Planejamento de Experimentos com várias Replicações em Paralelo em Grades Computacionais

Lourenço Alves Pereira Júnior
ljr@icmc.usp.br

Orientador:
Prof. Dra. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC, USP, como parte dos requisitos para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Julho de 2010

Planejamento de Experimentos com várias Replicações em Paralelo em Grades Computacionais

Lourenço Alves Pereira Júnior

Agradecimentos

Agradeço à Deus pela inspiração de poder ter chegado até aqui.

Agradeço ao meu pai, Lourenço, por todo amor, carinho e dedicação que teve comigo nesta etapa de minha vida, e apesar de ele não estar mais entre nós para poder presenciar este passo de minha caminhada, se faz presente em nossos corações, meu, de minha família e de todos os amigos. Agradeço também à minha mãe, Aparecida, pelo seu amor, pelo apoio emocional, pela ajuda financeira e por tudo que tem feito para mim em todos estes anos. Gostaria de agradecer à minha amada minha irmã, Lílian, por confiar em mim, pelo carinho, amizade e compreensão. Agradeço à minha amada namorada, Cidinha, por me amar, me compreender, me ajudar e me fazer tão bem nas horas em que eu preciso.

Agradeço à minha orientadora profa. Dra. Sarita pela confiança, dedicação e ajuda proporcionada ao longo destes anos que trabalhamos juntos.

Gostaria de agradecer meus amigos "das antigas" que de uma forma ou de outra contribuem em minha vida. Agradeço aos amigos Júlio, Marcelo e Michelle pelas discussões técnicas, filosóficas e culturais. Aos colegas de laboratório Jonathan, Bruno, Thiago, Felipe, Roberto, Augusto, Tott, Lucas, Bert, Matheus, Maycon, Priscila, Edwin, Pedro, Douglas, Ricardo, Kenji, Mário, Paulo, e todos outros pelos ótimos momentos de trabalho, lazer e descontração.

Aos professores do grupo Paulo Sérgio, Monaco, Regina, Marcos, Edson e Kalinka que sempre nos ajudam compartilhando seus conhecimentos.

Aos funcionários do ICMC/USP, em especial aos funcionários da Secretaria de Pós-Graduação pela atenção, convivência e suporte durante o período que convivemos.

Agradeço a CAPES pelo financiamento deste projeto.

Resumo

Este trabalho de mestrado apresenta um estudo de Grades Computacionais e Simulações Distribuídas sobre a técnica MRIP. A partir deste estudo foi possível propor e implementar o protótipo de uma ferramenta para Gerenciamento de Experimento em Ambiente de Grade, denominada *Grid Experiments Manager – GEM*, organizada de forma modular podendo ser usada como um programa ou integrada com outro *software*, podendo ser expansível para vários *middlewares* de Grades Computacionais. Com a implementação também foi possível avaliar o desempenho de simulações sequenciais com aquelas executadas em *cluster* e em uma Grade Computacional de teste, sendo construído um *benchmark* que possibilitou repetir a mesma carga de trabalho para os sistemas sobre avaliação. Com os testes foi possível verificar um ganho alto no tempo de execução, quando comparadas as execuções sequenciais e em *cluster*, obteve-se eficiência em torno de 197% para simulações com tempo de execução baixo e 239% para aquelas com tempo de execução maior; na comparação das execuções em *cluster* e em grade, obteve-se os valores para efeciência de 98% e 105%, para simulações pequenas e grandes, respectivamente.

Abstract

This master's thesis presents a study of Grid Computing and Distributed Simulations using the MRIP approach. From this study was possible to design and implement the prototype of a tool for Management of Experiments in Grid Environment, called Grid Experiments Manager – GEM, which is organized in a modular way and can be used as a program or be integrated with another piece of software, being expansible to varius middlewares of Computational Grids. With its implementation was also possible to evaluate the performance of sequencial simulations executed in clusters and a Computational testbed Grid, also being implemented a benchmark which allowed repeat the same workload at the systems in evaluation. A high gain turnaround of the executions was infered with those results. When compared Sequential and Cluster executions, the eficiency was about of 197% for thin time of execution and 239% for those bigger in execution; when compared Cluster and Grid executions, the eficiency was about of 98% and 105% for thin and bigger simulations, repectivelly.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ASDA	Ambiente de Simulação Distribuída Automático.
CMB	Chandy, Misra e Bryant.
CPU	*Central Processing Unit.*
CoG Kit	*Java Commodity Grid Kit.*
DNS	*Domain Name Service.*
ELF	*Executable and Linkable Format.*
FCFS	*First Come First Serve.*
GEM	*Grid Experiments Manager.*
GLOBUS	*Globus Toolkit.*
GNU	*GNU is Not Unix.*
GRAM	*Grid Resource Allocation and Management.*
ICMC-USP	Instituto de Ciências Matemáticas e de Computação – USP.
ICMP	*Internet Control Message Protocol.*
IP	*Internet Protocol.*
LaSDPC	Laboratório de Sistemas Distribuídos e Programação Concorrente.
LSF	*Load Sharing Facility.*
MPI	*Message Passing Interface.*
MPICH2	*Message Passing Interface Chameleon.*
MRIP	*Multiple Replication in Parallel.*
OGF	*Open Grid Forum.*
OGSA	*Open Grid Service Architecture.*
OGSI	*Open Grid Service Infrastructure.*
P2P	*Peer-to-peer.*
PBS	*Portable Batch System.*

PVM *Parallel Virtual Machine.*
QoS *Quality of Service.*
RAID *Redundant Array of Inexpansive Disks.*
RMI *Remote Method Invocation.*
SRIP *Single Replication in Parallel.*
SSH *Secure Shell.*
TCP *Transmission Control Protocol.*
UML *Unified Modeling Language.*
USP Universidade de São Paulo.
VO *Virtual Organization.*
XML *eXtensible Markup Language.*
WAN *Wide Area Network.*
WS *Web Service.*

Capítulo

1

Introdução

1.1 Motivação

A validação de sistemas computacionais através da medição e avaliação de desempenho é um grande desafio encontrado na área de Ciência da Computação. Os sistemas distribuídos têm uma rápida evolução e a cada momento exigem novas propostas, arquiteturas, topologias e uma melhor e mais eficiente comunicação entre as entidades envolvidas no sistema. Neste contexto, a avaliação e a validação destes sistemas requer o uso das mais variadas ferramentas de apoio, uma vez que a construção de protótipos destes sistemas se torna cada vez mais difícil, tanto por dificuldades de implementação quanto à inviabilidade devido a altos investimentos, financiando protótipos de sistemas.

A modelagem de sistemas computacionais é uma técnica de validação que pode ser utilizada tanto em projetos a serem desenvolvidos quanto em sistemas já em funcionamento. As características essenciais de um sistema são abstraídas em um modelo que o representa com certo grau de identidade. A simulação tem sido largamente adotada na indústria como um todo, não se restringindo apenas aos sistemas computacionais, devido à sua versatilidade (pode ser utilizada em diferentes situações), flexibilidade (é adaptável a novas e diferentes situações) e baixo custo (com um mesmo programa pode-se simular diferentes situações sem alterar o custo da simulação).

Uma simulação pode ser classificada como determinística ou estocástica, de acordo com os dados fornecidos na entrada. A maioria dos sistemas a serem simulados são modelados na forma de simulações estocásticas, onde os dados de entrada são gerados aleatoriamente a partir de uma distribuição de probabilidade. Devido à aleatoriedade dos dados de entrada, a simulação precisa ser repetida várias vezes de modo a garantir que as variáveis estimadas minimizem a influência das variáveis aleatórias. É aqui que a simulação apresenta uma grande desvantagem: o tempo de execução da simulação.

Com o objetivo de diminuir o tempo de processamento de uma simulação, pelo menos duas abordagens de simulação distribuída foram propostas e têm sido estudadas: SRIP (Single Replication in Parallel) (JEFFERSON, 1985; MISRA, 1986; FUJIMOTO, 2000) e Múltiplas Replicações em Paralelo (MRIP – Multiple Replication in Parallel) (HEIDELBERGER, 1988; EWING et al., 1999). Tanto uma como a outra abordagem fazem uso dos benefícios de uma outra área que vem apresentando grandes avanços: a Computação Paralela. O objetivo principal da Computação Paralela é aumentar o desempenho, diminuindo o tempo para se obter um resultado (KUMAR et al., 2001). O que diferencia as abordagens SRIP e MRIP é o modo como o modelo será distribuído nos vários processadores disponíveis para a simulação.

Na abordagem SRIP, o sistema é particionado em processos lógicos e cada processo lógico é mapeado em um processador. Estes processos se comunicam através da troca de mensagens. Para garantir que os eventos sejam simulados na devida ordem, diversos protocolos de sincronização foram desenvolvidos. O objetivo desses protocolos é sincronizar os mecanismos lógicos da simulação (tempo, eventos, etc) uma vez que a evolução de cada processo lógico depende dos eventos que podem ocorrer em outros processos lógicos.

Na abordagem MRIP o sistema não é particionado. Replicações independentes de um mesmo programa de simulação seqüencial são executadas em paralelo. Cada replicação envia seus resultados (variáveis estimadas) para um analisador global, onde as médias finais são calculadas. Quando a precisão requerida é atingida, a simulação é encerrada.

Independentemente da abordagem a ser considerada para a utilização da computação paralela no desenvolvimento da simulação, um ponto importante é o planejamento dos experimentos que serão realizados, ou seja, quais os parâmetros que serão analisados e como eles irão variar para efeitos de comparação e análise de desempenho.

Portanto, além da simulação ter que ser executada diversas vezes para se garantir que os resultados estejam estatisticamente corretos, é possível ainda que cada simulação tenha que ser executada diversas vezes, onde cada um possui diferentes valores para os parâmetros de entrada.

A pesquisa relacionada ao planejamento de experimentos e à execução de simulação distribuída utilizando grades é bastante interessante pois elas oferecem exatamente as

características necessárias para o desenvolvimento de simulações distribuídas MRIP.

1.2 Objetivo

Este projeto de mestrado tem por objetivo o estudo e a análise do desenvolvimento de simulações distribuídas em grades computacionais, definindo experimentos e utilizando a abordagem MRIP.

Para alcançar tal objetivo, os seguintes tópicos foram considerados:

- Estudo dos ambientes de gerenciamento de grades computacionais, com enfoque nos problemas de comunicação envolvidos nas áreas de simulação distribuída;
- Estudo da teoria envolvida na área de planejamento de experimentos;
- Definição de uma ferramenta que permita o planejamento dos experimentos e o desenvolvimento de simulações distribuídas utilizando a abordagem MRIP em grades computacionais;
- Avaliação do desempenho de simulações distribuídas que são executadas em grades, através do *Speedup* obtido.

1.3 Estrutura

Esta dissertação está organizada em 6 capítulos, os quais são brevemente descritos a seguir:

- No Capítulo 2 é apresentada uma revisão geral sobre a de Planejamento de Experimentos e como os diferentes tipos de experimentos podem ser executados, bem como os conceitos de Simulação e Simulação Distribuída nas abordagens SRIP e MRIP;
- O Capítulo 3 trata dos principais conceitos que fundamentam a abordagem adotada no projeto, definindo grades computacionais, destacando as principais diferenças com relação à Computação Distribuída e Clusters;
- O Capítulo 4 é discutido a proposta de uma ferramenta que canaliza todos os experimentos de uma simulação para a execução nos recursos computacionais disponíveis, bem como as conclusões da utilização de dois *middlewares* de computação em grade;
- No Capítulo 5 são discutidos a abordagem para avaliação de desempenho, o *benchmark*, os resultados e o *Speedup* obtido;

- No Capítulo 6 serão apresentados conclusões, contribuições e trabalhos futuros.

Capítulo

2

Avaliação de Desempenho

2.1 Considerações Iniciais

A avaliação de desempenho é uma área de pesquisa que pode ser aplicada em diversas áreas do conhecimento. Independente da área, um ponto essencial é planejar como será feita a avaliação e esse ponto é tratado no Planejamento de Experimentos. Especificamente para este projeto utiliza-se a simulação como uma técnica para avaliar o desempenho de sistemas computacionais.

Desse modo, este capítulo está organizado em duas partes principais que têm o objetivo de apresentar conceitos sobre Planejamento de Experimentos sobre a ótica da avaliação de desempenho de sistemas através de Simulação.

A primeira parte é dedicada ao Planejamento de Experimentos. A seção 2.2.1 traz uma breve descrição sobre o que é Planejamento de Experimentos e seus objetivos, apresentando uma terminologia proposta por Jain (1991) e identificando, também, alguns passos para execução de experimentos, além dos erros mais comuns que podem surgir durante a prática da aplicação dos conceitos envolvidos. A seção 2.2.2 versa sobre os tipos de projetos de planejamentos de experimentos mais comuns, abordando Projeto Simples, Projeto Fatorial Completo e Projeto Fatorial 2^2 e 2^k, que auxiliam o entendimento de

como pode ser a análise de um projeto Fatorial Completo. Também é descrito como é feito o tratamento das saídas de uma simulação na seção 2.2.3.

Na segunda parte, dedica-se à Simulação Distribuída, trazendo de forma geral os principais conceitos de Simulação (Seção 2.3.1); descrevendo o que é necessário para que se possa implementar uma simulação com êxito. Em seguida são apresentados métodos para a execução de simulações em paralelo (Seção 2.3.2).

2.2 Planejamento de Experimentos

Planejamento de Experimentos designa uma área de estudos da *Estatística* que desenvolve técnicas de planejamento e análise de experimentos. Existe um grande número de técnicas, as quais possuem vários níveis de sofisticação, bem como uma grande quantidade de ferramentas que oferecem suporte às condições necessárias para o planejamento de experimentos. Essas técnicas cobrem todas as possibilidades dos possíveis experimentos; seus diversos fatores, cada qual com seus diferentes níveis; tratamento de replicações; etc. Assim, são de suma importância dentro de Avaliação de Desempenho, pois possibilitam, por sua vez, uma melhor utilização de suas técnicas e suas ferramentas, assim como a análise dos resultados (SANTANA, 2007).

2.2.1 Introdução ao planejamento de experimentos

Muitas vezes experimentos são executados sem um planejamento prévio, o que pode levar à incerteza sobre os resultados obtidos. Isto se deve ao fato de não haver uma visão satisfatória do sistema a ponto de se ter conhecimento sobre todos os componentes que o formam e as interações entre cada um destes componentes. Assim, um conhecimento do sistema em observação é um fator básico na execução de uma avaliação de nível melhor. O Planejamento de Experimentos pode ser usado em casos onde avaliação de desempenho é necessária e apresenta técnicas que levam a melhores resultados com um mínimo de aferições obtidas através de experimentos (SANTANA, 2007).

A indústria se beneficia muito com estas técnicas, pois estas apresentam resultados confiáveis, além da economia de tempo e dinheiro. No entanto, elas exigem uma grande quantidade de cálculos a fim de serem obtidos intervalos de confiança satisfatórios. Ferramentas especialmente desenvolvidas não só contribuem no auxílio para estas tarefas, mas também se tornam fundamentais para a efetiva economia de tempo (SANTANA, 2007).

Os objetivos para o planejamento de experimentos são a maximização da informação com um número mínimo de experimentos e a identificação dos efeitos dos vários fatores

no resultado observado, determinando o quão significante é o efeito de um no resultado observado. Todos estes objetivos convergem para uma melhor qualidade dos resultados dos testes e para a obtenção de um projeto com desempenho superior em termos de suas características funcionais e de sua robustez (SANTANA, 2007).

Um experimento pode ser definido como um teste, ou uma série de testes, nos quais mudanças propositais são feitas nas variáveis de entrada do modelo de modo a se observar e identificar razões para mudanças nas variáveis de saída resultantes (MONTGOMERY, 2004). Na área de planejamento e análise de experimentos existem métodos específicos para que conclusões válidas possam ser obtidas.

Considerando um sistema a ser avaliado, as seguintes observações podem ser feitas:

- Quais são as variáveis de entrada?
- As variáveis escolhidas são as melhores?
- Quais são os parâmetros de saída?
- Qual é a influência das variáveis de entrada sobre os parâmetros de saída?

É apresentado um caso para simulação de um Servidor *Web* com quatro variáveis de entrada e dois parâmetros de saída descritos como:

- Variáveis de Entrada:
  - Taxa de chegada das requisições no servidor *Web*, que poderia seguir uma distribuição exponencial com média $0,1$ (sistema mais carregado) ou $2,0$ (sistema não muito carregado);
  - Tempo médio de atendimento do servidor *Web* para cada requisição, que poderia ser uma distribuição exponencial com média $0,018$ (servidor mais rápido) ou $0,036$ (servidor mais lento);
  - Política de escalonamento no servidor, que poderia ser FCFS (*first come, first serve*) ou com prioridade; e
  - Número de servidores, podendo ser 4 ou 6.
- Parâmetros de saída:
  - Tempo médio de atendimento para cada requisição; e
  - Taxa de requisições não atendidas.

Este exemplo será referenciado ao decorrer de todo capítulo, apresentando de forma mais clara os conceitos abordados.

2.2.1.1 Terminologia

Para um perfeito entendimento sobre as palavras utilizadas para referência a cada termo da área de Planejamento de Experimentos, os próximos parágrafos apresentam uma terminologia proposta por Jain (1991).

Variável ou Parâmetro de Resposta: é o resultado de um experimento e geralmente é usado para medir o desempenho do sistema. No exemplo, existem duas e pode-se identificá-las como tempo médio de atendimento e taxa de requisições não atendidas.

Fatores: são as variáveis que podem assumir vários valores e afetam o resultado da Variável de Resposta. Correspondem aos quatro ítens descritos como variáveis de entrada no exemplo.

Níveis: são os possíveis valores para cada fator. Um conjunto de níveis de uma mesma variável constitui um Fator. São os valores pré-determinados atribuidos a um fator em determinado experimento. Correspondem ao número de servidores no exemplo, pondendo assumir os valores quatro ou seis.

Fatores primários: são fatores que afetam com grande impacto uma Variável de Resposta. A taxa de chegada das requisições e seus respectivos tempos médios de atendimento afeta de forma considerável o valor das variáveis de resposta. Uma vez que taxa de chegada com valores pequenos e tempo médio de atendimento com valores altos implicam em uma sobrecarga no sistema, estes valores influenciarão diretamente os parâmetros de saída.

Fatores secundários: são fatores cujo impacto na Variável de Resposta não é significante ou não se tem interesse em quantificar. Em servidores *Web* distribuídos onde não exista uma diferenciação do tipo das requisições que chegam ao sistema e somente um ponto de rede para a chegada de requisições para o sistema é disponível, este ponto de rede se tornará um gargalo, pois nos dias atuais a velocidade de operação da rede é menor do que aquela presente no interior às máquinas que compõe o servidor. Neste caso, o fator número de servidores pode ser classificado como secundário.

Réplica: trata de repetições ou reexecuções de todo ou de parte dos experimentos. Seja o caso da simulação do exemplo com a configuração de um valor para cada fator. Sejam também os dados de entrada gerados de forma aleatória. Com uma execução não

é possível se ter uma certeza de que os valores dos parâmetros de saída expressam o comportamento do sistema observado. Em casos como este, valores que possuam um intervalo de confiança satisfatório são o resultado da média dos valores obtidos com várias execuções de um mesmo experimento.

Projeto: determina o número de experimentos a serem considerados, incluindo o número de fatores e níveis, a combinação entre os níveis e o número de replicações para cada experimento. O projeto é resultado de um estudo do sistema a ser avaliado levando em consideração os objetivos desejados e também a relação dos serviços oferecidos pelo sistema.

Interação: dois fatores interagem se o efeito de um depende do nível do outro. Sejam adicionados três fatores ao exemplo: número de discos, quantidade de memória e a quantidade de cache disponível. Não existiria uma interação entre o número de discos e a quantidade de cache disponível, pois o aumento ou a diminuição do número de discos não influenciaria o desempenho da quantidade de cache disponível, e vice-versa. Por outro lado, a quantidade de memória interagiria com a quantidade de cache disponível, uma vez que uma cache com valor mais alto apresentaria mais quantidades de acertos e por conseqüência refletiria como um melhor desempenho de acesso aos dados alocados na memória.

2.2.1.2 Etapas de um Experimento

Com os tópicos vistos até agora pode-se observar algumas etapas para o desenvolvimento de um bom experimento. Abaixo são listados estas etapas, as quais não seguem necessariamente a ordem de listagem (SANTANA, 2007):

- Estudar o sistema e definir os objetivos;
- Determinar os serviços oferecidos pelo sistema;
- Selecionar métricas de avaliação;
- Determinar os parâmetros que afetam o desempenho do sistema;
- Determinar o nível de detalhamento da análise;
- Determinar a caga de trabalho característica;
- Realizar a avaliação e obter os resultados.

O ponto crucial para a execução de um experimento não é somente ter o conhecimento sobre o sistema a ser modelado, mas também ter bem definida qual resposta deseja-se obter do experimento, ou seja, um objetivo. Assim, é possível ter ciência de qual é o escopo do sistema em observação e determinar de forma mais eficiente quais serviços serão oferecidos por ele, não sendo necessário que todos os serviços disponíveis sejam escolhidos como parte da análise. As métricas estão diretamente relacionadas com os objetivos e são elas que garantirão a resposta produzida pelo experimento. Um passo além ao conhecimento do sistema é ter conhecimento sobre os parâmetros que influenciam drasticamente o sistema e isto é obtido com teste preliminares e/ou estudos detalhados do sistema em questão já em funcionamento. Com isso, é possível limitar quão detalhada será a análise, a ponto de considerar parâmetros que influenciam de forma mínima ou quase nula, podendo, eventualmente, abrir mão destes objetivando simplicidade. Um mesmo sistema pode ter comportamentos diferentes quando submetido a diferentes cargas de trabalho, portanto é preciso ter em mente quais as características sobre as quais o sistema deve ser observado. Dependendo da carga aplicada sobre um sistema pode-se obter respostas de seu funcionamento em condições extremas (*stress* na utilização de seus recursos), bem como respostas quando em condições normais. Uma vez detalhado, com escopo definido e objetivo(s) especificado(s), o sistema sob observação é avaliado e os resultados dos experimentos são coletados para que a análise seja feita.

2.2.1.3 Erros comuns

Em Jain (1991) é identificada uma série de erros mais comuns que podem ocorrer durante a experimentação, os quais muitas vezes ocorrem por falta de experiência do analista que desenvolve o projeto. Os próximos parágrafos apresentam observações sobre estes erros.

Variação decorrida dos erros de cada experimento é ignorada. Como os valores utlizados para medição correspondem ao resultado de uma execução cujos valores de entrada foram gerados aleatoriamente, a cada nova execução, valores diferentes para o mesmo experimento são produzidos com freqüência. Decisões devem ser tomadas levando-se em conta a variação causada por estas variações.

Parâmetros importantes não são controlados. Parâmetros que influenciam o desempenho como, por exemplo, carga de trabalho, ambiente e parâmetros do sistema devem ser cuidadosamente estudados e somente alguns devem ser classificados como fatores e terem seus valores variados. Caso eles não sejam considerados, os resultados podem não apresentar valores significantes.

Projetos simples do tipo *one-factor-at-a-time* são usados. Neste tipo de projeto, são necessários muitos experimentos e na grande maioria das vezes resultam em valores que apontam uma mesma informação. Com a definição de um Projeto de Experimentos é possível obter resultados melhores (valores com intervalos de confiança menores) com a mesma quantidade de experimentos.

Efeitos de diferentes fatores são ignorados. Ocorre principalmente quando são variados de forma simultânea vários fatores a fim de diminuir o número de experimentos. A identificação da influência de determinado fator no desempenho se torna mais difícil.

Interações são ignoradas. Com freqüência, existe dependência entre fatores do sistema, e elas devem sempre ser observadas. A utilização de projetos do tipo *one-factor-at-a-time* dificulta, ou até mesmo impede, a identificação das interações.

Execução de muitos experimentos. A quantidade de experimentos é uma função do número de fatores e níveis. Uma melhor alternativa é desenvolver projetos menores ao invés de usar um número excessivo destes. Com a utilização de poucos fatores e níveis, é possível, em um primeiro instante, analisar os primeiros resultados e assim determinar uma alteração no projeto para que ele se torne mais eficiente. Ou seja, incrementar o projeto com novos fatores; mais níveis para determinados fatores; ou, ainda, exclusão de fatores.

2.2.2 Tipos de Experimentos

Com os parâmetros apresentados no exemplo da seção 2.2.1, uma primeira possibilidade para o planejamento dos experimentos a serem realizados sobre o sistema sendo analisado, poderia ser uma combinação aleatória das possíveis variáveis de entrada. Por exemplo: as variáveis de entrada seriam uma taxa de chegada das requisições com uma distribuição exponencial com média $0,1$, um tempo médio de atendimento com distribuição exponencial de média $0,036$, utilizando uma política FCFS e com 4 servidores. Após obter de maneira confiável os valores dos parâmetros de saída, pode-se obeservar, por exemplo, que a taxa de requisições não atendidas está muito alta, devido provavelmente ao fato dos servidores serem muito lentos. Pode-se então diminuir a média do tempo de atendimento dos servidores *Web* para $0,018$ e executar a simulação novamente para ver a influência desse fator nos resultados. Essa técnica é conhecida como abordagem *best-guess*. Essa é uma boa técnica quando o usuário possui um bom conhecimento técnico ou teórico do sistema. Essa técnica possui duas desvantagens: uma é quando a suposição inicial não produz os resultados desejados. Nesse caso, uma outra suposição pode também não levar

a resultados desejados e esse processo continuar por um período muito longo. A segunda é quando a suposição inicial produz um bom resultado e o usuário pára as tentativas, porém não há garantias de que a melhor solução foi encontrada (MONTGOMERY, 2004).

Existe uma série de modelos mais comuns para Planejamento de Experimentos que visam a descoberta de informações, que não são possíveis de obter com experimentos do tipo *best-guess*. A principio, versa-se sobre o Projeto Simples, que é ineficiente, porém muito usado pois resulta em pouco experimentos. Em seguida, versa-se sobre o Projeto Fatorial Completo, com respectivas abordagens para redução de experimentos. Por fim, apresenta-se como deve ser feita a análise de influência das variáveis de saída em projetos 2^2 e 2^k.

2.2.2.1 Projeto Simples

A abordagem deste tipo de projeto consiste em começar com uma configuração típica e depois variar um fator por vez (*one-factor-at-a-time*). Em um projeto com vários fatores e vários valores para cada nível é feita uma medição de desempenho com uma primeira configuração. Nos próximos experimentos é alterado um nível de um fator enquantos os níveis dos outros fatores são mantidos (JAIN, 1991).

Nesse caso, seleciona-se um conjunto de valores como ponto inicial, denominado linha base, e varia-se sucessivamente cada uma das variáveis, mantendo os outros fatores na linha base. Após a execução das simulações, uma série de gráficos podem ser elaborados com o objetivo de verificar o comportamento do sistema com a variação da variável de entrada. Um problema com essa abordagem é que ela não considera a possível interação entre as variáveis de entrada, tornando-a uma técnica não muito aconselhável (MONTGOMERY, 2004). Como conseqüência da não identificação das interações entre as variáveis de entrada, este tipo de projeto também não permite verificar quais fatores afetam o desempenho. Como são de fácil implementação e apresentam uma quantidade de experimentos razoavelmente menor se comparado às outras abordagens, este projeto é muito utilizado mesmo que seja estatisticamente ineficiente (JAIN, 1991).

Nesta abordagem é escolhida uma configuração padrão para os fatores e os experimentos consistem da alteração dos níveis de cada fator, um por vez, até que todos os níveis sejam executados. Pode ser feita uma estimativa de quantos experimentos serão necessários para a execução. Seja, por exemplo, um projeto onde existam k fatores e n_i níveis no fator i, tem-se:

$$n = 1 + \sum_{i=1}^{k} (n_i - 1)$$

Como exemplo seja um servidor de arquivo com quatro fatores: (1) processador com três níveis: Pentium IV, Athlon XP e Pentium IV com *Hyper Thread*; (2) Quantidade de memória com quatro níveis: 512MB, 1GB, 2GB e 4GB; (3) Quantidade de Cache com três: 256KB, 512KB, 1MB; (4) Número de Discos com tês: 2, 3 e 4. A quantidade de experimentos seria dez, pois:

$$n = 1 + (3 - 1) + (4 - 1) + (3 - 1) + (3 - 1) = 10$$

2.2.2.2 Fatorial Completo

Outra possibilidade é a abordagem fatorial completo onde as variáveis sofrem variações conjuntas, ao invés de uma por vez. Considerando o exemplo do servidor *Web* distribuído, suponha que somente os fatores número de servidores e tempo médio de atendimento são importantes. Neste caso, tem-se dois fatores, cada um com dois níveis, formando os cantos de um quadrado. Esse fatorial é denominado planejamento 2^2 fatorial. Com essa técnica, é possível avaliar a influência isolada de cada uma das variáveis e também determinar quando as variáveis interagem. Essa análise pode ser feita através de métodos de regressão linear e análise de variância (MONTGOMERY, 2004).

Um projeto do tipo Fatorial Completo explora todas as combinações possíveis de todos fatores e todos níveis do projeto de forma que pode ser feita uma estimativa de quantos experimentos são necessários para execução. Seja um projeto onde existam k fatores e n_i níveis no fator i, tem-se

$$n = \prod_{i=1}^{k} n_i$$

No caso do servidor de arquivos apresentado na seção anterior (2.2.2.1), tem-se:

$$n = 3(CPU) \times 4(mem) \times 3(cache) \times 3(discos) = 108$$

A grande vantagem é que todos os fatores podem ser avaliados e assim determinar o efeito que cada fator tem sobre as variáveis de saída bem como a interação entre eles. Por outro lado, o custo da avaliação é aumentada devido ao grande número de experimentos gerados (JAIN, 1991; SANTANA; SANTANA, 2007).

Uma estratégia para diminuir os custos é diminuir o número de níveis de cada fator. A redução dos níveis de cada fator para dois se torna interessante por que o experimento se torna 2^k e além de produzir um menor número de experimentos, a análise de seus

resultados é simplificada. Com os primeiros resultados é possível verificar quais são os fatores primários e por conseqüência explorar de forma mais consciente outros possíveis níveis para tais fatores.

Reduzir o número de fatores também contribui para a diminuição da quantidade de experimentos (SANTANA, 2007). Esta estratégia exige muito cuidado e deve ser executada após a estratégia do parágrafo anterior. Por tratar da eliminação de um fator, é necessário uma metodologia adequada, pois fatores primários podem ser eliminados e assim exercer grande influência nos parâmetros de saída.

O método fatorial parcial ajuda a diminuir a quantidade de experimentos (SANTANA, 2007). Trata-se da não execução de experimentos, nos quais sabe-se que não existe interação ou a interação existente é insignificante. No exemplo do servidor de arquivos, sabe-se que não existe uma interação entre número de discos e a quantidade de memória cache. Como na estratégia do parágrafo anterior, antes de qualquer decisão é necessário saber das interações dos fatores. O ponto positivo é que se ganha rapidez, uma vez que o número de experimentos é menor. Por outro lado, perde-se informações válidas decorrentes dos experimentos que não foram executados.

2.2.2.3 Fatorial 2^2

O projeto Fatorial 2^2 é uma especialização do projeto Fatorial 2^k, onde $k = 2$. Ter o número de fatores e os níveis de cada fator reduzidos para o valor dois, não só torna a análise da influência de cada variável e a interação entre elas mais simplificados, como também necessita de somente quatro experimentos.

No modelo para este projeto, assume-se A e B como rótulo para os dois fatores do sistema. A análise começa com a definição de duas variáveis x_A e x_B, como apresentado na tabela 2.1.

Tabela 2.1: Atribuição para os valores de x_A e x_B (JAIN, 1991)

Valor	*Descrição*
$x_A = -1$	Para o nível de A com menor valor
$x_A = 1$	Para o nível de A com maior valor
$x_B = -1$	Para o nível de B com menor valor
$x_B = 1$	Para o nível de B com maior valor

O modelo para este tipo de projeto é dado por (JAIN, 1991):

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_{AB}$$

Onde y representa o desempenho do sistema. Assim, y é igual à soma do desempenho

médio (q_0), da influência do fator A (q_A), da influência do fator B (q_B) e da influência da interação entre os fatores A e B (q_{AB}).

Portanto, seja a tabela 2.2 com os resultados do experimentos, onde a coluna y corresponde à medida de desempenho de cada um dos experimentos.

Tabela 2.2: Resultado de todos os experimentos possíveis (JAIN, 1991)

Experimento	A	B	y
1	-1	-1	y_1
2	1	-1	y_2
3	-1	1	y_3
4	1	1	y_4

Neste ponto é possível obter o resultado da influência de cada uma das variáveis seguindo o método descrito na tabela 2.3. A coluna I é toda completada com valor 1. As colunas A e B são as possíveis combinações dos valores das variáveis x_A e x_B. A coluna AB é o resultado da multiplicação das colunas A e B. A coluna y corresponde aos resultados dos experimentos. Cada coluna da linha $Total$ é a somatória da prévia multiplicação dos valores da coluna em questão com a coluna y. Por fim, a linha $Total/4$ apresenta os resultados que podem ser mapeados no modelo de desempenho apresentado no início desta seção. Assim, ao mapear cada coluna na fórmula pode-se obter conclusões sobre a influência das variáveis no desempenho final, bem como a interação entre elas. As colunas da tabela significam:

- I: influência do desempenho médio, correspondente ao valor de q_0 no modelo;
- A: influência do fator A, correspondente ao valor de q_A no modelo;
- B: influência do fator B, correspondente ao valor de q_B no modelo;
- AB: influência da interação entre os fatores A e B, correspondente ao valor de q_{AB} no modelo.

Tabela 2.3: Influência da variáveis de entrada no parâmetros de saída (JAIN, 1991).

I	A	B	AB	y
1	-1	-1	1	y_1
1	1	-1	-1	y_2
1	-1	1	-1	y_3
1	1	1	1	y_4
$\sum I \times y$	$\sum A \times y$	$\sum B \times y$	$\sum AB \times y$	Total
$\frac{\sum I \times y}{4}$	$\frac{\sum A \times y}{4}$	$\frac{\sum B \times y}{4}$	$\frac{\sum AB \times y}{4}$	Total/4

2.2.2.4 Fatorial 2^k

Uma vez entendido como é feita a análise do projeto Fatorial 2^2 ($k = 2$), é possível fazer uma generalização tal que seja possível aplicar as mesmas técnicas para um valor variável de k. Para esta análise, sejam k fatores com dois níveis cada, o que resultará em um total de 2^k experimentos, bem como 2^k valores de influência para o desempenho final. Estes valores de influência incluem interações entre $\binom{k}{2}$ variáveis, $\binom{k}{3}$ variáveis e assim por diante (JAIN, 1991).

Para fins de exemplificação, seja $k = 3$. A tabela 2.4 apresenta os mesmos conceitos apresentados para o fatorial 2^2 (seção 2.2.2.3). Tal tabela apresenta um total de oito (2^3) variáveis de influência em seu modelo.

Tabela 2.4: Exemplo de influência das variáveis de entrada no parâmetros de saída para um projeto 2^3 fatorial (JAIN, 1991).

I	A	B	C	AB	AC	BC	ABC	y
1	-1	-1	-1	1	1	1	-1	y_1
1	1	-1	-1	-1	-1	1	1	y_2
1	-1	1	-1	-1	1	-1	1	y_3
1	1	1	-1	1	-1	-1	-1	y_4
1	-1	-1	1	1	-1	-1	1	y_5
1	1	-1	1	-1	1	-1	-1	y_6
1	-1	1	1	-1	-1	1	-1	y_7
1	1	1	1	1	1	1	1	y_8
$\sum I \times y$	$\sum A \times y$	$\sum B \times y$	$\sum C \times y$	$\sum AB \times y$	$\sum AC \times y$	$\sum BC \times y$	$\sum ABC \times y$	Total
$\frac{\sum I \times y}{8}$	$\frac{\sum A \times y}{8}$	$\frac{\sum B \times y}{8}$	$\frac{\sum c \times y}{8}$	$\frac{\sum AB \times y}{8}$	$\frac{\sum AC \times y}{8}$	$\frac{\sum BC \times y}{8}$	$\frac{\sum ABC \times y}{8}$	Total/8

2.2.3 Tratramento dos Resultados

Neste ponto, é feita uma interseção entre a parte de Planejamento de Experimentos e a de Simulação Distribuída. Utiliza-se de fundamentos estatísticos para o tratamento das saídas das simulações apontadas na seção seguinte (vide 2.3).

Conceitualmente, todo sistema recebe uma entrada, processa-a e produz uma saída. Isto é valido para um sistema ecológico, para um sistema orgânico, para um sistema industrial e qualquer outra coisa que tenha este comportamento. Como será visto na seção 2.3, simulações executadas em computadores tem o mesmo comportamento. Assim, nos parágrafos seguintes são descritos como o tratamento dos valores de **saída** de um sistema é feito, restringindo o escopo para as simulações.

O termo simulação ainda deixa muito extenso seu significado. Portanto, é importante restringir mais o foco da explicação para as Simulações de Eventos Discretos executadas em computadores. Mais ainda, destas, consideremos aquelas que processam a entrada

parametrizada com valores pseudo-aleatórios, ou seja, simulações que produzem resultados estocásticos.

O termo aleatório é usado para indicar que valores gerados não fazem parte de uma sequência lógica e não podem repetir. Em outras palavras, os resultados não são previsíveis. Se simulações fossem executadas com valores aleatórios, uma dada execução poderia ser considerada um evento único e, portanto, teria um tratamento estatístco diferenciado.

Neste caso, o termo pseudo-aleatório faz menção a uma sequência de valores aleatórios, porém com a especialidade de serem previsíveis. Um gerador de números pseudo-aleatório produz números aleatórios e depois os transforma, obtendo uma sequência que se encaixe em uma determinada distribuição estatística (JAIN, 1991).

Outra característica dos geradores de números pseudo-aleatórios é receber um parâmetro que permita que uma sequência seja repetida. Este parâmetro tem o nome de semente (*seed*). Esta característica possibilita, por exemplo, que dois modelos diferentes, propostos para resolver o mesmo problema, sejam comparados com um mesmo nível de carga.

Um ponto negativo de se usar tais geradores é que sua aleatoriedade influência diretamente o valor de saída do sistema em questão. Isto significa que executar um experimento uma vez não garante um resultado preciso. Faz-se necessário, então, estimar o valor, analizando o conjunto de saída de várias execuções, sendo que uma semente diferente tenha sido usada para cada execução. Portanto, existirá um erro associado ao resultado estocástico de uma simulação. As estimativas são feitas por meio do **Intervalo de Confiança**.

O Intervalo de Confiança é uma medida de probabilidade que estima o intervalo de valores que o parâmetro estudado pode conter (MAGALHãES; LINA, 2005). Ele quantifica quão grande é o erro ou quão preciso é o resultado.

Neste contexto, confiança é a chance que o valor do parâmetro estudado esteja no intervalo calculado. Ela é representadapela seguinte fórmula:

$$C = 100 \times (1 - \alpha)\%$$

A estimativa é feita com o auxílio de medidas de posição e de dispersão. As primeiras produzem valores que informam o ponto de maior ocorrência dos valores estimados, ou seja, o valor médio. As segundas indicam quão aglomerados ou dispersos estão todos os valores de entrada (MAGALHãES; LINA, 2005). Assim, é descrito primeiramente sobre tais medidas para que finalmente seja apresentado o cálculo completo do Intervalo de Confiança.

As medidas de posição são representadas pelas: média, mediana e moda.

A medida mais popular quando se trata do escopo abordado por esta seção é a média (MAGALHãES; LINA, 2005). Em outras referências é possível encontrar médias com ponderações, onde cada valor do conjunto de dados a ser utilizado recebe um fator, o qual é usado como multiplicador. No entanto, esta não é uma característica de um conjunto de saídas de uma simulação, porque cada valor tem o mesmo peso, ou seja, um deles é tão representante da saída como qualquer outro elemento do conjunto. Assim sendo, a **média simples** é suficiente para estimar a tendência central das saídas de uma simulação. Portanto, sua fórmula é dada como segue:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

A mediana ocupa a posição central de todos os valores ordenados. Enquanto a moda é dada pelo valor mais frequente (MAGALHãES; LINA, 2005).

Medidas de tendência central não são suficientes para descrever ou discriminar diferentes conjuntos de dados, elas fornecem uma ideia do comportamento das variáveis (MAGALHãES; LINA, 2005). Neste ponto, as medidas de dispersão proporcionam um mecanismo que permite que este espaço seja complementado. No escopo deste trabalho, far-se-á uso da medida média simples, sendo a mediana e a moda não utilizadas.

Das medidas de dispersão, aquelas que contribuem bastante para o tratamento de dados da saída de simulações são: a variância amostral e o desvio-padrão.

A variância amostral mede a "distância" de cada valor do conjunto de dados em relação à média. Sendo sua fórmula (MAGALHãES; LINA, 2005):

$$var = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

O desvio-padrão corresponde à raiz quadrada da variância, como observado em sua fórmula (MAGALHãES; LINA, 2005):

$$dp = \sqrt{var}$$

Uma vez que se tenha informação sobre o comportamento de uma variável (média) e sua dispersão (variância e desvio-padrão), uma alternativa para o cálculo do Intervalo de Confiança é utilizar a tabela *t-student* para obter seu respectivo índice e calcular os valores inferior e superior do intervalo:

$$H = \bar{x} \pm (t_{1-\alpha/2,N-1} \times dp)$$

Onde:

- $\bar{x}$: média do conjunto de saídas das execuções de uma simulação;
- $t_{1-\alpha/2,N-1}$: índice obtido da tabela *t-student*;
- dp : desvio padrão.

Com o resultado desta fórmula é possível afirmar que se um novo conjunto de resultados da simulação for gerado, o valor médio desse conjunto estará no intervalo calculado, com uma probabilidade de erro α.

2.3 Técnicas para Avaliação de Desempenho

A simulação é uma técnica que pode ser usada para a avaliação de desempenho de sistemas de computação. Se o sistema a ser caracterizado não está disponível (quando ainda está em fase de projeto), a simulação apresenta uma forma de predizer seu desempenho e comparar várias alternativas de configurações. No caso de um sistema já em operação, a simulação também se torna atraente ao permitir que sejam feitas comparações desse sistema com diferentes cargas de trabalho e ambientes (JAIN, 1991).

2.3.1 Simulação

A simulação é a implementação de um modelo de um sistema. No contexto deste trabalho, um programa de computador. Ou seja, a simulação é um modelo do mundo real que foi implementado em uma linguagem de programação. Essa implementação é o veículo para a condução de experimentos que visam explorar o comportamento do sistema modelado dada uma série de parâmetros passados como entrada. Para que uma simulação seja bem sucedida, ela requer uma formulação precisa do sistema em estudo (modelo), e a correta implementação computacional do modelo e interpretação dos resultados (FORTIER; MICHEL, 2002).

2.3.1.1 Simulação analítica e Ambientes Virtuais

Fujimoto (2000) classifica a simulação em duas classes: ambientes virtuais e analítica. Os Ambientes Virtuais tendem a ser mais realísticos e, por conseqüência, também tendem a ser mais custosos para a implementação. Nesses ambientes, na maioria das vezes, o usuário participa interagindo com a simulação em andamento. Assim, a contagem do

tempo deve ser feita de forma mais próxima da realidade. Como exemplo para ambiente virtuais podem ser mencionados jogos e simulações militares.

As Simulações Analíticas se referem à abordagem clássica para simulação. No geral, não necessitam de interação humano-computador durante a execução de experimentos. O usuário é um observador do sistema simulado e muitas vezes seu papel se concentra na análise do resultados, uma vez que o modelo já foi implementado. Essas simulações possuem um maior grau de abstração e é desejável que suas execuções sejam rápidas, onde o tempo é cronometrado de forma lógica. Por ser mais abstrata e, logo mais simples que o mundo real, simulações analíticas tendem a ter um custo menor.

2.3.1.2 Modelagem

Modelagem do Tempo O controle do tempo é um elemento importante em um programa de simulação. É ele quem dá ordem aos eventos que ocorrem durante a execução do experimento na simulação. No entanto, deve haver uma distinção sobre a medição do tempo (FUJIMOTO, 2000):

- Tempo físico: corresponde ao tempo como é medido no mundo real e pode ser referênciado no sistema modelado;
- Tempo da simulação: se refere ao controle do tempo durante a execução da simulação. Geralmente implementado como uma variável do tipo real;
- Tempo de execução (*wallclock time*): se refere à medida do tempo durante a execução do programa simulador. Esta medida pode ser feita por consultas ao relógio físico do sistema, levando-se em consideração o instante que o programa começou sua execução e o horário atual.

Como o **Tempo de execução** (*wallclock time*) se refere ao tempo de execução do programa simulador e não pertence à lógica do modelo que a simulação propõem-se a resolver no escopo deste trabalho, ele não é tratado como um problema aqui. O **Tempo da simulação** pode ter uma correspondência lógica com o **Tempo físico**, ou seja, existir uma proporção dos valores atribuídos a esta varíavel com aquele medido por convenção, tempo seguido pelas pessoas. Por outro lado, existem muitos casos em que não há necessidade de um correspondência entre eles, e, assim, o tempo da simulação pode ser controlado conforme a necessidade do desenvolvedor.

O controle do Tempo simulado pode ser feito de duas formas (FUJIMOTO, 2000; FORTIER; MICHEL, 2002; BANKS et al., 1996):

- Movido a passos ou síncrono: o tempo é representado em seqüências com valores equivalentes (unidades) a serem movidas de forma seqüencial, unidade por unidade. A checagem da existência de eventos a serem executados é feita a cada unidade. Caso não exista(m) evento(s), a unidade de tempo é incrementada representando um passo adiante; e
- Orientado a evento ou assíncrono: a mudança na variável que controla o tempo é alterado por um evento, o qual informa qual o novo valor a ser considerado para a variável tempo.

Modelos estáticos e dinâmicos Modelos estáticos representam o sistema em um determinado espaço de tempo; um exemplo de simulação que faz uso deste tipo de modelo é a simulação de Monte Carlo. Já modelos dinâmicos descrevem o comportamento do sistema através do tempo, como exemplo pode ser considerada a simulação do funcionamento de um banco (BANKS et al., 1996).

Modelos determinísticos e estocásticos Um modelo pode ser classificado como determinístico ou estocástico, de acordo com os dados fornecidos na entrada. A maioria dos sistemas a serem simulados são transformados em simulações estocásticas, onde os dados de entrada são gerados aleatoriamente a partir de uma distribuição de probabilidade. Devido à aleatoriedade dos dados de entrada, a simulação precisa ser repetida várias vezes de modo a garantir que as variáveis estimadas minimizem a influência das variáveis aleatórias. Por outro lado, também existe a possibilidade de um sistema ser simulado tomando-se como base para os dados de entrada um conjunto fixo de dados. Nesta modalidade, o mais comum é a modelagem de um sistema existente e em operação, onde são coletadas informações (muitas vezes arquivos de log) para que sejam submetidas ao programa simulador (BANKS et al., 1996).

Modelos discretos e contínuos A classificação de um modelo como discreto ou contínuo leva em consideração as variáveis presentes no sistema. Caso a alteração de valor dessas variáveis seja feita em pontos específicos do Tempo simulado, o modelo é classificado como discreto. Em contra-partida, modelos matemáticos analíticos são exemplo de modelos contínuos, pois as variáveis têm seus valores alterados continuamente ao longo do Tempo de simulação.

2.3.1.3 Fases do desenvolvimento de uma simulação

Em BANKS et al. (1996) e BANKS (1998) é proposta uma série de passos que têm a função de guiar o desenvolvedor da simulação.

Formulação do problema e objetivos O desenvolvimento de um projeto de simulação inicia com a identificação de um problema que se deseja investigar. Nesta fase deve ser pesquisado sobre o sistema a ser simulado a fim de se ter o maior conhecimento possível sobre ele. Assim, é possível descrever de forma clara e consisa o problema. Esta descrição pode variar ao passo que o projeto é desenvolvido. A descrição também deve apresentar quais questões a simulação deve responder, para assim traçar os objetivos a serem alcançados.

Desenvolvimento do modelo Esta etapa pode ser considerada mais uma arte do que ciência. Esta arte está diretamente ligada à capacidade de abstração das características essenciais do problema por cada modelador, bem como sua capacidade de selecionar e/ou modificar suposições levadas em consideração durante o projeto, que caracterizam o sistema. O desenvolvimento deve seguir o padrão incremental onde cada versão implementa uma característica a mais no sistema, porém não deve exceder o escopo estabelecido no passo anterior. A inclusão de características que não contribuem para o alcance dos objetivos esperados aumenta complexidade do modelo e adiciona um tempo extra durante a fase de execução.

Coleta de dados Esta fase deve ser realizada em paralelo com a anterior. Nela devem ser estudados detalhes sobre os dados do sistema. No caso da simulação de um banco, podem ser coletadas informações sobre a taxa de chegadas de clientes no estabelecimento, o tempo médio necessário para o atendimento de cada cliente e o tamanho médio das filas, e.g.

Implementação do modelo Nesta fase é feita a tradução do modelo para uma linguagem de computação em específico. Existem opções para escolha de linguagens especialmente desenvolvidas para simulação, assim como as de propósito geral. Também existem bibliotecas para linguagens de propósito geral com funcionalidades específicas para simulação. Essas bibliotecas contribuem para um ganho de produtividade no desenvolvimento, uma vez que disponibilizam uma interface de programação bastante próxima da técnica de modelagem utilizada.

Verificação A verificação consiste em analisar se o modelo foi perfeitamente implementado. Esta fase geralmente é trabalhosa e exige muita depuração. É aconselhado que a verificação seja feita de forma contínua em pequenas partes do modelo, mesmo que ele ainda não tenha sido concluído. Assim, aos poucos o modelo como um todo pode ser verificado.

Validação A validação tem a finalidade de assegurar que o modelo proposto representa o sistema do mundo real e assim garantir que as suposições feitas para delinear o modelo são válidas. Deve ser avaliada a validade operacional que verifica se a saída do programa de simulação tem a precisão necessária para a aplicação e o domínio desejado. A validação dos dados deve assegurar que os dados necessários para a construção do modelo, sua avaliação e testes são precisos.

Planejamento de Experimentos As alternativas nas quais o programa de simulação deve executar são especificadas nesta fase. Esta fase tem ligação direta com o capítulo de Planejamento de Experimentos (cap. 2). Descisões sobre como obter um maior número de informações possíveis com um menor número de execuções são tomadas nesta fase, bem como o número de réplicas e o tamanho da simulação.

Execução e análise dos resultados Nesta fase o programa é distribuído na plataforma de execução para que sejam gerados os primeiros resultados de desempenho. Muitas vezes as simulações são estocásticas e seus resultados devem atingir um pré-determinado intervalo de confiança. A análise dos resultados indica se novas execuções devem ser iniciadas para que esse intervalo de confiança seja alcançado.

Emissão dos relatórios Os resultados de todas as análises devem ser formatados claramente e concisamente em um relatório final. A partir deste relatório o usuário poderá tomar descisões, dado o comportamento do sistema em um determinado cenário.

2.3.2 Simulação Distribuída

Com o objetivo de diminuir o tempo de processamento de uma simulação, duas abordagens foram propostas e têm sido estudadas: *Single Replication in Parallel* (SRIP) (JEFFERSON, 1985; MISRA, 1986; FUJIMOTO, 2000) e Múltiplas Replicações em Paralelo (*Multiple Replication in Parallel* – MRIP) (HEIDELBERGER, 1988; EWING et al., 1999). Tanto uma como a outra abordagem fazem uso dos benefícios de uma outra área que vem apresentando grandes avanços: a Computação Paralela. O objetivo principal da Computação Paralela é aumentar o desempenho, diminuindo o tempo para se obter um resultado (KUMAR et al., 2001). O que diferencia as abordagens SRIP e MRIP é o modo como o modelo será distribuído nos vários processadores disponíveis para a simulação.

Na abordagem SRIP, o sistema é particionado em processos lógicos e cada processo lógico é mapeado em um processador. Estes processos se comunicam através da troca de mensagens. Para garantir que os eventos sejam simulados na devida ordem, diversos

protocolos de sincronização foram desenvolvidos. O objetivo dos protocolos é sincronizar os tempos da simulação uma vez que a evolução de cada processo lógico depende dos eventos que podem ocorrer em outros processos lógicos.

Na abordagem MRIP o sistema não é particionado. Replicações independentes de um mesmo programa de simulação seqüencial são executadas em paralelo. Cada replicação envia seus resultados (variáveis estimadas) para um analisador global, onde as médias finais são calculadas. Quando a precisão requerida é atingida, a simulação é encerrada.

Nas próximas seções serão discutidos alguns detalhes sobre cada uma das abordagens.

2.3.2.1 *Single Replication in Parallel* – SRIP

O desenvolvimento de uma simulação discreta distribuída usando a abordagem SRIP não é fácil. Isto pode ser observado analisando as estruturas básicas de uma simulação seqüencial:

1. As variáveis de estado que armazenam o estado do sistema;
2. Uma lista de eventos que armazena todos os eventos que foram escalonados mas ainda não simulados;
3. Um relógio global que controla o andamento da simulação.

Cada evento contém um *timestamp* que denota quando uma mudança no sistema irá ocorrer. A lista de eventos é classificada em ordem crescente do tempo de chegada. O programa de simulação consiste basicamente de um *loop* principal que repetidamente retira o evento com menor *timestamp* da lista de eventos e processa este evento. Processar um evento implica em executar algum código para efetivar a mudança de estado e, se necessário, escalonar novos eventos para o futuro. é muito importante notar nesta abordagem que o evento com menor *timestamp* (E_{min}) deve ser o próximo evento a ser processado. Isto garante que os eventos sejam simulados em ordem cronológica no tempo de simulação (MACDOUGALL, 1987). Se um evento (E_x) que contém um *timestamp* maior que E_{min} for processado, este pode alterar as variáveis utilizadas por E_{min} permitindo que o futuro influencie o passado. Erros desse tipo são chamados de erros de causa e efeito (FUJIMOTO, 2000).

A natureza seqüencial da lista de eventos e o relógio global limitam o potencial de paralelismo da simulação. A solução é, portanto, eliminar a lista de eventos em sua forma tradicional, e este é o objetivo da simulação SRIP (REED et al., 1988).

Na simulação SRIP o sistema é visto como um conjunto de processos lógicos PL_0, PL_1, ..., PL_n, cada um representando um processo físico. Toda a interação entre os

processos físicos é modelada por mensagens (eventos com *timestamp*) enviadas entre os correspondentes processos lógicos. Cada processo lógico contém uma parte do estado correspondente ao processo físico a que corresponde, bem como um relógio local que denota quanto a simulação avançou.

Em uma simulação SRIP pode-se garantir que erros de causalidade não ocorrem se, e somente se, cada processo lógico executar eventos em ordem crescente de *timestamp*.

Para evitar que os erros de causa e efeito ocorram, diversos mecanismos de sincronização têm sido propostos. Tais mecanismos podem ser divididos basicamente em duas categorias: conservativos e otimistas. Os protocolos conservativos impedem a possibilidade de qualquer erro de causalidade, determinando quando é seguro processar um evento, ou seja, determinando quando todos os eventos que poderiam afetar os outros já foram processados. Já os protocolos otimistas utilizam a técnica de detecção e recuperação. Neste caso, os erros de causalidade são detectados e um mecanismo denominado *rollback* é utilizado na recuperação desse erro (FUJIMOTO, 2000).

Historicamente, o primeiro mecanismo de sincronização para simulação distribuída seguia a abordagem conservativa e foi desenvolvido independentemente por CHANDY e MISRA (1979) e por BRYANT (1977), sendo então denominado CMB (Chandy, Misra e Bryant).

A principal característica deste protocolo é a determinação de quando é seguro executar um evento, ou seja, o evento só é executado quando há garantias de que nenhum outro evento com *timestamp* menor chegará para ser executado. O processo fica bloqueado enquanto isso não é garantido, o que pode gerar uma queda no desempenho e situações de *deadlock*.

Os mecanismos conservativos diferem entre si na maneira como o *deadlock* é tratado. Alguns previnem o *deadlock* e outros deixam que ele ocorra e depois utilizam algum mecanismo para recuperar o sistema.

O protocolo CMB pode previnir o *deadlock* com a utilização de mensagens nulas ou da transmissão de mensagens nulas sob demanda. Uma outra abordagem consiste em não prevenir, mas detectar e recuperar os processos em *deadlock*.

Uma técnica importante utilizada nos protocolos conservativos é o *lookahead*. *Lookahead* pode ser definido como o intervalo de tempo em que um processo pode garantir o que acontecerá no futuro com absoluta certeza (MISRA, 1986) ou a capacidade de prever o que acontecerá, ou o que não acontecerá, no tempo de simulação futuro (FUJIMOTO, 2000). Um *lookahead* de L representa que o processo pode garantir que nenhuma mensagem nova de evento será criada com *timestamp* menor do que $T + L$, onde T é o tempo de simulação do processo.

Os protocolos otimistas, ao contrário dos conservativos, não impedem os erros de causa e efeito, não precisando determinar quando é seguro processar um evento. Quando um erro é detectado um procedimento é chamado para recuperar o sistema e voltar a um estado seguro (*rollback*). A vantagem destes mecanismos é a possibilidade de explorar completamente o paralelismo em situações onde erros poderiam ocorrer, mas não ocorrem.

O mecanismo *Time Warp* é o mais conhecido protocolo otimista. Foi proposto por Jefferson (1985) e baseia-se no paradigma de Tempo Virtual (*Virtual Time*).

2.3.2.2 *Multiple Replication in Parallel* – MRIP

A técnica *Multiple Replications in Parallel* (MRIP) é uma alternativa quando se deseja utilizar diversos processadores para realizar uma simulação. Ao contrário das técnicas utilizadas na SRIP, na técnica MRIP várias instâncias (denominadas a partir daqui como replicações) de um mesmo programa de simulação seqüencial são executadas independentemente em diferentes processadores (GLYNN; HEIDELBERGER, 1992; EWING et al., 1999). Deste modo, primeiramente os dados do período transiente devem ser eliminados em cada replicação e, após atingir o período de estimação dos parâmetros, os resultados produzidos por cada replicação começam a contribuir na análise da variável estimada. Esses resultados são enviados para um analisador global que calcula o intervalo de confiança da(s) variável(is) estimada(s) e quando esta(s) atinge(m) a precisão requerida, a simulação é encerrada (EWING et al., 1999). Devido ao fato de que cada replicação é a execução de uma simulação seqüencial, esta deve utilizar algum método para cálculo de intervalo de confiança.

A abordagem MRIP pode ser amplamente utilizada em qualquer sistema, independente do sistema possuir paralelismo ou não. As replicações só não são aplicadas quando (GLYNN; HEIDELBERGER, 1992):

1. O modelo é grande o suficiente de modo que uma replicação não pode ser executada em um único processador, em um tempo razoável;
2. A variância de cada replicação é muito pequena de modo que os resultados são quase determinísticos. Neste caso, replicar é um desperdício de processamento.

Apesar do método ser conceitualmente simples, alguns cuidados devem ser tomados em relação ao número de processadores (replicações), ao tamanho das replicações e à quantidade de observações a serem eliminadas no período transiente de modo a gerar um intervalo de confiança válido dos parâmetros observados (GLYNN; HEIDELBERGER, 1992).

Todo processo de estimação tem um pouco de variabilidade e esta variabilidade é normalmente medida pela variância do estimador (normalmente, estimadores da média e da variância). Heidelberger (1988) provou que as *bias* (outra componente que mede a variabilidade) dos estimadores mais comuns aumentam proporcionalmente ao número de processadores. Desse modo, quando esses métodos são utilizados para analisar resultados paralelos, seus desempenhos precisam ser avaliados, pois, devido ao efeito da paralelização, a combinação linear dos valores estimados tende a convergir para o valor errado.

As vantagens na utilização da MRIP são (EWING et al., 1999):

- Pode ser aplicado a qualquer programa de simulação, sem a necessidade de paralelizá-lo ou modificá-lo;
- Uma vez que as simulações são independentes, se n replicações estão sendo executadas em n processadores, n resultados serão produzidos enquanto na versão seqüencial somente um seria produzido, ou seja, produz um *speedup* aproximadamente igual ao número de processadores;
- Tolerância a falhas, uma vez que são várias instâncias de um mesmo programa sendo executadas. Um cuidado especial deve ser tomado quanto ao analisador global pois se ocorre algum problema neste analisador, a simulação pára.

2.4 Considerações Finais

Este capítulo abordou uma parte da teoria de Planejamento de Experimentos que tem grande importância na proposta deste trabalho. Primeiramente foram introduzidos conceitos chave sobre o tema, que apontam os objetivos e a conceituação nos moldes do escopo deste trabalho. Em seguida, foram apresentados projetos de experimentos mais comuns, bem como um método para análise da influência dos fatores nas variáveis de saída e também a interação entre estes fatores. E, finalizando esta parte, apresentou-se o tratamento dos resultados de uma simulação.

Este capítulo abordou também, em sua segunda parte, conceitos gerais de Simulação e seu desenvolvimento para implementação em computadores e também sua extensão para execução em vários computadores em paralelo. A Simulação considerada neste projeto é a Simulação de Eventos Discretos com modelos dinâmicos e contínuos no tempo, estocásticos e de estados discretos. Leva também em consideração a abordagem MRIP para execução em paralelo devido à ausência de comunicação entre as réplicas de cada experimento gerado. Esta decisão é justificada devido ao fato de que o intuito deste trabalho é execução de Simulação em ambiente de Computação em Grade, o qual é interconectado por uma

rede de alta latência. A aplicação de simulação SRIP em tal ambiente pode comprometer o desempenho da simulação ao ponto de poder inviabilizá-la.

Capítulo

3

Computação em Grade

3.1 Considerações Iniciais

Computação em Grade é um paradigma de computação que provê uma infraestrutura para computação e gerenciamento de dados para a concepção de 'sociedades' globais, sendo aplicada em diversas áreas como: de negócios, governamental, científica e de entretenimento. As grades propiciam um ambiente virtual para compartilhamento de recursos através da Internet integrando recursos de comunicação, informação e computação (BERMAN et al., 2003).

Este capítulo apresenta os principais conceitos relacionados com a teoria que embasa Computação em Grade, apresentando primeiramente uma perspectiva histórica da evolução do paradigma (seção 3.2), passando em seguida para a descrição das principais características encontradas em sistemas que implementam tal paradigma (seção 3.3). Também são descritos aspectos sobre a arquitetura de uma grade (seção 3.4). Por fim, é apresentada uma breve descrição e comparação de Computação em Grade com outros domínios da área de Sistemas Distribuídos (seção 3.5).

3.2 Histórico e visão geral

3.2.1 Definição

Como apresentado nas sub-seções seguintes (3.2.2, 3.2.3, 3.2.4), uma definição que todos aceitem e entendam como Computação em Grade ainda não existe, algo semelhante ao que também acontede com Sistemas Distribuídos. No entanto, a seguinte definição é bastante completa:

"Uma infra-estrutura de *hardware* e *software* distribuída geograficamente em larga escala composta por recursos heterogêneos interconectados através de uma rede. Estes recursos são pertencentes, compartilhados e administrados por múltiplas organizações. Eles também são coordenados de forma a prover transparência, confiabilidade, persistência e consistência para uma grande faixa de aplicações. Estas aplicações podem implementar algum tipo de computação distribuída para se obter aumento de *throughput*, computação sobre demanda, computação intensiva de dados, computação colaborativa e multimídia." (BOTE-LORENZO et al., 2004)

Desta definição podem ser destacados os seguintes aspectos:

1. Larga escala;
2. Distribuição geográfica;
3. Heterogenidade;
4. Compartilhamento de recursos;
5. Múltipla administração;
6. Coordenação de recursos;
7. Acesso transparente;
8. Confiabilidade;
9. Consistência;
10. Persistência.

Houve muita evolução sobre aquilo que é considerado uma grade no cenário atual. A seguir estão algumas considerações históricas e suas respectivas relações com os conceitos enumerados anteriormente.

3.2.2 Primeira geração

Segundo Berman et al. (2003), a primeira geração de ambientes de Computação em Grade teve como principal objetivo foi interconectar centros de super-computação nos Estados Unidos.

Smarr e Catlett (1992) definiram o termo metacomputador, o qual significa um ambiente que provê acesso transparente a vários super-computadores de forma a serem operáveis ou permitirem acesso transparente. Um passo adiante foi dado por Grimshaw et al. (1992), que propôs um sistema para a interoperabilidade de sistemas heterogêneos pertencentes a múltiplos domínios adiministrativos para fins de compartilhamento de recursos. Tal sistema foi nomeado como metasistema. Grimshaw et al. (1997) definem que tais sistemas tendem a ser geograficamente distribuídos e podem não ser somente compostos de computadores, mas também outros dispositivos, como telescópios.

Naquela época, a definição de um sistema semelhante às grades que temos hoje correspondia a supercomputadores virtualmente conectados por rede, disponibilizando ambientes de execução onde redes de alta velocidade são usadas para conectar computadores de alto desempenho, base de dados, instrumentos científicos, e avançados sistemas de visualização, muitas vezes distribuídos geograficamente (FOSTER; KESSELMAN, 1997).

3.2.3 Segunda geração

A segunda geração destaca-se pela evolução dos metasistemas. Tais sistemas foram responsáveis por possibilitar o provimento de serviços de computação de alto desempenho por meio de ferramentas mais genéricas e de fácil implementação. Nesta geração surgiram várias ferramentas e utilitários que possibilitavam um nível mais alto tanto no ponto de vista de operabilidade do usuários, quanto para o desenvolvimento de aplicações. Assim como uma evolução considerável dos escalonadores de recursos típicos para um metasistema e a criação de *Brokers* específicos para o tipo de acesso exigido. O grande foco ficou por conta da criação de *middlewares* para sistemas escaláveis e heterogêneos para grandes volumes de dados e recursos distribuídos em larga escala (BERMAN et al., 2003).

A *National Computational Science Alliance* denomina um protótipo desenvolvido como Grade Nacional de Tecnologia (*National Technology Grid*) em analogia ao sistema de energia elétrico também organizado em forma de grade (STEVENS et al., 1997). O ponto de intersecção que leva a esta analogia é a idéia de que como a energia energia não pode ser armazenada indefinidamente, o ciclos de processamento de um computador também não podem. Assim, como uma residência se conecta à infraestrutura que provê energia elétrica para consumí-la, um computador se conectaria à infraestrutura para consumir

ciclos compoartilhados de processamento de outros computadores.

Neste estágio de evolução são destacados um aumento na escala dos computadores conectados a um ambiente de grade. Também pôde-se notar uma evolução e o aumento da eficiência na implementação de questões como confiabilidade, consistência e persistência dos dados. Uma vez que tais ambientes possuiam vários domínios adiministrativos de participação em estágios dinâmicos de tempo (FOSTER; KESSELMAN, 1999a, 1999b).

3.2.4 Terceira geração

A geração atual das grades, a terceira, destaca-se principalmente pela virtualização (detalhes na seção 3.3.3) dos recursos a fim de que a interface seja provida por Serviços *Web*. Nesta época também é adotado o termo Organização Virtual (detalhes na seção 3.4).

Foster et al. (2001) apontam que o verdadeiro e específico problema acerca do conceito de grade é a coodernação do compartilhamento de recursos, bem como a solução de problemas de organizações virtuais dinâmicas e multi-institucionais, onde o foco deve ser mantido em questões de compartilhamento de recursos e suas respectivas coordenações. Com isto, é possível delimitar melhor o escopo que um ambiente de computação em grade tem.

Basicamente, o estágio atual consiste da evolução dos middlewares para componentes mais reutilizáveis, focando possibilitar uma integração de somente as partes necessárias para compor as funcionalidades do ambiente em formação. Um passo além foi a definição de uma arquitetura para o desenvolvimento de ambientes de Computação em Grade.

Ao decorrer deste capítulo são comentados detalhes sobre os ambientes de Computação em Grade e que servem como uma visão mais detalhada sobre esta geração.

3.3 Caracterísca

Nesta seção serão detalhadas características dos ambientes de Computação em Grade.

3.3.1 Classificação

Segundo Chede (2004), as grades podem ser classificadas de acordo com sua abrangência, semelhantemente ao que acontece em redes de computadores. Existe uma evolução das redes de computadores considerando-se sua expansão, desde as redes locais e *intranets* para um nível de conectividade restrito aos limites da instituição que pertencem, às *ex-*

tranets com redes de empresas parceiras se interconectando, até a Internet, com conexões amplas. Seguindo esta mesma linha para classificação, as grades podem ser classificadas:

- Intragrids: comumente denominados grades de departamentos, ou "*Enterprise Grids*." Elas são restritas aos limites da rede diponibilizada pela instituição;
- Extragrids: são grades construídas entre empresas parceiras de negócio. Também podem ser conhecidas como "*Partner Grids*;"
- Intergrids: têm uma abrangência global. Fazem uso da Internet a fim de criar uma grade capaz de interconectar participantes geograficamente situados em qualquer local do planeta.

3.3.2 Tipos de uso

Os ambientes de Computação em Grade podem ser classificados de acordo com sua principal forma de utilização. Entretanto, não existe uma perfeita aceitação sobre esta classificação ou taxonomia. Em Bote-Lorenzo et al. (2004) é apresentada uma classificação bem detalhada. No entanto, a principal distinção pode ser feita entre Grades Computacionais e Grades de Dados (STOCKINGER, 2007).

Grades Computacionais têm como principal forma de uso o processamento. Nesta modalidade, os recursos compartilhados consistem basicamente do processador. Cada participande da grade disponibiliza para os outros participantes ciclos de processamento (cluster, supercomputador, etc) a fim de reduzir o tempo total de computação de uma determinada tarefa, a qual é grande o suficiente a ponto de sua execução ser muito demorada quando executada apenas com os recursos locais disponíveis. Simulações, estudo de parâmetros e otimização combinatória são exemplos de aplicações típicas. Outro exemplo pode ser observado na computação intensiva de dados, na qual o objetivo é extrair e sintetizar informações a respeito de uma grande quantidade de dados utilizando diversos recursos em paralelo.

Grades de Dados têm o foco no armazenamento distribuído dos dados bem como na manutenção de réplicas. Neste caso, o fator que leva uma instituição a se integrar à grade é a limitação de infraestrutura local, a qual não viabiliza o armazenamento de dados devido à sua enorme quantidade, assim como a criação de réplicas. Uma Grade de Dados visa disponibilizar um ambiente que permita a criação de banco de dados altamente distribuídos. O ambiente deve ser capaz de distribuir de forma balanceada os dados de acordo com a capacidade compartilhada por cada participante. Devem também sintetizar e recuperar estes dados que geralmente constituem repositórios distribuídos de dados, bibliotecas digitais e banco de dados. Um exemplo pode ser observado em um cenário

onde existem várias instituições com o objetivo de compartilhar dados científicos entre si. Outro cenário, é uma grade onde os participantes disponibilizam como recurso dispositivos de armazenamento e não somente informação.

Uma classificação adicional que deve ser citada é a de Grades de Serviços (BUYYA et al., 2002). Trata-se de uma proposta mais flexível e que permite uma amplitude maior para a especialização de acordo com a necessidade da entidade que visa a implementação de uma infraestrutura de grade. Este tipo de classificação, mesmo sendo mais ampla, ainda é bastante válida, pois o serviço diponibilizado pode ser implementado pelo participante. Outro fator importante é que atualmente o padrão de fato para construção de ambientes de grades trata-se do `Globus Toolkit`, o qual apresenta uma arquitetura orientada a Serviços *Web*. A infraestrutura implementada pode prover serviços de forma colaborativa, serviços multimídia e sob-demanda.

Devido à complexidade e à especificidade dos requisitos de cada tipo de ambiente de grade, o mais comum é que projetos sejam criados com o objetivo de solucionar uma modalidade em específico.

3.3.3 Propriedade

A seguir são listadas as propriedades mais comuns para um ambiente de Computação em Grade (STOCKINGER, 2007):

- Colaboração;
- Agregação;
- Virtualização;
- Orientação a Serviço;
- Heterogeneidade;
- Controle descentralizado;
- Padronização e interoperabilidade;
- Transparência de acesso;
- Escalabilidade;
- Reconfigurabilidade;
- Segurança;

- Suporte a aplicações;
- Modelo de Computação;
- Modelo de licensas de uso;
- Procedimentos e políticas;
- Auditoria.

A **colaboração** é um dos conceitos chaves para a existência dos ambientes de Computação em Grade. O principal conceito é que cada participante da grade compartilhe algum recurso não necessariamente exclusivo, podendo haver mais de um provedor para um recurso em específico. Assim, o resultado da união de todos os recursos disponibilizados pelos participantes produziria um valor superior ao valor do conjunto destes participantes se atuassem de forma individual, ou seja, a colaboração visa uma sinergia. No entanto, garantir uma justiça (igualdade de doação) entre os participantes bem como a perfeita implementação é um desafio. Desafio o qual não inviabiliza tal implementação.

Os benefícios que uma grade, formada e em operação, proporciona são maiores do que a soma das partes que as compõem, sendo que a capacidade individual de cada recurso é preservada. De um ponto de vista mais global em direção ao ambiente de Computação em Grade, a **agregação** permite que aplicações maiores sejam executadas de forma mais rápida. Do ponto de vista local, permite que novas aplicações sejam executadas. Se existe uma agregação de recursos de processamento, logo existirá uma disponibilidade maior para computação. Se recursos não existentes em uma instituição participante se tornam presentes na grade, possivelmente existirão novas abordagem para execução.

Devido ao fato de que os participantes de uma grade estão normalmente geograficamente dispersos, existe uma necessidade de acesso remoto aos recursos. Os recursos podem ser tanto dados como recursos computacionais, porém não limitados a estas categorias. Os ambientes de Computação em Grade devem prover, na medida do possível, uma interface que tem como o objetivo esconder a complexidade do recurso físico. Esta interface forma uma camada abstrata entre o cliente e o recurso físico. Tal abordagem é denominada **virtualização**.

A virtualização é baseada na possibilidade de gerenciamento de convenções de nomes, informações de estado, métodos de acesso e operações remotas independentemente do recurso remoto. As necessidades para ambientes de Computação em Grade podem ser (STOCKINGER, 2007):

- Virtualização de nomes em um contexto (*name spaces*). Permitir a existência de nomes lógicos para os recursos, usuários, arquivos remotos;

- Virtualização do acesso. Permitir gerenciamento a autenticação e autorização ao recurso remoto;
- Virtualização de restrições. Permitir o gerenciamento no controle de acesso ao recurso remoto;
- Virtualização de operação. Disponibilizar uma interface para operação do recurso;
- Virtualização da rede. Disponibilizar a comunicação através dos dispositivos da rede como *firewalls*, balanceadores de carga, redes virtuais privadas (*private virtual networks*);
- Gerenciamento da latência. Minimizar o número de mensagem enviadas através de *wide area network* (WAN);
- Federação. Interoperar por ambientes heterogêneos de Computação em Grade, no que diz respeito à solução que permite a integração no ambiente. Por exemplo, a interoperação entre um ambiente onde está instalado o Globus com outro onde está instalado o OurGrid. O que requer autenticação estilo *Shibboleth* (INTERNET2, 2008) para a verificação de autenticidade verificada com o ambiente o qual o cliente pertence.

Pela facilidade fornecida pela virtualização, os recursos podem ser tratados como **serviços**. Em conjunto com a dispersão geográfica outro fator é constante: a **heterogeneidade**. Como cada participante compartilha o recurso que tem, não existe uma forma de padronizar ou até mesmo alterar o recurso de forma que ele se torne homogêneo à grade. Logo, sempre haverá uma variedade de componentes de hardware e software com diferentes características de desempenho e latência.

Outro ponto é que uma Grade não possui um único dono, o qual possui controle total perante o sistema como um todo. Os componentes da Grade estão sobre o controle de múltiplos domínios. Assim, o controle do sistema é feito de forma **descentralizada**.

Igualmente ao funcionamento da Internet, à qual atualmente é baseada em protocolos que permitem a interação entre os diversos ponto da rede, o mesmo acontece em um ambiente de Computação em Grade. Quando existe uma **padronização**, todos aqueles participantes que aderem à grade são capazes de interagir, ou seja, garantem a **interoperabilidade**. Esta padronização também é refletida nos serviços de forma que permitam não somente nas interações entre o cliente e o provedor, mas também nas tarefas dos usuários. O sucesso da implementação de um ambiente de Computação em Grade depende diretamente desta padronização, uma vez que os recursos são heterogêneos. Uma grade sem padrões é sem utilidade, uma vez que não existe uma garantia de entrada (dados de entrada), trabalho (processamento) e retorno (dados de saída).

O uso de uma grade deve permitir acesso à infraestrutura sem ter um conhecimento apurado no que diz respeito à sua arquitetura ou topologia de rede. Esta **transparência** é bastante relacionada com a virtualização.

A **escalabilidade** é uma questão que deve ser tratada hereditariamente, pois foi herdado dos Sistemas Distribuídos e da Computação Paralela. A Computação em Grade pode ser considerada como uma especialização de Sistemas Distribuídos e Computação Paralela. Da mesma forma, questões de **reconfigurabilidade** podem ser citadas, onde deve ser permitido que novos participantes se juntem ou se desliguem do ambiente sem atrapalhar o seu funcionamento.

Segurança é uma presença garantida em uma ambiente de Computação em Grade. Somente usuários ou aplicações autorizados têm acesso a um recurso, onde este acesso é sempre restrito. Em um ambiente real de Computação em Grade a segurança é a primeira coisa que um participante tem que aceitar e respeitar. Ela também tem a função de negar acesso aos recursos para usuários 'externos' e não participantes.

Não existe uma restrição clara quanto ao **modelo de programação** diretamente inserido no ambiente de Computação em Grade. Entretanto, existe a condição de que cada participante está, na maioria dos casos, inteligado por meio de uma rede de alta latência. Isto não impede de haver comunicação (troca de informação) entre os participantes, porém o ambiente se torna propício a modelos que converjam para abordagens conhecidas como *bag-of-tasks*. Isto influencia a forma como as aplicações são desenvolvidas, ou seja, o **suporte a aplicações**. O ambiente de grade deve ser dirigido pelos dados, de forma que as aplicações que se adaptem a eles.

Questões sobre a **licensa de software** também devem ser consideradas. Pela origem acadêmica e pela ausência de ônus na utilização de *softwares* de código aberto, a licensa de código aberto é o tipo de licensa mais comum. **Procedimentos e políticas** devem ser seguidas para uma perfeita interação entre os participantes. Neste aspecto, ainda existem modelos econômicos que podem ser seguidos. Outro ponto importante é ter a possibilidade de examinar o ambiente para verificar a eficiência em que ele opera, bem como levantar dados sobre quais operações foram feitas e por quem foram feitas. **Auditoria** é uma propriedade que permite tais procedimentos.

3.4 Arquitetura de uma Grade

A arquitetura de um ambiente de Computação em Grade pode ser representada como um modelo de camadas. A figura 3.1 ilustra a representação em camadas e a relaciona com a arquitetura (ou pilha de protocolos) da Internet.

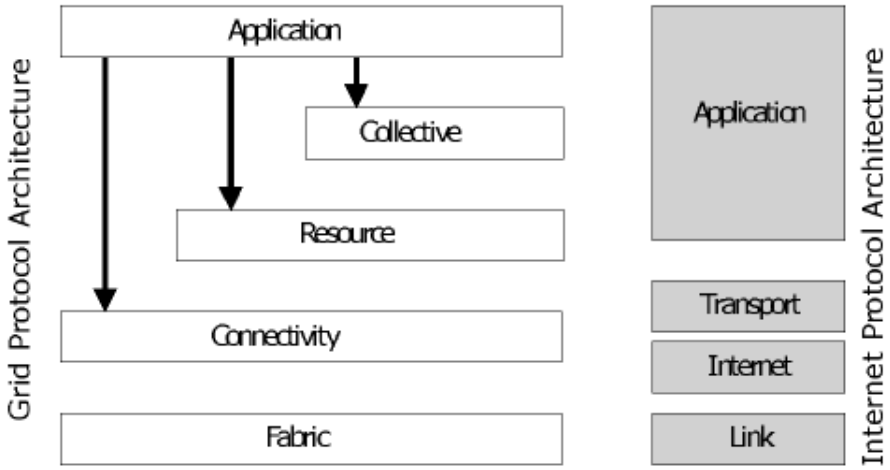

Figura 3.1: Comparação entre os modelos arquiteturais em camada de Grades e Internet (FOSTER et al., 2001)

A camada *Fabric* abriga os recursos que são compartilhados para uso com os demais participantes da Grade como, por exemplo, sensores, processadores, discos, licenças de software, etc, e também os componentes que implementam as operações locais requisitadas pelas camadas superiores (FOSTER et al., 2001). A forma com a qual a camada *Fabric* é implementada em um provedor de recursos reflete diretamente a maneira com a qual os clientes interagem com os recursos compartilhados pois, se essa camada oferecer operações mais sofisticadas de compartilhamento (reserva de recurso, por exemplo), o acesso e utilização desses recursos podem ser realizados de uma forma mais elaborada.

Sendo a Computação em Grade um paradigma de computação distribuída, são necessários mecanismos de comunicação e segurança. Para isso, a camada *Connectivity* hospeda os principais protocolos e APIs dessa natureza que são requisitados por uma Grade. Alguns protocolos bem conhecidos na Internet são encontrados nessa camada como, por exemplo, TCP, IP, ICMP, entre outros. A autenticação de usuários em uma organização virtual (ver sub-seção 3.4.1) também é implementada nesta camada.

A camada *Resource* faz uso dos protocolos da camada *Connectivity* para efetuar a negociação, iniciação e monitoração de recursos por meio das operações disponíveis na camada *Fabric*. Sendo assim, a camada *Resource* oferece protocolos e APIs que possibilitam a obtenção de informações de carga, configuração, entre outros, de um recurso compartilhado. Além disso, nessa camada os mecanismos de negociação permitem que sejam estabelecidos os requisitos da utilização dos recursos como, por exemplo, Qualidade de Serviço (QoS).

Logo acima da camada *Resource* é encontrada a camada *Collective*. Enquanto a primeira tem a função da manipulação de um único recurso, a segunda é responsável pelo gerenciamento de um conjunto deles. Sendo assim, a camada *Collective* oferece

mecanismos que disponibilizam serviços de diretório (que possibilitam a descoberta de recursos no ambiente de Computação em Grade), co-alocação de vários recursos para uma determinada finalidade, entre outros.

A camada *Application* hospeda as aplicações de usuários que operam no ambiente de Computação em Grade. Essa camada pode interagir diretamente com qualquer uma das três camadas inferiores, devido ao fato de que elas oferecem APIs pré-definidas para essa interação.

3.4.1 Organizações Virtuais

Devido à natureza multi-institucional da Computação em Grade, foi criado o conceito de Organizações Virtuais (*VO*). Trata-se de grupos que têm o objetivo em comum de compartilhar não somente seus recursos, mas também atores (*roles*) com papéis para a resolução de um problema em comum. Estes grupos podem conter um recurso, como um cluster, ou vários. Assim sendo, uma Grade é composta por várias Organizações Virtuais, as quais podem ser identificadas por pertencer a um único domínio administrativo, de forma que um acesso coordenado aos recursos possa ser implementados a partir dela.

O termo virtual neste contexto se refere a uma organização lógica em relação ao mundo real. Por exemplo, sejam três organizações físicas com especialidades distintas: computação, física e química. A partir destas, podem ser formadas duas Organizações Virtuais, onde uma tem o foco de compartilhar recursos ociosos de processamento, cujos participantes são os especialistas em computação e física, e outra formada por especialistas em química e física, que têm o foco no desenvolvimento de um combustível econômico e inofencivo ao meio ambiente. Portanto, uma única entidade física pode participar de uma ou mais Organizações Virtuais. Como ilustrado no exemplo, seu propósito, escopo, tamanho, duração e estrutura podem variar. Mais ainda, elas podem ser dinâmicas pois não existe uma restrição que uma nova organização física possa se juntar a uma organização virtual.

Outro ponto diz repeito ao domínio administrativo. Seguindo o exemplo, podem existir três domínios administrativos, mas quando virtualizados se tornam dois. Neste caso, o domínio administrativo de uma organização virtual é o controle de acesso aos recursos compartilhados que cada Organização física disponibiliza.

O compartilhamento aqui referido vai além de prover acesso direto a computadores, programas, dados, sensores, instrumentos científicos, ferramentas de análise e outros recursos. Se refere, necessariamente, ao acesso altamente controlado para provedores de recursos e consumidores. Portanto, é necessário definir de forma clara e cuidadosa o que vai ser compartilhado, quem tem autorização para compartilhar e sobre quais condições o compartilhamento se dá efetivamente. Considerando questões operacionais, a interação

entre os provedores não se estabelece exclusivamente de forma cliente-servidor, mas de forma ponto-a-ponto, pois provedores podem ser consumidores e vice-versa (FOSTER; KESSELMAN, 2003; RIPEANU et al., 2008; FOSTER et al., 2001).

3.5 Comparação com outros domínios de Computação Distribuída

3.5.1 Computação Distribuída

Computação em Grade é uma tipo de Computação Distribuída. No entanto, existem classificações que julgam Computação em Grade ora como sendo uma generalização da Computação Distribuída, ora como uma especialização. O fato é que sempre existe uma aplicação que consome ou requer mais recursos do que aqueles disponíveis em uma única máquina local e onde uma estratégia para se suprir esta falta é 'juntar'os recursos computacionais através de uma rede de computadores, seja ela pertencente a um consórcio, companhia ou instituição acadêmica. A união destes recursos só é possível através da rede (GRIDCAFé, 2008). Quando o termo Computação Distribuída é usado, a referência é feita a um conjunto de computadores que trabalham juntos para resolver um problema. As principais diferenças entre Computação em Grade e Computação Distribuída podem ser observadas por dois ítens (STOCKINGER, 2007):

- Escalabilidade:

  Ao passo que a Computação Distribuída sempre opera com contextos bilaterais, a Computação em Grade opera de forma N-lateral. O termo contexto significa principalmente segurança, arquitetura e modelo de programação. Bi-lateral e N-lateral se relacionam com a quantidade de participantes que podem utilizar o recurso disponibilizado.

  Número de organizações envolvidas: a quantidade variável de participantes e a pouca dependência entre as partes que compõem a grade.

- Transparência: o papel do ambiente de Computação em Grade em oferecer uma visão transparente em relação ao sistema como uma todo é mais difícil em relação à Computação Distribuída. No caso da grade, deve-se sempre ter em mente que vai existir heterogeneidade e pouco pode-se afirmar sobre a arquitetura dos recursos que participarão.

3.5.2 Internet (*WEB*)

A Computação em Grade tem adotado muitas das técnologias da Internet e dos Serviços *Web*, uma vez que serviço de grade (*Grid service*) é um Serviço *Web* com características adicionais. Entretanto, não existe um acordo entre a fronteira de Computação em Grade e Internet, se é que existe uma. Um ambiente de Computação em Grade é um sistema que é desenvolvido sobre o padrão da Internet, ou seja, um *overlay*. Assim, a Internet provê protocolos para a 'ligação' de recursos, ao passo que Computação em Grade contribui com funcionalidades mais avançadas, tais como execução de tarefas, gerenciamento de dados, segurança nas operações, etc. Do ponto de vista da grade, Serviços *Web* não é uma área de pesquisa diferente, mas sim uma parte dele (STOCKINGER, 2007).

3.5.3 Aglomerado (*clusters*)

Muitas são as divergências no sentido de classificar um cluster como um tipo de Grade Computacional. Os *clusters* surgiram nos anos 80 e consistem no agrupamento de máquinas que executam uma aplicação e utilizam aplicações responsáveis pela comunicação entre processos como, por exemplo, o PVM (*Parallel Virtual Machine*) ou MPI (*Message Passing Interface*) (BUYYA et al., 2002). O termo Grade surgiu por volta de 1995 mesmo tendo o Condor (CONDOR, 2005) realizado computação altamente distribuída ainda em 1985. Apesar de também consistirem na união de máquinas com o objetivo de aumentar a oferta de recursos, quando se fala em grade, geralmente refere-se a um conjunto de máquinas espalhadas geograficamente e que pertençam a mais de um domínio administrativo (FOSTER; KESSELMAN, 1999c). Dessas duas últimas características citadas surgem outras como maior heterogeneidade e compartilhamento de recursos, além da inexistência de um controle centralizado no sistema. O que se discute é se a falta de algumas dessas características acarreta a não-classificação de um sistema como sendo uma Grade Computacional. O que se pode afirmar é que as grades, devido à sua natureza, necessitam de serviços especiais para simular um computador virtual (CIRNE, 2002). Todos os nós da grade são 'autonomos' enquanto no caso de cluster não são (STOCKINGER, 2007). São os casos dos serviços de gerenciamento de recursos, os quais submetem e monitoram a execução de aplicações na grade, do serviço de informação, que coleta dados sobre a utilização dos recursos do sistema, do serviço de gerenciamento de dados, o qual transfere arquivos de uma máquina para outra, e do serviço de segurança necessário, por exemplo, para autenticar de forma única os usuários membros da grade através das diversas instituições fornecedoras de recursos.

3.5.4 Ponto-a-ponto (*peer-to-peer*)

O maior destaque para este tipo de computação é observado no ambiente onde computadores (na maior parte computadores *desktops* utilizados nos domicílios) se conectam em uma rede lógica (*overlay*) sobre a Internet. Uma vez conectados, compartilham arquivos. Um usuário estabelece o que é compartilhado, no caso um diretório com arquivos, e pode acessar os arquivos que outros usuários disponibilizaram. Para saber onde existe a informação que ele procura, é usado um sistema de descoberta que opera de forma distribuída, não existindo um ponto central no sistema. Estas características de controle descentralizado chama muita a atenção da comunidade de Computação em Grade. As duas áreas são distintas, porém cada vez mais a Computação em Grade implementa algoritmos de Computação ponto-a-ponto para oferecer um ambiente menos dependente de pontos centrais.

3.5.5 Comparação: *Grid, Cluster e Peer-to-peer*

A tabela 4.1 destaca diferenças entre técnicas que comumente dificultam a distinção de cada um destes domínios.

Tabela 3.1: Tabela comparativa entre Grades, Aglomerados e Sistemas Ponto-a-ponto

Característica	**Cluster**	**Grid**	**P2P**
Máquinas	*Commodity*	*High-end computers*	Bordas da rede (*desktops*)
Proprietários	Único	Múltiplos	Múltiplos
Descoberta de serviços	Predefinido	Indices: centralizado e distribuído	Descentralizado
Gerenciamento de usuários	Centralizado	Descentralizado	Descentralizado
Gerenciamento de Recursos	Centralizado	Distribuído	Distribuído
Escalonamento	Centralizado	Descentralizado	Descentralizado
Interoperabilidade	*VIA based*	Em progresso (ex: WSRF)	Sem padrões
Imagem única do sistema	Sim	Não	Não
Escalabilidade (valores variáveis)	Centenas	Milhares	Milhões
Capacidade	Garantida	Alta e variável	Variável
Throughput	Médio	Alto	Muito alto
Latência da rede	Baixa	Alta	Alta

Fonte: Stockinger (2007)

3.6 Implementações de sistemas de Computação em Grades

Nesta seção são brevemente descritas duas implementações que foram utilizadas neste trabalho: o Globus Toolkit e o Ourgrid.

3.6.1 Globus Toolkit

O *Globus Toolkit* [1] é tido como o padrão de `facto` para a construção de grades. Ele implementa a especificação *Open Grid Service Architecture* (OGSA), proposta pelo comitê *The Globus Alliance* (GLOBUS, 2005). A principal ideia foi criar um padrão que permitisse a integração de diferentes implementações de *middlewares* de Computação em Grade. Especificamente, ela propõe mecanismos nos quais permitam que Organizações Virtuais criem, gerenciem e formem aplicações compostas de outros serviços (FOSTER et al., 2002).

São definidos serviços básicos operacionais para a Computação em Grade. Este serviços são implementados como Serviços Web (*Web Services*) no Globus e denominados *Grid Services*, os quais são Serviços Web tradicionais que seguem convenções estabelecidas na OGSA e suportam interfaces padronizadas para garantir algumas operações adicionais, como o gerenciamento do ciclo de vida do serviço (MPOG, 2006).

Como mostrado na figura 3.2, o Globus possui um conjunto de componentes – denominado *Common Runtime* – que formam o coração de sua implementação, permitindo que aplicações sejam efetivamente excutadas em recursos; que serviços sejam disponibilizados (*deployment*); que instruções de entrada e saída sejam passadas para o sistema operacionais em questão. Para que aplicações sejam escalonadas, sua execução monitorada, é disponível o *GRAM5*. Para que os dados sejam transferidos de forma segura, eficiente e com alto desempenho, existem os componentes de gerenciamento de dados. Quanto ao que tange a segurança, as permissões e as autenticações, são implementados os componentes de segurança.

Embora exista a impressão de o Globus ser um *middleware* integrado e monolítico, a realidade é que ele foi projetado como modular, podendo ser instalados ou implantados somente aqueles componentes desejados. Em relação ao presente trabalho, os componentes GRAM5 e GridFTP foram diretamente utilizados, portanto são feitas considerações sobre estes componentes.

Como já mencionado, o GRAM é o componente do Globus que permite a submissão

[1] Referenciado daqui em diante apenas como Globus

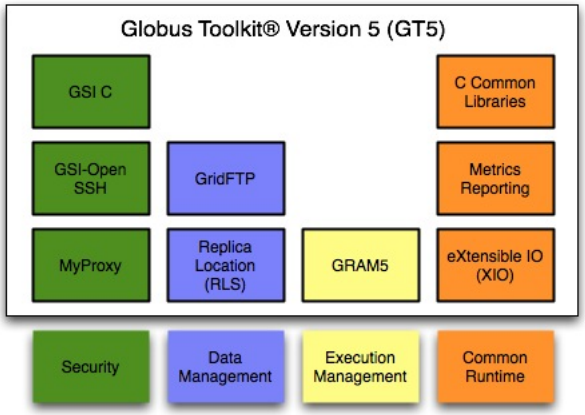

Figura 3.2: Organização dos componentes do Globus Toolkit versão 5

de processos dos usuários na grade. Assim, seu projeto contempla todos procedimentos para a execução de um processo, bem como a manipulação de entrada e saída dos dados utilizados, o seu monitoramento, os avisos de alteração de seus estados e o tratamento das saídas padrão e de erro.

Para executar uma aplicação o usuário faz a submissão através de um dos programas disponibilizados pelo Globus (dentre pode ser citado o `globusrun-ws`), o qual a encaminha para a grade que vai controlar o ciclo de vida do(s) processo(s) que serão criado(s). Esta submissão ou requisição é primeiramente analisada pelo *gatekeeper*, o qual faz a autenticação do usuário requisitor e verifica a disponibilidade para seu atendimento. Uma vez que todos os previlégios sejam concedidos, esta requisição é encaminhada para o *job manager* que invoca o componente que instancia o(s) processo(s) daquela requisição, colocando-os em execução em um *hardware*. Este componente é configurado de acordo com a instalação, sendo comuns o uso do `fork` (chamada de sistema presente nos sistemas Unix-like), `Condor` (escalonador de processos usado em *clusters*), `PBS` (outro escalonador), entre outros.

Em relação ao ciclo de vida de um processo no Globus, um processo pode ocupar os seguintes estados:

Unsubmitted: corresponde a um processo que ainda não foi submetido para execução;

StageIn: a submissão foi aceita, porém todos os dados necessários para a execução ainda não estão presentes. Neste estado, os executáveis e os dados necessários estão sendo transferidos para o recurso executor;

Pending: uma vez que todos os requisitos estejam resolvidos para a execução, o processo entra no estado pendente, aguardando para que seja escalonado;

Active: após escalonado, em um momento este processo será alocado em algum recurso para que seja processado, tornando seu estado como ativo;

Suspended: igualmente aqueles processos dos sistemas operacionais, os quais podem sofrer preempções, este estado configura o processo no estado de suspenso, aguardando para que seja novamente alocado para execução;

StageOut: com a execução terminada, todos os dados gerados pelo processamento, seus arquivos de saída, são encaminhados para o meio de armazenamento final para posterior recuperação pelo usuário;

CleanUp: neste estado é feita a remoção de todos os arquivos gerados pelos estados `StageIn` e `StageOut`;

Done: indica que o processo foi finalizado com sucesso;

Failed: indica que houve uma falha na execução e o processo não executou naturalmente.

O *GridFTP* é um serviço inspirado no protocolo FTP, porém incrementado com aspectos importantes para troca de arquivos em ambiente de grade. Estes aspectos contemplam a transferência confiável e segura de grandes volumes de dados de forma, podendo um arquivo ser distribuído em vários recuruso de uma grade, sendo transferido simultaneamente. Por ser um serviço bastante utilizado, sua instalação é integrada com toda solução Globus, podendo ser instalado individualmente. Este fato possibilita criar pontos dedicados de armazenamento em uma grade, sendo a este ponto associado um *hardware* especial capaz de garantir alto desempenho.

Seu modo de operação permite que hajam operações entre um arquivo armazenado localmente e outros armazenados servidor `GridFTP`; um arquivo remoto seja copiado para a máquina local; e, que operações servidor a servidor sejam feitas. Por exemplo, sejam três organizações virtuais, situadas nas cidades São Paulo, São Carlos e Ribeirão Preto. Um usuário localizado em São Carlos pode solicitar que um arquivo armazenado em um recurso disponível em São Paulo seja transferido para outro disponível em Ribeirão Preto.

As operações com o servidor podem ser feitas diretamente pelo comando `globus-url-copy` e também pela especificação de uma submissão. Neste último caso, as operações serão executadas quando o processo estiver em estado `StageIn`.

3.6.2 Ourgrid

O `Ourgrid` é um `middleware` para computação em grade de código aberto, com uma comunidade aberta a colaboradores que compartilham recursos computacionais ociosos.

O escopo do projeto foi reduzido para aplicações *bag-of-tasks*, o que simplifica a implementação em alguns aspectos. A idéia é reunir de forma simples, rápida, segura e produtiva recursos computacionais como, por exemplo, *clusters* de laboratórios de pesquisa.

O padrão de comunicação utilizado é o ponto-a-ponto, permitindo acesso direto de um usuário a um recurso compartilhado por outro. Segundo seu site (OURGRID, 2007), pessoas não usam seus recursos computacionais incessantemente, mas seu uso alterna entre estados de processamento e ocioso. Neste ponto, o `Ourgrid` propõe que usuários se juntem em uma rede de favores que permite utilizar os recursos ociosos de outros usuários para garantir uma resposta mais rápida de suas execuções. O recurso compartilhado por um usuário não será utilizado caso ele esteja em uso.

O *Ourgrid* possui quatro objetivos principais (CIRNE et al., 2006), a saber:

Rapidez: o tempo de execução um processo ou um conjunto de processos deve ser menor do que se tivesse sido executado localmente, pois se isto não acontecer, o uso do ambiente é desnecessário;

Simplicidade: os usuários de um ambiente de grade querem gastar o menor tempo possível configurando os recursos para compartilhamento. Assim, o procedimento para que o ambiente se torne uma ferramenta produtiva deve ser o menor possível;

Escalabilidade: além de seu conceito natural, o termo também se refere a questões administrativas, significando a inexistência de negociações humanas, bem como definições de papéis e regras de acesso, uma vez que estes procedimentos podem acarretar em um atraso para a utilização dos recursos;

Segurança: por sua natureza de compartilhamento ponto-a-ponto, deve ser capaz de lidar com o acesso de usuários que embora desconhecidos participam da grade.

O figura 3.3 ilustra os componentes deste *middleware*. A "comunidade" é formada por componentes denominados *Peers*, os quais assumem o papel de gardião de um sítio compartilhado. São eles quem recebem requisições de execução e tomam a descisão de aceitá-la ou não. Todos os recursos compartilhados são cadastrados neles, de forma que possam ser consultados e monitorados. Os *Workers* assumem o papel de unidades processantes, ou recursos compartilhados. Quando um requisição chega em um site, ela é alocada a um *Worker* ocioso para que seja processada. O componente *MyGrid* é o ponto de acesso que o usuário temp para interagir com a grade, fazendo submissão de tarefas, monitorando sua execução e coletando os resultados gerados.

No nível do *middleware*, o termo *bag-of-tasks* significa que uma aplicação executando em um *Worker* não comunica com outra, independente se ela seja alocada no mesmo

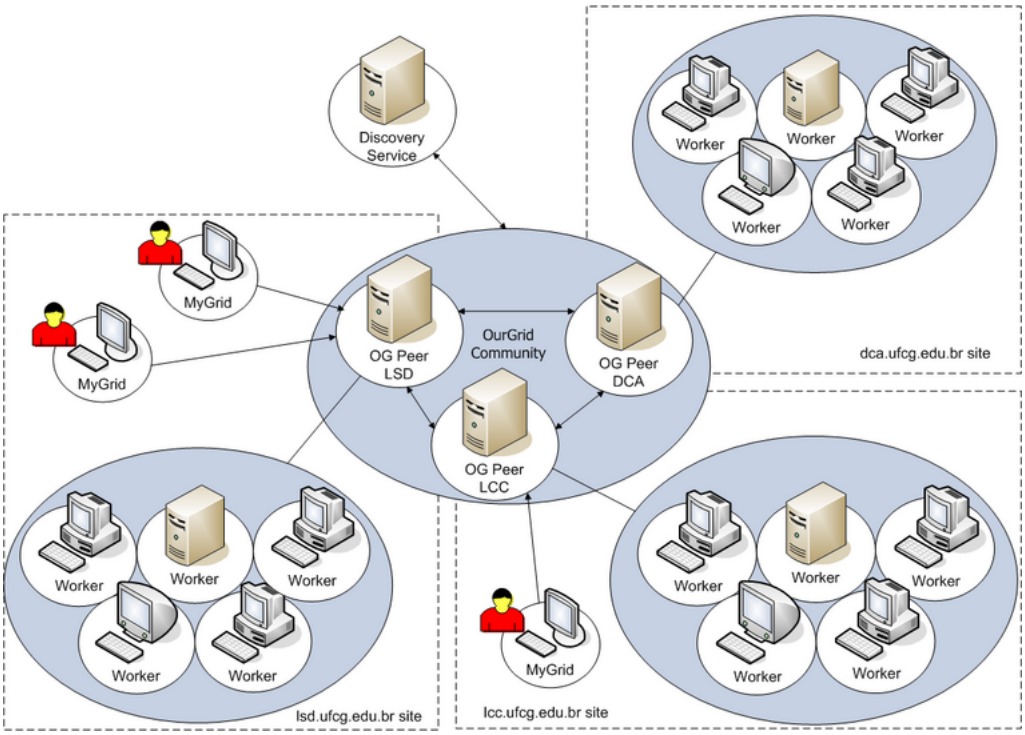

Figura 3.3: Componentes do Ourgrid.

recurso ou em outro. No entanto, não existe uma restrição explícita quanto a isto. Por exemplo, seja um sítio que compartilha seu *cluster* instalando um *Worker* para cada nodo, e possua instalado uma biblioteca para troca de mensagens. Um requisição, ao chegar neste sítio, poderá instanciar outras réplicas do programa executado e elas se comunicarem. Porém, um outro site com recursos que possuam regras de acesso bem definidas e limitações de uso (por exemplo, um *firewall*), a mesma aplicação poderá não funcionar. Como conclusão pode-se observar que, formalmente para o ambiente a comunicação entre tarefas não existe, mas tecnicamente elas são possíveis desde que sejam suportadas.

3.7 Considerações sobre o mapeamento das aplicações em Grades Computacionais

Os passos para implementação de uma Grade Computacional são os mais diversos e englobam vários aspectos sociais (formação de um grupo com interesses comuns), administrativos (jurídicos, economicos, burocráticos, etc.) e técnicos. Sobre o escopo técnico, eles permeiam pela padronização da arquitetura que os serviços da grade em questão vai implementar; pelo padrão que sua infraestrutura deve se moldar; resultando em uma Organização Virtual que instancia uma grade. No entanto, outros procedimentos e decisões precisam ser tomados para que o efetivo compartilhamendo dos recursos aconteça. Dentre eles, destaca-se a organização do *hardware*, a qual mapeia como cada aplicação

será alocada nos recursos disponíveis, podendo ser implementada de forma individual por cada participante de uma Organização Virtual. Assim, nesta seção são discutidos 3 tipos organização focados em Grades Computacionais, não sendo as possíveis organizações de *hardware* limitada a esse número.

Basicamente, são citadas 3 implementações de *middlewares* para computação em grade, a saber: `Globus Toolkit`, `Ourgrid` e `GridGain`. O `Globus` é mais flexível quando comparado a estes outros, pois possui uma vasta gama de componentes que podem ser moldados de forma a se adaptar às necessidades técnicas de um sítio em específico, provendo vários aspectos como aqueles já mencionados na seção 3.6.1. O `Ourgrid` por ter um escopo reduzido à aplicações *bag-of-tasks*, implementa um modelo fixo, porém eficiente para o propósito que se propõe. O `GridGain` é um modelo mais simples e de baixo nível, possibilitando ao desenvolvedor definir o comportamento desejado, em muitos casos, comunicação ponto-a-ponto. Assim, nos próximos parágrafos são mostrados alguns pontos para a organização do *hardware* a ser compartilhado.

O `Globus Toolkit` é um conjunto de ferramentas que facilitam a implementação de um ambiente de computação em grade. Um esboço de uma possível configuração utilizando o Globus é mostrada na figura 3.4. Esta abstração contempla um computador que permite acesso ao domínio compartilhado, referenciado como "*Globus front-end*." Ao receber uma requisição, esta é repassada para os recursos locais capazes de processá-la. Uma vez processados, o estado da requisição é alterado, e o escalonador comunica o componente solicitante que o processo terminou. Os resultados ficam disponíveis para serem recuperados pelo usuário dono da execução.

O `Globus`, no entanto, possibilita que outras abordagens sejam implementadas. É possível que cada recurso de um domínio aceite requisições diretamente, dispensando o uso do *front-end*. É possível que um serviço *web* receba requisições e as repasse para recursos específicos, conforme os parâmetros recebidos por ele.

O `Ourgrid` disponibiliza a configuração mostrada na figura 3.5. Os *workers* não são acessíveis diretamente e recebem solicitações somente do *peer* a que está associado. Para que uma submissão seja efetuada, um cliente envia uma requisição para seu *peer* de registro, e ao recebê-la, este tenta alocá-la no site local, e caso não haja disponibilidade, ele repassa a solicitação daquelas tarefas remanescentes para outro *peer* vizinho.

O `Ourgrid` é bastante rígido quanto a infraestrutura para sua implementação, impossibilitando outras combinações.

Por fim, o último exemplo citado é o modelo implementado pelo `GridGain` (GRIDGAIN, 2009). Cada processo iniciado pode comunicar com outro na topologia, ficando livre o papel que cada nodo componente da grade pode assumir. Este *middleware* é projetado especificamente para aplicações Java, no entanto, possui mecanismos que possi-

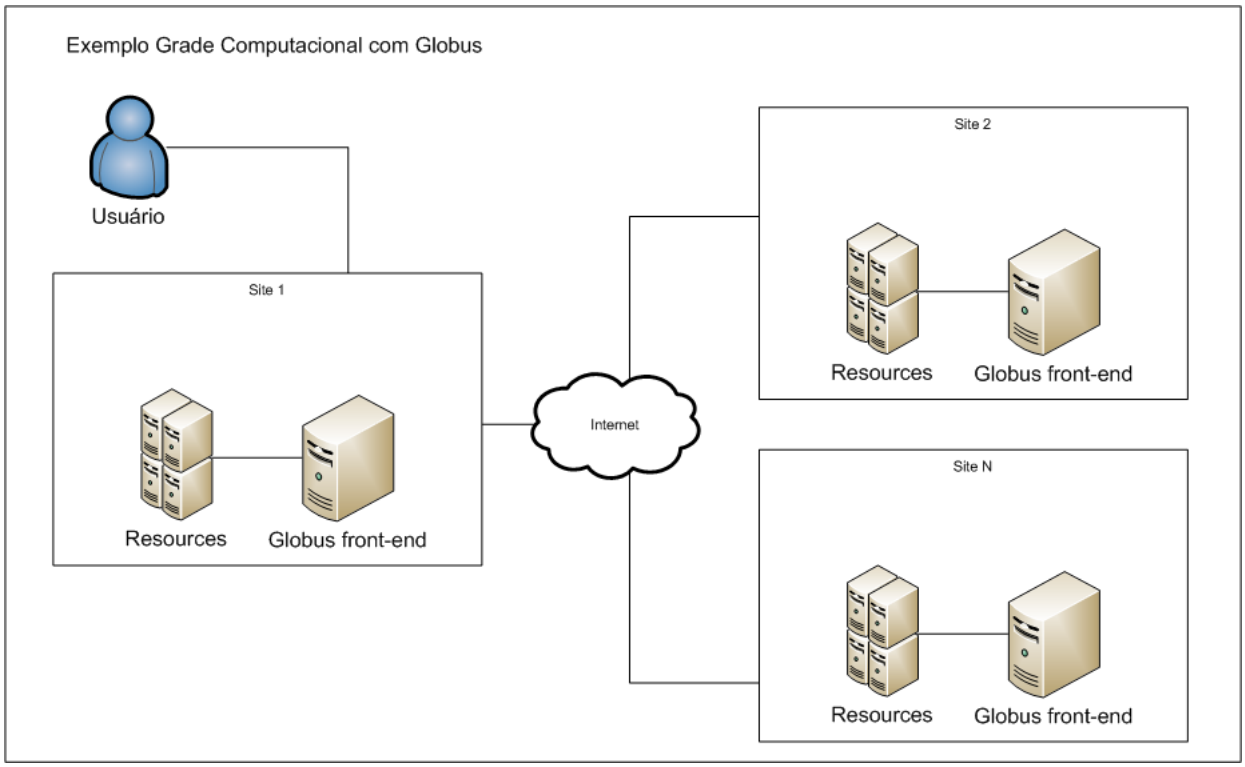

Figura 3.4: Topologia de uma grade Globus

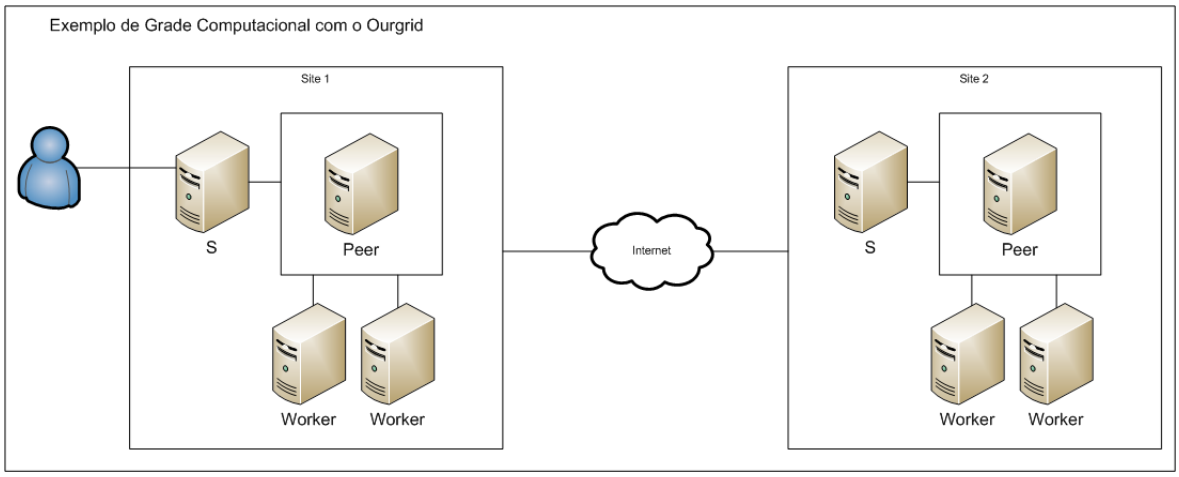

Figura 3.5: Topologia de uma grade Ourgrid

bilitam a execução de programas nativos feitos em Fortran, C, C++ ou outra linguagem de programação. A figura 3.6 mostra um esboço de uma grade organizada pelo `GridGain`. Embora permita uma abordagem mais flexível quanto à topologia da grade, o projeto é uma biblioteca desenvolvida em Java e a aplicação alvo para execução deve ser instrumentada com diretivas (*annotations*) em seu código fonte.

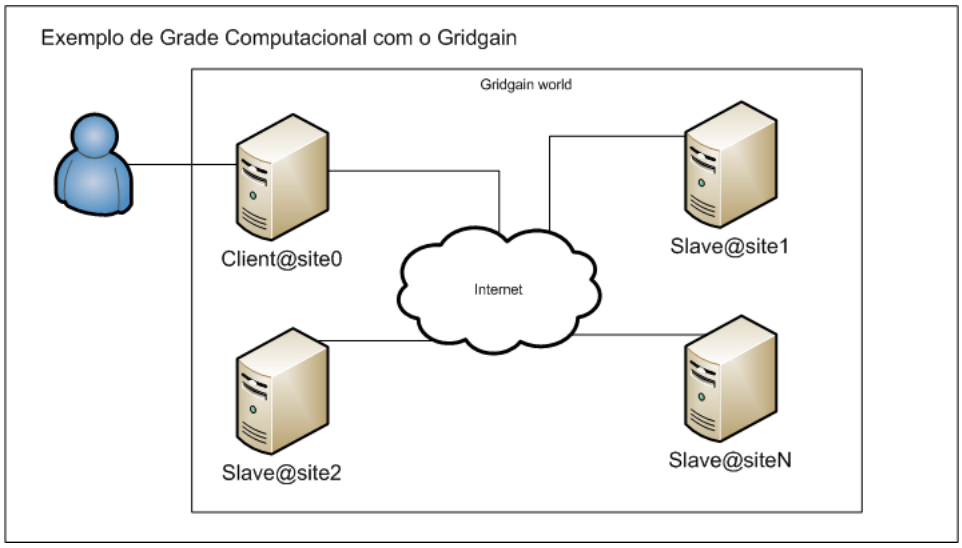

Figura 3.6: Topologia de uma grade Gridgain

3.8 Considerações Finais

Neste capítulo foram estudados os conceitos gerais sobre o paradigma de computação em grade. Primeiramente, definiu-se um ambiente de computação em grade e seu histórico de evolução até o presente estágio de desenvolvimento. Em seguida foram abordados alguns conceitos úteis no que se diz respeito às características e à arquitetura de uma grade mais comumente utilizadas no meio acadêmico. Por fim, discutiu-se a relação da computação em grade com outros domínios da área de Sistemas Distribuídos.

Ao final, foi comentado sobre dois *middlwares* para computação em grade, utilizados como objeto de estudo neste trabalho. Exemplificando, em seguida, possíveis forma de como os processos que são executado em uma grade podem ser alocados nos recursos. O próximo capítulo objetiva a descrição dos principais conceitos de Computação em Grade, que é um ponto chave para o desenvolvimento deste projeto de mestrado.

Capítulo

4

Gerenciador de Experimentos para Ambientes de Computação em Grade

4.1 Considerações Iniciais

A computação em grade no estado atual pode abranger uma grande gama de aplicações, podendo existir compartilhamento de processamento, dados e outros dispositivos não computacionais, como telescópios científicos. As formas de acesso a estes recursos também podem variar, como, por exemplo, via serviços *web*, RMI, Corba, TCP, etc. A computação em grade abrange outros aspectos como segurança, teorias econômicas para uso justo dos recursos, virtualização de recursos, entre outros, os quais não são o foco de estudo neste trabalho, tendo estes aspectos como mecanismos de funcionamento que não influenciam os pontos abordados por ele.

Como uma abordagem para resolver um dos pontos deste trabalho é proposta uma ferramenta para gerenciamento experimentos para que sejam executados em um ambiente de computação em grade. Especificamente, são experimentos de simulação de eventos discretos que usam a técnica MRIP para paralelismo e controle da execução.

Estas simulações não fazem uso de instrumentos científicos específicos. No entanto,

podem necessitar de algum acesso a dados para serem usados como entrada para o modelo e podem também produzir outros dados como resposta. Mas a característica principal é o uso de processamento.

Desta forma, o escopo para computação em grade deste trabalho fica reduzido para aquelas focadas em compartilhamento de recursos computacionais, ou seja, Grades Computacionais. Nas próximas seções são apresentados a solução proposta e os estudo de casos dos dois ambientes de grade estudados neste projeto.

4.2 O gerenciador de experimentos

No projeto de *middleware* para grades computacionais não existe um modelo que seja obrigatório para sua implementação. Como exemplo pode ser citadas três implementações distintas: `Globus Toolkit`, `Ourgrid` e `Gridgain`, como já discutido na seção 3.7.

Os 3 exemplos mencionados não contemplam todas as possibilidades para a organização do *hardware* para um ambiente de computação em grade, mas apenas ilustram casos prováveis quando se trata de grades computacionais. Existe um trabalho de padronização feito pelo Open Grid Forum (OGF, 2005), o qual visa prover protocolos para tarefas comuns a grades, possibilitando que implementações diferentes comuniquem entre si.

Do ponto de vista do desenvolvedor de uma simulação, a utilização de um ambiente de grade mostra-se atraente sobre o aspecto da produção de resultados, uma vez que há a possibilidade de adicionar recursos computacionais além daqueles disponíveis e de seu acesso. Para efetivamente ter acesso a estes recursos, o desenvolvedor, neste contexto, usuário da grade, deve conhecer as particularidade do sistema em questão e fazer a submissão de uma simulação. A figura 4.1 ilustra este esquema. Caso o usuário deseje utilizar recursos disponibilizados por um outro ambiente de grade que utilize um *middleware* diferente daquele já usado, o usuário terá de se adaptar a este, a fim de poder utilizar os novos recursos diponibilizados. Portanto, se o usuário resolve usar várias grades, ele deve aprender as particularidades de cada um destes ambientes.

A figura 4.2 ilustra a proposta de uma ferramenta que centraliza a submissão de simulações em ambientes de computação em grade, adicionando uma camada entre o usuário e os ambientes disponíveis para uso. O usuário acessa um ponto que repassa as requisições para as respectivas grades. Para a submissão são usadas premissas abstratas, onde as particularidades dos ambientes em questão ficam por conta da ferramenta. A esta camada foi nomeada de Gerenciador de Experimentos para Ambientes de Computação em Grade, no entando para fins de simplificação é adotada a sigla GEM, acronimo do inglês *Grid Experiments Manager*.

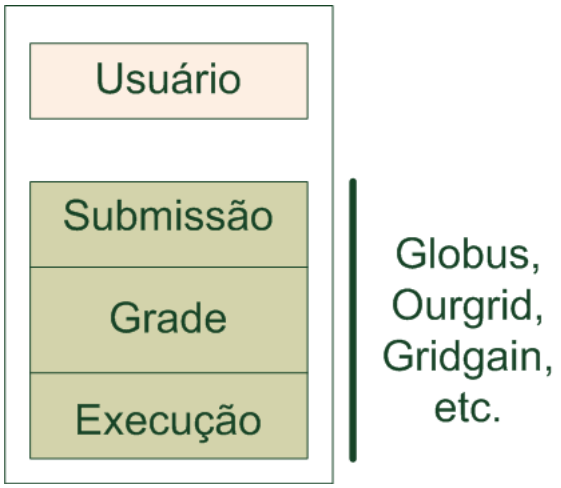

Figura 4.1: Como é feita a interação entre um usuário com uma grade.

Esta proposta permite que sejam adicionadas extensões específicas para cada *middleware* disponível, sendo módulos capazes de lidar com todo o processo de submissão adicionados ao sistema em forma de *plugins*. Questões de segurança quando do acesso devem ser fixadas e configuradas previamente, possibilitando a interação entre da camada com o ambiente de grade, como ilustrado na figura 4.2.

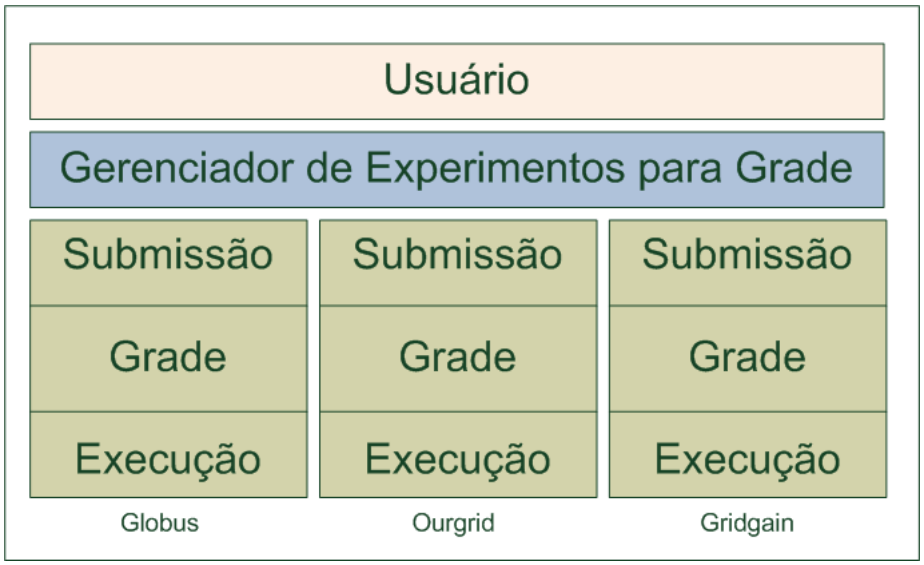

Figura 4.2: Ferramenta para Gerenciamento de Experimentos aplicada a Grades Computacionais.

O projeto da ferramenta GEM contempla a possibilidade de ela ser usada em modo isolado (*stand-alone*), bem como integrada a outras ferramentas. A motivação para seu desenvolvimento partiu do ASDA (BRUSCHI et al., 2004), um ambiente integrado de desenvolvimento de simulações distribuídas. Em sua interface gráfica é possível descrever o modelo em redes de fila e, em seguida, gerar o código da simulação pronto para execução segundo a técnica MRIP. É conveniente para o usuário que, além de gerar o

código, seja possível também um mecanismo para sua execução. Ainda no escopo do ASDA, a ferramenta GEM provavelmente seria integrada a um módulo de planejamento de experimentos, recebendo parâmetros necessários para gerar o conjunto de processos a serem iniciados nos recursos computacionais. Portanto, ao receber a simulação, gerar seus experimentos e executá-los, o produto da computação pode ser repassado de volta ao ASDA para que seja feita a respectiva análise estatística.

Figura 4.3: Exemplo de integração com o ASDA.

Como ilustrado na figura 4.3, o GEM funcionaria como a ponte de ligação entre um sistema desenvolvido para usuários de simulação e os recursos computacionais de uma grade. Cada componente pode executar distribuídamente em locais diferentes, comunicando-se entre através de uma rede.

Um ponto importante no desenvolvimento de sistemas distribuídos é tratar com a forma de conexão entre as partes que o compõem. A figura 4.4 ilustra a conectividade assumida para que todos os componentes do GEM possam funcionar de forma integrada. O usuário terá acesso a um módulo cliente – representado por um computador portátil – que recebe a descrição dos experimentos a serem executados. Os experimentos são processados e enviados ao módulo executor – representado por um computador servidor –, o qual analisa os recursos disponíveis e envia um conjunto de execuções para cada recurso disponível. Após o processamento pela grade o resultado é armazenado no agente gerenciador de experimentos para posterior acesso pelo agente instanciador.

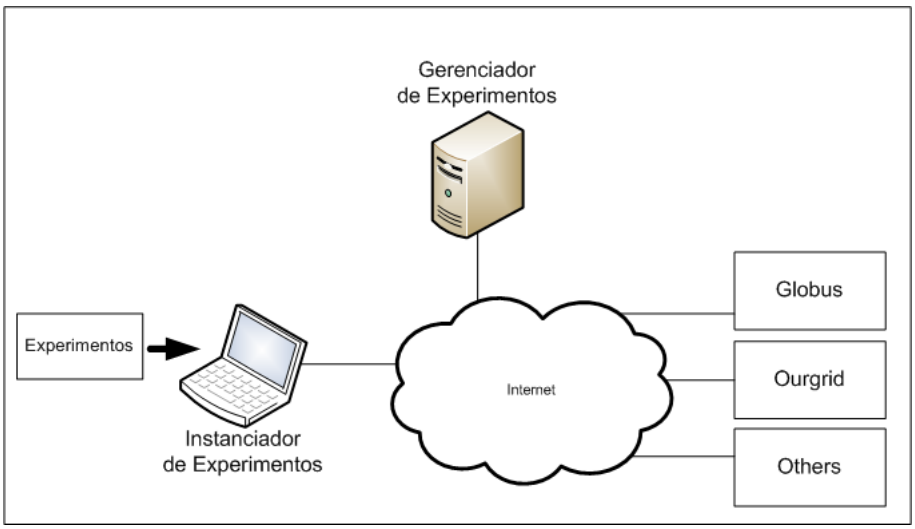

Figura 4.4: Arquitetura do GEM

Nas próximas sessões são descritos detalhes dos agentes, bem como o mecanismo de comunicação implementado.

4.2.1 Agente Instanciador

O módulo denominado agente instanciador foi implementado como no modelo [1] ilustrado na figura 4.5. Seu objetivo é receber o arquivo XML que descreve o experimento, processá-lo e enviar ao agente gerenciador de experimentos o conjunto de comandos necessários para execução de todo projeto de experimento, bem como os arquivos binários e os de entrada e de saída; e, posteriormente, obter as respostas da simulação do repositório de resultados, armazenado no agente gerenciador.

As configurações necessárias, como, por exemplo, as de endereço e porta do agente gerenciador, são feitas pelo componente `ConfigurationMgmt`, sendo responsável por ler o arquivo de configuração e fixar os devidos valores para execução.

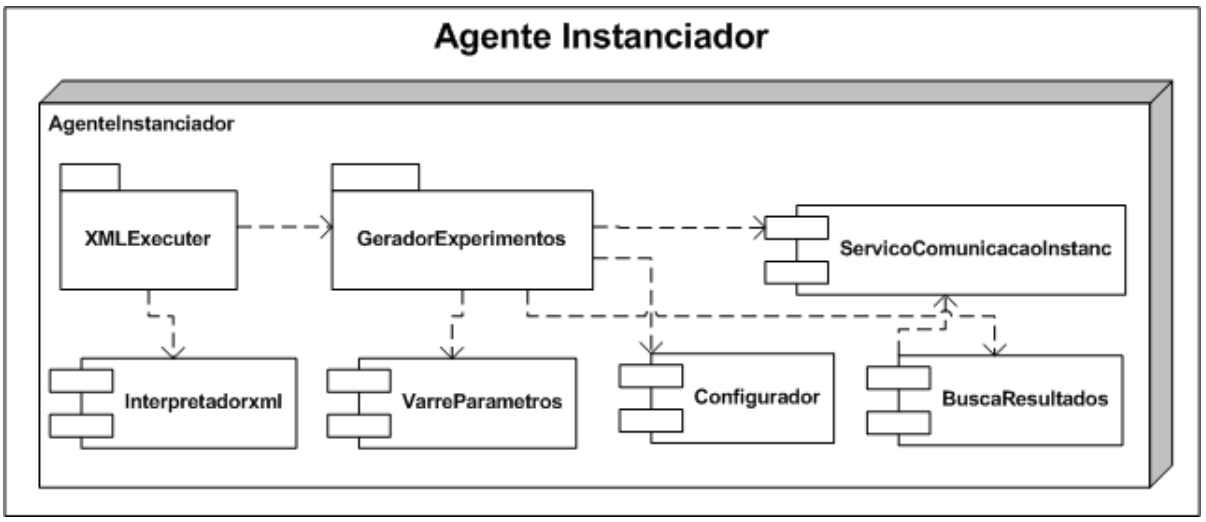

Figura 4.5: Anatomia do Agente Instanciador

Ao ser executado, o agente instanciador recebe um arquivo XML que descreve os parâmetros a serem passados para o programa da simulação. O componente que trata e mapeias as informações descritas no arquivo XML para objetos é denominado `Interpretadorxml`. O esquema XML deve prover mecanismos de descrição para:

O experimento: cada experimento pode ser rotulado com o nome do proprietário e o do projeto de experimentos a fim de ser possível um controle pelo usuário;

Os arquivos: cada experimento pode ter um ou mais executáveis que serão utilizados, podendo variar de acordo com a arquitetura de destino. Dessa forma, deve haver

[1] É usado o Diagrama de Instalação (Deployment Diagram) da UML para a representação dos componentes de *software* da ferramenta GEM.

a possibilidade de se vincular um arquivo executável para um recurso. Da mesma forma, devem ser especificados os arquivos de entrada e de saída, quando existentes;

Os parâmetros: lista de parâmetros cujos valores serão variados de forma fatorial completa. Um parâmetro pode ser do tipo incremental, o qual é descrito com os valores inicial, final e de incremento. Ou, caso contrário, pode ser no formato de uma lista preenchida com os valores possíveis;

A linha de comando: cada linha de comando é formada por três partes, a saber: o prefixo, os parâmetros e o posfixo. Nesta *tag* são especificados os valores prefixo e posfixo, sendo os parâmetros processados pelo componente `VarreParametros`.

Na figura 4.6 é listado um exemplo de arquivo esquematizado para descrever um experimento que possui um arquivo com dois arquivos executáveis (linhas 12 e 13), sendo vinculados a dois recursos específicos (linhas 12 e 13), um arquivo de saída (linha 14), um parâmetro do tipo incremental (linhas 16 a 22) e outro com valores em forma de lista (linhas 23 a 29).

Para que haja o processamento da descrição do projeto de experimento e a geração da lista de comandos necessários, o componente `Interpretadorxml` lê o arquivo de entrada e após mapear as descrições em objetos, esses são passados para o componente `VarreParametros`, o qual irá combinar todas as possibilidades dos valores dos parâmetros e formar um conjunto de linhas de comando, as quais serão executadas nos recursos. Na figura 4.7, o algoritmo exemplifica, de forma recursiva, como as variações são combinadas, produzindo as linhas de comandos.

Ao final, a variável `cmdLines` conterá o conjunto necessário para a execução. Para que sejam enviados ao agente gerenciador, os arquivos necessários são compactados e transferidos utilizando o serviço de comunicação (apresentado na sessão 4.2.3).

4.2.2 Agente Gerenciador de Experimentos

O agente gerenciador, como ilustrado pelo modelo da figura 4.8, recebe do agente instanciador um conjunto de linhas de comando e os arquivos para execução. Seu objetivo é dividir essas linhas de comando entre os recursos disponíveis; enviá-las para execução; e obter o resultado, armazenando-o localmente para posterior recuperação pelo cliente.

Ao ser iniciado esse agente recebe configurações do diretório base para armazenamento do repositório e dos recursos disponíveis.

O repositório corresponde a uma estrutura de diretórios local que possibilita a identificação de experimentos, sendo que para cada experimento existe um subdiretório de

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:experiment
    xmlns:tns="http://www.example.org/gemschema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/gemschema gemschema.xsd ">
  <tns:owner>Lourenço</tns:owner>
  <tns:projectname>t0125</tns:projectname>
  <tns:executable>/u/sim</tns:executable>
  <tns:prefix>--method="BM"</tns:prefix>
  <tns:postfix>out.txt</tns:postfix>
 <tns:targets>
    <tns:file type="bin" resource="lasdpc" file="/tmp/sim64"/>
    <tns:file type="bin" resource="providers" file="/tmp/sim431"/>
    <tns:file file="/tmp/out.txt" type="output"/>
  </tns:targets>
  <tns:sequentialparameters>
    <tns:seqparameter order="1">
      <tns:begin>0</tns:begin>
      <tns:end>10</tns:end>
      <tns:increment>1</tns:increment>
    </tns:seqparameter>
  </tns:sequentialparameters>
  <tns:valueparameters>
    <tns:valparameter order="0">
      <tns:value>FCFS</tns:value>
      <tns:value>SJF</tns:value>
      <tns:value>RR</tns:value>
    </tns:valparameter>
  </tns:valueparameters>
</tns:experiment>
```

Figura 4.6: Exemplo de Projeto de Experimentos

armazenamento dos arquivos necessários à execução. Pode ser que um usuário decida executar um mesmo experimento mais de uma vez e, para isto, o agente cria um subdiretório para cada execução feita de um experimento. Portanto, ao receber uma mensagem do agente instanciador contendo a descrição de um experimento, essa estrutura é criada no sistema de arquivo acessado localmente, e os dados recebidos armazenados.

Uma vez iniciado e após ter recebido uma requisição para execução de um projeto de experimentos, o agente gerenciador extrai a lista de experimentos da especificação do experimento e a divide proporcionalmente – por meio das classes `EscalonadorImpl` e `Metricas` – entre cada recurso disponível, observando o programa executável compatível com o respectivo recurso. Ao sinal de inserção na lista de experimentos, o agente cria um novo diretório, observado o último já existente, faz a conexão com o recurso computacional e dispara o conjunto (ou lote – *batch*) de experimentos a ser executado, que do ponto de vista da grade é uma aplicação. Ao terminar sua execução, um sinal é enviado de volta ao agente e ao recebê-lo, o processa obtendo os arquivos de reposta, armazenando-os localmente no diretório daquela execução. No agente gerenciador são mantidas duas

```
1 String prefix = getPrefix(); // E.g. "/user/grid/simulation "
2 String postfix = getPostfix(); // E.g. " > output.txt"
3 List<Parameter> parameters = getParameters(); // List os parameters
4 List<String> cmdLines; // generated command lines.
5
6 private void generateExperiments(String params,
7         List<Parameter> parameters) {
8     if (parameters.size() == 0) {
9         cmdLines.add(prefix + params + postfix);
10        return;
11    }
12
13    Parameter param = parameters.get(0);
14    List<Parameter> newList = parameters.subList(1, parameters.size());
15    String newParams;
16
17    if (param.isIncremental()) {
18        for (int value = param.getBegin(); value < param.getEnd();
19                value += param.getIncrement()) {
20            newParams = params + " " + value;
21            generateExperiments(newParams, newList);
22        }
23    } else {
24        for (String value : param.getValues()) {
25            newParams = params + " " + value;
26            generateExperiments(newParams, newList);
27        }
28    }
29 }
```

Figura 4.7: Algoritmo recursivo que gera a instância de um experimento

listas para cada recurso disponível: uma para os experimentos pendentes a execução e outra para os experimentos em execução. Em um ambiente de grade é provável que os dados da submissão trafeguem pela Internet, assim, ao processar a lista de experimentos de uma simulação, cada execução é composta de vários experimentos, visando minimizar os efeitos de uma conexão de latência maior. Por exemplo, sejam três recursos, com 5, 3 e 2 computadores respectivamente em cada recurso, e um projeto de experimentos com 50 variações. Serão geradas 3 execuções, com um total de 25, 15 e 10 variações, alocadas respectivamente para os recursos com 5, 3 e 2 computadores. A figura 4.9 ilustra a comunicação existente entre o agente gerenciador e os recursos. Ao finalizar todas as execuções do projeto, os dados ficam disponíveis para o agente instanciador.

No escopo do presente trabalho, `recurso` é um ponto acessível pela rede, capaz de receber um conjunto de arquivos e processá-los. Esta definição não rígida, sendo possível um recurso se apresentar como uma Organização Virtual (VO) contendo vários nodos e cada um, ou parte deles, com o *middleware* de grade instalado, assim como, um conjunto de computadores com um máquina que recebe as requisições para processar. O foco foi

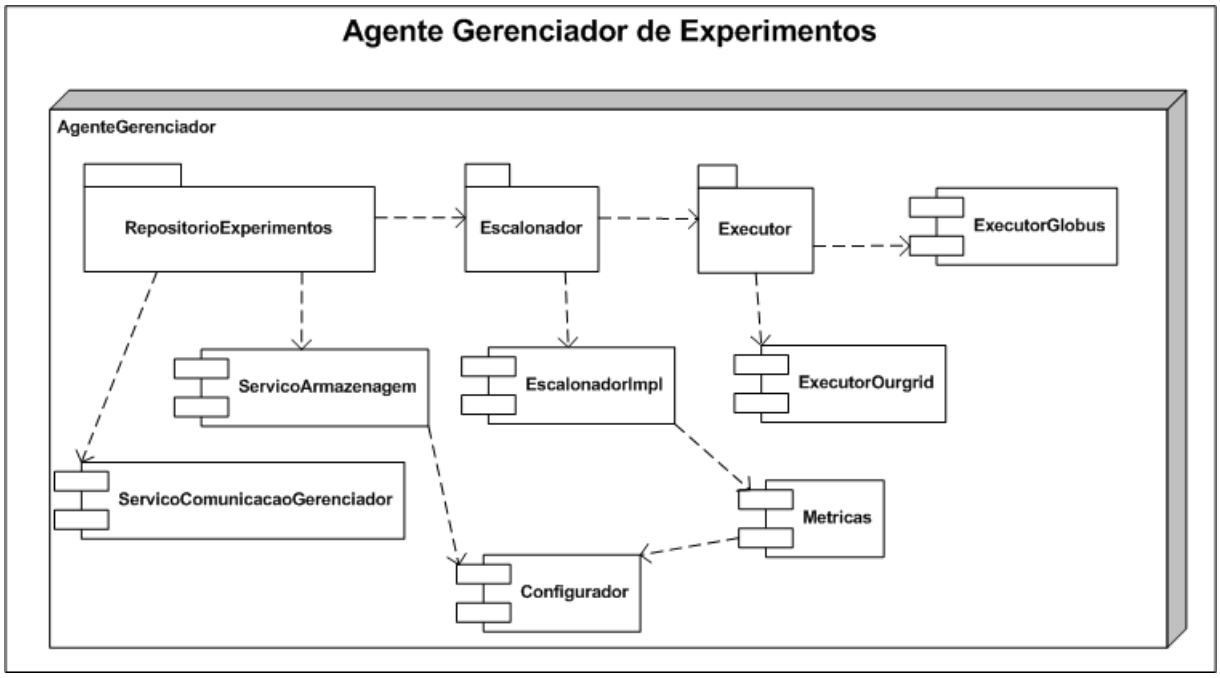

Figura 4.8: Anatomia do Agente Gerenciador de Experimentos

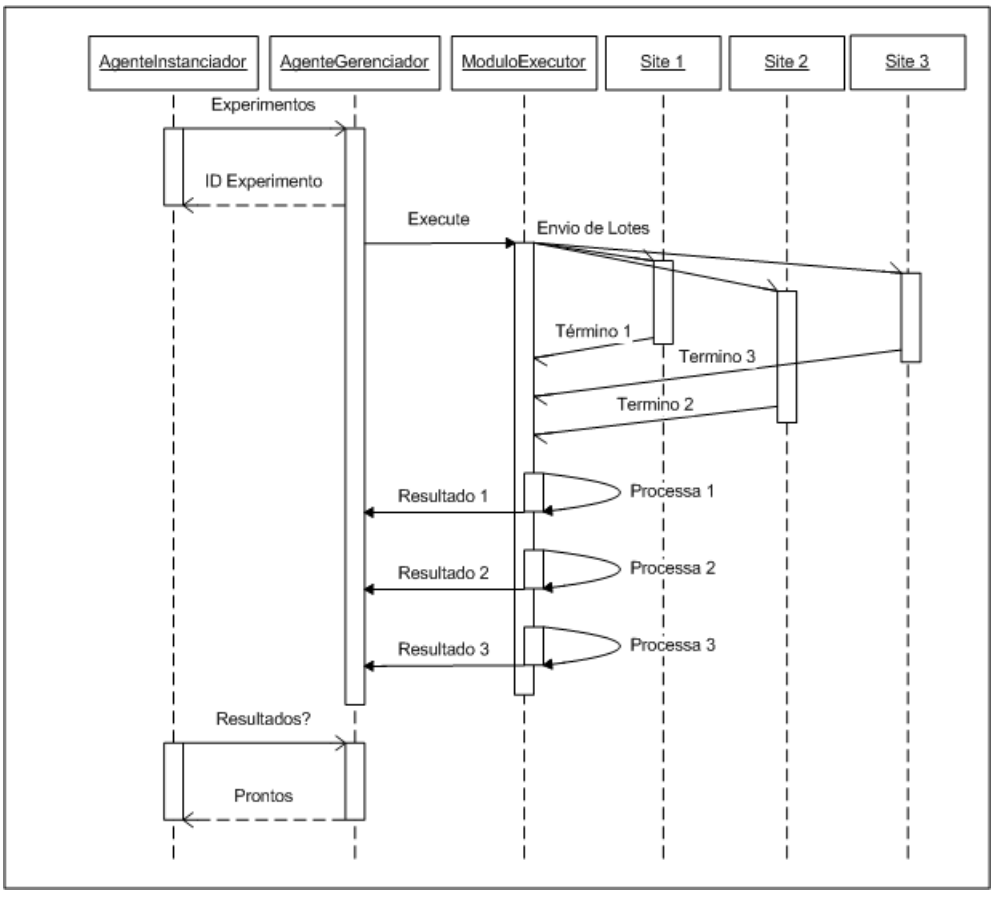

Figura 4.9: GEM: Diagrama de sequência da execução de um projeto de experimentos.

considerar o recurso mapeado como uma VO, a qual possui um ponto de acesso para submissão de tarefas.

Como já mencionado, as configurações dos recursos disponíveis são definidas em um arquivo de inicialização. O agente gerenciador quando iniciado carrega também módulos que fazem a transferência dos arquivos corretos ao recurso e a submissão do conjunto de comando no ambiente de grade para o qual ele, o módulo, foi exclusivamente desenvolvido. Esses módulos não fazem parte da aplicação, sendo eles caracterizados como um projeto a parte cujo desenvolvimento possui uma interface em comum com o agente gerenciador. Assim, um módulo pode ser carregado e suas operações invocadas pelo agente. A figura 4.10 ilustra o modelo de desenvolvimento, onde há um pacote para que implementa a funcionalidade de repasse da aplicação para o ambiente em grade, e os módulos como porções de *software* necessárias para o funcionamento correto da solução.

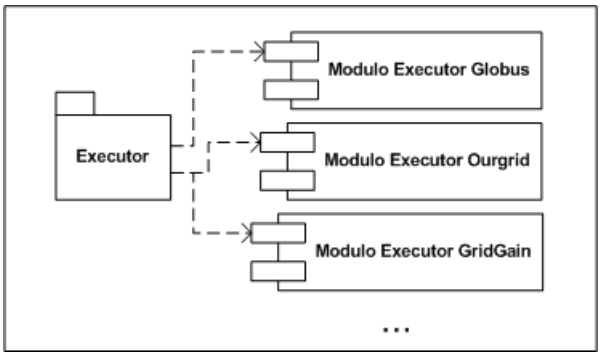

Figura 4.10: Mecanismo de acoplagem de módulos específicos para cada *middleware de computação em grade.*

O conjunto de premissas básicas a serem implementadas pelos módulos são:

- **void readConfiguration(confFile):** esta função deve ler um arquivo no formato XML que deve conter o nome do recurso, o dados para conexão e submissão de processos (endereço IP e porta, usuário, senha, arquivo para autenticação se houver), o sistema operacional, a arquitetura e um campo extra para informações adicionais;
- **boolean start():** esta função ativa o módulo e inicia quando necessário mecanismo extra para recebimento de notificações de término de execução;
- **boolean connect():** esta função deve estabelecer uma conexão com o recurso;
- **boolean authenticate():** esta função faz a autenticação, liberando ao módulo executor a submissão e obtenção de resultados;
- **void disconnect():** esta função fecha uma conexão estabelecida com o recurso;
- **void submit(experimentsBatch):** esta função envia um conjunto de experimentos para a execução;

- **void getResults(experimentBatch):** esta função obtém os arquivos de saída para uma determinada execução.

Portanto, para que o agente gerenciador seja iniciado, devem ser passados os módulos de cada ambiente de computação em grade e os arquivos de configuração para cada recurso presente.

Um ponto de falha para a solução é o agente gerenciador porque ele funciona como um ponto centralizador. Como neste trabalho o foco foi implementar um protótipo e testá-lo em um ambiente controlado, esta solução não foi implementada. Porém, soluções são possíveis para resolver este problema. No nível de rede um servidor DNS pode resolver a questão de o executor estar alocado em um endereço IP, permitindo que várias cópias (*mirrors*) sejam disponibilizadas. Cada requisição é enviada para uma cópia somente, e caso ela falhe, um outro endereço pode ser usado para garantir a disponibilidade do serviço. No nível de aplicação pode-se implementar um algoritmo para réplica das informações e outro para coordenação daquela máquina que responderá como o servidor ativo, ficando responsável pelas transações com o usuário.

4.2.3 Sistema de Comunicação

O sistema de comunicação é um componente presente no agente instanciador e no agente gerenciador. Seu objetivo é proporcionar um mecanismo simples e efeciente para troca de informações entre processos no GEM. Como a proposta da ferramenta contempla a troca de informações entre recursos que estão disponíveis em pontos distribuídos através da Internet, o sistema de comunicação foi projetado para utilizar de protocolos comuns em tal meio, basicamente: o IP e o TCP.

Cada ponto, ou nodo, é identificado por um endereço IP e respectiva Porta. A comunicação ocorre por meio de troca de mensagens, as quais possuem um campo do nodo que originou a mensagem, um para o nodo de destino, outro para identificar o rótulo da mensagem, e um que contém o objeto que a mensagem leva consigo, o *pay load*. Para que um software inicie um nodo que se comunique com outros, foi especificado um conjunto de quatro operações básica que o torna operante, a saber:

- `start()`: a qual inicia um soquete em IP e Porta específicos;
- `stop()`: a qual cessa o servidor;
- `send(Message)`: a qual envia uma mensagem a ser trasmitida para o nodo de destino;
- `Message receive()`: a qual recebe uma mensagem destinada

Uma vez que servidores estejam iniciados, toda comunicação é feita pelas diretivas `send` e `receive`, enviando ou recebendo uma mensagem respectivamente. A existência do campo rótulo na mensagem permite que sejam explorados aspectos de um protocolo de comunicação. Portanto, ao trocar mensagem um nodo especifica um rótulo que é utilizado naquele outro nodo receptor a fim de tratar de forma correta o conteúdo levado pela mensagem. A comunicação feita do agente instanciador com o agente gerenciador é composta por 4 tipos de mensagens:

- **EXECUTION_REQUEST**: a qual requer a execução de um projeto de experimento, causando a criação dos diretórios necessários para o processamento. Recebe como parâmetro de entrada um objeto que descreve o projeto de experimento e como de resposta envia para o cliente o respectivo identificador do projeto de experimento;
- **REEXECUTION_REQUEST**: a qual requer uma nova execução de um projeto experimento. O parâmetro passado é o identificador do projeto de experimentos, retornado pela mensagem EXECUTION_REQUEST e como resposta é enviado o número da execução;
- **RESULTS_REQUEST**: a qual requer o diretório de resultados de uma determinada execução. Caso tenha sido enviado um EXECUTION_REQUEST, deve ser passado o número 1; caso tenha sido enviado um REEXECUTION_REQUEST, deve ser passado o número da execução agendada. O agente gerenciador compacta o diretório e o envia como resposta à mensagem;
- **GET_LIST_OF_RESOURCES**: a qual retorna a lista formada pelos recursos disponíveis para utilização.

Para esta proposta foi implementado um protótipo escrito na linguagem Java, que contempla comunicação entre o cliente e executor via TCP. Os componentes de comunicação utilizaram o padrão de projeto `Abstract Factory` [2], de forma que outras abordagens possam ser adicionadas (por exemplo serviços *web*). O módulo implementado foi para o `Globus` (descrito na seção 4.4.2). E o ambiente para testes e validações constou de três *clusters*, como será discutido no capítulo 5.

4.3 O ASDA MRIP

O simulador MRIP foi desenvolvido por Bruschi et al. (2004) na linguagem de programação C e utilizando a biblioteca PVM. No presente trabalho, ele foi portado para

[2]Este padrão de projeto é capaz de gerar objetos conforme um parâmetro passado. Por exemplo, `Fac.getInstance("TCP")` retornaria um sistema de comunicação que utilizaria o protocolo TCP. Já `Fac.getInstance("WS")` retornaria outro que utilzaria *Web Services*.

MPICH2, e a organização de seu código fonte reestruturada, transformando-o em uma biblioteca estática de funções que pode ser compilada em conjunto com a simulação desenvolvida, permitindo execuções sem dependências extras.

O objetivo do ASDA MRIP é permitir que uma simulação sequencial seja replicada paralelamente em unidades processantes. A execução das réplicas deve ser controlada por um programa que recebe os dados do estado de cada uma delas, enviados periodicamente em um intervalo de tempo pre-determinado. O estado de uma réplica é composto basicamente pela variável de resposta da simulação e sua variância até aquele instante de sua execução. O programa centralizador é denominado Analizador Global, pois ele centraliza a variável de resposta de todas as réplicas e toma a descisão de continuar ou parar suas execuções em função dos valores recebidos.

O ASDA MRIP é um programa paralelo do tipo mestre/escravo, tendo o Analisador Global o papel de mestre e as simulações replicadas o papel de escravos. Para iniciar a execução do Analizador Global, ou mestre, devem ser passados como o parâmetros: o programa da simulação sequencial adaptado e seus parâmetros específicos, o número de cópias a serem iniciadas e o valor para a primeira semente (*seed*) usada no gerador de números pseudo-aleatórios.

Uma vez iniciada a execução, o mestre dispara os programas escravos e aguarda seus retornos. A cada retorno, adiciona em uma lista uma estrutura que armazena o nome da máquina hospedeira, o identificador do sistema de passagem de mensagem, uma estrutura para armazenar o estado do escravo e outras variáveis de controle que visam informar se a réplica está esperando por uma resposta, se ela está viva e se ela já solicitou uma confirmação para execução.

Uma réplica, por sua vez, começa sua execução enviando os dados para o mestre, e, em seguida, esperando por uma semente. Ao recebê-la, configura o sistema analisador local, disponibilizado pela biblioteca MRIP, e inicia a simulação.

Simulações de eventos discretos tendem a ser iterativas. Desta forma, a cada iteração é feita a avaliação da variável de resposta. Em um número de iterações proporcional ao número de réplicas é feita uma cópia do estado atual, a qual é enviada para o programa mestre. Em retorno desta cópia, uma mensagem é passada, indicando continuação ou término da execução.

O Analisador Global recebe os estados de cada uma das réplicas e processa, em função da precisão da variável de resposta produzida por todos os escravos, a necessidade de continuar ou parar a simulação.

A estratégia para adaptar uma simulação sequencial ao simulador MRIP é instrumentar seu código fonte em dois pontos: a) no ínicio de sua execução, fazendo um registro no

programa mestre; b) no meio da simulação, geralmente no ponto final de uma iteração, enviando uma cópia do estado atual para o mestre. Devido a esta instrumentação, a biblioteca MRIP passar a ser requerida pelo projeto da simulação sequencial, o qual, após sua compilação, se tansformará em um novo programa, adaptado para execuções controladas pelo analizador global.

A fig. 4.11 mostra o ambiente ideal para a execução de simulações MRIP. Cada experimento deste trabalho será uma execução semelhante a esta, onde cada escravo processa uma simulação sequencial, possuindo uma semente diferente das demais. Cada réplica comunica somente o Analisador Global, sendo inexistente comunicações réplica-réplica.

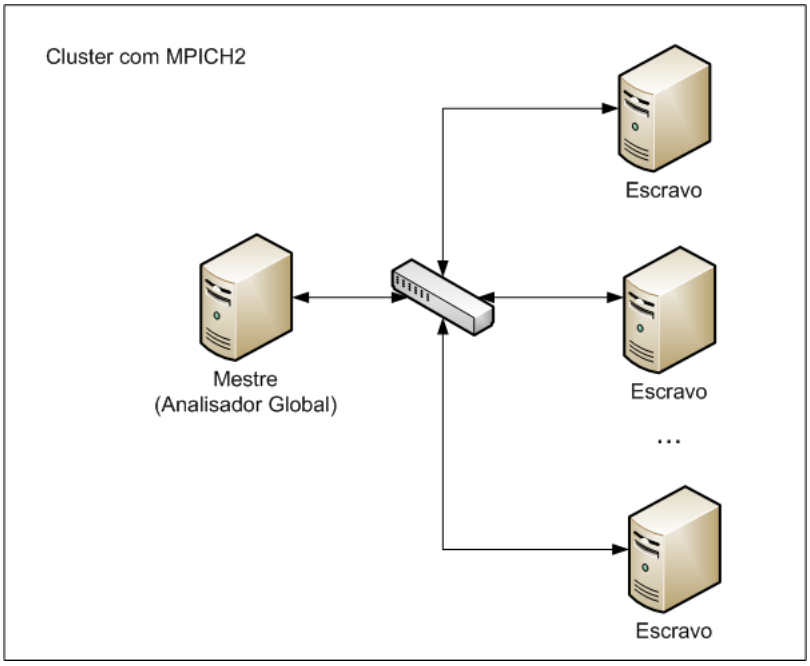

Figura 4.11: Topologia de uma simulação MRIP.

4.4 Implementação dos módulos do Executor

4.4.1 Estudo de caso: Ourgrid

O *middleware* para computação em grade `OurGrid` foi objeto de estudo na fase inicial deste trabalho. Esperava-se executar com sucesso o simulador MRIP nesse ambiente, mas os estudos mostraram que o modelo utilizado na concepção do projeto, *bag-of-taks*, não contempla com perfeição os requisitos para a execução do simulador. Nos próximos parágrafos são descritos detalhes técnicos do estudo e, por fim, apresentadas as limitações encontradas.

Os primeiros estudos foram feitos com a versão 4.0-alpha em Setembro de 2008, apresentando problemas desde o nível de aplicação (*bugs*), até mesmo no nível de infraestrutura, onde foram identificados falhas na execução do sistema de virtualização `vserver`. Naquele momento, tal projeto estava em migração de sua versão anterior, 3.3, para a versão em desenvolvimento, alpha – o que pode justificar a utilização insatisfatória. Há, também, no próprio site do desenvolvedor (OURGRID, 2007), um ambiente de grade aberto a colaboradores, mas devido a questão de transição de versões, este ambiente não estava disponível.

Como alternativa, o ambiente de testes utilizado, em Outubro de 2008, foi o `ShareGrid` (SHAREGRID, 2007), projeto italiano aberto à comunidade mundial com fins de compartilhamento de recursos computacionais, estruturado sobre o `Ougrid` versão 3.3.1.

Para a participação na grade foi disponibilizada uma máquina com endereço IP público, uma referência a ela por um DNS e conectividade em portas TCP de números altos, inerente da utilização do RMI (*Remote Method Invocation*) como mecanismo de chamadas remotas. Esta máquina hospedava tanto o componente `Peer` quanto o `Worker`.

A partir desta máquina foram disparados testes preliminares, verificando a execução destes nas máquinas que compunham a grade. Naquele instante de tempo, haviam recursos que executavam o Microsoft Windows, o Sun Solaris e o Linux, sendo possível especificar um executável especial para cada um dos sistemas operacionais em questão, descritos no arquivo de definição de uma submissão. Tais testes eram executáveis do padrão `ELF`, utilizados pelo Linux, que processavam um cálculo e sua saída era direcionada para um arquivo. Todas submissões executaram a contento, sendo alocadas no sistema operacional correto.

O teste para a verificação de compatibilidade com o ASDA MRIP com o ambiente foi a execução de um programa que utilizava a padrão MPI, implementado pela biblioteca MPICH2. Durante sua execução, o programa escrevia na saída padrão o identificador do ambiente de passagem de mensagem (seu *rank*). Este teste se mostrou ineficiente porque somente a máquina local foi capaz de executá-lo. Em outros recursos que compunham a grade não havia a biblioteca MPICH2 instalada.

Outro problema que poderia sugir é que, mesmo se houvesse a biblioteca requerida pelo executável instalada em todos ou alguns dos *workers* de um determinado *peer*, não existe uma garantia de comunicação entre os *workers*. Deste modo, é provável que uma aplicação, ao chegar em um sítio que supra todos os requisitos operacionais, execute de forma concorrente e local os processos disparados.

O `OurGrid` foi desenvolvido para execução de aplicações `Bag-of-Taks`, aplicações não trocam mensagens com outras cópias ou programas, nem necessitam de um programa que coorderne a execução. A execução do simulador MRIP possui um processo que controla

a execução e as simulações comunicam com que este processo. Por essas razões este ambiente de grade não foi compatível com o escopo do presente trabalho.

4.4.2 Estudo de caso: Globus

Com a experiência da utilização do `Ourgrid` foram identificados potênciais problemas inerentes do uso de computação em grade, principalmente aqueles de compatibilidade dos arquivos executáveis e bibliotecas requeridas com os recursos de destino. Quando comparado ao `Ourgrid`, o `Globus` não é diferente nesse aspecto. No entanto, apresenta mais recursos e um escopo maior, permitindo a integração direta do MPICH2 com o *middleware*, além de uma quantidade maior de protocolos que facilitam a tarefa de compartilhar um recurso, bem como a comunicação entre eles. No presente trabalho, utilizou-se um ambiente controlado em laboratório. Apesar de o escopo do `Globus` ser maior e mais flexível, problemas como aqueles encontrado no `Ourgrid` podem acontecer, mas existe uma probabilidade maior deles serem contornados, não inviabilizando a solução.

O objetivo deste módulo é fazer a autenticação no ambiente de grade, enviar os arquivos necessários para a execução, submeter um conjunto de tarefas e enviar um sinal de término para o agente gerenciador. Para alcançar tal objetivo várias abordagens podem ser realizadas. Dentre elas, optou-se por uma mais simples. Partiu-se do princípio que o acesso ao *front-end* ou ponto de acesso ao recurso, ou Organização Virtual, é feito via `SSH`. Assim, um usuário que fosse utilizar desse recurso, efetuaria *login* na máquina de acesso ao ambiente através desse protocolo, se autenticaria e executaria sua tarefa.

O `Globus` foi estruturado como ilustrado na figura 3.4. A submissão é feita em uma máquina denominada "*front-end*". Como será discutido no próximo capítulo, o ambiente de testes constou de 3 *clusters* configurados igualmente, cada um possuindo uma máquina com conectividade com o agente gerenciador e somente nela está instalado o `Globus Tookit`. Foi configurado um usuário com as credenciais necessárias para a execução de processos localmente. Para a execução de um processo faz-se necessária a autenticação, que foi feita pelo seguinte comando:

```
myproxy-logon -s frontend
```

Uma vez autenticado, o ASDA MRIP é disparado pelo comando:

```
globusrun-ws -submit -c /mpi/mpiexec /user/mrip-master \
    "/user/simulation 2 0 1 2" 7 2048
```

Esta execução é diferente daquela feita diretamente pelo `mpiexec` porque ela é tratada pelo *middleware* em questão, o qual faz uma série de verificações e inicia o processo

conforme o mecanismo configurado. A figura 4.12 ilustra, citando o uso de dois escalonadores (LSF e LoadLeveler) e um utilizando a premissa `fork` do sistema operacional, como é fluxo de chamadas entres os componentes que efetivamente executam uma tarefa no ambiente. Muitas vezes, opta-se por não instalar bibliotecas nos nodos que processam ("*workers*"), possibilitando que o *front-end* fique livre das cargas impostas pelos processos executantes no *cluster* compartilhado com a grade. Desta forma, o agente gerenciador envia um arquivo contendo um conjunto de comandos que disparam sequencialmente as simulações.

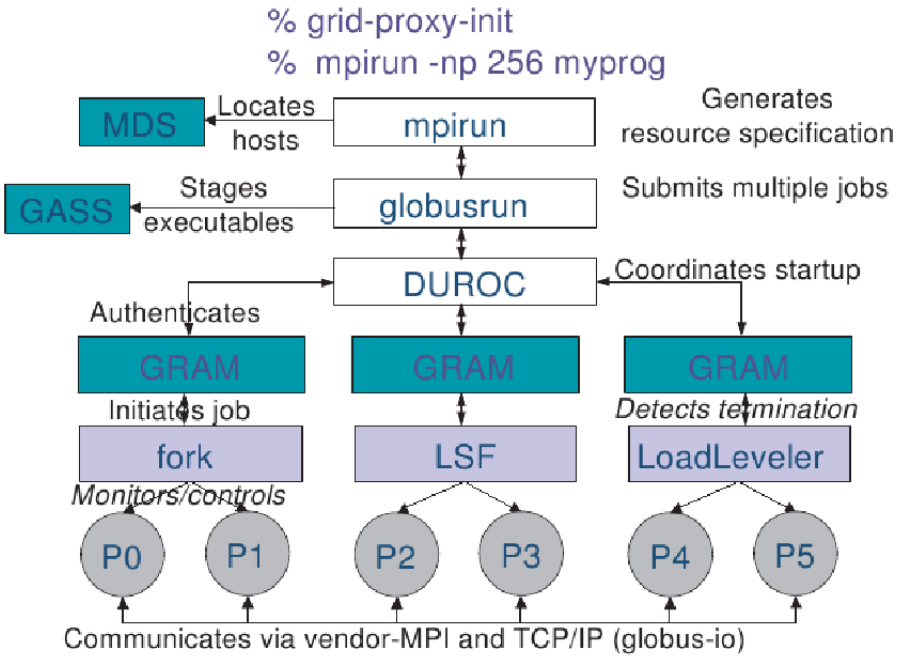

Figura 4.12: Execução de um processo no Globus.

Uma parte do processo de execução do ASDA MRIP em grade é o sinal de término da execução, a qual é implementada da seguinte forma: um servidor TCP é iniciado na função `start` do módulo `GEM-Globus` do agente gerenciador, o qual recebe um objeto contento um texto com o nome do recurso e um *byte* com valor 1, significando o término da execução. De posse desse objeto, o agente gerenciador identifica o recurso e retira da lista de execuções o conjunto de experimentos.

Em cada *cluster* há um diretório compartilhado ("/user") visível a todos nodos. Assim, os arquivos necessários à execução são transferidos para este local. Ao final da execução, os arquivos de resposta são recuperados da mesma forma. O protocolo utilizado para transferência de arquivos é o *GridFTP*, distribuído conjuntamente com o `Globus Toolkit`, e operados pelo agente gerenciador.

4.5 Trabalhos Relacionados

Os trabalhos apresentados nesta seção são sobre propostas para o gerenciamento de experimentos em ambiente de computação em grade.

4.5.1 JAMES II

Em Leye et al. (2008) o autor descreve uma abordagem para execução de simulações estocásticas em ambiente de computação em grade. O conceito principal é a criação de um servidor de simulação, o qual é instanciado no recurso capaz de processar e outros serviços que servem como estágios para a divisão e gerenciamento dos experimentos a serem executados. Assim, um experimento para ser executado para por um serviço denominado `ParallelSimulationRunner`, o qual cria um novo processo capaz de controlar um conjunto de experimentos. Por sua vez, este processo encaminha um experimento para outro serviço que executa uma aplicações com característica mestre/escravo. O componente mestre envia o experimento para ser processado pelo servidor de simulação, o qual processará a requisição e devolverá para o servidor mestre. Ao final de todos os experimentos os dados resultantes são retornados ao processo. Toda a execução é controlada por um outro componente que retorna informações sobre o estado da execução para o processo requerente. É uma abordagem interessante e permite um controle preciso sobre cada passo para a execução de uma simulação em um ambiente de grade, no entanto não faz parte do trabalho um repositório de simulações, nem menção sobre um controle dos recursos disponíveis para a execução.

4.5.2 The Java CoG Kit Experiment Manager

Em Laszewski et al. (2006) é descrito um módulo do CoG Kit (COG-KIT, 2004) especializado no gerenciamento de experimentos executados em uma grade construída sobre o Globus. Possui mecanismos de *checkpoint* automáticos que prevê o salvamento do estado de uma simulação para uma possível reexecução a partir daquele ponto de parada; gerenciamento transparente das saídas padrão e de erro, salvando as no diretório de um experimento; controle de versão, permitindo ao usuário que possa duplica um experimento; metadados sobre os experimentos, tornando fácil o gerenciamento por humanos das requisições feitas. Possui um componente cliente que submete, recebe e armazena os resultados de experimentos processados. E uma parte servidora que se comunica com a grade solicitando execuções. Não são mencionada integrações com o *middleware* de computação em grade a fim de saber quais recursos estão disponíveis, nem o estado de cada um deles. Outro ponto não referenciado é sobre como associar um tipo de executável com

um tipo de arquitetura, o que aumentaria as possibilidade de nodos capazes de processar as execuções.

4.5.3 Comparação entre trabalhos relacionados e o GEM

A tabela 4.1 mostra de forma sintética alguns pontos comparáveis entre as propostas estudadas. A coluna "Específico para Simulação" refere-se a exclusividade de utilização da ferramenta para execução de programas de simulação, não possibilitando a execução de outros programas; muito embora, o GEM tenha sido estudado no escopo de Simulações Distribuídas neste presente trabalho, outros tipos de programas podem ser utilizados, aumentando as possibilidades de sua aplicação. "Uso geral" refere-se a possibilidade de execução de vários tipos de aplicação. A coluna "Gerencia Experimentos" refere a capacidade de gerenciar os experimentos dos usuários de forma transparente. E, "Multi-Recursos" refere-se a possibilidade de utilização de vários sítios com diferentes *middlewares* de Computação em Grade.

Tabela 4.1: Comparativos entre trabalhos relacionados e o GEM

Trabalho	Específico para Simulação	Uso Geral	Gerencia Experimentos	Multi-Recursos
JAMES II	Sim	Não	Não	Não
CoG Kit	Não	Sim	Sim	Não
GEM	Não	Sim	Sim	Sim

4.6 Considerações Finais

O desenvolvimento da ferramenta GEM proporcionou a aplicação dos conceitos vistos na seção 2.2, possibilitando que a geração dos experimentos e sua execução seja feita de forma automatizada. Outro ponto é o contato com os ambientes de computação em grade, permitindo o estudo de uma grade real implementada pelo `Ourgrid` e a utilização do `Globus`. Houve também a experiência de migração e padronização do código do ASDA MRIP, possibilitando sua efetiva utilização modular. A implementação do GEM corresponde a um protótipo necessitando de futuros incrementos, como uma integralização maior com o `GRAM5` para submissão de aplicações na grade, bem como a monitoração de recursos e o estado de aplicações.

O capítulo seguinte apresenta uma avaliação de desempenho usando uma simulação sequencial e o ASDA MRIP em paralelo, sendo executado em *cluster* e em um ambiente de grade sob coordenação do GEM, quanto ao disparo das aplicações.

Capítulo

5

Experimentos e Resultados

5.1 Considerações Iniciais

Este capítulo descreve uma avaliação do desempenho que a ferramenta GEM oferece ao usuário. Para tal, foi calculado o tempo de uma execução sequencial, o tempo de uma execução ASDA MRIP em um *cluster* e o tempo de uma execução em ambiente de grade. São apresentados a abordagem para avaliação de desempenho e o *benchmark* criado para este fim. O objetivo é apurar o quanto de ganho foi possível obter utilizando as ferramentas de execução em paralelo.

5.2 Abordagem para a avaliação de desempenho

A avaliação de desempenho apresentada neste capítulo visa comparar o tempo de resposta – o tempo de execução – de uma simulação sequencial e da mesma alterada para execução distribuída em *cluster* e em ambiente de grade.

Para que o tempo apurado seja corretamente medido, e que a mesma carga imposta nas modalidades distribuída e sequencial seja aplicada, foi proposto um *benchmark*. Com ele é possível repetir uma determinada carga de trabalho em ambientes diferentes, des-

considerando o sistema operacional, bibliotecas de funções específicas, etc.

Outro ponto abordado foi o tamanho da simulação, ou seja, o tempo necessário para que ela fosse executada. Para que isto seja possível, cálculos numéricos foram inseridos em um ponto específico do programa de simulação, de forma a estipular tamanhos diferentes para suas execuções.

Uma vez apurados os resultados das modalidades distribuída e sequencial, foi feita uma comparação destes resultados, levando-se em consideração a quantidade de unidade processadoras e os tempos de resposta. Isto permitiu uma avaliação do aumento de desempenho alcançado, bem como a eficiência da utilização das unidades disponíveis para processamento.

5.2.1 O Benchmark

O *benchmark* apresentado é uma simulação inspirada no modelo proposto por Patterson et al. (1988) que descreve o RAID (*Redundant Array of Inexpansive Disks*). O modelo implementado pode ser visualizado na figura 5.1. Cada entrada do sistema corresponde a um arquivo a ser processado pelo RAID. O primeiro passo é a chegada em um centro de serviço denominado controlador, o qual decide como o arquivo será processado e o encaminha para um dos discos disponíveis, representado pelo conjunto de centros de serviços do nível seguinte. Dependendo do nível a ser utilizado, o comportamento do tratamento é diferente, podendo haver sincronismo de discos, replicação de dados, controles de paridade, etc. O tamanho do arquivo, o tipo de operação a ser realizada (leitura ou escrita) e a quantidade de discos também influenciam o desempenho oferecido pelo modelo. Desempenho, no contexto deste programa corresponde a quantidade de arquivos processados por unidade de tempo. Assim, a variável de resposta observada é a quantidade de arquivos em um determinado tempo. Sua codificação foi feita em C++, usando-se a biblioteca SIMPACK (FISHWICK, 1992).

A biblioteca SIMPACK possui objetos e funções que facilitam a implementação de simulações de eventos discretos. Ela possui um mecanismo de tempo lógico e lista de eventos futuros. Tal mecanismo permite processar eventos em paralelos, sobre um determinado ponto no tempo lógico.

Um evento é representado pelo objeto TOKEN, o qual possui um conjunto básico de informações e procedimentos, e um ponteiro para um objeto de qualquer tipo, permitindo que um objeto TOKEN t entre no sistema, e a ele seja acoplada uma estrutura com informações extras. No *benchmark* cada evento trás consigo informações sobre o tipo de operação (leitura ou escrita), o tamanho do arquivo e a unidade na qual será atendido.

Outro ponto sobre a SIMPACK, é que ela possui um módulo para geração de nú-

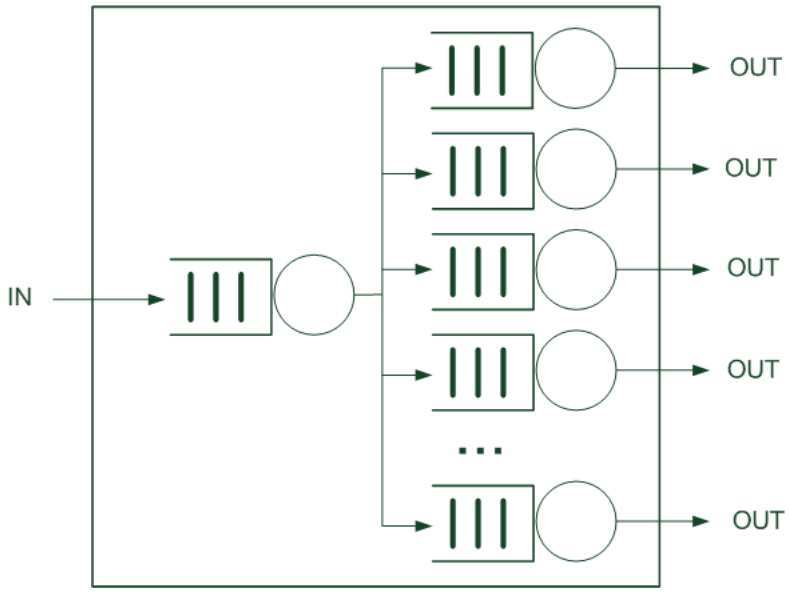

Figura 5.1: Modelo de Rede de Filas para o *Benchmark*

meros pseudo-aleatórios, garantindo independência de plataforma, quando da geração de números pseudo-aleatórios. O problema em questão é que em um ambiente altamente distribuído e com vários níveis de administração, não há uma garantia de que no sistema receptor haverá a mesma biblioteca padrão C para executar a função pseudo-aleatória.

Por exemplo, há implementações feitas pela Microsoft, pelo projeto GNU, pelo FreeBSD, pelo NetBSD, pelo openSolaris, etc. Há somente a especificação de uma interface para a função `random`, mas não sobre sua implementação, ficando livre seu desenvolvimento. Ao usar uma função do sistema operacional, sequências diferentes podem ser geradas. Estas sequências impactam diretamente no programa simulado, porque o resultado delas é usado para definir quando um evento chega ao sistema, qual o tamanho do arquivo, qual operação, entre outros fatores. Portanto, usar o gerador de números pseudo-aleatórios da biblioteca SIMPACK elimina tal problema, uma vez que seu comportamento (a sequência de números gerados) será o mesmo independente da plataforma de execução, desde que seja feita sua compilação prévia para execução.

Como ilustrado na figura 5.1, o modelo implementado possui vários centros de serviços, cada um com seu próprio gerador de números pseudo-aleatórios. Isto proporciona uma grande variabilidade nos tempos de resposta, permitindo que muitos eventos cheguem ao sistema.

Como mostrado na figura 5.2, a implementação é feita através de um *loop* que controla por quanto tempo (tempo lógico) a execução acontecerá. Assim que um evento é processado, já é adicionado um próximo, especificando um tempo aleatório para sua chegada. Basicamente, este mecanismo pode ser esboçado como:

Na implementação o valor $TIME\_LIMIT = 100.000$ permite que cerca de 52.000

```
while (Future::SimTime() < TIME_LIMIT) {
    Estatus es = Future::NextEvent();
    switch (es.event_id) {
    case BEGIN_TOUR: begin_tour();
        break;
    case REQUEST_CONTROLLER: request_controller();
        break;
    case FORWARD_UNIT: forward_to_unit();
        break;
    case REQUEST_DISK: request_disk();
        break;
    case RELEASE_DISK: done_end_tour();
        break;
    default: cout << "Unrecognised event!\n";
    }
}
```

Figura 5.2: *Loop* principal de uma simulação

(cinquenta e dois mil) eventos sejam processados, em outras palavras,cinquenta e duas mil iterações são feitas no referido laço de repetição. Esse valor, 100.000, foi definido com base nos trabalhos desenvolvidos no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC). Sendo passados parâmetros para o tempo médio de chegada de requisição de escrita ou leitura de arquivos, tempo que a controladora de RAID leva para receber e escolher uma unidade para processar, tempo de escrita e tempo de leitura. No código da simulação, essas variáveis se apresentam como constantes, sendo seus valores estipulados como mostrado na figura 5.3

```
static const int TIME_LIMIT          = 100000;
static const double MEAN_ARRIVAL_TIME = 1.875;
static const double FORWARD_OVERHEAD  = 0.625;
static const double WRITING_SPEED     = 2.5;
static const double READING_SPEED     = 0.75;
```

Figura 5.3: Declaração de constantes

Como mencionado anteriormente, buscou-se inserir um cálculo que permitisse tamanhos diferentes para uma mesma simulação. A solução proposta foi adicionar uma chamada a outra função quando o evento *REQUEST_DISK* é processado. Esta outra função foi descrita como mostrado na figura 5.4.

Também como comentado anteriormente, a função `random` adiciona uma carga extra

```
1  void big_o() {
2  // set the length of the simulation. Uncomment one of them.
3  #define THIN
4  //#define SMALL
5
6  #ifdef THIN
7      return;
8  #endif
9
10     stream(15);
11     int j = 0;
12
13 #ifdef SMALL
14     for (int i = 0; i < (5 * 100); i++) {
15         for (int k = 0; k < (5 * 50); k++) {
16 #endif
17             j += i * random(1, 65535) * k;
18         }
19     }
20 }
```

Figura 5.4: Mecanismo para controle do tempo de controle de uma simulação

de processamento, envolvendo operações aritméticas – *CPU bound*.

5.2.2 Experimentos

Os experimentos se dividem em 2: os experimentos do usuário e aqueles usados para a validação da ferramenta GEM. Estes primeiros correspondem ao conjunto de experimentos que um usuário precisa processar, ao passo que o segundo são execuções sequenciais e paralelas usadas para medir o desempenho da ferramenta. Nas próximas seções são comentados sobre os dois experimentos.

Experimentos do usuário

Os experimentos propostos para o *benchmark* contemplam os seguintes fatores e níveis:

- Fator: nível do RAID

  Níveis: de 0, 1 e 5

- Fator: número de discos

  Níveis: 4, 6 ou 8 unidades

- Fator: tamanho dos arquivos

Níveis: 25% pequenos e 75% grandes; 50% pequenos e 50% grandes; 75% pequenos e 25% grandes

- Fator: tipo da operação

  Níveis: 25% escrita e 75% leitura; 50% escrita e 50% leitura; 75% escrita e 25% leitura

Os termos pequenos e grandes foram usados para uma escala de 1 a 200. A determinação de qual valor será atribuído para o tamanho de um arquivo durante a simulação possui dois passos: 1) gera-se um número aleatório que resulta em um valor que indica se o arquivo será grande ou pequeno; 2) sabendo o tamanho do arquivo, gera-se outro número aleatório na faixa específica de cada classificação, sendo de 1 a 100 para pequenos e de 101 a 200 para grandes. Se o número gerado para o tamanho do arquivo estiver nos valores da primeira metade classificada, este será classificado como pequeno, e, caso contrário, classificado como grande. Assim, estes termos são relativizados ao escopo deste trabalho e, para fins de praticidade, eles serão referenciados desta forma.

Os experimentos, quando planejados de forma fatorial, somam:

$$n = 3(RAID) \times 3(discos) \times 3(arquivos) \times 3(leitura/escrita) = 81$$

A tabela 5.1 apresenta todos os experimentos do usuário. Os números representam a variação dos níveis de cada fator mapeados em números sequenciais, com o intuito de facilitar a representação. Esta tabela possui 4 linhas e 3 colunas. Cada célula é composta por outras 7 linhas e 4 colunas. Cada coluna de uma célula representa respectivamente os fatores nível RAID, número de discos, tamanho dos arquivos e tipo da operação. Com exceção do nível, os outros fatores recebem valores que variam entre 0, 1 e 2, representando cada uma de suas possíveis combinações.

Experimentos para validar a ferramenta GEM

Nesta seção é feita menção ao projeto de experimentos para execução a fim de se obter o tempo de respostas de várias execuções. Isto se faz necessário para efeitos comparativos: tempo de execução para simulações sequenciais × tempo de execução para simulações distribuídas MRIP.

O projeto de experimentos é o seguinte:

- Tipo da simulação
  - Sequencial

  - Em um Cluster
  - Grade

- Tamanho da simulação
  - Inalterado
  - Pequeno (ou com carga extra)

Analogamente à seção 5.2.1, o termo *pequeno* é usados por uma questão de praticidade e é válido para o escopo deste projeto. Este termo foi assim definido quando em comparação com aqueles de tamanho denominado inalterado. Ele corresponde a um executável compilado conforme a quantidade de cálculos feitos a cada iteração do laço de repetição do *benchmark*. Para referência sobre os cálculos feitos em cada dos níveis, vide código exposto na seção 5.2.1.

Até que uma execução estável e confiável para a aferição dos tempos fosse obtida, muitos testes foram feitos, revelando também muitos problemas. A raiz de todos os problemas residiu na incompatibilidade de bibliotecas e arquitetura das máquinas. Seja naquela de desenvolvimento, executando Ubuntu Linux 32 bits, seja naquelas executoras, ora executando no Ubuntu Linux 8.04 64 bits, ora Ubuntu Linux 9.10 64 bits.

Um código compilado na máquina de desenvolvimento pode executar naquela executora, desde que haja bibliotecas que ofereçam suporte ao programa. Mas nem sempre essa

Tabela 5.1: Tabela dos experimentos e respectiva utilização

0 0 0 0	1 0 0 0	5 0 0 0
0 0 0 1	1 0 0 1	5 0 0 1
0 0 0 2	1 0 0 2	5 0 0 2
0 0 1 0	1 0 1 0	5 0 1 0
0 0 1 1	1 0 1 1	5 0 1 1
0 0 1 2	1 0 1 2	5 0 1 2
0 0 2 0	1 0 2 0	5 0 2 0
0 0 2 1	1 0 2 1	5 0 2 1
0 0 2 2	1 0 2 2	5 0 2 2
0 1 0 0	1 1 0 0	5 1 0 0
0 1 0 1	1 1 0 1	5 1 0 1
0 1 0 2	1 1 0 2	5 1 0 2
0 1 1 0	1 1 1 0	5 1 1 0
0 1 1 1	1 1 1 1	5 1 1 1
0 1 1 2	1 1 1 2	5 1 1 2
0 1 2 0	1 1 2 0	5 1 2 0
0 1 2 1	1 1 2 1	5 1 2 1
0 1 2 2	1 1 2 2	5 1 2 2
0 2 0 0	1 2 0 0	5 2 0 0
0 2 0 1	1 2 0 1	5 2 0 1
0 2 0 2	1 2 0 2	5 2 0 2
0 2 1 0	1 2 1 0	5 2 1 0
0 2 1 1	1 2 1 1	5 2 1 1
0 2 1 2	1 2 1 2	5 2 1 2
0 2 2 0	1 2 2 0	5 2 2 0
0 2 2 1	1 2 2 1	5 2 2 1
0 2 2 2	1 2 2 2	5 2 2 2

é uma garantia. Mesmo quando em arquiteturas iguais (64 bits) houve falhas nas execuções. Essas ocorreram devido a incompatibilidades entre a biblioteca `libc++` disponível nas versões 8.04 e 9.10 do Ubuntu Linux.

Os ambientes de desenvolvimento e os de execução compartilham muitas características, das quais pode-se afirmar que essas arquiteturas são relativamente homogêneas. Quando comparado a um ambiente de computação em grade maior, este detalhe pode ser um fator de complicação, pois é perfeitamente admissível encontrar uma grade que possua recursos dos mais variados tipos, desde arquiteturas IBM PC 32 e 64 bits, passando por processadores Sun Sparc e Ultra-Sparc, podendo ser encontrados processadores IBM Cell, até mesmo computadores Cray. Além do fator arquitetura, uma execução pode esbarrar em diferentes tipos de sistemas operacionais, incluindo incompatibilidade entre suas versões. Mais ainda, incompatibilidade de diferentes versões de bibliotecas, que podem estar instaladas em uma mesma arquitetura, um mesmo sistema operacional, em uma mesma versão.

A solução encontrada foi de levar o código fonte do *benchmark* para o recurso desejado e efetuar a compilação, produzindo o programa executável. Assim, de posse deste, na submissão do experimento, pode ser feito uma regra de forma que para um determinado nodo seja associado um determinado executável. Assim, ao invocar a execução o agente gerenciador de experimentos verifica qual recurso vai utilizar e envia o programa apropriado.

Isto resolve o problema da execução para o ambiente deste trabalho. Ambientes de grade tendem, além das discrepancias de arquitetura e sistemas, a serem dinâmicos. Um recurso pode ficar disponível por um tempo indeterminado, podendo durar semanas, dias, e até meses. Logo, um conjunto de executáveis compilado em um determinado instante pode não funcionar em outro, decorrido algum tempo. Ademais, estes mesmos ambientes podem ter dezenas de diferentes arquiteturas, e será demasiado dispendioso o desenvolvedor lidar com a disponibilização de executáveis que sempre sejam compatível com cada recurso disponível no ambiente de grade em questão.

Desta forma, com base na experiência adquirida na execução deste trabalho, pode-se afirmar que é recomendado a utilização de linguagem(s) de programação mais portável(is), como é o caso daquelas que executam em plataformas de máquinas virtuais. A exemplo pode-se citar: Java, Python e .Net.

A linguagem Java pode ser é utilizada para tal portabilidade, como é o caso do Globus Toolkit ter a adotado para construção de parte das ferramentas disponíveis. Outra linguagem que garante portabilidade é a Python, embora não tenha um bom suporte para aplicações com multiplas linhas de execução (*multi-thread*). No caso .Net, ainda existe pouco, mas pode ser que exista algo no futuro, uma vez que muitas máquinas executam

o sistema operacional Microsoft Windows. O .Net em si não é um tecnologia produzida pela Microsoft, e sim uma especificação, como o caso Java. Assim, pode haver várias implementações para .Net e, teoricamente, os códigos seriam compatíveis entre si. Como exemplo de implementações pode ser citado a própria versão da Microsoft e o projeto Mono (MONO, 2004).

Essas linguagens possuem um desempenho pior quando comparadas àquelas frequentemente usadas em *clusters* como Fortran, C e C++. Portanto, sua adoção pode implicar em um menor desempenho nas execuções quando observadas separadamente, sem levar em consideração o ambiente de grade. Deste modo, pode se abrir mão de parte do desempenho obtido por linguagens de alto desempenho em função de outras com menor ganho, mas esta perda pode ser balanceada por uma quantidade maior de pontos processantes, o que, no final, pode garantir produção maior de resultados.

5.2.3 Ambiente de testes

O ambiente de testes foi composto por 3 *clusters* distintos, com redes de comunicação diferentes e independentes. As máquinas possuem as configurações conforme mostrado na tabela 5.2.

Tabela 5.2: *Clusters* disponíveis

Cluster	Nodos	CPU	Memória	HD	Sist. Operacional	Globus
Providers	3	Intel Core 2 DUO	7 GB	230 GB	Ubuntu Linux	4.0.8
Birds	5	Intel Core 2 QUAD	3 GB	160 GB	Ubuntu Linux	4.0.8
LaSDPC	7	Intel Core 2 DUO	6 GB	450 GB	Ubuntu Linux	4.0.8

Topologicamente, eles formam uma rede como na figura 5.5. Quando agrupados em forma de grade, cada *cluster* corresponde a um nodo. O agente gerenciador de experimentos, como precisa alcançar todos os nodos da grade, bem como ser alcançado pelo agente instanciador, foi instalado na máquina `frontend` do *cluster* LaSDPC. Trata-se de um ambiente estruturado em laboratório e totalmente controlado, diferentemente daqueles reais, onde possuem um comportamento dinâmico.

Portanto, os experimentos foram executados neste ambiente, onde cada planejamento de experimentos foi executado 10 vezes, sendo calculada uma média aritmética de seus tempos de resposta e também o intervalo de confiança com nível de confiança de 95%.

Os experimentos sequenciais foram executados no *cluster* LaSDPC. Para cada experimento considerou-se 10 sementes diferentes para garantir resultados estatisticamente confiáveis. Como o total de experimentos somam um valor de 81, o conjunto de execuções sequenciais foi de 810. Cada conjunto, então, foi executado, 10 vezes a fim de se obter

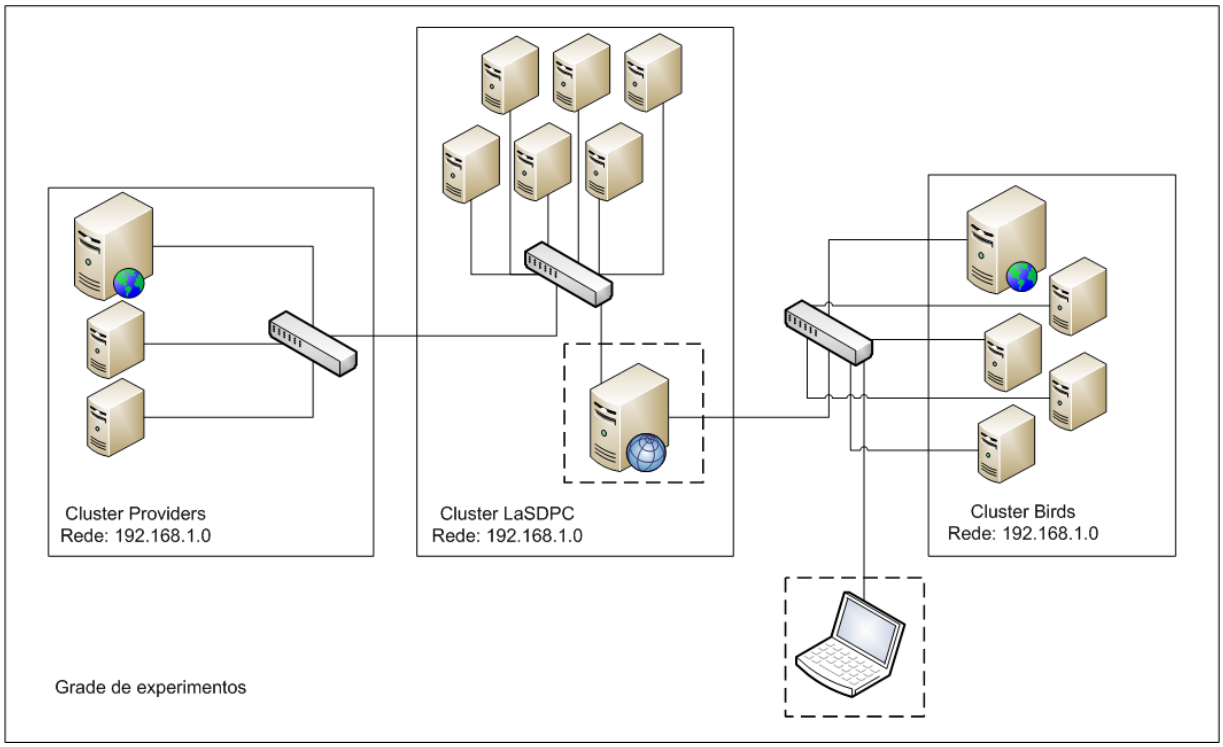

Figura 5.5: Topologia dos *clusters* disponíveis para experimentos

um resultado estatítisco confiável do tempo de resposta, resultando em 8100 execuções. Para os experimentos distribuídos no *cluster* e em grade, a quantidade de sementes para cada experimento é determinada em função da quantidade de nodos, e é gerenciado pelo Analisador Global do ASDA MRIP. Assim, cada experimento é composto por uma execução, que intrinsicamente possui n nodos de réplicas executando em paralelo, totalizando 81 execuções, e, como nos experimentos sequenciais, este conjunto foi executado 10 vezes para obter-se um resultado estatístico confiável. Os experimentos distribuídos na grade processam uma quantidade proporcional dos experimentos do usuário a quantidade de nodos disponível em cada ponto da grade. Assim, a maioria dos experimentos executou no *cluster* LaSDPC, outra fração pelo *cluster* Birds e uma menor pelo *cluster* Providers. O tempo de resposta para as execuções na grade foi o maior tempo obtido da execução em cada *cluster* (LaSDPC, Bird e Providers).

5.3 Resultados

Esta seção apresenta os resultados obtidos das execuções dos experimentos da seção 5.2.2 e discute alguns pontos sobre os resultados.

5.3.1 *Speedup* e Eficiência

Speedup é a medida usada para verificar se o desempenho de uma implementação paralela de um algoritmo sequencial aumentou o tempo de execução na mesma proporção dos processadores utilizados. O valor é obtido pela razão entre o tempo da execução sequencial pelo tempo da execução paralela, a saber

$$S_p = \frac{T_1}{T_p}$$

Onde:

T_1**:** tempo de execução sequencial

T_p**:** tempo de execução em p processadores

Outra medida utilizada é a Eficiência, a qual informa a porcentagem de utilização dos processadores. Valores próximos a 100% indicam uma maior utilização de todos os processadores presentes no sistema, deixando-os o mínimo possível em estado ocioso. Ao passo que valores menores indicam a presença de processadores não utilizados. O índice de eficiência pode ser obtido pela fórmula

$$E_p = \frac{S_p}{p}$$

Onde:

p**:** corresponde ao número de processadores

S_p**:** corresponde ao *Speedup* obtido

5.3.2 Comparação dos resultados sequenciais × *cluster* LaSDPC

As simulações sequenciais funcionam como no código apresentado na seção 5.2.1, o qual possui um laço de repetição principal. A condição de parada é um determinado valor constante correspondente a um tempo lógico desejado e mesmo que já tivesse sido alcançado um resultado estatísticamente confiável, a execução continuaria.

Isto aconteceu no caso do *benchmark* proposto neste trabalho. A execução sequencial iterava mais vezes do que aquelas coordenadas pelo simulador MRIP. Assim, os ganhos com *speedup* e eficiência foram altos. Estes ganhos não são ilusórios pois proporcionam ao usuário a quantidade certa de processamento para alcançar respostas a confiabilidade

desejada. Mais ainda, o ASDA MRIP retira do usuário a responsabilidade de avaliar tal propriedade estatística. Não sendo necessária, execuções extras, caso o grau de confiança final não tenha sido alcançado.

A tabela 5.3 mostra os resultados obtidos.

Tabela 5.3: Speedup e eficiência

	Sequencial	**LaSPC**	**Speedup 1**	**Eficiência 1**
Inalterado	151.45 min.	7.27 min.	20.83	297.54%
Pequeno	350.77 min.	14.76 min.	23.77	339.53%

Em resumo, são apresentados dois ganhos na adoção do ASDA MRIP. O primeiro é um *speedup* próximo ao linear – o que garantiria valores para a coluna *speedup 1* próximos a 7, e para a eficiência próximos a 100% –, uma vez que as simulações sequenciais são replicadas em paralelo, tendo um ponto de sincronismo individual entre uma instância e o servidor mestre, não prejudicando as outras que podem continuar processando. O segundo é o fato de processar somente o quantidade necessária para alcançar resultados confiáveis, possibilitando um tempo de execução menos, mas com resultados estatisticamente equivalentes. Tal fato proporcionou um ganho de *speedup* e eficiência bem acima daqueles ideais.

5.3.3 Comparação dos resultados *cluster* × grade

O cluster LaSDPC possui 7 nodos e quando os outros *clusters* são adicionados para formar a grade, o total de nodos resulta em 15. Para o cálculo do *speedup* o procedimento será o mesmo, dividindo-se o valor obtido pelos 15 nodos que compõem a grade, por aquele valor obtido pela execução no *cluster* LaSDPC, com 7 nodos. No entanto, para que a eficiência seja calculada corretamente, primeiro é necessário calcular a proporção de processadores adicionados (de 7 para 15) quando considerado o ambiente de grade.

$$p = \frac{N_{grade}}{N_{cluster}} \Rightarrow \frac{15}{7} \Rightarrow 2.14$$

Onde:

p**:** corresponde a quande proporcional de processadores adicionados;

N_{grade}**:** corresponde ao número de nodos presente na grade;

$N_{cluster}$**:** corresponde ao número de nodos presente no *cluster*.

Desta forma a tabela 5.4 utiliza o valor 2.14 para a variável p da fórmula do eficiência. Esse valor representa também o valor ideal para o *speedup*, representando que a adição de

dois *clusters* (`Birds` e `Providers`, que somam 8 nodos) àquele LaSDPC (com 7 nodos) é equivalente a adição de 2.14 processadores, considerando que o LaSDPC representa uma unidade processante.

O índice alcançado para os experimentos "inalterados" foi um pouco inferior ao *speedup* ideal, 2.14. No entanto, ainda com um desempenho ótimo devido ao escalonamento proporcional dos conjuntos de experimentos submetidos à grade, uma vez que os todos nodos são homogêneos. No caso daqueles com tamanho pequeno houve um ganho extra de desempenho, também influenciado pelo escalonamento proporcional.

Tabela 5.4: Speedup e eficiência

	LaSPC	**Grade**	**Speedup 2**	**Eficiência 2**
Inalterado	7.27 min.	3.44 min.	2.12	98.72%
Pequeno	14.76 min.	6.51 min.	2.27	105.77%

Em resumo, os ganhos de desempenho estão relacionados com o aumento recursos disponíveis e a distribuição proporcional das tarefas entre eles.

5.3.4 Consolidação dos resultados

Nesta seção são apresentados todos os resultados obtidos de forma consolidada. Como mencioando na seção 5.2.3, foram feitas várias execuções e a partir dos resultados foram calculados a média aritmética e o intervalo de confiança. Assim, os valores de cada células apresentados nas tabelas 5.5 e 5.6 representam os valores médios obtidos de todos os experimentos e as linhas numeradas de 1 a 10 indicam a execução. Na parte inferior das tabelas estão destacados os valores médio, desvio padrão e intervalo de confiança com confiabilidade de 95%.

Tabela 5.5: Resultados consolidados para simulações inalteradas. (valores em minutos)

Execução	**Sequencial**	**Cluster**	**Grade**
1	148.73	6.97	3.22
2	149.44	7.00	3.01
3	151.50	7.29	3.67
4	151.93	7.19	3.79
5	151.89	7.37	3.53
6	152.23	6.94	3.09
7	152.07	7.01	3.30
8	152.07	7.99	4.03
9	152.59	6.82	3.06
10	152.07	8.12	3.67
Média	151.45	7.27	3.44
Desvio Padrão	1.29	0.45	0.35
Int. Confiança	0.80	0.28	0.22

5.4 Considerações Finais

Neste capítulo foram detalhados: a implementação do *benchmark* para avaliação do desempenho, o modelo implementado e considerações sobre as ferramentas utilizadas para sua concepção. O planejamento de experimentos dos usuários e aqueles usados para a avaliação também foram descritos, bem como o ambiente utilizados para os testes.

O resultados se mostraram ótimos, uma vez que o ASDA MRIP proporciona naturalmente um ganho de desempenho alto por se tratar de aplicações sequenciais replicadas em paralelo, característica a qual é aliada à utilização eficiente do tempo a ser simulado, evitando que uma simulação seja mais longa que o necessário, e vice-versa. Para o usuário, automatiza o processo de aferição da confiabilidade estatística da variável de resposta observada na simulação.

Na perspectiva da computação em grade, os resultados são mostraram ótimos, pois a abordagem de escalonamento utilizada garantiu um divisão proporcional das tarefas a serem executadas no ambiente de testes, implementado em laboratório e controlado.

Na comparação das simulações sequenciais para aquelas executadas em *cluster* houve um *speedup* extra, além daquele ideal, garantindo uma eficiência de cerca de 197% para as simulações sequenciais quando executadas em *cluster*. Este ganho foi maior quando as simulações tiveram seus tempos aumentados, garantindo cerca de 239% de ganho extra. Quando comparadas execuções em *cluster* e em grade, não houve um ganho extra de *speedup* para as simulações com tamanho inalterado, no entanto elas se aproximaram bastante daquele valor ideal, eficiência de cerca de 98%. Já as simulações de tamanho pequena proporcionaram um ganho extra de 5% além do valor ideal.

Tabela 5.6: Resultados consolidados para simulações com tamanho pequeno. (valores em minutos)

Execução	**Sequencial**	**Cluster**	**Grade**
1	348.80	14.72	6.42
2	350.09	14.90	6.75
3	350.22	13.85	6.09
4	351.22	14.86	6.43
5	350.06	15.19	6.47
6	351.67	14.57	6.28
7	351.45	14.62	6.44
8	351.42	14.78	6.90
9	352.14	14.32	6.64
10	350.65	15.78	6.72
Média	350.77	14.76	6.51
Desvio Padrão	0.99	0.51	0.24
Int. Confiança	0.62	0.31	0.15

O próximo capítulo irá apresentar as conclusões obtidas com o desenvolvimento deste trabalho, as contribuições obtidas e os trabalhos futuros que podem ser considerados para a melhora das peças de *software* propostas nesta dissertação.

Capítulo

6

Conclusão

6.1 Conclusões

O objetivo deste trabalho foi estudar ambientes de computação em grade disponíveis e executar simulações distribuídas que usam a abordagem MRIP. Outro ponto de investigação foi o estudo da teoria envolvida na área de Planejamento de Experimentos, de forma que estes conceitos pudessem ser aplicados na concepção de uma ferramenta que gerencie os experimentos e suas respectivas execuções no ambiente em questão. Para a implementação deste projeto de mestrado foram estudados dois *middlewares* para implementação de grades computacionais, que motivou a arquitetura para a implementação da ferramenta. Por fim, foi feito um estudo do desempenho de simulações sequenciais e que utilizaram recursos distribuídos.

A ferramenta para Gerenciamento de Experimentos em Ambiente de Grade – GEM (*Grid Experiments Manager*) – foi implementada em três componentes principais: 1) situado do lado do cliente o qual recebe a solicitação de execução de um projeto de experimentos e o encaminha para um repositório de projetos e resultados; 2) outro denominado agente executor que faz a ligação entre a requisição do usuário com os recursos disponíveis, escalonando as tarefas proporcionalmente entre eles; 3) o módulo de encaminhamento de execuções, o qual recebe um conjunto de experimentos de um projeto maior e o submete

a um ambiente de computação em grade para finalmente ser processado.

Os 2 *middlewares* estudados foram o `Ourgrid` e o `Globus`, sendo o `Ourgrid` estudado primeiramente devido sua implementação contemplar aplicações do tipo *bag-of-tasks*. Estas aplicações são semelhantes às replicações feitas pelo ASDA MRIP, do ponto de vista que um tarefa não se comunica com outra, no entanto, cada tarefa comunica-se com um servidor mestre o qual coordena a execução. O fato de não haver um ponto de comunicação entre duas tarefas inviabilizou a utilização deste *middleware*. O `Globus` foi escolhido por sua flexibilidade para configuração de um ambiente de grade, possibilitando a implementação de um ambiente compatível para as execuções necessária no presente trabalho. O `Ourgrid` se mostrou inadequado para o escopo deste trabalho, mas como discutido na seção 6.3, ele pode fornecer vantangens quando utilizado por outros tipos de aplicações.

Dentre os trabalhos relacionados, não foi encontrada uma proposta voltada para o controle das execuções integradas com vários ambiente de computação em grade. Embora, tenha sido implementado o módulo para `Globus`, outros poderão ser construídos no futuro, o que acrescenta transparência na gerência e na execução de experimentos de um usuário.

Do ponto de vista do desempenho, o ganho no tempo de execução, *Speedup*, foi ótimo, superior ao índice ideal em três casos e quase ideal em um caso. Este ganho se deve ao fato de o programa mestre do ASDA MRIP garantir que as iterações das execuções sejam controladas até o limite suficiente para que a variável de resposta tenha a confiabilidade estatística desejada, ao passo que em uma simulação sequencial, o usuário não tem este controle, podendo uma simulação executar por um tempo maior do que aquele necessário.

6.2 Contribuições

As principais contribuições realizadas com o desenvolvimento deste trabalho de mestrado são listadas a seguir:

Um protótipo para um Gerenciador de Experimentos em Ambiente de Grade: o GEM pode ser considerada a maior contribuição deste trabalho, uma vez que permite aglutinar vários recursos computacionais para que possam ser utilizados, possibilitando a implementação de escalonadores, permitindo também a criação de um repositório de experimentos e seus respectivos dados de resposta;

Avaliação de desempenho para simulações MRIP: outra contribuição relevante obtida foi estudo que mostrou a eficiencia da utilização de simulações distribuídas com abordagem MRIP, implementadas pelo ASDA;

Melhorias na bilbioteca de simulação MRIP: o código da biblioteca foi reestrutu-

rado podendo ser utilizado em outros programas mais facilmente, permitindo que haja uma compilação para cada arquitetura, diferentemente, da organização anterior, a qual exigia explicitamente a inclusão de todos os código objetos para compilação.

6.3 Trabalhos futuros

O protótipo implementado neste projeto de mestrado ainda precisa de incrementos e melhorias para que possa ser efetivamente utilizado como uma ferramenta de produtividade por usuários. Dentre as melhorias, pode-se destacar:

Implementação de um escalonador integrado com o ambiente de grade: estudar as formas de eficientemente integrar um serviço de monitoração com o objeto escalonador do GEM e com isto permitir que decisões melhores sejam tomadas. As informações de monitoração dizem respeito a quantidade de nodos disponível em cada sítio, a potência computacional de cada nodo, a carga de trabalho atual, a taxa de transferência da rede, etc. Todos estes dados podem ser combinados de forma a proporcionar uma melhor alocação dos experimentos a serem executados;

Estender o escopo do projeto para executar outras aplicações: o GEM é proposto como uma solução para simulações que executam o ASDA MRIP, mas seu uso pode ser estendido para atender outras;

Melhorias no sistema de comunicação: o sistema de comunicação usa o protocolo TCP com portas altas, comumente bloqueadas por *firewalls*, o que pode impedir a comunicação entre os componentes da solução. Assim, propõe-se um estudo de outras alternativas para comunicação, como, por exemplo, a utilização de Serviços *Web*;

Melhorar o módulo GEM-Globus para a utilização Java-CoG Kit: estudar mecanismos de integração com o Java-CoG Kit, de forma a aproveitar todos os recursos disponíveis pelo Globus de forma mais simples e eficiente;

Portar a biblioteca ASDA MRIP para Java: o porte garantiria uma possibilidade maior de execuções com sucesso, ficando, na maioria dos casos, isentas de erro em tempo de execução devido a questões de arquitetura e bibliotecas.

Referências Bibliográficas

BANKS, J. Principles of simulation. In: BANKS, J. (Ed.). *Handbook of Simulation: principles, methodology, advances, applications, and practice*. New York, NY, USA: John Wiley and Sons, Inc., 1998. ISBN 0471134031.

BANKS, J.; CARSON, J. S.; NELSON, B. L. *Discrete-event system simulation*. 2nd. ed. Upper Saddle River, New Jersey 07458: Prentice-Hall, 1996. ISBN 0132174499.

BERMAN, F.; FOX, G.; HEY, A. J. G. *Grid Computing: Making the Global Infrastructure a Reality*. New York, NY, USA: John Wiley & Sons, Inc., 2003. ISBN 0470853190.

BOTE-LORENZO, M. L.; DIMITRIADIS, Y. A.; GOMEZ-SANCHEZ, E. Grid characteristics and uses: a grid definition. In: *First European Across Grids Conference*. Santiago de Compostela, Spain: [s.n.], 2004. LNCS 2970, p. 291–298. Lecture Notes in Computer Science.

BRUSCHI, S. M.; SANTANA, R. H. C.; SANTANA, M. J.; AIZA, T. S. An automatic distributed simulation environment. In: *WSC '04: Proceedings of the 36th conference on Winter simulation*. [S.l.]: Winter Simulation Conference, 2004. p. 378–385. ISBN 0-7803-8786-4.

BRYANT, R. E. *Simulation of packet communication architecture computer systems*. [S.l.]: Cambridge, MA, USA, 1977.

BUYYA, R.; KRAUTER, K.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. *Software - Practice and Experience*, v. 32, n. 2, p. 135–164, 2002.

CHANDY, K. M.; MISRA, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 5, p. 440–452, 1979. ISSN 0098-5589.

CHEDE, C. T. *Grid computing: um novo paradigma computacional*. Rio de Janeiro, RJ, Brazil: Brasport, 2004. ISBN 8574521930.

CIRNE, W. Computation grids: Architectures, technologies and applications. *Terceiro Workshop em Sistemas Computacionais de Alto Desempenho*, 2002.

CIRNE, W.; BRASILEIRO, F.; ANDRADE, N.; COSTA, L.; ANDRADE, A.; NOVAES, R.; MOWBRAY, M. Labs of the world, unite!!! *Journal of Grid Computing*, Springer, New York, USA, v. 4, n. 3, p. 225–246, 2006. ISSN 1570-7873.

COG-KIT. *Java CoG Kit*. 2004. Http://www.cogkit.org. Último acesso em 25 de Fevereiro de 2010.

CONDOR. *Projeto Condor*. 2005. Https://www.cs.wisc.edu/condor. Último acesso em 25 de Fevereiro de 2010.

EWING, G. C.; PAWLIKOWSKI, K.; MCNICKLE, D. Akaroa-2: Exploiting network computing by distributing stochastic simulation. In: *Proceeding of European Simulation Multiconference (ESM?99)*. [S.l.: s.n.], 1999. p. 175–181. Warson – Poland.

FISHWICK, P. A. Simpack: getting started with simulation programming in c and c++. In: *WSC '92: Proceedings of the 24th conference on Winter simulation*. New York, NY, USA: ACM, 1992. p. 154–162. ISBN 0-7803-0798-4.

FORTIER, P. J.; MICHEL, H. *Computer Systems Performance Evaluation and Prediction*. Newton, MA, USA: Butterworth-Heinemann, 2002. ISBN 1555582605.

FOSTER, I.; KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, v. 11, n. 2, p. 115–128, 1997.

FOSTER, I.; KESSELMAN, C. *Computational grids*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. 15–51 p. ISBN 1-55860-475-8.

FOSTER, I.; KESSELMAN, C. The globus project: a status report. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 15, n. 5-6, p. 607–621, 1999. ISSN 0167-739X.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan-Kaufman, 1999.

FOSTER, I.; KESSELMAN, C. *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. ISBN 1558609334.

FOSTER, I.; KESSELMAN, C.; NICK, J. M.; TUECKE, S. The physiology of the grid. *Global Grid Forum*, 2002.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 15, n. 3, p. 200–222, 2001. ISSN 1094-3420.

FUJIMOTO, R. M. *Parallel and Distributed Simulation Systems*. [S.l.]: John Wiley and Sons, Inc., 2000.

GLOBUS. *Projeto Globus*. 2005. Https://www.globus.org. Último acesso em 25 de Fevereiro de 2010.

GLYNN, P. W.; HEIDELBERGER, P. Analysis of initial transient deletion for parallel steady-state simulations. *SIAM J. Stat. Comput*, v. 13, n. 4, p. 904–922, 1992.

GRIDCAFé. 2008. Último acesso em 25 de Fevereiro de 2010. Disponível em: <gridcafe.web.cern.ch>.

GRIDGAIN. *GridGain*. 2009. Último acesso em 25 de Fevereiro de 2010. Disponível em: <www.gridgain.org>.

GRIMSHAW, A. S.; WEISSMAN, J. B.; WEST, E. A.; LOYOT, J. E. *MetaSystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems*. Charlottesville, VA, USA, 1992.

GRIMSHAW, A. S.; WULF, W. A.; TEAM, C. T. L. The legion vision of a worldwide virtual computer. *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 1, p. 39–45, 1997. ISSN 0001-0782.

HEIDELBERGER, P. Discrete event simulation and parallel processing: Statistical properties. *SIAM J. Stat. Comput*, v. 9, n. 6, p. 1114–1132, 1988.

INTERNET2. *Shibboleth*. 2008. Último acesso em 25 de Fevereiro de 2010. Disponível em: <shibboleth.internet2.edu>.

JAIN, R. *The Art of Computer Systems Performance Analysis*. [S.l.]: John Wiley and Sons, Inc., 1991.

JEFFERSON, D. R. Virtual time. *ACM Transactions on Programming Languages and Systems*, v. 7, n. 3, p. 404–425, 1985.

KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. *Introduction to Parallel Computing*. [S.l.]: John Wiley and Sons, Inc., 2001.

LASZEWSKI, G. V.; ZIMNY, P.; TRIEU, T.; ANGULO, D. The java cog kit experiment manager. *Second International Workshop on Grid Computing Environments*, 2006.

LEYE, S.; HIMMELSPACH, J.; JESCHKE, M.; EWALD, R.; UHRMACHER, A. M. A grid-inspired mechanism for coarse-grained experiment execution. In: *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. Washington, DC, USA: IEEE Computer Society, 2008. p. 7–16. ISBN 978-0-7695-3425-1.

MACDOUGALL, M. H. *Simulating computer systems: techniques and tools*. Cambridge, MA, USA: MIT Press, 1987. ISBN 0-262-13229-X.

MAGALHãES, M. N.; LINA, A. C. P. de. *Noções de Probabilidade e Estatística*. 6th. ed. São Paulo, SP, BRA: Edusp, 2005. ISBN 85-314-0677-3.

MISRA, J. Distributed discrete-event simulation. *Computer Surveys*, v. 18, n. 1, p. 39–65, 1986.

MONO. *Mono Project*. 2004. Http://www.mono-project.org. Último acesso em 25 de Fevereiro de 2010.

MONTGOMERY, D. C. *Design and Analysis of Experiments*. 6th. ed. [S.l.]: John Wiley and Sons, Inc., 2004.

MPOG. *Guia de Estruturação e Administração do Ambiente de Cluster e Grid*. 2006. Http://www.governoeletronico.gov.br/anexos/guia-de-cluster. Último acesso em 25 de Fevereiro de 2010.

OGF. *Open Grid Forum*. 2005. Http://www.ogf.org/. Último acesso em 25 de Fevereiro de 2010.

OURGRID. *OurGrid Project*. 2007. Http://www.ourgrid.org/. Último acesso em 25 de Fevereiro de 2010.

PATTERSON, D. A.; GIBSON, G.; KATZ, R. H. A case for redundant arrays of inexpensive disks (raid). In: *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1988. p. 109–116. ISBN 0-89791-268-3.

REED, D. A.; MALONY, A. D.; MCCREDIE, B. Parallel discrete event simulation using shared memory. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 4, p. 541–553, 1988. ISSN 0098-5589.

RIPEANU, M.; SINGH, M. P.; VAZHKUDAI, S. S. Virtual organizations. *Internet Computing, IEEE*, v. 12, n. 2, p. 10–12, 2008. ISSN 1089-7801.

SANTANA, M. J.; SANTANA, R. H. C. Avaliação de desempenho – planejamento de experimentos. Notas de aula. Último acesso em 25 de Fevereiro de 2010. 2007. Disponível em: <lasdpc.icmc.usp.br>.

SHAREGRID. *ShareGrid*. 2007. Http://dcs.mfn.unipmn.it/sharegrid/. Último acesso em 25 de Fevereiro de 2010.

SMARR, L.; CATLETT, C. E. Metacomputing. *Commun. ACM*, ACM, New York, NY, USA, v. 35, n. 6, p. 44–52, 1992. ISSN 0001-0782.

STEVENS, R.; WOODWARD, P.; DEFANTI, T.; CATLETT, C. From the i-way to the national technology grid. *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 11, p. 50–60, 1997. ISSN 0001-0782.

STOCKINGER, H. Defining the grid: a snapshot on the current view. *J. Supercomput.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 42, n. 1, p. 3–17, 2007. ISSN 0920-8542.