulação, num sistemas de servidores dispersos num ambiente distribuído. A figura 2.3 ilustra um exemplo da modelagem do sistema computacional empregando esse conceito. Conforme mostrado na figura, as redes do cliente ao servidor, do servidor ao cliente, e os servidores são representados pelas filas Q_{ns}, Q_{nr} e Q_s, respectivamente. As taxas de serviços de Q_{ns}, Q_{nr} e Q_s representam a largura de banda de cada uma das redes e o poder de processamento do servidores, respectivamente (A e A' denotam um mesmo cliente, mas distinguidos para melhor entendimento). Mais detalhes sobre esta modelagem podem ser encontrados em [19] e em [20].

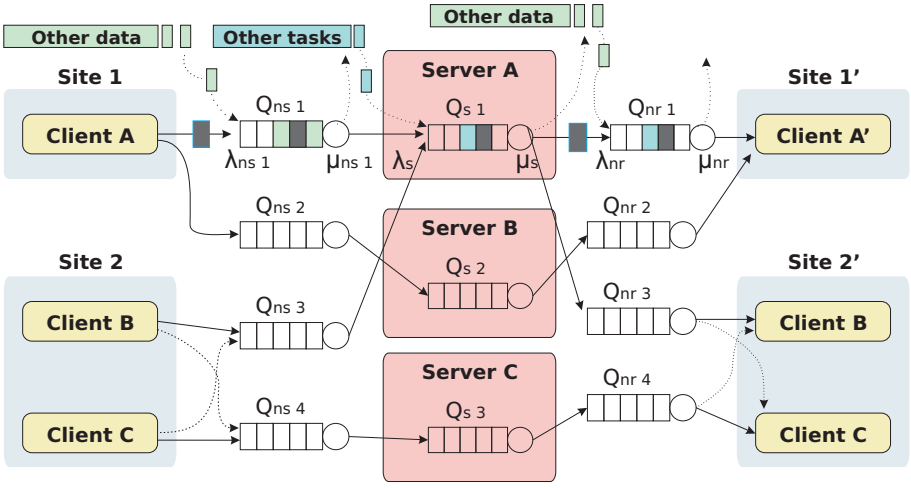


Figura 2.3: Exemplo de modelagem do ambiente computacional global por redes de filas

2.2.1 Arquitetura do Bricks

O simulador discreto orientado a eventos Bricks consiste basicamente de um *ambiente computacional global* e de uma *unidade de escalonamento*, que coordenam o comportamento do sistema. Este primeiro componente compreende o escalonador de tarefas e os módulos responsáveis pela monitoração e predição dos recursos, além de um sistema de banco de dados que mantém as informações coletadas sobre os recursos. O segundo compreende o conjunto de clientes, que solicitam a execução das tarefas, os servidores, que executam as tarefas solicitadas, e a rede que os interliga. A figura 2.4 ilustra a arquitetura do Bricks

descrita em termos desses dois componentes. Nas próximas seções, os módulos que formam ambos componentes básicos serão detalhados.

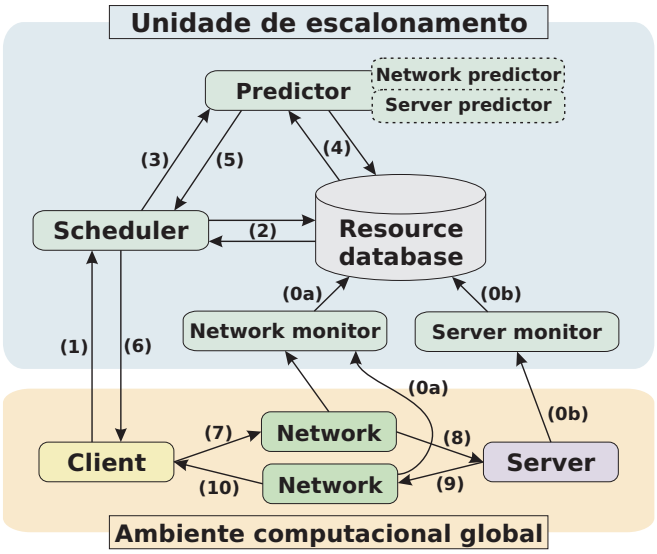


Figura 2.4: Arquitetura do Bricks

Ambiente computacional global

O ambiente computacional global representa a plataforma sobre o qual a simulação será realizada e é formado pelos seguintes módulos:

Client : representa o *host*, por meio do qual tarefas computacionais são inicializadas pela aplicação do usuário.

Network : representa a rede de interconexão dos clientes e servidores, e é parametrizado por sua largura de banda, congestionamento e sua variação ao longo do tempo.

Server : denota os recursos computacionais do sistema distribuído, e é parametrizado por seu poder computacional, carga de trabalho e sua variação ao longo do tempo.

Tanto a rede, quanto os servidores são modelados na forma de filas, cujas estratégias de processamento (políticas de atendimento) podem ser substituídas. A tarefa invocada pelo usuário (cliente) é modelada em termos de duração da computação e da comunicação, sendo implementada pelo Bricks da seguinte forma:

- Quantidade de dados transmitidos de/para um servidor em função da transmissão da tarefa pela rede.
- Número de instruções (operações) executadas pela tarefa.

Neste simulador, o modelo de comunicação é especificado em uma linguagem de *script* própria do Bricks. Duas familias principais de modelos suportadas pelo Bricks, são citadas nesse estudo: a primeira família assume que o congestionamento da rede é representado pela quantidade de dados de entrada oriundos de tráfego externo gerado por outros nós do sistema. Neste caso, é necessário especificar a largura de banda média dos canais de comunicação e sua variância e o tamanho médio dos dados provenientes de outros nós. Quanto menor for o tamanho de pacote especificado no modelo, maior será a precisão da simulação, mas também maior será o custo computacional da simulação.

Na segunda família, a variação da largura de banda é determinada por parâmetros observados em uma rede real. Embora necessite de informações externas à simulação, a rede modelada comporta-se de forma bastante fiel a uma rede real.

O modelo de servidor adotado pelo Bricks, atua da seguinte forma: as tarefas são processadas de acordo com o modelo FCFS (*First-Come, First-Served*); sua carga de trabalho pode ser especificada e simulada não apenas pela taxa de chegada de tarefas, mas também por meio de parâmetros observados em um cenário real.

Unidade de escalonamento

O segundo componente básico da arquitetura do Bricks provê um *framework* de escalonamento para *grids*. Os módulos que o constituem representam aspectos comuns em *grids* computacionais:

Network Monitor : mede a largura de banda e a latência dos canais de comunicação do *grid*. Os valores aferidos são armazenados no módulo *Resource DB*.

Server Monitor : mede o desempenho (em termos de poder de processamento), carga de trabalho e disponibilidade dos servidores. Os valores aferidos também são armazenados no *Resource DB*.

Resource DB : atua como um banco de dados para informações úteis ao escalonamento de tarefas. Os dados nele armazenados são acessados pelo *Predictor* e *Scheduler*, para realizar as decisões de previsão e escalonamento.

Predictor : recupera a informação armazenada a respeito dos recursos a cada determinado intervalo de tempo e realiza previsões sobre a disponibilidade de recursos. A informação sobre as previsões é utilizada no escalonamento de novas tarefas.

Scheduler : aloca uma dada tarefa invocada pelo usuário em um servidor adequado, tomando decisões com base nas informações sobre recursos e previsões providas pelo *Resource DB* e *Predictor*.

A *unidade de escalonamento* é implementada na forma de uma API, de forma que seus módulos podem ser alterados ou reescritos pelos usuários. Por exemplo, o módulo *Scheduler* pode ser alterado para acomodar outras políticas de escalonamento, e o *Predictor* pode ser substituído de forma a incorporar um módulo de predição externo, tal como o NWS (*Network Weather Service*), ferramenta que monitora e efetua previsões sobre o comportamento de sistemas computacionais fisicamente em experimentos reais.

2.2.2 Descrição do funcionamento da simulação

Como mencionado anteriormente, o *ambiente computacional global* e a *unidade de escalonamento* coordenam conjuntamente a simulação do comportamento do sistema. Na figura 2.4 foram apresentados os componentes da arquitetura do simulador, e através dela, os passos da simulação realizada pelo Bricks serão detalhados:

0a. O *NetworkMonitor* checa periodicamente a rede, mensurando a sua largura de banda, e armazena os dados obtidos no *Resource DB*.

0b. O *ServerMonitor* periodicamente consulta os servidores, obtendo informações sobre o seu estado, e também armazena as informações coletadas no *Resource DB*.

1. Quando o cliente invoca uma tarefa, esta consulta o escalonador (*Scheduler*) para saber qual é o servidor mais adequado para execução da tarefa. Para isso, são forne-

cidas ao *Scheduler* informações a respeito das características da tarefa. A definição do servidor mais apropriado dependerá da política de escalonamento adotada.

2. O *Scheduler* consulta o *Resource DB*, obtendo uma lista com o servidores existentes no *grid*.

3. O *Scheduler* consulta o *Predictor*. obtendo informações sobre a disponibilidade dos servidores consultados na etapa anterior, bem como informações sobre o estado dos canais de comunicação conectados aos servidores.

4. O *Predictor* obtém do *Resource DB* valores de medições efetuadas sobre a rede e servidores, e prevê a disponibilidade dos respectivos recursos no futuro.

5. Após realizar as previsões, o *Predictor* envia os resultados ao *Scheduler*.

6. Empregando a informação obtida, o *Scheduler* aloca a tarefa no servidor mais apropriado, e transmite a informação sobre a decisão de escalonamento, incluindo o identificador do servidor alocado ao cliente.

7. O cliente decompõe os argumentos a serem passados a tarefa em pacotes lógicos, e os injeta pela rede tendo como destino o servidor definido na etapa anterior.

8. Os pacotes transmitidos pelo cliente são processados em uma fila do canal de comunicação, com um tempo de duração dado por:

$$\frac{tamanho\ dos\ pacotes\ enviados}{largura\ de\ banda\ do\ canal}$$

9. Após todos os pacotes serem transferidos para o servidor, a tarefa é colocada na fila do servidor, e este executa a tarefa. O tempo de processamento da tarefa é dado por:

$$\frac{numero\ de\ instrucoes\ da\ tarefa}{poder\ computacional\ do\ servidor}$$

10. Após o término da execução da tarefa, os resultados são decompostos em pacotes, e estes são enviados pela rede tendo como o destino o cliente que invocou a tarefa. O tempo de transferência é dado por:

$$\frac{tamanho\ dos\ pacotes\ recebidos}{largura\ de\ banda\ do\ canal}$$

Após o recebimento de todos os pacotes, a tarefa invocada é considerada encerrada.

2.2.3 Modelagem da plataforma computacional e da aplicação

A documentação existente sobre o Bricks não apresenta muitos detalhes sobre a forma de modelagem da plataforma do *grid* e do programa de simulação. A modelagem da aplicação é realizada pela substituição ou alteração da SPI (*Service Provider Interface*) da *unidade de escalonamento*, uma API que descreve o algoritmo de escalonamento a ser empregado. O usuário pode, assim, modificar essa interface, de modo a adequá-la aos seus objetivos. Outra possibilidade, já anteriormente citada, é a substituição do módulo *Predictor* por um módulo de predição externo, como o NWS.

A modelagem da plataforma do *grid* é feita pela alteração de um arquivo escrito em uma linguagem de *script* própria do Bricks, possibilitando a especificação da topologia de rede, o conjunto de máquinas cliente e servidor, e os parâmetros que caracterizam os recursos computacionais e os canais de comunicação.

2.2.4 Implementação e documentação

O simulador Bricks é implementado em Java, sendo funcional em qualquer sistema operacional com suporte à JVM (*Java Virtual Machine*). Cada um de seus componentes é implementado em um módulo distinto. Essa modularização tem como objetivo facilitar a alteração do escalonador, além de viabilizar a incorporação de ferramentas externas, como o sistema NWS, uma ferramenta de monitoração baseada em observações passadas. O Bricks, como citado anteriormente, emprega uma linguagem de *script* para a modelagem da plataforma do *grid*. Há pouca documentação a respeito da ferramenta. A maior parte das informações coletadas neste estudo foram retiradas da página oficial do Bricks [21]. O código fonte do simulador não se encontra disponível e atualmente seu projeto está parado.

2.2.5 Aplicações

As aplicações do Bricks concentram-se no estudo de estratégias de escalonamento de tarefas. Em [22] o simulador Bricks é utilizado para avaliar uma nova estratégia de esca-

lonamento apropriada para o caso de multi-clientes e multi-servidores, empregando-se os mecanismos de *fallback* e correção de carga de trabalho.

2.3 Optorsim

A ferramenta Optorsim [11] é um simulador projetado como um *framework* modularizado, com o qual podem ser estudadas estratégias de escalonamento que utilizam o conceito de replicação de dados. Neste caso, a replicação significa criação de cópias de dados em recursos geograficamente distribuídos, e é uma das principais técnicas de otimização na redução do custo de acesso aos dados. Trata-se, dessa forma, de um simulador voltado para *grids* de dados, nos quais as transferências de dados constituem um importante fator limitante do desempenho das tarefas executadas. Na próxima seção será descrita a arquitetura do simulador.

2.3.1 Arquitetura do Optorsim

Uma importante questão no projeto da ferramenta foi a necessidade de modelar as interações dos componentes individuais do *grid* de dados de forma mais realista possível. Para isso, adotou-se a arquitetura de *grid* de dados desenvolvida pelo projeto *EU Data Grid* [23], como mostrado na figura 2.5.

De acordo com o este modelo, o simulador considera que o *grid* é formado por diversos *sites*, os quais provêm recursos computacionais e de armazenamento para as tarefas submetidas. Cada *site* possui zero ou mais *elementos de computação* (CE) e zero ou mais *elementos de armazenamento*. (SE) Um CE executa tarefas que utilizam dados em arquivos localizados nas SEs, e um componente denominado *Resource Broker* controla o escalonamento das tarefas nas CEs.

As decisões sobre a movimentação de dados entre *sites* são tomadas pelo *Replica Manager*. Dentro deste componente, a decisão de criar ou eliminar réplicas é controlada pelo *Replica Optimizer*, também chamado de Optor. O coração do Optor é o algoritmo de otimização de réplicas.

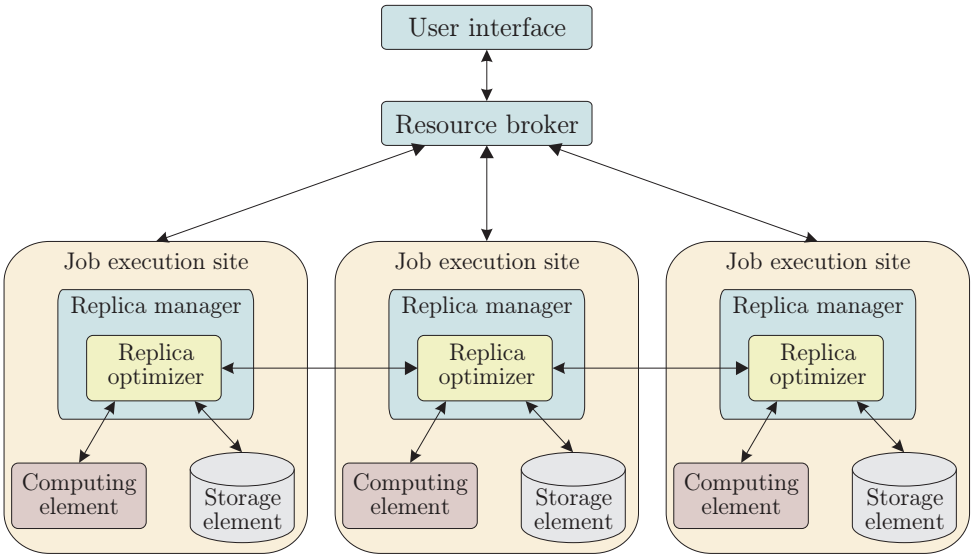


Figura 2.5: Arquitetura do *grid* de dados adotada pelo Optorsim

Internamente, cada CE é representado por uma *thread*. A submissão de tarefas é gerenciada por outra *thread*: o *Resource Broker*, que assegura que todo CE execute tarefas continuamente (sem haver ociosidade) tentando distribuir todas as tarefas em todos CEs. Quando o *Resource Broker* encontra um CE ocioso, ele seleciona uma tarefa para executar neste recurso.

Cada tarefa tem um conjunto de arquivos que pode solicitar. Para requisitar um arquivo, a tarefa deve fazer uma referência a ele, e existem dois tipos de referência: nome lógico de arquivo (LFN) e nome físico de arquivo (PFN). Um LFN é uma referência abstrata a um arquivo, independente de onde ele está armazenado e de quantas réplicas existem. O PFN referencia uma réplica de um LFN localizado em um dado *site*. Cada LFN tem um PFN para cada réplica no *grid*.

Uma tarefa tipicamente requisita um conjunto de LFNs para acesso aos dados. A ordem na qual os dados são solicitados é determinada pelo padrão de acesso. Os seguintes padrões de acesso são considerados:

Seqüencial : o conjunto de LFNs é ordenado, formando uma lista de requisições sucessivas.

Aleatório : arquivos são selecionados aleatoriamente, segundo uma distribuição de pro-

babilidade uniforme.

Caminho aleatório unitário : o conjunto é ordenado e sucessivas requisições de arquivos são feitas em elementos a uma unidade de distância do arquivo previamente requisitado (a direção é aleatória).

Caminho aleatório gaussiano : funciona da mesma maneira que o caminho aleatório unitário, mas os arquivos são selecionados de acordo com a distribuição de probabilidade gaussiana centrada no arquivo previamente requisitado.

Quando um arquivo é requisitado por uma tarefa, o arquivo LFN é usado para localizar a melhor réplica, por meio de um função específica do *Replica Optimizer* (`getBestFile`). Esta função avalia qual réplica pode ser transferida mais rapidamente para o *elemento de computação* que executará a tarefa, baseando-se na informação sobre as larguras de banda dos canais de comunicação. Para isso, checa-se o catálogo de réplicas, que mapeia todas as cópias de cada arquivo. Este catálogo é um serviço de *middleware* do *grid*, implementado pelo simulador como uma tabela de LFNs e seus respectivos PFNs. O objetivo desse esforço é reduzir o tempo de acesso aos dados e, com isso, diminuir o tempo de execução da tarefa.

A função `getBestFile` é uma chamada bloqueante, que pode provocar a replicação de dados no *elemento de armazenamento* do *site* no qual a tarefa será executada. Após a efetuação de uma replicação, o PFN da melhor réplica disponível é retornado à tarefa. Se não ocorre replicação, a melhor réplica é localizada em um *site* remoto e seu acesso ocorrerá usando E/S remota.

Tanto o tempo de replicação (caso ocorra), quanto o tempo de acesso ao arquivo são dependentes das características da rede durante a conexão. Em qualquer instante, a largura de banda disponível para a transferência é limitada pela menor largura de banda ao longo do caminho da transferência. Para transferências que empregam um mesmo elemento de rede, a largura de banda deste elemento é compartilhada de forma que cada transferência receba um porção igual.

2.3.2 Algoritmos de otimização

O algoritmos de otimização de réplicas compreendem o núcleo do componente *Replica Optimizer*. Tais algoritmos são implementados diretamente pela função `getBestFile`. Toda vez que um arquivo é requisitado à *unidade de armazenamento* remota, é criada uma cópia do arquivo no *site* do *elemento computacional* que o solicitou. Quando não há mais espaço nos *elementos de armazenamento* de um dado *site*, um arquivo deve ser eliminado para que a replicação ocorra.

A estratégia adotada para decidir qual arquivo deve ser eliminado é o que diferencia os algoritmos de otimização de réplicas. O simulador traz alguns exemplos de algoritmos simples e um mais complexo, de abordagem econômica [24].

2.3.3 Modelagem da plataforma do *grid* e das aplicações

No OptorSim, a plataforma do *grid* é especificada em um arquivo de configuração (`grid.conf`), o qual é lido pela aplicação. Neste arquivo descreve-se a topologia de cada *site* e detalhes de seus componentes:

- Número de *elementos computacionais* do *site* (caso existam).
- Número de *elementos de armazenamento* do *site* (caso existam).
- Tamanho dos *elementos de armazenamento* em *megabytes*.
- Matriz diagonal simétrica *site vs. site*, em que cada elemento indica a máxima largura de banda do canal que liga os *sites* correspondentes.

As informações sobre as tarefas a serem executadas são indicadas em outro arquivo de configuração (`jobs.conf`). Este arquivo contém informações sobre as tarefas e os arquivos (tamanho e localização) por elas utilizados, e a política de cada *site* (lista de arquivos que cada *site* irá aceitar).

Um outro arquivo de configuração é utilizado para passar os parâmetros de entrada da simulação, os quais especificam diferentes características da aplicação:

- O caminho dos arquivos de plataforma e de tarefas - `grid.conf` e `jobs.conf`.
- Tipo de usuário:

  Simple : submete tarefas em intervalos regulares de tempo. Esse intervalo pode ser ajustado.

  Random : submete tarefas em intervalos de tempo uniformemente distribuídos, com valor médio indicado.

  CMS DC04 : baseado no padrão de submissão criado no CMS *Data Challenge* 2004; submete tarefas de acordo com uma distribuição gaussiana centrada no horário 15:00h, com um número total de 700 tarefas.

- Estratégia de escalonamento do *Resource Broker*:

  Random : tarefas são escalonadas aleatoriamente a qualquer *unidade de computação* disponível.

  Queue Length : envia a tarefa para a *unidade de computação* com a menor fila de espera.

  File access cost : escalona a tarefa na *unidade de computação* de acordo com o custo de acesso a todos os arquivos (em termos de latência de rede).

  File access cost + job queue access cost : combina as duas últimas estratégias para calcular um valor de custo.

- Algoritmo de otimização a ser usado. Há diferentes algoritmos implementados pela ferramenta, como previamente citado.
- Gerador de padrão de acesso, que determina a ordem com a qual os arquivos são acessados pelas tarefas, como comentado na seção anterior.
- Distribuição inicial dos arquivos, em que se define a localização de cada arquivo original. Pode-se especificar uma distribuição aleatória dos arquivos pelos *sites* que constituem o *grid*.
- Intervalo de tempo entre submissões de tarefas, que dependerá do tipo de usuário escolhido.

- Semente do gerador de números aleatórios. Pode-se fixar a semente, para que em todas as execuções os números aleatórios gerados sejam os mesmos.
- Tamanho máximo da fila, isto é, máximo número de tarefas em espera para serem executadas em um CE.
- Sinalização sobre o uso ou não da interface gráfica provida pelo OptorSim.

2.3.4 Visualização dos resultados

Uma vez definidos os parâmetros dos arquivos de configuração, o Optorsim pode ser executado. Ao longo da execução da simulação, são mostrados em modo texto os detalhes das ocorrências dos eventos: gerador do evento (usuário, *resource broker* ou *site*). Em relação aos usuários, são mostradas informações sobre o nome da tarefa por ele submetida, o *resource broker* para o qual a tarefa é enviada e o horário da submissão. Quanto ao *resource broker*, é mostrado o nome da aplicação submetida, o site escolhido pela política de escalonamento adotada, e o horário em que ocorre esta atividade. Sobre o *site* é indicado o seu nome, o nome do CE responsável pela execução da tarefa, nome da tarefa a ser executada, tamanho da fila do respectivo CE, e o horário de início da execução da tarefa.

Ao final da simulação, importantes informações sobre o estado final do *grid* e estatísticas da simulação são mostradas. Sobre o *grid* simulado são exibidas as seguintes informações: número de réplicas criadas, porcentagem de uso dos CEs, número de leituras feitas em arquivos (originais ou réplicas) armazenados nos *sites*, e o tempo total de execução das tarefas.

Para cada *site* são mostradas as cópias de arquivos (réplicas) armazenadas em cada um de seus SEs, e algumas estatísticas: média de ocupação dos CEs, número de leituras de arquivos locais, número de arquivos acessados, tempo total de processamento das tarefas, capacidade de armazenamento dos SEs e o uso dessa capacidade. Também são exibidas quais tarefas foram executadas naquele *site*, bem como o tempo de execução das mesmas

Uma forma mais fácil de executar o simulador, visualizar os seus resultados e acompanhar o progresso da simulação é por meio da interface gráfica provida pelo Optorsim. Esta interface permite ajustar os parâmetros definidos no arquivo `parameters.conf`, ver o an-

damento da simulação em um terminal (como no modo texto), e visualizar uma estrutura lógica dos *sites* do *grid*. Os *sites* são organizados em um anel e são mostradas graficamente as requisições de dados feitas pelos *sites*. Outra importante funcionalidade desta interface gráfica é a criação de arquivos de imagem JPEG contendo gráficos e histogramas sobre diversas estatísticas sobre a utilização dos recursos.

2.3.5 Implementação e documentação

O OptorSim é implementado em Java, o que lhe confere uma maior portabilidade, e de forma bem modularizada. Cada componente da arquitetura do simulador é implementado em uma classe distinta. Além disso, a ferramenta implementa em diferentes classes o tratamento de cada parâmetro passado para a simulação.

Seu uso é relativamente simples, consistindo basicamente na configuração da plataforma do *grid*, das tarefas e arquivos por elas utilizados, e dos parâmetros de entrada da simulação. Portanto, o uso da ferramenta não envolve codificação do programa de simulação, podendo ser usada por usuários sem conhecimentos específicos em programação. Existe, ainda, a possibilidade de o usuário implementar novas políticas de otimização de réplicas.

Outra importante questão é a interface oferecida pelo OptorSim ao usuário. O simulador possui uma interface gráfica (GUI), por meio da qual o usuário pode especificar os parâmetros da simulação e visualizar o seu progresso em um terminal, e uma ilustração gráfica do *grid* indicado no arquivo de configuração. Por meio dessa interface, é possível também verificar informações estatísticas a respeito dos recursos e a saída da simulação.

O OptorSim é de código aberto e bem documentado. Além de publicações em periódicos sobre a ferramenta, informações sobre a sua instalação e uso podem ser encontradas na página oficial do projeto DataGrid, ao qual o OptorSim está vinculado [25]. Nesta página é possível também obter os códigos fonte e executável da ferramenta.

2.3.6 Aplicações

Os trabalhos de pesquisa que envolvem o uso do OptorSim visam tipicamente avaliar diferentes estratégias de escalonamento de tarefas e de replicação de dados em *grids* compu-

tacionais. Tais *grids* normalmente estão voltados à aplicações que envolvem transferências de enormes quantidades de dados, o que justifica o especial interesse na otimização dessas transferências como forma de reduzir o tempo de execução das tarefas.

Em [26] a ferramenta é empregada justamente com essa finalidade, utilizando uma topologia de *grid* baseada em um ambiente computacional dedicado a experimentos na área de física de altas energias, o qual compreende 20 *sites* situados nos Estados Unidos (Fermilab) e Europa (CERN). De maneira análoga, em [27] o Optorsim é utilizado para avaliar os resultados obtidos por diferentes estratégias de escalonamento e replicação em uma plataforma de *grid* baseada no *UK Grid* (GridPP), destinado a estudos sobre física de partículas no Reino Unido.

2.4 GangSim

O simulador GangSim [9] foi criado para auxiliar estudos de estratégias de escalonamento em ambientes de *grid*. Estes estudos procuram avaliar o impacto das políticas de alocação de recursos adotadas por *sites* e organizações virtuais (VO) no desempenho apresentado por tais *sites* e VOs. A ferramenta simula, assim, grandes grupos ou *gangs* de *sites* e usuários.

Em grades computacionais, uma VO é definida como o conjunto de instituições que compartilham seus recursos de forma coordenada e que atendem determinados requisitos [28]. Esses requisitos envolvem a definição de um único método de autenticação, de autorização, de acesso aos recursos, de descoberta de recursos, entre outros aspectos.

A ferramenta permite combinar componentes de simulação com instâncias da ferramenta de monitoração VO-Ganglia executando em recursos reais. O aspecto inovador do GangSim é a possibilidade de modelar não apenas *sites*, mas também usuários e projetistas de VOs, e sua habilidade em modelar políticas de uso em nível de *site* e VO. Na próxima seção, será descrita a arquitetura da ferramenta.

2.4.1 Arquitetura do GangSim

O ambiente computacional simulado pelo GangSim é formado por dois componentes básicos:

Sites : abrigam os recursos que executarão as tarefas submetidas pelos usuários. Além dos recursos, possuem escalonadores de tarefas local e global, e um componente responsável por assegurar que a política de alocação de recursos será cumprida.

VOs : locais dentro do *grid* onde se encontram os usuários, um componente de monitoração, e um outro componente que desempenha funções das políticas de alocação de recursos adotadas pela VOs.

O ambiente simulado pelo GangSim, é mostrado na figura 2.6. Esta figura ilustra, ainda, o modelo de alocação de recursos, no qual recursos dos *sites* são alocados pelas VOs. Os rótulos `C` e `S` denotam, respectivamente, recurso de computação e recurso de armazenamento.

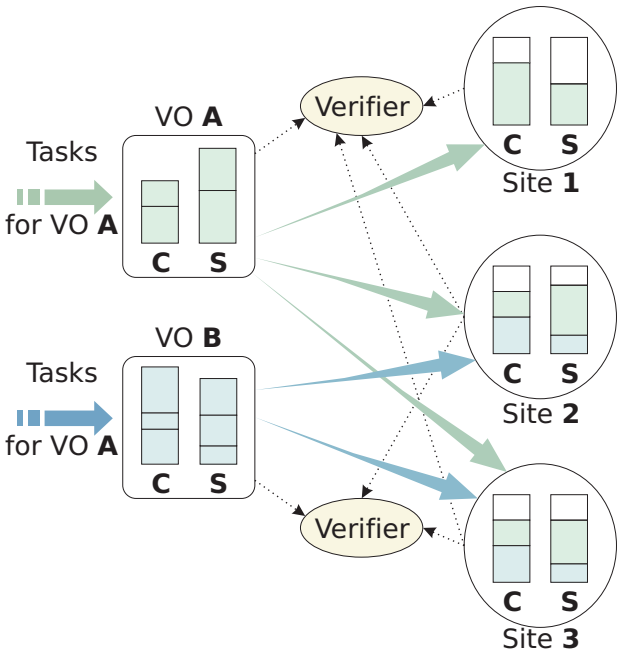


Figura 2.6: Esquema do modelo de alocação de recursos usado pelo GangSim

O GangSim simula um gerenciamento de infra-estrutura, no qual políticas que definem a alocação de recursos dentro das comunidades (VOs) e a alocação de recursos através

das VOs em *sites* individuais interagem para determinar a decisão final de alocação de recursos computacionais individuais (CPU, disco e rede). As políticas usadas são expressas em termos de regras associadas aos *sites*, VOs, grupos e usuários para diferentes formas de agregação dos recursos disponíveis. Por exemplo, uma política adotada por uma VO pode estabelecer a seguinte regra: um grupo de usuários A terá disponível 50% dos recursos alocados pela VO, enquanto um *site* pode definir uma política que reserva 20% de seus recursos para a VO B. A figura 2.7 ilustra um exemplo de alocação dos recursos de um *site* a uma VO, e a política de divisão do uso desses recursos pelos diferentes grupos de usuários da VO.

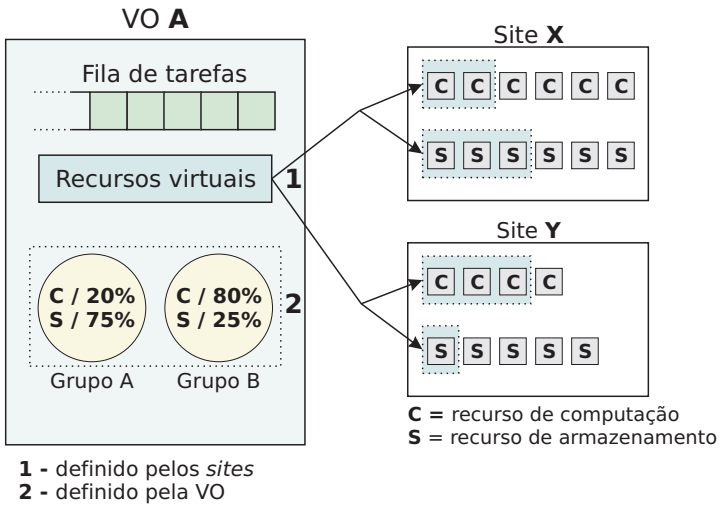


Figura 2.7: Exemplo de alocação de recursos e de política de distribuição em uma VO

A ferramenta modela o ambiente simulado por meio dos seguintes elementos encontrados em *grids* reais: infra-estrutura de submissão de tarefas, infra-estrutura de monitoração, e uma política de uso da infra-estrutura. Os seus principais componentes são: escalonadores externos (ES), escalonadores locais (LS), escalonadores de dados (DS), pontos de distribuição do monitoramento (MDP), pontos de políticas de coerção dos *sites* (S-PEP) e das VOs (V-PEP). Os *sites* agregam diversos nós computacionais e as VOs agregam usuários, que podem ser posteriormente reunidos em grupos.

Os componentes da arquitetura do GangSim, e as suas relações são mostrados na figura 2.8. Como é possível observar, o escalonamento das tarefas é feito de forma descentralizada. Cada VO adota a sua própria política de escalonamento de tarefas, e contribui para a

distribuição das informações de monitoração.

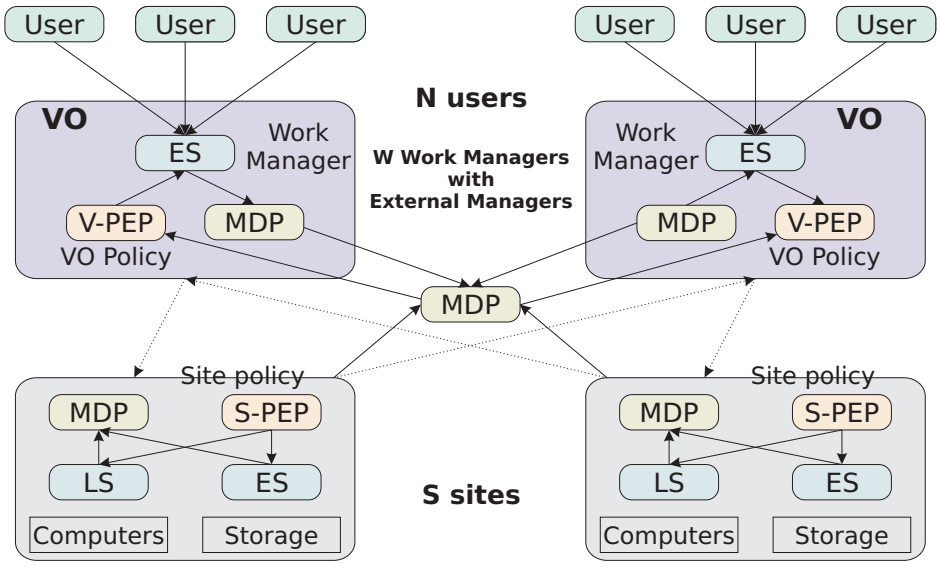


Figura 2.8: Componentes da arquitetura do GangSim e suas relações

Um *site* é caracterizado pelo número de CPUs, poder computacional de cada CPU, número de discos, tamanho de espaço em disco, canais de comunicação e a largura de banda destes canais. Cada característica é descrita em um arquivo de configuração, que é lido na inicialização do programa de simulação. Além das informações sobre a rede interna de cada *site*, devem ser descritas as características da rede instituída entre os *sites*. É definida também, a política de escalonamento usada pelo *site*, e a parcela de tempo de CPU, espaço em disco, largura de banda cada VO pode usar.

Uma VO é composta por um conjunto de grupos de usuários. Usuários submetem tarefas, que podem ser agrupadas em conjuntos chamados *carga de trabalho*. Além disso, um conjunto de arquivos de dados, correspondentes aos elementos de dados solicitados para a execução das tarefas, são distribuídos entre os *sites* antes de a simulação ser executada. A cada grupo de usuários definido, um conjunto de políticas é associado para definir o que cada VO disponibilizará para o grupo.

Escalonadores externos (ES), escalonadores locais (LS) e escalonadores de dados representam pontos nos quais as diversas decisões de escalonamento são tomadas. Um ES organiza as tarefas submetidas por usuários em uma fila, e seleciona seleciona o melhor

candidato para executar cada tarefa. Uma vez escalonada para um dado *site*, a tarefa é transferida do ES para o LS associado ao *site* escolhido.

Os pontos de monitoração de dados (MDPs), representam a infra-estrutura de nós de monitoração, que coleta diversas informações sobre o consumo dos componentes do *grid*. Tais informações são obtidas dos escalonadores locais e externos, filtrados, e distribuídos de maneira uniforme.

Os pontos de coerção das políticas (PEPs) são responsáveis pela correta execução das políticas. Eles reúnem dados de monitoração e outras informações relevantes a suas operações, e então as utilizam para assegurar que a alocação de recursos será realizada da forma especificada pela política adotada. Há dois tipos de PEPs: o S-PEP, residente em todos os *sites*, opera de forma contínua, no sentido de que todas as tarefas são imediatamente descartadas quando os requisitos da política não são mais atendidos; o V-PEP opera de forma similar ao S-PEP, tomando decisões de políticas de coerção considerando as especificações das VOs para a alocação de recursos aos grupos da VO ou o tipo de trabalho executado pela VO. Os V-PEPs interagem com os S-PEPs e escalonadores para garantir o atendimento dos requisitos da política.

O GangSim implementa diversos algoritmos e estratégias utilizadas para atualizar o estado de diferentes componentes do *framework*. Há algoritmos para a seleção e atribuição de tarefas, replicação de arquivos, e algoritmos que consideram os custos associados a cada operação e para a condução das tarefas através do *framework*. Todos estes algoritmos são implementados em um único módulo, o qual é invocado quando uma decisão concernente ao estado de um componente necessita ser considerada.

As tarefas são submetidas pelos usuários às filas do ES de acordo com uma política específica. Uma vez definidos os *sites* que executarão as tarefas, estas são transferidas às fila do LS. Uma tarefa pode ser rejeitada pelo LS, e neste caso a tarefa retorna à fila do ES para reentrar no processo de submissão. Se uma tarefa solicita arquivos maiores que a capacidade do *site*, ela é rejeitada pelo ES.

Os simulador associa diferentes custos de tempo a cada operação bem ou mal sucedida. O seguintes intervalos de tempo são contabilizados durante a submissão da tarefa: tempo para entrar na fila de submissão, tempo para atribuição de *site* (decisão de escalonamento

global), tempo para a transferência ao *site*, tempo para atribuição de nó dentro do *site* (decisão de escalonamento local) e, por fim, tempo de transferência da tarefa até este nó (considerando o tamanho do executável e dos arquivos de dados necessários).

2.4.2 Modelagem da plataforma do *grid* e das aplicações

A especificação da carga de trabalho, que caracteriza o conjunto de tarefas a serem simuladas, e do ambiente do *grid* é realizada por meio de ferramentas específicas oferecidas pelo GangSim. A modelagem do programa de simulação baseia-se na especificação das políticas de alocação de recursos nos *sites* e nas VOs. Essas políticas podem ser descritas em uma interface *Web* ou em um arquivo de configuração.

A modelagem da plataforma do *grid* segue o procedimento adotado pelo VO-Ganglia, no qual um arquivo de configuração é usado para especificar o número de VOs e *sites*, o poder computacional das máquinas dos *sites*, e as capacidades dos sistemas de armazenamento de dados destes *sites*.

2.4.3 Implementação e documentação

Como mencionado anteriormente, o GangSim deriva-se de uma ferramenta chamada VO-Ganglia, e é simulador discreto orientado a eventos. A sua implementação foi realizada de maneira modularizada, de modo que podemos dividir o simulador nos seguintes componentes:

Simulator Modules : conjunto de módulos escritos em linguagem Perl, que representam o núcleo da ferramenta, e simulam os ESs, os *hosts* de submissão, LSs e *sites*.

Task Assignment Policies : conjunto de algoritmos invocados para efetuar o escalonamento de tarefas em *sites* (escalonamento global) , escalonamento de tarefas em nós (escalonamento local). Estes algoritmos são chamados por vários componentes: S-PEPs, LSs, V-PEPs e ESs.

Metric Aggregators : um conjunto de rotinas que agregam métricas baseando-se em regras, tais como concatenação de *strings*, computação do inteiro mediano ou média

inteira.

Grid components : conjunto de funções e estruturas de dados para a simulação dos componentes do *grid*. Por exemplo, filas são representadas por arranjos de tipos específicos, *sites* são modelados por uma lista de capacidades físicas (armazenamento, computação e largura de banda) dos nós, e as cargas de trabalho são mantidas em estruturas de fila, enquanto passam de um estágio a outro no ambiente de simulação.

Environment State Keeper : conjunto de estruturas de dados que guardam os dados usados para a simulação do sistema. A informação armazenada é em sua maioria relativa ao estado das cargas de trabalho, estado dos componentes do *grid*, e utilização correntes dos recursos do sistema.

Interface : conjunto de *scripts* CGI que reúnem o estado da simulação executada pelo GangSim em um arquivo HTML. Os resultados da simulação ficam disponíveis em tabelas gravadas em arquivos e também na interface gráfica. Além disso, a interface *Web* permite navegar e visualizar de forma simples as estatísticas sobre variações do estado dos diversos componentes do *grid*.

O simulador GangSim é *opensource*, e pode ser obtido em [29]. A sua implementação, baseada em *scripts* Perl e interface *Web*, o torna acessível remotamente. Junto com o simulador, estão presentes todas as dependências necessárias para a sua instalação e execução. Assim como o OptorSim, o uso do GangSim não envolve a codificação do programa de simulação. No entanto, a documentação sobre a ferramenta é bem escassa. Não há manuais descrevendo a forma de uso do simulador. A documentação a respeito da ferramenta de monitoração VO-Ganglia constitui o principal fonte de informações para o desenvolvimento de simulações com o GangSim.

2.4.4 Aplicações

Não há muitas publicações sobre trabalhos que fazem uso do GangSim. Basicamente, a ferramenta pode ser usada para simular cargas de trabalho síncronas e assíncronas. No primeiro caso, todas as VOs submetem suas tarefas quase ao mesmo tempo. Na segunda situação, as VOs submetem suas rajadas de tarefas em diferentes instantes de tempo.

2.5 GridSim

O GridSim [8], atualmente na versão 4.1, é uma *toolkit* que propõe facilitar a simulação de diferentes classes de recursos heterogêneos, usuários, aplicações e escalonadores. Ele pode ser usado para simular escalonadores de tarefas, para sistemas distribuídos como *clusters* (um único domínio administrativo) e *grids* (múltiplos domínios administrativos).

As principais funcionalidades e características da ferramenta são descritas a seguir:

- O simulador possibilita a modelagem de recursos heterogêneos.
- Recursos podem ser modelados de forma que sejam compartilhados no tempo e no espaço. Ou seja, os recursos computacionais podem ser sistemas monoprocessados, multiprocessados de memória compartilhada, ou multiprocessados de memória distribuída. Os dois primeiros empregam o conceito de compartilhamento no tempo, devido ao fato de o sistema operacional ser multitarefa. O último, explora o conceito de compartilhamento no espaço, adotando um modelo de escalonamento interno distinto.
- A capacidade dos recursos é definida de acordo com o padrão estabelecido pelos *benchmarks* SPEC (*Standard Performance Evaluation Corporation*), isto é, em termos de MIPS (milhões de instruções por segundo).
- Recursos podem estar localizados em quaisquer zonas de fuso-horário.
- Finais de semana e feriados podem ser mapeados, dependendo do horário local do recurso para modelar carga de trabalho não referente ao *grid* (carga de trabalho local).
- Recursos podem ser reservados antecipadamente.
- *Grids* de dados podem ser simulados. A ferramenta permite modelar os diversos aspectos de um *grid* de dados, como a distribuição dos arquivos de dados entre os recursos, gerenciamento das réplicas, entre outros aspectos relevantes. Isso significa que o GridSim pode ser utilizado para avaliar estratégias de otimização de réplicas em *grids* onde as aplicações requisitam grandes volumes de dados.

- Recursos e outras entidades podem ser conectados em uma topologia de rede. O simulador permite modelar diferentes aspectos da rede de interconexão do *grid*. Além disso, ele incorpora o conceito de tráfego de fundo, útil na modelagem de uma rede pública, onde há problemas de congestionamento.

- Tarefas podem ser heterogêneas, e essencialmente de computação (CPU *intensive*) ou de dados (I/O *intensive*).

- Não há limite de número de tarefas que podem ser submetidas a um recurso.

- Múltiplas entidades de usuários podem submeter tarefas para execução simultaneamente em um mesmo recurso, que pode ser de tempo ou espaço compartilhado. Esta característica ajuda na construção de escalonadores, que podem usar diferentes modelos dirigidos por requisitos de *Grid Economy* [30].

- Suporte à simulação de escalonadores estáticos e dinâmicos.

- Estatísticas de todas as operações realizadas podem ser gravadas, e então analisadas usando métodos de análise estatística oferecidos pelo GridSim.

2.5.1 Arquitetura do GridSim

A arquitetura do simulador foi concebida de forma modular e em camadas. Esta arquitetura multi-camadas e suas aplicações são mostradas na figura 2.9.

De baixo para cima, a primeira camada refere-se a interfaces escaláveis entre o simulador e a máquina virtual Java (JVM), cuja implementação está disponível para sistemas mono e multiprocessados. A segunda camada é composta pela infra-estrutura básica de simulação orientada a eventos, construída usando as interfaces providas na camada inferior. Uma implementação de infra-estrutura de simulação orientada a eventos, de uso bastante difundido e que serve de base para o GridSim, é o SimJava [31]. A terceira camada diz respeito à modelagem e simulação das entidades básicas do *grid*, como recursos e serviços de informação. Nesta camada, também é realizada a modelagem da aplicação, o acesso uniforme a interface, e existe um *framework* para a criação de entidades em alto nível. A quarta camada diz respeito à simulação de agregadores de recursos do *grid*, chamados de

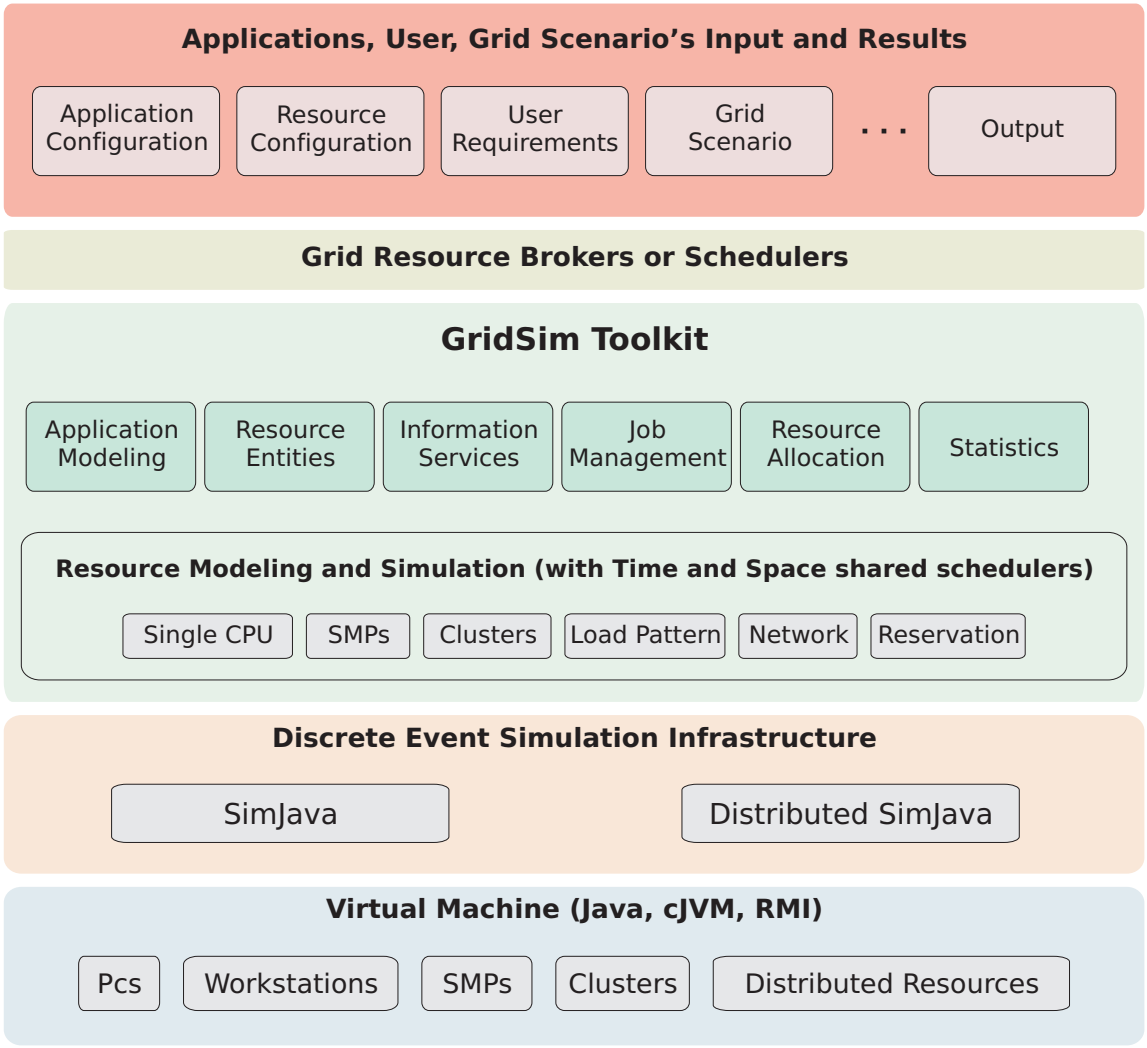


Figura 2.9: Arquitetura multi-camadas do GridSim e seus componentes

resource brokers ou escalonadores. A camada mais acima na arquitetura do GridSim está focada na aplicação e na modelagem dos recursos sob diferentes cenários, e faz uso dos serviços providos pelas duas camadas inferiores para avaliar as políticas de escalonamento e gerenciamento de recursos.

O SimJava é um pacote de simulação discreta orientada a eventos de propósito geral. Simulações no SimJava contêm um número de entidades, cada uma sendo executada em uma *thread* própria. A simulação realizada pelo GridSim é baseada, portanto, na definição das entidades, que são descritas a seguir:

User : cada instância desta entidade representa um usuário do *grid*, o qual é caracterizado pelo tipo de tarefa que cria (tempo de execução da tarefa, número máximo de réplicas, etc), pela estratégia de otimização adotada no escalonamento (minimização do custo, do tempo, de ambos, etc), taxa de atividade (quão freqüentemente o usuário cria uma nova tarefa) e fuso-horário.

Broker : cada usuário é conectado a uma instância de uma entidade *Broker*. Toda tarefa de um usuário é primeiramente submetida ao seu *broker*, e este então escalona a tarefa de acordo com a política de escalonamento adotada pelo usuário. O *broker* também é responsável por obter uma lista de recursos disponíveis e tenta otimizar a política de escalonamento do usuário.

Resource : cada instância desta entidade representa um recurso do *grid*, e possui os seguintes atributos: número de processadores, custo de processamento, velocidade de processamento, política interna de escalonamento (compartilhamento no tempo ou no espaço), fator de carga local, e fuso-horário.

Grid Information Service : provê o serviço de registro dos recursos existentes no *grid*, mantendo uma lista atualizada dos recursos disponíveis. Os *brokers* consultam esta entidade para obter informações sobre a disponibilidade, configuração e estado dos recursos.

Input/Output : o fluxo de informações entre as entidades do GridSim se dá por meio das entidades de entrada e saída (E/S). O uso de entidades separadas para entrada e saída permite que as demais entidades modelem canais de comunicação *full-duplex* e multi-usuários.

A modelagem da rede de interconexão do *grid* é feita descrevendo-se cinco componentes: *links*, caracterizados por largura de banda, latência e unidade máxima de transmissão (MTU), instâncias das entidades *Input* e *Output* [32], roteadores, pacotes e escalonadores de pacotes. Na figura 2.10, a estrutura de rede simulada pelo GridSim e as relações entre os componentes citados são mostradas.

A entidade *Output* é responsável por dividir as mensagens em pacotes com tamanho dado pelo MTU do canal, enquanto a entidade *Input* combina diferentes pacotes em um

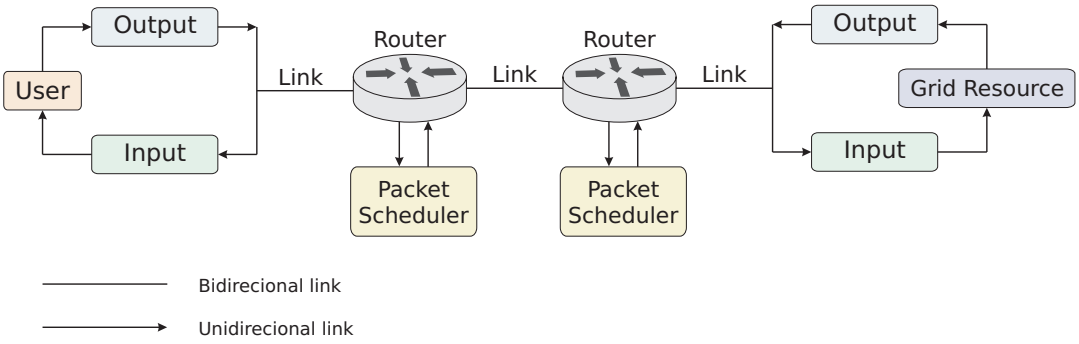


Figura 2.10: Estrutura de rede simulada pelo GridSim

fluxo, e o envia como uma porção de dados a uma entidade do *grid*. Outra funcionalidade deste par de entidades é a atuação como um *buffer* de contenção de pacotes à espera da liberação do *link* para a transmissão.

Um roteador no GridSim é representado por uma classe abstrata, por meio da qual é possível implementar qualquer algoritmo de roteamento. O roteamento pode ser feito empregando-se tabelas estáticas ou métodos dinâmicos, como o RIP (*Routing Information Protocol*) e o OSPF (*Open Shortest Patch First*).

Um pacote de rede no GridSim pode pertencer a duas categorias: *NetPacket*, usado para encapsular dados transmitidos na rede, e *InfoPacket*, dedicado à coleta de informações sobre a rede durante a execução da simulação, sendo equivalente ao protocolo ICMP (*Internet Control Message Procotol*) em redes reais.

O escalonador de pacotes é o componente responsável por decidir a ordem de envio dos pacotes em um canal de transmissão. Três estratégias de escalonamento são nativamente implementadas na ferramenta: FIFO (*First-In, First-Out*), WQF (*Weighted Fair Queuing*) e SCQF (*Self Clocked Fair Queuing*). Com este componente, torna-se possível priorizar determinadas classes de pacotes, oferecendo QoS às aplicações submetidas pelos usuários.

Outra importante funcionalidade do GridSim é a possibilidade de gerar tráfego de fundo, ou seja, tráfego não oriundo das aplicações do *grid*. A existência deste tipo de tráfego é justificada pelo fato de nem toda aplicação executada em um recurso do *grid* ser específica do ambiente de *grid*, caracterizando-se como uma aplicação local, que também consome recursos (CPU, rede, etc...). Ao considerar a existência de tráfego de fundo, o GridSim

confere um maior realismo às simulações. Este tráfego pode ser gerado através de funções existentes no SimJava, por meio do qual são especificados o intervalo entre chegadas de pacotes, tamanho dos pacotes e o número de pacotes gerados em cada intervalo, de acordo com alguma das distribuições de probabilidade implementadas na biblioteca de simulação.

O GridSim possui, ainda, uma extensão para a simulação de *grids* de dados, visando a avaliação estratégias de gerenciamento de dados sob diferentes cenários. Esta extensão permite a modelagem dos seguintes componentes [33]:

Files : são os arquivos (réplicas ou originais), e possuem os seguintes atributos: informação armazenada, nome do proprietário, tamanho do arquivo, data de modificação. Estes atributos são enviados a um outro componente, o catálogo de réplicas (*Catalogue Replica*).

***Catalogue Replica* (RC)** : possui a função de armazenar informações sobre os arquivos (metadados) e prover o mapeamento entre o nome de um arquivo e suas localizações físicas no *grid*. Diversos catálogos de réplicas podem coexistir em um mesmo *grid*. Um catálogo pode adotar o modelo centralizado, no qual o RC gerencia as informações sobre todos os arquivos e trata de todas as consultas, a exemplo do modelo seguido por programas de compartilhamento de arquivos *peer-to-peer*. Um outro modelo possível é o hierárquico, no qual os RCs são organizados em uma estrutura de árvore, de tal forma que cada RC faz o mapeamento entre nomes de arquivos e uma lista de RCs folhas. Os nós folha fazem o mapeamento entre nomes de arquivos e os recursos que os armazenam. Além destes dois modelos nativos do GridSim, outros modelos podem ser projetados e implementados pelo usuário.

Resource : um recurso no *grid* de dados é caracterizado por seus sistemas de armazenamento (modelado pela capacidade de armazenamento e atrasos na leitura e escrita), gerenciador de réplicas (responsável por registrar os conjuntos de dados do recurso nos RCs) e política de alocação (componente responsável pela execução da tarefa do usuário).

Por fim, pode-se considerar o GridSim como uma ferramenta de simulação de alto nível, projetada para investigar interações e interferências entre decisões de escalonamento

tomadas por escalonadores (*brokers*) distribuídos.

2.5.2 Modelagem da plataforma do *grid* e das aplicações

A modelagem das entidades participantes do *grid* e das tarefas que deverão ser submetidas é feita por meio da implementação, em Java, de um programa que faça uso da biblioteca provida pelo GridSim.

A criação e manipulação das tarefas é feita através do pacote *Gridlet*, que contém informações relacionadas ao tamanho computacional da tarefa, às operações de E/S, ao tamanho dos dados de entrada e de saída, e ao usuário criador da tarefa. Estes atributos associados às tarefas podem ser gerados aleatoriamente, por meio da classe *GridSimRandom*.

A interação entre as entidades do *grid* é modelada por meio de eventos, que são gerados pelas entidades e podem significar uma requisição de serviço a outra entidade, um atendimento de serviço solicitado por outra entidade, ou uma atividade interna. Os eventos gerados podem ser classificados como síncronos ou assíncronos. Um evento é síncrono, quando a entidade fonte do evento espera até que a entidade destinatária efetue todas as ações associadas ao evento em questão. Se a entidade fonte continua com outras entidades após gerar o evento, então este evento é classificado como assíncrono. Eventos internos de uma entidade (fonte e destino são idênticos) são necessariamente assíncronos, para evitar a ocorrência de *deadlocks*. Eventos externos podem ser síncronos ou assíncronos. Uma entidade pode solicitar, por exemplo, informações ao GIS (*Grid Information Service*) sobre os recursos disponíveis no *grid*, criando um evento de requisição. Esse modelo de interação também é importante em atividades como escalonamento de tarefas pelo *broker*, escalonamento realizado internamente pelo recurso, entre outras.

Os recursos do sistema são modelados criando-se o que se denomina elementos de processamento (PEs), com diferentes velocidades. Um ou mais PEs podem ser combinados para criar uma máquina mono ou multiprocessada. Analogamente, uma ou mais máquinas podem ser combinadas para criar um recurso do *grid*. Assim, os recursos podem ser sistemas monoprocessados, multiprocessadores de memória compartilhada (SMP), ou *cluster* de computadores, em que a memória é distribuída. Os recursos monoprocessados e os multiprocessados de memória compartilhada são tipicamente gerenciados por um sis-

tema operacional de tempo compartilhado, que usam alguma política de escalonamento para prover um ambiente multitarefas aos seus usuários. Os sistemas multiprocessados de memória distribuída são gerenciados por sistemas de filas chamados escalonadores de espaço compartilhado. Tais sistemas usam políticas de alocação de recursos como FCFS (*First-Come,First-Served*), SJFS (*Shortest-Job-First-Served*), etc.

A especificação da plataforma computacional e das tarefas que executarão no *grid* é realizada, portanto, mediante a codificação de um programa em Java, que faça uso das classes e métodos que permitem modelar as aplicações e recursos. A interação entre as entidades também é feita por meio de métodos específicos, que abstraem o conceito de simulação orientada a eventos.

2.5.3 Implementação e documentação

O GridSim é implementado em linguagem Java, no topo da biblioteca de simulação SimJava, utilizando o paradigma de programação orientada a objetos. A ferramenta implementa diversas classes que modelam componentes de sua arquitetura, além de classes e métodos úteis na parte estatística (geração de números aleatórios, extração de estatísticas sobre o uso e estado dos recursos), que facilitam a modelagem dos dados de entrada e a interpretação dos resultados.

Cada componente da *toolkit* do GridSim é executado por uma *thread* distinta. Este mecanismo de execução *multithreaded* procura conferir uma maior escalabilidade ao simulador. Outro aspecto importante é a existência de uma interface gráfica denominada *Visual Modeler* [34], cujo objetivo é prover uma interface amigável para criar e modificar facilmente diferentes modelos de simulação. Os modelos criados via interface gráfica podem ser gravados em arquivos XML, para posterior edição ou alteração.

A ferramenta Gridsim é *opensource* e apresenta uma vasta documentação. Informações podem ser obtidas nas publicações sobre o GridSim e suas extensões (suporte a *grids* de dados e interface gráfica), e na sua página oficial na internet [35]. Através deste *site*, o código fonte do simulador pode ser obtido, bem como exemplos de modelos de simulação e as publicações.

2.5.4 Aplicações

Os trabalhos que fazem uso do GridSim exploram especialmente o estudo do comportamento políticas de escalonamento estabelecidas por escalonadores distintos. No entanto, alguns trabalhos destacam-se por explorar aspectos da rede de interconexão do *grid*. Algumas das principais aplicações da ferramenta são apresentadas a seguir:

- Em [36] é apresentado um estudo sobre a provisão de qualidade de serviço (QoS) em um ambiente de *grid*. Para isso, é proposto um escalonador de tarefas que leva em conta o estado da rede em suas decisões. Os experimentos dessa nova estratégia foram realizados no GridSim.

- Em [33] o GridSim é utilizado como suporte ao estudo de estratégias de gerenciamento de dados e réplicas em *grids* de dados. Para tornar este estudo viável, os autores do trabalho propõem também uma extensão atualmente incorporada ao GridSim, específica para a modelagem de *grids* de dados.

- Em [37] é estudada a influência da inclusão de serviços diferenciados de rede (tráfego de fundo, roteamento, coleta de informações sobre a rede, etc) na simulação de *grids* computacionais. Estes serviços foram incorporados ao GridSim, com o objetivo de auxiliar o suporte à qualidade de serviço.

2.6 MicroGrid

O MicroGrid [38, 39] foi proposto inicialmente como um simulador, mas os seus princípios e implementações atuais claramente o tornam pertencente à classe dos emuladores de *grid*. Esta ferramenta foi criada para facilitar o estudo da interação entre aplicações, *middleware*, recursos e redes em um ambiente de *grid*

O Microgrid emula um ambiente de *grid* virtual, modelando a rede, os recursos e os serviços de informação de forma transparente. Com isso, objetiva-se proporcionar meios para a realização de experimentos controláveis e reprodutíveis de gerenciamento de recursos dinâmicos, uma atividade crítica no exame de ambientes de recursos compartilhados propostos para cenários futuros dos *grids* computacionais [40].

2.6.1 Arquitetura do MicroGrid

A funcionalidade básica do MicroGrid é simular o comportamento de aplicações em ambiente de *grid* virtual. A ferramenta suporta aplicações que fazem uso da infra-estrutura de *middleware* do Globus Grid. A arquitetura básica do MicroGrid é mostrada na figura 2.11, e cada componente na figura corresponde a um dos maiores desafios na construção de um *grid* virtual fiel a um *grid* real.

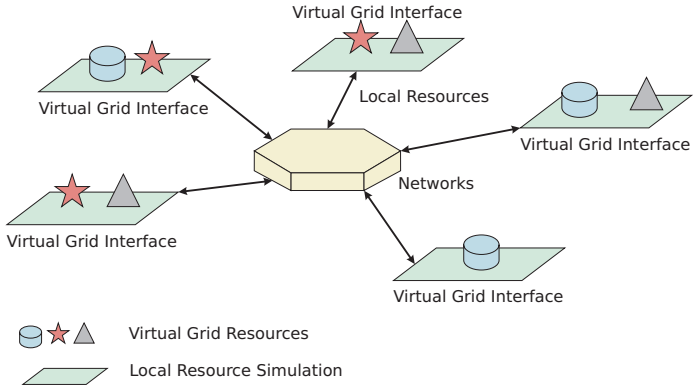


Figura 2.11: Diagrama do MicroGrid: os recursos locais provêm um ambiente de recursos em um *grid* virtual e o simulador de rede captura a interação entre os recursos locais virtualizados

Estes desafios constituem, portanto, a base da arquitetura da ferramenta. O primeiro deles, a virtualização, provê a ilusão de um ambiente de *grid* virtual. Para isso, o MicroGrid monitora e intercepta todos os usos diretos dos recursos ou serviços de informação feitos pelas aplicações. Em particular, é necessário intermediar todas as operações que identificam recursos pelo nome ou utilizam informações obtidas sobre eles. A seguir, são descritos os dois níveis de virtualização realizados pela ferramenta.

Virtualização de recursos : o MicroGrid virtualiza processamento, memória, rede, discos e quaisquer outros recursos de um *host* virtual que sejam usados pelo sistema. Cada *host* virtual é mapeado para um *host* real empregando-se uma tabela de conversão de endereços IP virtuais para endereços IP reais. Todas as chamadas de biblioteca relevantes são interceptadas e mapeadas do ambiente virtual para o real, usando esta tabela. Estas chamadas incluem `gethostbyname`, `bind`, `send`, `receive` e `process creation`. Pela interceptação das chamadas, uma aplicação real pode ser executada

de forma transparente em um *host* virtual, cujo endereço IP e *hostname* são virtuais.

Virtualização dos serviços de informação : os serviços de informação são críticos para a descoberta de recursos e para o uso eficiente deles em um *grid*. Uma vez que o *middleware* suportado pelo MicroGrid é o Globus, o problema recai na virtualização do GIS (*Globus Information Service*). Esta virtualização mantém a forma como as informações são solicitadas, obtidas e manipuladas pelas aplicações. Além disso, ela facilita a identificação e organização das entidades do *grid* virtual.

O segundo desafio diz respeito à coordenação global dos recursos virtuais modelados. Uma vez que as capacidades e características dos recursos virtuais e as dos recursos reais que os emularão podem ser diferentes, deve ser realizado um balanceamento. Este procedimento é realizado, portanto, com base na relação entre as características dos recursos reais e a dos recursos virtuais neles mapeados, e resulta no cálculo de uma taxa de simulação (TS) para cada recurso. Por exemplo, se um dado recurso real despende x unidades de tempo para executar uma dada aplicação, quando virtualizado o mesmo recursos despenderá $x.TS$ unidades de tempo. Dada essa diferença, torna-se necessário virtualizar também o tempo, com base na taxa de simulação calculada.

Por fim, o terceiro desafio refere-se à simulação dos recursos. Dois tipos de recursos são considerados: recursos de computação e de comunicação. A simulação de recursos de computação foca-se na simulação da capacidade de processamento dos *hosts*. A simulação do meio de comunicação, ou seja, da rede é realizada pelo emulador de rede chamado MaSSF, que provê módulos para a modelagem detalhada de protocolos como IP, TCP, UDP, OSPF e BGP4. Além disso, este emulador de rede permite modelar parâmetros como largura de banda, latência e topologia de rede.

2.6.2 Modelagem da plataforma do *grid* e das aplicações

A especificação do ambiente de *grid* a ser emulado é efetuada pela modelagem dos recursos virtuais e reais. Para isso, são utilizados arquivos de configuração, que utilizam uma linguagem própria para a sua formatação. No arquivo `mygrid.dml` é descrita a rede virtual. Um grande número de parâmetros podem ser configurados. A topologia de rede

é definida por uma coleção de redes interligadas por roteadores. Cada rede é formada por uma coleção de *links*. E cada *link* é caracterizado pelas entidades (ou nós) que interligam, largura de banda e latência. Para cada roteador, é possível definir as suas interfaces (quantidade e largura de banda associada a cada uma delas), o protocolo e a tabela de roteamento que utiliza.

As máquinas reais são identificadas no arquivo `machine.dml`, em termos dos seguintes atributos: endereço IP ou *hostname*, sistema operacional e número de processadores. No arquivo `mygrid.phost` são especificadas as capacidades dos recursos reais em termos de poder de processamento (em MHz ou FLOPS) e quantidade de memória. E, finalmente, no arquivo `mygrid.vhost` são definidas as capacidades dos recursos virtuais, em termos de capacidade de processamento, nome virtual, quantidade de memória. Além disso, é possível indicar neste último arquivo, para cada máquina virtual, se ela executará um *gatekeeper* ou não.

As aplicações que executarão no *grid* virtual, por sua vez, são aplicações reais, construídas segundo a *toolkit* Globus [41]. Por meio da virtualização realizada pelo MicroGrid, as aplicações reais enxergam apenas os recursos virtuais modelados, permitindo a avaliação do comportamento das mesmas sob diferentes cenários. Uma vez construídas as aplicações reais, especifica-se o conjunto de aplicações que deverão ser executadas no *grid* virtual. Para cada aplicação deve-se indicar o caminho do código binário, horário em que ela será submetida, o *host* que originará a submissão e os parâmetros de entrada da aplicação. Além disso, é possível definir os arquivos para os quais serão redirecionadas as saídas e mensagens de erro das aplicações.

2.6.3 Implementação e documentação

O MicroGrid é implementado em C, utiliza arquivos XML para especificação das aplicações Globus, e emprega arquivos formatados para a identificação dos recursos reais e o seu mapeamento para os recursos virtuais do *grid* emulado. Para o funcionamento do emulador, é necessário a instalação e configuração da biblioteca de passagem de mensagem MPI, pois uma vez que a emulação envolve recursos reais distribuídos, a execução do MicroGrid é paralela. Por meio do MPI, são inicializados os *daemons* de controle da execução

das aplicações no *grid* virtual. Além disso, os *daemons* são responsáveis pela inicialização dos *gatekeepers* indicados nos arquivos de configuração.

A ferramenta é *opensource*, e sua documentação, embora não muito detalhada, descreve os procedimentos básicos de instalação, dependências, estrutura dos arquivos de configuração, forma de execução do emulador e como usá-lo conjuntamente com o Globus Grid. Na página oficial do projeto [42], podem ser obtidos o código fonte do MicroGrid, e as publicações sobre ele.

2.6.4 Aplicações

O MicroGrid é utilizado fundamentalmente para emulação de coleções arbitrárias de recursos e redes, como forma de viabilizar o estudo de *testbeds* sob condições controláveis e reprodutíveis, ou até mesmo para estudar a eficácia de ambientes de alto desempenho disponíveis atualmente. O MicroGrid suporta a execução de aplicações para *grids* não alteradas, que podem ser escritas em C/C++ (com ou sem MPI), Perl e Python. Estas aplicações usualmente fazem uso do *middleware* de *grid* oferecido pelo Globus, tornando o MicroGrid uma importante ferramenta de avaliação de *grids* cuja infra-estrutura é baseada neste sistema.

3 Comparação entre as ferramentas

O processo de comparação entre as ferramentas abordadas é feito com base nos quesitos mais relevantes no projeto de um simulador ou emulador de grades computacionais. Primeiramente, na tabela representada pela figura 3.1, é apresentada uma comparação quanto aos aspectos básicos de implementação, funcionamento, portabilidade, documentação e licença de uso.

A questão de linguagem de implementação é importante quando se avalia a facilidade de extensibilidade da ferramenta, ou seja, o quão fácil a ferramenta pode ser modificada ou ter suas funcionalidades ampliadas. Além disso, a linguagem escolhida está associada à portabilidade do simulador, à velocidade de execução das simulações, à facilidade de construção de interface gráfica, e à forma de acesso aos mesmos. O GangSim, por exemplo, consiste de um conjunto de *scripts* Perl, uma linguagem interpretada de aprendizado relativamente difícil e de lenta execução quando comparada a linguagens como C ou Java. No entanto, a possibilidade de se utilizar seus *scripts Perl* como *scripts* CGI (*Common Gateway Interface*), faz com que o GangSim possa ser acessado remotamente via interface Web, a qual é facilmente construída. Em outro extremo está o SimGrid, construído em linguagem C, o que favorece a sua manutenção e velocidade de execução. No entanto, torna-se difícil a construção de uma interface gráfica em sua linguagem nativa que não o torne dependente de plataforma.

O quesito interface influencia diretamente a facilidade de uso da ferramenta. Entre os simuladores e emuladores analisados, o OptorSim apresenta a interface mais bem projetada. A sua interface em modo texto mostra detalhadamente o andamento da simulação,

e após o final desta, apresenta um conjunto amplo de estatísticas sobre o uso dos recursos. Estas mesmas informações podem ser obtidas pela interface gráfica provida pelo OptorSim, que também permite a visualização dos resultados sob a forma de gráficos. Estas funcionalidades facilitam fortemente o processo de análise de saída da simulação.

A documentação também está relacionada à facilidade de uso. Uma boa documentação é fundamental, em nível de usuário, para o correto entendimento do uso da ferramenta. Em nível de desenvolvedor, a documentação auxilia a compreensão do comportamento do *software*, sendo essencial para que se possa estender ou modificar as suas funcionalidades. Algumas ferramentas são muito bem documentadas, o que estimulou e facilitou a construção de extensões que ampliaram as suas funcionalidades. Neste quesito, o GangSim, se destaca negativamente, por não oferecer nenhuma documentação detalhando como instalá-lo e utilizá-lo.

A licença de uso diz respeito à acessibilidade dos usuários à ferramenta. Todas as ferramentas abordadas neste estudo são de código livre, o que significa que podem ser utilizadas e alteradas livremente. De fato, não há soluções proprietárias conhecidas, que se proponham a oferecer um ambiente de simulação completo e fácil de usar.

O mecanismo de execução, por sua vez, tem relação direta com a escalabilidade da ferramenta. O modo de execução *multithreaded* confere uma maior escalabilidade ao simulador. Neste quesito, destaca-se o GridSim, uma vez que ele faz uso do pacote de simulação *SimJava*, que possui uma versão para execução distribuída. O uso do *Distributed SimJava* em conjunto com uma máquina virtual Java também distribuída (como o Jessica2 [43], o AJVM [44], entre outros), viabiliza a execução paralela das simulações. O emulador MicroGrid implica em uma execução paralela. Isto, no entanto, não é feito com o intuito de prover maior escalabidade à ferramenta. O paralelismo é um pré-requisito para a emulação de sistemas distribuídos (considerando-se mais de uma máquina empregada na emulação).

Quanto à criação do modelo de simulação, três quesitos são analisados:

- Modelagem da lógica de funcionamento da simulação, isto é, da forma como o gerenciamento dos recursos e serviços do *grid* é realizado. Isto envolve, por exemplo, a definição do algoritmo de escalonamento de tarefas, o gerenciamento das informações coletadas sobre os recursos do *grid*, entre outras atividades, que poderão ser

modeladas segundo o interesse do usuário e das capacidades do simulador. Alguns simuladores pré-determinam uma estrutura básica para a modelagem dos serviços prestados pelo *grid*, definindo quais serviços devem existir no sistema, restando ao usuário ajustar algumas configurações destes serviços.

- Modelagem da plataforma do *grid*, que consiste na definição de quais componentes estarão presentes no sistema e sua disposição, e na descrição da topologia de rede que interliga tais componentes. Em geral, especifica-se o conjunto de *hosts* que executarão as tarefas (definindo-se também as suas capacidades de processamento e armazenamento), e tais *hosts* podem estar englobados dentro de *sites*. Determina-se ainda a rede (modelando os *links* e o roteamento inter e intra *sites*).

- Modelagem das tarefas, que envolve a definição de como as tarefas são geradas (taxa com que são geradas e os *hosts* ou usuários originadores) e das suas características, como tamanho computacional, arquivos de entrada (quando necessários), etc.

A comparação entre as ferramentas em termos dos três quesitos apresentados, é mostrada nas tabelas indicadas nas figuras 3.2 e 3.3.

Por fim, nas tabelas apresentadas pela figuras 3.4 e 3.5, com base nas análises feitas no capítulo 2, os pontos fortes, deficiências e principais funcionalidades das ferramentas são apresentados. Dois aspectos funcionais das ferramentas são abordados individualmente: possibilidade de gerar tráfego e computação de fundo e integração de ferramentas ou informações externas. Ambos estão intimamente relacionados com a questão de realismo da simulação. O primeiro permite representar mais fielmente cenários reais, nos quais os recursos existentes nos *grids* podem ser utilizados localmente, caracterizando um uso não associado às tarefas do ambiente de *grid*. A integração com ferramentas externas, por sua vez, permite adicionar funcionalidade extras, visando aumentar o grau de realismo das simulações, além de auxiliar no processo de integração de informações externas ao ambiente de simulação.

O SimGrid admite a modelagem de tráfego e computação de fundo através da especificação de *traces*, os quais basicamente indicam flutuações no desempenho dos recursos. Cada *trace*, nesta ferramenta, compreende um arquivo no qual é indicado um padrão de

variação da disponibilidade de um recurso. Esta variação é representada pela fração indisponível do recurso ao longo de um intervalo de tempo periódico. Para a correta modelagem dos *traces* normalmente são utilizadas informações obtidas do *grid* que se deseja simular, via ferramentas de monitoração, como NWS, NetPerf, entre outros. Logo, a modelagem do tráfego e computação de fundo é habilitada por informações coletadas por ferramentas externas.

De forma similar, o OptorSim permite modelar variações na disponibilidade dos enlaces. Para isso, cria-se *traces*, nos quais são indicadas as variações temporais da disponibilidade dos *links*. Assim como o SimGrid, as informações necessárias para especificar os *traces* são tipicamente obtidas por meio de ferramentas de monitoração.

O Bricks, por sua vez, é mais flexível quanto à integração de ferramentas externas. Este simulador permite a substituição de cada componente de sua SPI (*Service Provider Interface*) por um módulo equivalente que implemente uma interface com uma ferramenta externa. Assim, algumas das funções dos módulos padrão do simulador podem ser desempenhadas por ferramentas externas. Isto é especialmente útil para a validação das simulações realizadas. A única (e atual) versão do Bricks implementa a integração com o sistema NWS, que monitora e realiza predições de sistemas distribuídos baseado em informações passadas sobre a utilização dos recursos.

Características \ Ferramentas	SimGrid	Bricks	OptorSim	GangSim	GridSim	MicroGrid
Tipo de ferramenta	Simulador	Simulador	Simulador	Simulador	Simulador	Emulador
Linguagem de implementação	C	Java	Java	Perl	Java	C
Plataformas suportadas	Linux, MacOs, Windows	Qualquer S.O com suporte à JVM	Qualquer S.O com suporte à JVM	Linux	Qualquer S.O com suporte à JVM	Linux/Alpha em um Grid Globus
Interface	Modo texto	Modo texto	Modo texto e GUI	Modo texto	Modo texto e GUI (limitada)	Modo texto
Documentação	Muito boa	Regular	Boa	Fraca	Muito boa	Regular
Licença	Opensource	Opensource	Opensource	Opensource	Opensource	Opensource
Mecanismo de simulação	Sequencial	Sequencial	Multithreaded	Sequencial	Multithreaded	Paralelo

Figura 3.1: Tabela comparativa dos quesitos básicos das ferramentas

Características \ Ferramentas	SimGrid	Bricks	OptorSim
Arquitetura do *grid*	Definida pelo usuário	Definida pela ferramenta	Definida pela ferramenta
Especificação da simulação/emulação	**Codificação em C e configuração de arquivo XML** (construção de um programa em C, que implementa os processos especificados no arquivo deployment.xml)	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (modelagem é realizada de forma indireta pela configuração da plataforma do *grid* e especificação das tarefas)	**Arquivos de configuração escritos em linguagem definida pela ferramenta** (indica-se a política de escalonamento adotada, tipo de usuário do *grid*, estratégia de otimização de réplicas, existência ou não de tráfego de fundo, etc.)
Modelagem da plataforma do grid	**Configuração de arquivo XML** (especificação dos recursos de computação, *links* de comunicação e roteamento entre os nós no arquivo platform.xml)	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (indica-se para cada componente da arquitetura do *grid*, os seus atributos e quantidade. Modela-se a topologia de rede pela especificação da interconexão dos nós)	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (modela-se cada *site* em função dos seus CEs e SEs e uma matriz representa a conectividade e largura de banda entre os *sites*)
Modelagem das tarefas	**Codificação em C e/ou configuração em arquivo XML** (A especificação da quantidade e atributos das tarefas pode ser feita no programa em C e/ou no arquivo deployment.xml)	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (modelagem feita via especificação dos usuários do *grid*, que envolve a definição da taxa de submissão de tarefas e do tamanho computacional das mesmas)	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (modela-se a coleção de arquivos acessados por cada tarefa e a distribuição inicial deles no grid, o seu tamanho computacional e outras características)

Figura 3.2: Tabela comparativa dos princípios de modelagem adotados pelas ferramentas SimGrid, OptorSim e Bricks

Características \ Ferramentas	GangSim	GridSim	MicroGrid
Arquitetura do *grid*	Definida pela ferramenta	Definida parcialmente pelo usuário	Definida pelo usuário, segundo a concepção do Globus Grid
Especificação da simulação/emulação	**Arquivo de configuração escrito em linguagem definida pela ferramenta ou via interface WEB** (especificação da política de alocação de recursos nas VOs)	**Codificação do programa de simulação em Java** (modela-se a interação entre os recursos criados, os usuários que gerarão as tarefas, a submissão das tarefas nos recursos, a estratégia de escalonamento utilizada, etc.)	**Codificação de aplicações reais para o Grid Globus e uso de arquivo de configuração em linguagem definida pela ferramenta** (O modelo resulta da execução programada, segundo informações constantes no arquivo de configuração, das aplicações reais nos recursos virtualizados)
Modelagem da plataforma do grid	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (modelagem é feita indicando-se os VOs e *sites*. Para os sites, define-se o seu poder computacional das máquinas e a sua capacidade de armazenamento. A interconexão entre os sites e Vos também é especificada e caracterizada)	**Criação dos recursos no programa Java** (especificação dos componentes básicos da arquitetura do *grid*)	**Arquivos de configuração escritos em linguagem definida pela ferramenta** (modela-se o mapeamento entre recursos reais e virtuais, os atributos destes, indica-se os possíveis *gatekeepers*. Modela-se a topologia de rede, indicando os *links* e seus atributos, as subredes, e o roteamento dentro das subredes e entre elas)
Modelagem das tarefas	**Arquivo de configuração escrito em linguagem definida pela ferramenta** (indicação das taxas com que as tarefas são geradas pelas VOs e submetidas aos *sites*, e as características de cada tarefa)	**Codificação das tarefas no programa Java** (utilizando-se a classe própria para isso, chamada Gridlet. As informações sobre o tamanho computacional, operações de E/S, tamanho dos dados de entrada e de saída e o originador da tarefa podem ser gerados aleatoriamente por meio da classe GridSimRandom)	**Codificação de aplicações reais para o Grid Globus e uso de arquivo de configuração em linguagem definida pela ferramenta** (para cada aplicação identifica-se o *host* virtual que submeterá a aplicação, os parâmetros de entrada desta, e o horário de submissão)

Figura 3.3: Tabela comparativa dos princípios de modelagem adotados pelas ferramentas GridSim, GangSim e MicroGrid

Características \ Ferramentas	SimGrid	Bricks	OptorSim
Pontos fortes	- Flexibilidade; - Facilidade de depuração; - Reusabilidade de código; - Modelagem de tráfego e computação de fundo; - Boa documentação.	- Facilidade de uso; - Extensibilidade da API; - Flexibilidade na integração com ferramentas externas; - Portabilidade.	- Facilidade de uso; - Utilidades para a análise dos resultados; - GUI bem projetada; - Modelagem de tráfego de fundo; - Portabilidade.
Deficiências	- Dificuldade de uso; - Interface pouco amigável; - Ausência de funcionalidades que auxiliem a modelagem da plataforma; - Pouco escalável.	- Interface pouco amigável; - Formatação dos resultados dificulta a análise de saída; - Modelagem da plataforma do *grid* limitada - Falta de suporte e desenvolvimento do simulador.	- Aplicação restrita; - Pouca flexibilidade na modelagem da plataforma do *grid;* - Modelagem da rede dentro dos *sites* é limitada; - Modelagem do padrão de submissão das tarefas é limitada.
Funcionalidades	- Facilita o estudo de estratégias de escalanamento de tarefas; - Provê componente que auxilia o desenvolvimento de aplicações reais para *grids* (GRAS); - Provê componente que auxilia o desenvolvimento de programas MPI (SMPI); - Provê componente para a avaliação de tarefas paralelas em sistemas distribuídos (SIMDAG).	- Voltado ao estudo de algoritmos de escalonamento de tarefas; - Permite a integração de ferramentas externas ao simulador; - Estabelece um modelo de *grid*, que facilita o análise das interações cliente-servidor;	- Voltado ao estudo de algoritmos de escalonamento de tarefas que empregam o conceito de replicação; - Adota um modelo de *grid*, que facilita a avaliação de *grids* de dados; - Provê interface gráfica para facilitar a construção de simulações e sua análise;
Computação/tráfego de fundo	Sim / Sim	Não / Não	Não / Sim
Incorporação de ferramentas ou informações externas	Traces (modelados a partir de ferramentas de monitoração)	Ferramentas de monitoração, benchmarking e predição.	Traces (modelados a partir de ferramentas de monitoração)

Figura 3.4: Tabela representativa dos pontos fortes, deficiências e funcionalidades do SimGrid, Bricks e OptorSim

Características \ Ferramentas	GangSim	GridSim	MicroGrid
Pontos fortes	- Adiciona o conceito de VO às simulações; - Interface *Web* facilita e simplifica o acesso e uso do simulador.	- Grande flexibilidade; - Permite uma modelagem detalhada dos componentes do sistema; - Boa documentação; - Escalabilidade; - Portabilidade.	- Maior grau de realismo; - Virtualização dos recursos e serviços é feita de forma transparente ao usuário; - Possibilidade de avaliação de aplicações reais.
Deficiências	- Falta de documentação; - Apesar da interface *Web*, o uso é complicado; - Difícil entendimento da forma de funcionamento do simulador.	- Ausência de interface amigável; - Necessidade de codificação do programa de simulação; - Modelagem da plataforma do grid e das tarefas deve ser feita no próprio programa de simulação	- Pouco escalável; - Necessidade de uso de diversos recursos reais. - Não portável. - Documentação pouco detalhada
Funcionalidades	- Voltado para a avaliação do impacto das políticas de alocação de recursos adotadas por *sites* e Vos; - Possibilidade de integrar instâncias de simulação com a ferramenta de monitoração VO-Ganglia - Acesso remoto via interface Web	- Possibilita o estudo de diversos aspectos relacionados a um *grid* ou sistemas paralelos, como *clusters*; - Permite a modelagem de aplicações baseadas em diferentes modelos de paralelismo; - Possui extensões que facilitam o estudo de *grids* de dados e aspectos específicos da rede do *grid*, como provisão de QoS.	- Permite a criação de um ambiente computacional distribuído virtual, para o estudo da sua escalabilidade, resposta a falhas, e outros comportamentos. - Viabiliza o estudo de aplicações reais em um ambiente virtual. - Permite um alto grau de detalhamento dos recursos da rede
Computação/tráfego de fundo	Não / Não	Não / Sim	Não / Não
Incorporação de ferramentas ou informações externas	Ferramenta de monitoração VO-Ganglia	Não	Não

Figura 3.5: Tabela representativa dos pontos fortes, deficiências e funcionalidades do SimGrid, Bricks e OptorSim

4 Conclusões

Neste estudo foram apresentadas e comparadas algumas das principais ferramentas de simulação de grades computacionais. A partir da descrição dos simuladores abordados, foi possível identificar e delimitar os seus mecanismos de funcionamento, aplicabilidade, limitações, e pontos fortes.

Ao traçar estas propriedades de cada simulador, e compará-los entre si, foi possível determinar os requisitos que devem ser atendidos por um ambiente de simulação completo. Estes requisitos dizem respeito aos seguintes aspectos:

- Linguagem de programação utilizada na codificação do simulador, que deve priorizar a portabilidade, escalabilidade, e facilidade de manutenção do código.
- Interface, que deve ser projetada de forma a facilitar o uso do simulador pelos usuários. A existência de uma interface gráfica é algo de grande importância, não apenas por facilitar o uso da ferramenta, mas também por auxiliar a análise da saída da simulação, ao exibir os resultados de forma mais clara e compreensível.
- Documentação, que deve auxiliar no entendimento da forma de funcionamento e uso do simulador, bem como descrever aspectos de sua implementação, facilitando a sua extensibilidade.
- Mecanismo de execução, que, preferencialmente, deve ser paralelo, distribuído ou *multithreaded*, de forma a conferir uma maior escalabilidade à simulação e reduzir o seu tempo de execução.
- Modelagem do programa de simulação. O simulador deve prover mecanismos que facilitem a modelagem da lógica de funcionamento da simulação, da plataforma do

grid e das tarefas. Além disso, o simulador deve permitir uma descrição detalhada dos componentes do *grid* (usuários, *hosts*, rede, etc), tornando a simulação mais realista.

- Geração de tráfego e computação de fundo, e agregação de ferramentas e informações externas. Os dois primeiros ítens conferem um tratamento mais realista do ambiente computacional do *grid*, enquanto o último permite o uso de ferramentas ou dados reais, também com a finalidade de aumentar o grau de confiabilidade da simulação.

Este trabalho, portanto, permitiu a determinação dos parâmetros necessários para o desenvolvimento de um ambiente de simulação que satisfaça tais requisitos, auxiliando o estudo de diversos aspectos de um *grid*, como: formas adequadas de utilização de seus recursos, avaliação de seu desempenho sob diferentes cenários, análise do comportamento de algoritmos em diferentes tipos de grids, entre outros.

Referências Bibliográficas

1 FOSTER, I.; KESSELMAN, C. High-performance schedulers. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 279–309, 1999. Disponível em: <http://www.globus.org/alliance/publications/papers/chapter2.pdf>.

2 QUETIER, B.; CAPPELLO, F. A survey of grid research tools: simulators, emulators and real life platforms. In: *17th IMACS World Congress (IMACS 2005)*. Paris, França: [s.n.], 2005. Disponível em: <www.lri.fr/quetier/papiers/imacs2005-survey.pdf>.

3 SILVA, D. P. da; CIRNE, W.; BRASILEIRO, F. V. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In: *Euro-Par 2003: International Conference on Parallel and Distributed Computing*. [s.n.], 2003. Disponível em: <http://www.ourgrid.org/twiki-public/pub/OG/OurPublications/WQReplication.pdf>.

4 CASANOVA, H. et al. Heuristics for scheduling parameter sweep applications in grid environments. In: *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*. Washington, DC, Estados Unidos: IEEE Computer Society, 2000. p. 349. ISBN 0-7695-0556-2. Disponível em: <ieeexplore.ieee.org/iel5/8971/28471/01271423.pdf>.

5 SANTOS-NETO, E. *Escalonamento de Aplicações que processam Grande Quantidade de Dados em Grids Computacionais*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, Março 2004. Disponível em: <http://www.lsd.ufcg.edu.br/ẽlizeu/articles/StorageAffinity.v5.pdf>.

6 SANTOS-NETO, E. et al. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In: GMBH, S.-V. (Ed.). *10th Workshop on Job Scheduling Strategies for Parallel Processing*. New York: Lecture Notes on Computer Science, 2005. v. 3277, p. 210–232. Disponível em: <http://www.cs.huji.ac.il/f̃eit/parsched/p-04-11.ps.gz>.

7 CASANOVA, H. Simgrid: A toolkit for the simulation of application scheduling. In: *1st International Symposium on Cluster Computing and the Grid (CCGrid 2001)*. [s.n.], 2001. Disponível em: <http://grail.sdsc.edu/papers/simgrid_ccgrid01.ps.gz>.

8 BUYYA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, v. 14, p. 1175–1220, Novembro-Dezembro 2002. Disponível em: <http://www.gridbus.org/papers/gridsim.pdf>.

9 DUMITRESCU, C. L.; FOSTER, I. Gangsim: a simulator for grid scheduling studies. In: *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*. Washington, DC, Estados Unidos: IEEE Computer Society, 2005. v. 2, p. 1151–1158. ISBN 0-7803-9074-1. Disponível em: <http://people.cs.uchicago.edu/ cldumitr/docs/GangSim.pdf>.

10 TAKEFUSA, A. et al. Overview of a performance evaluation system for global computing scheduling algorithms. In: *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, Estados Unidos: IEEE Computer Society, 1999. p. 11. ISBN 0-7695-0287-3. Disponível em: <http://ninf.apgrid.org/papers/hpdc99takefusa/hpdc99takefusa.pdf>.

11 BELL, W. et al. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, p. 403–416, 2003. Disponível em: <https://edms.cern.ch/file/392802/1/OptorSimIJHPCA2003.ps>.

12 CARON, E.; GARONNE, V.; TSAREGORODTSEV, A. *Evaluation of Meta-scheduler Architectures and Task Assignment Policies for High Throughput Computing.* [S.l.], maio 2005. Also available as LIP Research Report 2005-27. Disponível em: <ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-5576.pdf>.

13 GAY, J.-S.; CANIOU, Y. Simbatch: an api for simulating and predicting the performance of parallel resources and batch systems. In: . [s.n.], 2006. Also available as INRIA Research Report 6040. Disponível em: <https://hal.inria.fr/inria-00115880>.

14 CASANOVA, H.; LEGRAND, A.; MARCHAL, L. Scheduling distributed applications: the simgrid simulation framework. In: *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*. IEEE Computer Society, 2003. Disponível em: <ieeexplore.ieee.org/iel5/8544/27003/01199362.pdf>.

15 QUINSON, M. Gras: A research & development framework for grid and p2p infrastructures. In: IASTED (Ed.). *18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*. [s.n.], 2006. Disponível em: <http://hal.inria.fr/inria-00108389>.

16 SIMGRID. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://simgrid.gforge.inria.fr/>.

17 JR., J. N. F.; JR., A. M.; OLIVEIRA, L. J. de. Avaliação de algoritmos de escalonamento em grids para diferentes configurações de ambiente. *Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance 2007)*, Julho 2007. Disponível em: <http://www.dcce.ibilce.unesp.br/spd/pubs/EEJNelson.pdf>.

18 RENARD, H.; ROBERT, Y.; VIVIEN, F. Data redistribution algorithms for heterogeneous processor rings. *Int. Journal of High Performance Computing Applications*, v. 20, n. 1, p. 31–43, 2006. Disponível em: <http://hal.inria.fr/inria-00070785/en/>.

19 AIDA, K. et al. Performance evaluation model for job scheduling in a global computing system. In: *7th IEEE International Symposium on High Performance Distributed Computing*. [s.n.], 1998. p. 352–353. Disponível em: <http://ninf.apgrid.org/bricks/contents/hpdc99.html>.

20 AIDA, K. et al. Performance evaluation model for scheduling in global computing systems. *International Journal of High Performance Computing Applications*, Sage Publications, Inc., Thousand Oaks, CA, Estados Unidos, v. 14, n. 3, p. 268–279, 2000. ISSN 1094-3420.

21 BRICKS. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://ninf.apgrid.org/bricks/>.

22 TAKEFUSA, A. et al. A study of deadline scheduling for client-server systems on the computational grid. In: *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2001. p. 406. Disponível em: <http://portal.acm.org/citation.cfm?id=876538>.

23 BOSIO, D. et al. Eu datagrid data management services. In: *UK e-Science All Hands Conference*. Nottingham, Reino Unido: [s.n.], 2003. Disponível em: <http://citeseer.ist.psu.edu/bosio03eu.html>.

24 ERNEMANN, C.; HAMSCHER, V.; YAHYAPOUR, R. Economic scheduling in grid computing. In: *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*. Londres, Reino Unido: Springer-Verlag, 2002. p. 128–152. ISBN 3-540-00172-7. Disponível em: <http://portal.acm.org/citation.cfm?id=689702>.

25 OPTORSIM. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html>.

26 CAMERON, D. G. et al. Evaluating scheduling and replica optimisation strategies in optorsim. In: *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*. Washington, DC, Estados Unidos: IEEE Computer Society, 2003. p. 52. ISBN 0-7695-2026-X. Disponível em: <http://portal.acm.org/citation.cfm?id=951948.952041>.

27 CAMERON, D. G. et al. Uk grid simulation with optorsim. In: *e-Science All-Hands Meeting*. Nottingham, Reino Unido: [s.n.], 2003. Disponível em: <https://edms.cern.ch/file/404058/1/allhands2003paper.ps>.

28 FOSTER, I. T. The anatomy of the grid: Enabling scalable virtual organizations. In: *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*. Londres, Reino Unido: Springer-Verlag, 2001. p. 1–4. ISBN 3-540-42495-4. Disponível em: <http://portal.acm.org/citation.cfm?id=699437>.

29 GANGSIM. *Página oficial do projeto, visitada em 12 de outubro de 2007.* Disponível em: <http://people.cs.uchicago.edu/~cldumitr/GangSim/>.

30 PARASHAR, M.; LEE, C. (Ed.). *The Grid Economy*, v. 93, n. 3 de *Special Issue on Grid Computing*, (Special Issue on Grid Computing, 3). New York, Estados Unidos: IEEE Press, 2005. Disponível em: <http://www.gridbus.org/papers/ieee-grideconomy.pdf>.

31 HOWELL, F.; MCNAB, R. Simjava: A discrete event simulationlibrary for java. In: SOCIETY FOR COMPUTER SIMULATION. *First International Conference on Web-based Modelling and Simulation.* San Diego, California, Estados Unidos, 1998. Disponível em: <http://citeseer.ist.psu.edu/136392.html>.

32 SULISTIO, A. et al. Constructing a grid simulation with differentiated network service using gridsim. In: *6th International Conference on Internet Computing (ICOMP'05).* Las Vegas, Estados Unidos: [s.n.], 2005. Disponível em: <http://www.gridbus.org/papers/gridsim_net.pdf>.

33 SULISTIO, A. et al. *A Toolkit for Modelling and Simulation of Data Grids with Integration of Data Storage, Replication and Analysis.* Melbourne, Australia, Novembro 2005. Disponível em: <http://citeseer.ist.psu.edu/730931.html>.

34 SULISTIO, A. et al. Visual modeler for gridmodelling and simulation (gridsim) toolkit. In: *3rd International Conference on Computational Science (ICCS 2003).* Springer Verlag Publications (LNCS Series), 2003. Disponível em: <http://www.gridbus.org/papers/vmgridsim.pdf>.

35 GRIDSIM. *Página oficial do projeto, visitada em 12 de outubro de 2007.* Disponível em: <http://www.gridbus.org/gridsim/>.

36 CAMINERO, A.; CARRIÓN, C.; CAMINERO, B. On the improvement of the network qos in a grid environment. In: *MCG '06: Proceedings of the 4th international workshop on Middleware for grid computing.* New York, NY, Estados Unidos: ACM Press, 2006. p. 18. ISBN 1-59593-581-9. Disponível em: <http://portal.acm.org/citation.cfm?id=1186675.1186695>.

37 SULISTIO, A. et al. On incorporating differentiated levels of network service into gridsim. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 23, n. 4, p. 606–615, 2007. ISSN 0167-739X. Disponível em: <http://www.gridbus.org/papers/gridsim-fgcs.pdf>.

38 XIA, H. et al. The microgrid: Using emulation to predict application performance in diverse grid network environments. In: *Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (CLADE'04).* IEEE Press, 2004. Published in conjunction with the Thirteenth IEEE International Symposium on High-Performance Distributed Computing, Honolulu, Hawaii, June 2004. Disponível em: <http://vgrads.rice.edu/papers/vgrads2/attach/clade04xia.pdf>.

39 SONG, H. J. et al. The microgrid: A scientific tool for modeling computational grids. *Sci. Program.*, IOS Press, Amsterdam, Holanda, v. 8, n. 3, p. 127–141, 2000. ISSN 1058-9244. Disponível em: <http://www-csag.ucsd.edu/papers/MicroGrid-00.pdf>.

40 CONCURRENT SYSTEMS ARCHITECTURE GROUP, UNIVERSITY OF CALIFORNIA. *MicroGrid 2.4.6 User Guide*. 0.46. ed. San Diego, Estados Unidos, Dezembro 2004. Disponível em: <http://www-csag.ucsd.edu/projects/grid/mgrid2-user.pdf>.

41 FOSTER, I.; KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, p. 115–128, 1997. Disponível em: <ftp://ftp.globus.org/pub/globus/papers/globus.pdf>.

42 MICROGRID. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://www-csag.ucsd.edu/projects/grid/microgrid.html>.

43 JESSICA2. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://i.cs.hku.hk/ wzzhu/jessica2/index.php>.

44 AJVM. *Página oficial do projeto, visitada em 12 de outubro de 2007*. Disponível em: <http://www.csc.uvic.ca/ jbaldwin/diva/>.