SimGrid x SAN: Um Estudo Comparativo de Ferramentas de Avaliação de Desempenho de Plataformas Distribuídas

Mateus Raeder

Porto Alegre
2009

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

SimGrid x SAN: Um Estudo Comparativo de Ferramentas de Avaliação de Desempenho de Plataformas Distribuídas

Mateus Raeder

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

Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

R134s Raeder, Mateus
SimGrid x SAN : um estudo comparativo de ferramentas de avaliação de desempenho de plataformas distribuídas / Mateus Raeder. – Porto Alegre, 2009.
65 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes.

1. Sistemas Distribuídos. 2. Simulação e Modelagem em Computadores. 3. Avaliação de Desempenho (Informática). I. Fernandes, Luiz Gustavo Leão. II. Título.

CDD 004.36

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

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

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "*SimGrid* x *SAN*: Um Estudo Comparativo de Ferramentas de Avaliação de Desempenho de Plataformas Distribuídas", apresentada por Mateus Raeder, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 16/01/09 pela Comissão Examinadora:

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

Prof. Dr. Paulo Henrique Lemelle Fernandes – PPGCC/PUCRS

Prof. Dr. Marilton Sanchotene de Aguiar – UCPel

Homologada em 16/06/09, conforme Ata No. 010 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Dedico este trabalho para Marcia e Rogério Raeder. Mais do que pais, são grandes amigos, sempre incansáveis e pacientes durante toda minha caminhada.

AGRADECIMENTOS

Para agradecer individualmente todos aqueles que, de alguma maneira ou outra, ajudaram-me a finalizar esta etapa, seria necessário muito espaço. Entretanto, os agradecimentos pontuais que faço aqui estendem-se a todos estes que me ajudaram nesta caminhada. Primeiramente, fica aqui o agradecimento aos meus pais, Marcia e Rogério. Quantas vezes acordavam para ir trabalhar e lá estava eu, sentado no mesmo lugar, fazendo alguma coisa que, normalmente, era para o mesmo dia. Durante o primeiro ano eram os trabalhos das disciplinas. Durante o segundo ano, a dissertação. E durante todo este tempo, não deixaram de me apoiar um segundo sequer. Palavras não são nada perto do que eu tenho a agradecê-los. Além deles, algumas peças foram extremamente importantes durante este tempo. O primeiro deles, sem dúvida, é o meu orientador, Prof. Dr. Luiz Gustavo Leão Fernandes. Desde a época da minha graduação (quando também foi meu orientador) até hoje, ele não mede esforços para defender todos os seus orientandos, sempre encontrando uma alternativa para os problemas que surgem. O Gustavo é, com certeza, mais do que um orientador, é um amigo. E tenho certeza que seria capaz até de escrever alguma parte do meu trabalho se fosse necessário. Fundamentais, também, foram meus colegas Thiago Nunes e Márcio Castro. Obrigado por tudo. E quando digo "tudo", é dentro e fora do Mestrado, pois desde um trabalho até uma festa, nós estávamos sempre juntos. Thiago, obrigado pelo companheirismo de sempre e por sempre acreditar em mim. Márcio, obrigado por me ajudar sempre que possível, principalmente quando o tempo estava apertando. Dois grandes amigos que não se encontra em qualquer lugar. Fica aqui também o meu grande agradecimento a duas pessoas que não poderiam faltar: Andriele e Piccoli. Obrigado por me aturarem, aguentando quando eu ficava saturado de alguma coisa e resolvia ir incomodar vocês lá embaixo. Piccoli, obrigado pelas inúmeras caronas e pelas divertidas madrugadas de troca de informações do tipo "falta muito aí?". Andriele, obrigado pela grande amizade que temos, foste muito importante em diversos momentos. Também gostaria de agradecer aos meus amigos Pedro Velho, Lucas Baldo e Márcio Dorn, que foram fonte de muito conhecimento durante longos anos. Também fica meu agradecimento pelos bons momentos proporcionados pela presença de alguns amigos, como Mariana Kolberg, Rafael Nemetz, Éverton Alexandre e Elder Macedo. A todos estes e aos demais que não pude citar aqui, fica minha eterna gratidão pela grande força e pela grande amizade que será, com certeza, para a vida inteira.

RESUMO

Avaliação de desempenho de sistemas distribuídos trata-se de uma importante ferramenta durante a fase de desenvolvimento. Esta tarefa tem por objetivo mostrar ao usuário (pesquisador, programador, analista, etc.) o comportamento previsto para o seu sistema, antes mesmo que este esteja implementado, evitando custos com tempo de implementação e de eventuais reconstruções de código. Neste contexto, existem maneiras muito utilizadas na comunidade científica para dar suporte a avaliação de desempenho, como Simulação e Modelagem Analítica. Entretanto, a criação e a execução destes modelos nem sempre é realizada de maneira trivial, pois uma importante fase na modelagem é a obtenção dos parâmetros do sistema alvo, devido ao fato de que quanto mais fiel for o modelo, mais próximo do real serão os resultados. Assim, este trabalho visa realizar uma análise comparativa entre duas ferramentas de avaliação de desempenho (SimGrid e SAN) através de duas abordagens diferentes (Simulação e Modelagem Analítica, respectivamente). Para isto, os parâmetros das aplicações executadas no SimGrid foram mapeados para o modelo SAN, procurando desta forma obter uma equivalência nas modelagens para que seus resultados possam ser comparados. Após a apresentação dos resultados obtidos, alguns aspectos interessantes sobre as duas maneiras de avaliação de desempenho são discutidos.

Paravras-chave: avaliação de desempenho, programação distribuída, modelagem analítica, simulação, SAN, SimGrid.

ABSTRACT

Distributed systems performance evaluation is an important tool during the development phase. This task objective is to show to the user (researcher, developer, analyst, etc.) the predicted behavior of his system, even before that such system is implemented, avoiding additional costs with implementation efforts and possible code reconstructions. In this context, there are methods widely utilized on the scientific community to support the performance evaluation, such as Simulation and Analytical Modeling. However, the creation and execution of these models is not always performed in a trivial way, because an important step on the modeling is the obtaining of the target systems parameters, due to the fact that the more accurate is the model, the more closely to the real case the results will be. Thus, this work intends to do a comparison analysis between two performance evaluation tools (SimGrid and SAN) through two distinct approaches (Simulation and Analytical Modeling, respectively). For such purpose, the parameters of the executed applications on the SimGrid were mapped to a SAN model, seeking for an equivalence on the modeling in order to compare their results. Following to the presentation of the obtained results, some interesting aspects about the two performance evaluation tools are discussed.

Keywords: performance evatuation, distributed systems, analytical modeling, simulation, SAN, SimGrid.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

SAN	Sthocastic Automata Network	15
UCSD	University of California at San Diego	20
API	Application Programming Interface	23
MSG	Meta-SIMGRID	23
GRAS	Grid Reality And Simulation	24
PEPS	Performance Evaluation of Parallel Systems	29

Sumário

1 Introdução

Os avanços nos diversos ramos da Ciência (tais como Medicina, Física, Química, Engenharia e Geologia) foram concebidos através da utilização de ferramentas computacionais. Entretanto, simulações de grande porte constantemente demandam um alto poder computacional para que sejam realizadas. Com a finalidade de evitar os elevados custos da compra de supercomputadores, a utilização de aglomerados de computadores pessoais de baixo custo surge como uma alternativa atraente.

A partir de então, uma vasta gama de tecnologias criadas para explorar recursos vem surgindo, como clusters, grades computacionais, etc.

Apesar da utilização destas plataformas distribuídas serem alternativas atraentes, o desenvolvimento de aplicações em tais ambientes é particularmente difícil, devido ao grande número de variáveis envolvidas e diferentes tipos possíveis de falhas. Plataformas mais distribuídas (como grades computacionais, por exemplo) são espalhadas em diversos domínios administrativos, trazendo como conseqüência flutuações na disponibilidade dos recursos. Logo, a condução de experimentos repetitivos para aplicações com tempos de execução relativamente altos torna-se impossibilitada, dificultando a avaliação de desempenho de aplicações reais, principalmente considerando as inúmeras possibilidades de configuração existentes.

Esta avaliação de desempenho, no entanto, é uma tarefa imprescindível para a validação da implementação realizada. Esta importância se deve ao fato de que, quando se deseja desenvolver determinado sistema para um dado propósito, a expectativa é de que ele se comporte de maneira adequada, apresentando um bom desempenho e sendo capaz de executar em um tempo aceitável.

Existem diversas formas de avaliação de desempenho, normalmente divididas em três grandes grupos:

- *Medições*: na qual o programa deve estar previamente implementado para que medidas sejam coletadas de suas execuções;
- *Simulações*: na qual um modelo computacional do sistema é criado para observar os fatores que interferem no seu comportamento;
- *Modelagem analítica*: que se trata de uma representação matemática do sistema a ser analisado.

Neste estudo, utilizar-se-á **simulação** e **modelagem analítica**, com o intuito de comparar tais para efetuar a avaliação de desempenho de aplicações para plataformas distribuídas.

O formalismo analítico a ser utilizado neste trabalho será Rede de Autômatos Estocásticos (SAN - *Sthocastic Automata Network*) [1], por tratar-se de um formalismo mais recente e fortemente baseado na teoria de Cadeias de Markov [2], porém expressando os modelos Markovianos de forma mais fácil, compacta e eficiente.

Na modelagem analítica, entretanto, as métricas de desempenho são geradas através de parâmetros indicados no modelo. Assim, experimentos realizados através de simulação surgem como uma escolha bastante viável para a obtenção precisa de tais parâmetros. Simulações podem ser repetidas, são cofiguráveis e geralmente demandam menos tempo que execuções reais. Este tipo de abordagem vem sendo amplamente utilizada para apontar decisões corretas durante o desenvolvimento de *software* para ambientes de grades computacionais. Uma das ferramentas mais conhecidas para a simulação de algoritmos em grades é o *SIMGRID* [3], que será utilizada neste trabalho para a realização dos objetivos nele propostos.

Na comunidade científica, alguns trabalhos utilizando a modelagem SAN para a avaliação de desempenho de sistemas computacionais podem ser encontrados. Um deles trata-se de um trabalho realizado por Maraculescu e Nandi [4]. Nele, os autores apresentaram uma nova metodologia para a modelagem de aplicações de sistemas de níveis. O decodificador de vídeo MPEG-2 foi a aplicação utilizada para validar e exemplificar as vantagens desta nova abordagem. O formalismo SAN foi utilizado para modelar a aplicação, e foi comprovado que esta possui um comportamento estacionário através de diferentes probabilidades, o que permitiu mapeamentos mais eficientes da aplicação. Os próprios autores consideraram SAN como uma ferramenta eficiente para modelar a comunicação entre os processos, e ainda relataram a vantagem de que o número de estados não explode.

Outro trabalho interessante com a utilização de SAN trata-se do trabalho descrito em [5]. Neste trabalho, os autores propuseram o uso de Redes de Autômatos Estocásticos para desenvolver modelos que se aplicassem a programas do tipo *mestre/escravo*, considerando dois padrões de comportamento para descrever a comunicação entre os mestres e os escravos: *síncrono* e *assíncrono*. Como caso de estudo, o algoritmo de *region growing* conhecido como *Propagation* [6] foi utilizado. Esta aplicação trabalha com interpolação de imagens, para gerar uma visão virtual entre dois pontos iniciais distintos. A aplicação foi modelada utilizando SAN para validar a estratégia de implementação escolhida. Segundo os autores, a modelagem de programas paralelos é facilitada com o formalismo SAN, e pôde dar uma visão geral dos resultados que serão obtidos com a paralelização do *Propagation* de acordo com alguns fatores, como sincronia/assincronia e grão fino/grosso.

O trabalho realizado por Mokdad et. al. [7] também utiliza a modelagem SAN. Neste trabalho, os autores apresentaram um novo algoritmo de roteamento, para melhorar a qualidade de entrega de pacotes em redes de baixa latência. Os autores mostram os benefícios desta nova proposta de algoritmo através da avaliação de desempenho de modelos SAN, além de ressaltar o fato de que o formalismo SAN é muito utilizado para sistemas complexos, quando a utilização de Cadeias de Markov é inviável.

1.1 Motivação

A implementação de aplicações para sistemas computacionais (em diversas ocasiões) não é trivial, principalmente quando se tratam de aplicações paralelas e distribuídas. Estudos sobre o desempenho apresentado por tais aplicações antes mesmo de sua implementação de fato, são de grande importância para que as soluções propostas sejam previamente validadas, evitando problemas relacionados com reconstruções do sistema. Uma vez previsto o comportamento da aplicação desejada, alterações na implementação, nos parâmetros ou nas configurações da solução proposta podem ser realizados (se necessário) para que o desempenho real apresente-se conforme o esperado.

Devido ao constante avanço das plataformas distribuídas e sua crescente utilização, a construção de simuladores facilita ao usuário a avaliação dos sistemas propostos. Isto se deve ao fato de que há uma maior flexibilidade de configuração e diversificação dos parâmetros do ambiente, flexibilidade esta que se torna bastante complexa de ser realizada na prática.

Na modelagem analítica, por sua vez, as métricas são obtidas através de modelos construídos com certos parâmetros que vêm da aplicação. Uma das grandes dificuldades encontradas na construção e resolução deste método, no entanto, é exatamente a definição precisa destes parâmetros, o que acarreta em incertezas nos resultados obtidos aravés dos modelos.

Com a ferramenta de simulação *SIMGRID*, os parâmetros providos pelo simulador podem agregar grande valor na construção do modelo SAN das aplicações, tendo em vista as configurações mais bem definidas desta ferramenta.

1.2 Objetivos

O objetivo do trabalho a ser realizado é **comparar o comportamento e os resultados obtidos através da criação de Modelos Analíticos (com a utilização da ferramenta SAN) de aplicações para plataformas distribuídas, com aqueles obtidos através da utilização de uma abordagem de Simulação (através da ferramenta *SimGrid*)**. Com isto, obter-se-á uma análise das funcionalidades, analisando ainda as facilidades e os obstáculos encontrados no desenvolvimento de cada um dos tipos de modelagem. Neste contexto, poderá ser avaliado quão semelhantes e quão úteis tornam-se estas ferramentas para o usuário final, que deseja avaliar o desempenho de sua(s) aplicação(ões).

O intuito da modelagem com SAN, neste contexto, é procurar parametrizar da maneira mais fiel possível o modelo criado para as aplicações, de acordo com os parâmetros obtidos com o simulador, com a finalidade de comparar e validar a utilização de tal modelagem para este fim. Ao final do estudo, acredita-se que será possível extrair uma conclusão fortemente aceitável sobre as abordagens de Simulação e Modelagem Analítica utilizando SAN.

1.3 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma:

- **Capítulo 2:** apresenta conceitos básicos de duas das principais metodologias de avaliação de desemepenho (Simulação e Modelagem Analítica);
- **Capítulo 3:** introduz as ferramentas utilizadas para a realização dos experimentos conduzidos neste trabalho;
- **Capítulo 4:** descreve como é feita a associação entre os parâmetros de entrada dos modelos SAN e do SimGrid;
- **Capítulo 5:** detalha três casos de estudos utilizados neste trabalho;
- **Capítulo 6:** apresenta os experimentos realizados sobre os estudos de caso introduzidos no Capítulo 5;
- **Capítulo 7:** aponta as conclusões deste trabalho e indica possíveis trabalhos futuros.

2 Avaliação de Desempenho

Sistemas computacionais devem apresentar certas características para que suas implementações sejam de real valia (facilidade de utilização, de manutenção, etc). Dentre estas características, pode-se destacar o desempenho, que se trata do rendimento demonstrado pela aplicação ao realizar suas respectivas funções.

Este desempenho, então, é um critério fundamental na criação destes sistemas, principalmente quando se tratam de ambientes paralelos ou distribuídos. Nestes ambientes, tanto usuários quanto projetistas, analistas e desenvolvedores desejam que seus sistemas apresentem o melhor desempenho com o menor custo possível, o que tem resultado em uma grande evolução dos sistemas de baixo custo e alto desempenho [8].

Embora processadores e memórias atuais já atinjam desempenhos muito mais satisfatórios quando comparados aos que apresentavam antes, as pesquisas realizadas na área de avaliação de desempenho das aplicações não se detêm apenas a aspectos tecnológicos. Deste modo, diversos modelos e métodos foram (e continuam sendo) criados para realizar análises através de diferentes abordagens.

Neste contexto, para realizar a análise de desempenho de um determinado sistema, primeiramente deve-se construir um modelo condizente com ele. Construir um bom modelo que realmente auxilie na análise de desempenho, contudo, não é uma tarefa trivial e requer considerações cuidadosas. Segundo [9], um bom modelo deve incluir todos os detalhes necessários para descrever o comportamento do sistema com a maior exatidão possível. Além disso, o modelo deve ser de simples resolução, excluindo a maior quantidade de parâmetros que se consiga. Note, no entanto, que quando se incluem todos os detalhes do comportamento do sistema, aumenta-se a complexidade de resolução do modelo, tornando a tarefa de selecionar os parâmetros necessários um passo importante na construção do modelo.

Um modelo, então, trata-se de uma coleção de atributos e um conjunto de regras que governam como estes atributos interagem [10]. No âmbito da avaliação de desempenho de programas desenvolvidos para serem executados em ambientes paralelos, devido a complexidade natural inferida nestas plataformas (comunicação e sincronização, por exemplo), tais modelos devem ser suficientemente claros e simples, abstraindo ao máximo estas complexidades.

Predição e avaliação de desempenho são assuntos que caminham juntos. Pode-se dizer, inclusive, que a tarefa de predizer o desempenho de uma aplicação trata-se de um tipo especial de avaliação.

Quando se deseja desenvolver um determinado sistema (principalmente tratando-se de programas paralelos), um dos principais objetivos é a obtenção de um desempenho satisfatório.

Entretanto, para os programadores, analistas e toda a equipe que estiver envolvida na implementação deste sistema, não é interessante que se descubra que o sistema desenvolvido não apresentou o desempenho esperado somente quando a versão encontrar-se implementada e for executada.

Conforme visto anteriormente, avaliação de desempenho apresenta o comportamento da aplicação, de acordo com os fatores que se deseja analisar. Predição de desempenho, por sua vez, é o ato de anunciar com antecedência este comportamento. Assim sendo, o objetivo da predição também é compreender a maneira como o sistema se comporta, mas este comportamento é visualizado antes mesmo que a aplicação seja implementada. Com isto, gargalos e situações adversas ao que se espera podem ser encontrados e solucionados antes mesmo da construção do sistema.

Existem duas abordagens principais para o processo de análise de desempenho, as quais estão diretamente relacionadas com o estágio de desenvolvimento do sistema. A primeira delas refere-se ao estágio posterior à construção do sistema, quando ele encontra-se totalmente implementado. Assim, ao menos um protótipo deve previamente estar implementado.

A segunda abordagem refere-se a uma avaliação que (geralmente) é realizada antes da implementação da aplicação, sem a necessidade da existência de código-fonte e nem sequer protótipos. Neste caso, a avaliação está fortemente ligada à predição de desempenho, pois as técnicas são utilizadas na parte inicial no processo de desenvolvimento, e o comportamento da aplicação é previamente examinado antes mesmo de sua implementação.

A seguir, são descritas as duas técnicas para a modelagem do processo de análise de desempenho utilizadas neste estudo: **Simulação** e **Modelagem Analítica**.

2.1 Simulação

Quando se utiliza esta técnica, pretende-se simular o comportamento do sistema, para que se tenha uma idéia do seu desempenho. Com a utilização desta abordagem, pode-se realizar predições, pois não há a necessidade do sistema pronto. Para tanto, um modelo do sistema que se deseja implementar (ou mesmo de um sistema já implementado) é criado, para facilitar e observar os fatores que interferem no seu comportamento.

Os sistemas modelados podem alterar suas características com o tempo ou não, diferenciando simulações estáticas e dinâmicas, respectivamente. Conforme descrito em [8,9], diferentes tipos de simulação são encontrados, tais como emulação, Monte Carlo, *trace-driven*, *discret-event* e *execution-driven*.

Entretanto, é comum que modelos de simulação falhem, apresentando resultados incorretos. Alguns dos erros mais comuns são citados a seguir, e um maior detalhamento e alternativas para a resolução deles podem ser encontrados em [8]. São eles: nível de detalhes inapropriado, linguagem de programação inadequada, falta de verificação nos modelos, modelos sem a devida

validação, condições iniciais impróprias, simulações muito curtas, gerações fracas de números aleatórios e seleção imprória da semente inicial (primeiro valor aleatório gerado, fornecido pelo analista).

Em simulações, as aplicações paralelas são caracterizadas por eventos globais separados por computações locais. Eventos (computações) locais não interferem e não são interferidos por outros processos do sistema, diferentemente dos eventos globais. O acesso a memória compartilhada e computações para a tarefa de sincronização são algumas das operações de um evento global [9].

Neste contexto, simuladores surgem como ferramentas interessantes para auxiliar na avaliação de desempenho de sistemas reais. Existem diferentes ferramentas utilizadas para simular plataformas de computação distribuída, dentre as quais pode-se citar algumas mais notáveis, tais como *OptorSim* [11], *GridSim* [12], *MicroGrid* [13], e *SIMGRID* [14].

O primeiro simulador (*OptorSim*), é implementado em Java e trabalha com réplicas de *jobs*. Este simulador é utilizado principalmente para comparar diferentes estratégias de escalonamento com a utilização destas réplicas, simulando o comportamento de ambientes distribuídos conforme alguns parâmetros definidos (topologia de rede e conjunto de *jobs*, por exemplo).

Desenvolvida na Universidade da Califórnia, em São Diego (UCSD - *University of California at San Diego*), a ferramenta *MicroGrid*, na realidade, trata-se de um emulador. Ela permite a execução de aplicações construídas com o *Globus* [15], para que o comportamento destas aplicações seja conhecido. O *MicroGrid* emula uma grade *Globus*, e pode ser utilizado como uma ferramenta complementar para verificar resultados obtidos através de simulações com execuções reais das aplicações.

Outra ferramenta bastante difundida, trata-se do *GridSim*, que também simula ambientes de grades computacionais, monitorando seus recursos. Utiliza-se do pacote *SimJava* (simulação baseada em eventos), o que torna-o escalável e portável.

Finalmente, para este trabalho escolheu-se a ferramenta *SIMGRID*, que será descrita na Seção 3.1.

2.2 Modelagem Analítica

Um modelo analítico é uma representação matemática de um sistema computacional [16]. Nesta modelagem, métricas de desempenho da aplicação são geradas através de parâmetros do modelo realizado.

Este tipo de técnica vem sido cada vez mais utilizada para modelar sistemas paralelos, devido à facilidade de utilizá-las e ao seu baixo custo. Este custo, entretanto, trata-se tanto do custo com ferramentas para o desenvolvimento da própria modelagem quanto do custo ao final da análise dos resultados por ela gerados, pois o sistema ainda não foi implementado (diferentemente de medições, por exemplo). As técnicas baseadas em modelagem analítica abstraem

as características do sistema paralelo, modelando-as como um conjunto de parâmetros ou de funções parametrizadas [17].

A modelagem analítica pode ser dividida em três tipos: modelagem com parâmetros escalares, modelagem com funções e análise estatística [17].

Na primeira delas, utiliza-se um conjunto de parâmetros escalares que irão representar o comportamento do sistema paralelo sobre determinadas condições. A dificuldade deste modelo é a escolha dos parâmetros corretos e importantes. Uma escolha mal formulada diminuirá a precisão dos resultados do modelo. Por outro lado, estes parâmetros simplificam o comportamento do sistema, levando a naturais imprecisões.

O tipo de modelagem realizado com funções, por sua vez, utiliza-se de funções matemáticas ao invés de parâmetros em seus modelos. Tal abordagem pode se tornar bastante complexa por ter que determinar a forma e os coeficientes das funções. A modelagem baseada em parâmetros descrita previamente, trata-se de uma caso especial de modelagem através de funções, na qual as funções são constantes.

Finalmente, a abordagem através de análise estatística é usualmente utilizada para analisar sistemas paralelos quando as características da carga de trabalho são bem conhecidas. Nesta modelagem, o comportamento assintótico do sistema alvo é o que se deseja representar. O desenvolvimento destes modelos, no entanto, pode necessitar de um conhecimento mais específico sobre estatística.

Como exemplo de ferramentas que realizam tal tipo de análise, podemos citar:

- **Teoria de Filas:** modelos baseados em filas são úteis para a análise de sistemas nos quais conflitos ocorrem quando diversas entidades tentam acessar simultaneamente o mesmo recurso [18]. Existem dois tipos de rede de filas: rede de filas aberta e rede de filas fechada. Em uma rede aberta, existem chegadas externas de clientes ao sistema e saídas dos clientes em determinados pontos da rede. Neste tipo de modelagem, há uma variação na quantidade total de clientes. Por outro lado, em redes fechadas o serviço é realizado e o cliente continua no sistema, voltando para algum ponto do mesmo. Não há, portanto, chegadas de clientes externos e tampouco saída dos que se encontram no sistema.

- **Cadeias de Markov:** Conforme [2], uma Cadeia de Markov trata-se de um conjunto de estados e transições entre estes estados. Estas transições são modeladas por processos estocásticos, e possuem ou uma probabilidade associada (quando tratam-se de processos de tempo discreto) ou uma taxa (no caso de processos de tempo contínuo). Cadeias de Markov de tempo contínuo permitem que as transições possam ocorrer em qualquer instante de tempo, enquanto as de tempo discreto (como o próprio nome diz) ocorrem em pontos discretos de tempo.

- **Redes de Petri:** Rede de Petri é uma poderosa ferramenta para modelar e analisar concorrência e sincronização de sistemas paralelos, pois permite uma visualização das coopera-

ções entre as diversas entidades [18]. Tornou-se um formalismo famoso pela simplicidade de sua modelagem e pela capacidade de descrever sistemas extensos.

A ferramenta escolhida para a realização das modelagens neste trabalho foi Redes de Autômatos Estocásticos, que será descrita na Seção 3.2.

3 Ferramentas

Para a realização dos objetivos propostos neste trabalho, duas ferramentas foram utilizadas: o simulador SimGrid e a ferramenta de Modelagem Analítica SAN. Nesta seção, tais ferramentas serão apresentadas, ressaltando seus principais aspectos e características.

3.1 SimGrid

A primeira versão do simulador *SIMGRID* (chamada de *SIMGRID v1.0*) era uma versão mais simples, e está descrita em [14]. Foi desenvolvida por Henri Casanova, que modificou uma implementação previamente realizada, criando uma estrutura de simulação mais genérica e com uma API (*Application Programming Interface*) amigável. Após algum tempo, uma nova camada chamada *MSG* (*Meta-SIMGRID*) foi desenvolvida, acrescentando *threads* e também a idéia de processos sendo simulados independentemente. No final do ano de 2003, o *SIMGRID* passou a executar em sua versão *SIMGRID v2*. A partir da versão 3, outros diferentes módulos foram adicionados à implementação (alguns destes sendo tratados a seguir). O simulador encontra-se, ultimamente, em sua versão 3.2.

3.1.1 Componentes do SimGrid

O *SIMGRID* é uma das ferramentas mais conhecidas para a simulação de ambientes de programação distribuída, e neste estudo será utilizado para ambientes de grades computacionais. Alguns de seus principais componentes podem ser vistos na Figura 1, cada um com uma funcionalidade específica.

Figura 1 – Componentes do SimGrid.

O primeiro componente trata-se do *MSG* supracitado. É um ambiente de programação simples, incluído desde a segunda versão do simulador, e foi o primeiro ambiente de programação distribuída provido no *SIMGRID*. Neste componente, algumas funcionalidades básicas são oferecidas, tais como o gerenciamento de *hosts* e tarefas, e é normalmente utilizado para a simulação de aplicações distribuídas genéricas.

O próximo componente trata-se do *SMPI*, que se refere ao ambiente de programação para aplicações construídas com a utilização da biblioteca MPI. Aplicações implementadas em MPI serão executadas no simulador através deste módulo, sem que sua implementação tenha que ser modificada.

O componente *SURF* é responsável por providenciar todas as funcionalidades de simulação da plataforma virtual. É o kernel de todos os componentes do *SIMGRID*, tratando-se assim de um módulo de bastante baixo nível.

O último componente corresponde ao *GRAS* (*Grid Reality And Simulation*). Este módulo é utilizado por usuários que desejam desenvolver aplicações reais, ou seja, aplicações com a finalidade de que, após o término das simulações, o programa realmente venha a ser distribuído sem a necessidade de alterações no código.

As aplicações deste documento foram implementadas sobre os módulos MSG e GRAS, pois são os que provêm uma visão mais real do ambiente de plataformas distribuídas.

3.1.2 Aplicações e Arquivos de Configuração

As aplicações executadas no SimGrid são implementadas utilizando a linguagem de programação C. Tratam-se de programas parecidos com os usuais, porém, com algumas abstrações e funções que não realizam em si a tarefa, mas sim simulam o seu possível comportamento. Quando as execuções no simulador são realizadas, usualmente ao final é relatado o tempo total de simulação, e impressões na tela podem ser realizadas durante a simulação para mostrar os passos que estão sendo realizados pelos processos.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <host id="Nodo1" power="137333000"/>
  <host id="Nodo2" power="98095000"/>
  <link id="1" bandwidth="3430125" latency="2"/>

  <route src="Nodo1" dst="Nodo2"><link:ctn id="1"/></route>

  <route src="Nodo2" dst="Nodo1"><link:ctn id="1"/></route>

</platform>
```

Figura 2 – Exemplo de arquivo *platform.xml* do SimGrid.

O ponto mais importante para este estudo neste contexto, trata-se dos arquivos de configuração. Ao executar uma aplicação no SimGrid, dois arquivos XML [19] de configuração devem

ser informados para que a simulação seja feita de maneira correta. O primeiro deles chama-se *platform.xml*. Trata-se de um arquivo no qual são descritas informações sobre a topologia da rede. Todos os nodos presentes na simulação devem ser descritos neste arquivo, e para cada processo ainda deve ser adicionada a informação sobre o seu poder computacional. Além disto, informações como a latência do(s) link(s), largura de banda e a maneira pela qual os nodos são interligados devem ser descritas. A Figura 2 apresenta um exemplo deste arquivo.

O segundo arquivo trata-se do *deployment.xml*. É um arquivo no qual são descritas informações para a aplicação, tais como parâmetros de entrada e informações sobre a porta para comunicação com os demais processos. Para cada processo do sistema (descrito no arquivo *platform.xml*), deve haver no mínimo uma linha neste arquivo de *deployment*, fazendo referência a sua função no processo (funções estas definidas no código da aplicação). Estas funções podem ser se o processo é o mestre ou o escravo, se o processo é o cliente ou o servidor, se é o coordenador ou o subordinado, etc. A Figura 3 apresenta um exemplo de arquivo de *deployment* para o arquivo de *platform* citado acima.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">

  <process host="Nodo1" function="receiver">
     <argument value="4000"/>
     <argument value="1"/>
  </process>

  <process host="Nodo2" function="sender">
     <argument value="Nodo1:4000"/>
     <argument value="512"/>
  </process>

</platform>
```

Figura 3 – Exemplo de arquivo *deployment.xml* do SimGrid.

3.2 SAN

A modelagem utilizando Redes de Autômatos Estocásticos [1] data da década de 80, e surge como uma importante alternativa para a modelagem de sistemas paralelos. Seu nome original é *Stochastic Automata Network* (SAN), e trata-se de um formalismo baseado na teoria de Cadeias de Markov, expressando modelos Markovianos de maneira mais intuitiva e compacta. Através da modelagem com SAN, medidas de desempenho podem ser obtidas, tais como *throughput*, atraso de sincronização, tempo de resposta, dentre outras, mesmo antes da implementação da aplicação.

O princípio da utilização do formalismo SAN é traduzir (ou abstrair) um sistema real em sua totalidade em diversos módulos, de maneira independente, modelando diversos subsistemas. Estes subsistemas, obviamente, terão pontos de interação em alguns momentos para que

o objetivo final do conjunto como um todo seja alcançado. Entretanto, para a obtenção das medidas de desempenho de um sistema, é necessário que o modelo criado seja resolvido. Nesta seção, a maneira de construção de modelos SAN é detalhada, visando gerar as estimativas de desempenho do sistema real.

Neste documento, será apresentada uma visão mais intuitiva da maneira pela qual os modelos SAN são construídos, indicando para considerações e explicações mais formais a leitura da referência [1].

3.2.1 Descrição do Formalismo

Conforme citado anteriormente, o formalismo de Redes de Autômatos Estocásticos representa o sistema do qual se deseja obter estimativas através de módulos, abstraindo o sistema real em diferentes subsistemas. Estes módulos (que são definidos na forma de autômatos estocásticos) são formados basicamente por três componentes: **estados**, **transições** e **eventos**. Na Figura 4 é apresentado um primeiro exemplo de um modelo SAN com 3 autômatos.

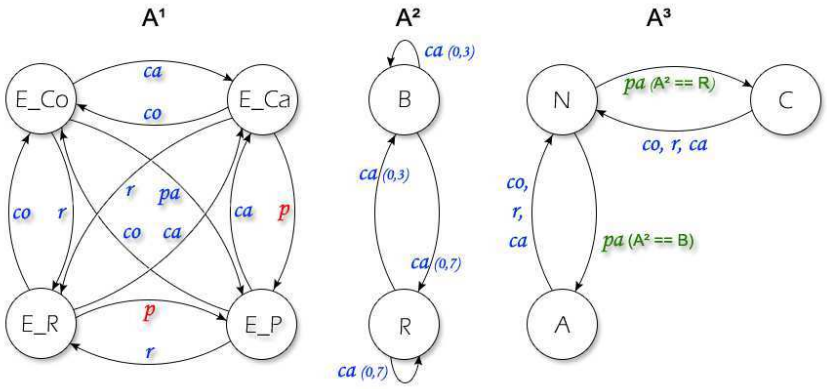

Figura 4 – Exemplo de uma Rede de Autômatos Estocásticos.

Um **estado** na modelagem SAN é representado graficamente por um círculo. Cada um destes refere-se a um determinado comportamento apresentado pelo sistema, e o conjunto deles representa o conjunto destes comportamentos. Quando o sistema altera seu comportamento, seu estado é alterado. Tal situação (mudança de um estado para outro) é chamada de **transição**, e é representada por um arco ligando os estados origem e destino (note que se trata de um *grafo dirigido*, pois as arestas possuem setas que indicam sua direção). Entretanto, para que uma mudança de estado aconteça, um **evento** deve ocorrer. Assim, quando um evento do sistema ocorre, o modelo altera seu estado, realizando transições de acordo com as probabilidades associadas a cada um destes eventos. A ocorrência destes eventos depende de **taxas** associadas a cada um deles.

No exemplo da Figura 4, percebe-se a existência de três autômatos: A^1 (com 4 estados),

A^2 (com 2 estados) e A^3 (com 3 estados). Além disso, o modelo SAN apresentado possui 5 eventos, que são listados na tabela 1. Em um modelo SAN podem ser diferenciados dois tipos de eventos: **locais** e **sincronizantes**. Os eventos *locais* alteram somente o estado interno de um autômato, não interferindo no estado que os outros autômatos encontram-se. Por exemplo, o evento p do autômato A^1 apenas modifica-o internamente, não influenciando em nada os autômatos A^2 e A^3 (note que eventos locais apenas aparecem em um único autômato). Por outro lado, eventos ditos *sincronizantes* apresentam a característica de não apenas alterar o seu estado, mas também de alterar o estado dos outros autômatos do modelo. São as referidas interações entre os diferentes subsistemas. Os eventos *ca*, *r*, *co* e *pa* são sincronizantes no exemplo da Figura 4, pois quando estes eventos ocorrem, mais de um autômato altera seu estado (perceba que os eventos sincronizantes aparecem em dois ou mais autômatos).

Tabela 1 – Eventos da Modelagem SAN para o Exemplo 1

Tipo	**Evento**	**Taxa**
local	*p*	$\tau 1$
sincronizante	*ca*	$\tau 2$
sincronizante	*r*	$\tau 3$
sincronizante	*pa*	$\tau 4$
sincronizante	*co*	$\tau 5$

Para alguns eventos, uma probabilidade é associada. No exemplo apresentado, nota-se que do estado B do autômato A^2, na ocorrência do evento *ca*, há uma probabilidade de 70% de o autômato alcançar o estado R, e uma probabilidade de 30% de continuar no estado B. Além disto, funções podem ser definidas para os eventos, como acontece no autômato A^3. O evento *pa* neste autômato, depende do estado do autômato A^2. Caso A^2 encontre-se em B, o evento *pa* levará A^3 do estado N para o estado A. Caso contrário (A^2 em R), o evento *pa* acarreta na transição do estado N para o estado C.

Dois tipos de análises podem ser realizadas ao resolver um modelo SAN: estacionária e transiente. Na análise estacionária, os resultados esperados são as probabilidades de permanência em cada estado (ou de combinações de estados desejados pelo usuário). Com isto, pode ser realizada uma análise do comportamento da aplicação modelada com as taxas e probabilidades definidas no modelo, possibilitando que alterações nestes valores sejam realizados, para melhorar o desempenho, diminuir gargalos, etc. Já na análise transiente, é possível estimar o tempo médio de execução da aplicação modelada. Este estudo dedicou-se a utilizar **análise transiente**, para realizar comparações nos tempos de execução previstos no simulador SimGrid e na modelagem SAN.

3.2.2 Parametrização de Modelos SAN

Para completar a criação de modelos SAN, ainda restam as definições das taxas associadas a cada evento. As taxas de um evento representam a frequência com que este evento acontece a cada unidade de tempo. Para exemplificar, a Figura 5 representa uma possível modelagem para a requisição de uma tarefa. Neste simples modelo, o autômato *A* envia uma mensagem para o autômato *B*, que realiza um processamento sobre esta mensagem e retorna o resultado para o autômato *A*.

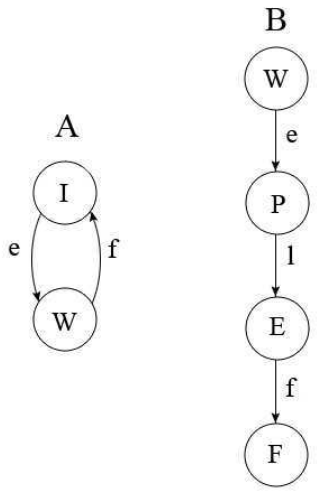

Figura 5 – Simples exemplo de uma Rede de Autômatos Estocásticos.

O estado *I* do autômato *A* representa o estado no qual este autômato está parado. Na ocorrência do evento sincronizante *e*, o autômato *A* passa para o estado *W* (aguardando) e o autômato *B* passa do seu estado *W* para o estado *P* (processando). Quando o autômato *B* termina o processamento, ocorre o evento local *l*, que faz com que este autômato passe para o estado *E*, no qual enviará a mensagem. Finalmente, na ocorrência do evento sincronizante *f* (representando o envio da mensagem de *B* para *A*), o autômato *B* vai para o estado *F* (final), enquanto o autômato *A* volta para o estado *I*.

Supondo que o tempo de envio da mensagem do autômato *A* para o autômato *B* (evento *e*) seja de 20 segundos, o tempo de processamento do autômato *B* (evento *l*) seja 37 segundos e o tempo de envio do autômato *B* para o autômato *A* (evento *f*) também seja de 20 segundos, e que a unidade de tempo é em segundos, tem-se:

- **taxa do evento e:** a cada unidade de tempo (1 segundo), este evento ocorre 1/20 vezes, ou seja, a taxa deste evento será de 0,05;
- **taxa do evento l:** a cada unidade de tempo (1 segundo), este evento ocorre 1/37 vezes, ou seja, a taxa deste evento será de 0,027;

- **taxa do evento f:** a cada unidade de tempo (1 segundo), este evento ocorre 1/20 vezes, ou seja, a taxa deste evento será de 0,05.

Desta maneira, calcula-se as taxas para os eventos, baseando-se no tempo em que leva para executar. Para encontrar este tempo, o usuário pode realizar medições (por exemplo, através de um programa que colete o tempo de envio da mensagem), realizar cálculos (através da utilização de valores encontrados em manuais da rede, por exemplo) ou até mesmo empiricamente. Cabe ressaltar que a obtenção dos valores corretos é de extrema importância na fase de modelagem, para representar da maneira mais fiel possível a aplicação.

Além destas taxas, existem ainda as chamadas taxas funcionais. Elas possuem este nome por serem executadas em função de alguma condição. Por exemplo, se o evento *l* (do autômato *B*) possuísse uma taxa funcional, ela poderia ser dada por:

- **taxa funcional do evento l:** se o autômato *A* estiver no estado *W*, então a taxa será de 1/37;

Como pode ser visto, o evento *l* ocorre em função do estado do autômato *A*.

3.2.3 PEPS

Para a execução dos modelos SAN, utilizou-se a ferramenta PEPS (*Performance Evaluation of Parallel Systems*) [20]. Nesta seção, serão abordados de maneira simples e intuitiva alguns aspectos relativos a sintaxe da ferramenta, que serão utilizados no decorrer do trabalho.

O arquivo de entrada do PEPS tem a extensão san (*nome_do_arquivo.san*). Neste arquivo, são descritos os autômatos, os estados, transições, eventos, taxas e todos os demais aspectos referentes ao modelo SAN. Abaixo, encontram-se algumas estruturas e expressões utilizadas neste documento:

- **st AUT == EST**, onde *AUT* é o nome do autômato e *EST* é o estado do autômato. Logo, significa que o autômato *AUT* deve estar no estado *EST*. Esta estrutura é utilizada para definições de taxas funcionais e estados inicial e final;
- **st AUT != EST**, significando que o autômato *AUT* não deve estar no estado *EST*;
- **st AUT1 == EST1 && st AUT2 != AUT1**, onde a estrutura && significa que as duas expressões devem ser verdadeiras;
- **nb[AUT(1)..AUT(i)] EST == N**, onde *N* é um número inteiro. Esta estrutura representa que o número de autômatos *AUT(i)* que encontram-se no estado *EST* deve ser igual a *N*.

Todas estas expressões quando são utilizadas resultam em verdadeiro ou falso (1 ou 0, respectivamente). Logo, são utilizadas para definições de taxas funcionais, como no exemplo abaixo:

- **(st AUT == EST) * 2**, significando que caso o autômato *AUT* encontre-se no estado *EST* (resultando no valor 1), a taxa associada ao evento será 1 * 2 (ou seja, 2). Caso contrário, se a expressão retornar valor 0, o evento não ocorre, pois terá sua taxa alterada para o valor 0.

3.2.4 Análise Transiente

Conforme relatado anteriormente, a análise transiente foi utilizada neste trabalho para realizar a obtenção de tempos de execução através da modelagem SAN. Nesta seção, será descrito o procedimento para a realização da análise transiente.

O primeiro passo para a realização da análise transiente é a definição de um **estado inicial** e de um **estado final** para o modelo SAN. Por exemplo, no modelo SAN da Figura 5, pode-se determinar - na sintaxe da ferramenta PEPS (Seção 3.2.3) - os estados inicial e final como seguem:

- **estado inicial:** *(st A == I) && (st B == W)*;
- **estado final:** *(st A == I) && (st B == F)*.

Ou seja, o estado inicial acontece quando o autômato *A* encontra-se no estado *I* e o autômato *B* encontra-se no estado *W*, enquanto no estado final o autômato *A* encontra-se no estado *I* e o autômato *B* no estado *F*.

A ferramenta PEPS possui métodos iterativos para a resolução de sistemas de equações. Assim sendo, na execução do modelo no PEPS, o usuário informa um tempo para que seu modelo seja executado. Ao final da execução para este tempo determinado, obtem-se como resultado um vetor de probabilidades com todos os estados do modelo SAN. Analisa-se, então, se a probabilidade do modelo estar no estado final encontra-se em 99%. Caso isto não ocorra, este vetor de probabilidades é recarregado pelo PEPS (para continuar da iteração anterior), e é realizada uma nova execução. Este procedimento é repetido até que o a probabilidade do estado final seja de 99%.

Cada uma destas execuções parciais (antes de chegar ao critério de parada de 99%) gera uma determinada probabilidade para o estado final do modelo. Devido ao fato de que a cada execução tem um tempo associado, tem-se que o tempo parcial de cada execução é dado por: **probabilidade parcial do estado final * tempo informado para execução = tempo parcial**.

Tabela 2 – Exemplo de cálculo de tempo com análise transiente.

Tempo do método transiente	**Probabilidade**	**Parcial (p)**
1	*0,15*	τ*(0,15 - 0) * 1 = 0,15*
2	*0,25*	τ*(0,25 - 0,15) * 2 = 0,2*
3	*0,36*	τ*(0,36 - 0,25) * 3 = 0,33*
4	*0,48*	τ*(0,48 - 0,36) * 4 = 0,48*
5	*0,67*	τ*(0,67 - 0,48) * 5 = 0,95*
6	*0,79*	τ*(0,79 - 0,67) * 6 = 0,72*
7	*0,88*	τ*(0,88 - 0,79) * 7 = 0,63*
8	*0,92*	τ*(0,92 - 0,88) * 8 = 0,32*
9	*0,99*	τ*(0,99 - 0,92) * 9 = 0,63*
-	-	$\tau Total = \sum_1^9 p = 4,41$

Entretanto, para cada nova iteração, a probabilidade da execução anterior já foi calculada. Logo, antes do cálculo do tempo parcial, é necessário descontar da probabilidade atual, a probabilidade anterior, como segue: **[(probabilidade parcial)*i* - (probabilidade parcial)*i-1*] * tempo = tempo parcial**.

Assim, com todas as probabilidades parciais conhecidas (desde a primeira até a última execução), basta realizar o somatório destes valores e obter-se-á o valor do tempo para o modelo SAN. Na Tabela 2 é apresentado um exemplo simbólico de cálculo do tempo com a análise transiente.

4 Mapeamento de Parâmetros

Para avaliar e comparar os resultados obtidos através da simulação e da modelagem analítica, algumas aplicações foram escolhidas, modeladas e executadas em suas respectivas ferramentas (previamente citadas neste documento). Antes de apresentar as aplicações, algumas definições serão realizadas, objetivando formalizar e unificar o entendimento do processo realizado.

4.1 Níveis: Middleware x Aplicação

Conforme descrito anteriormente, um modelo SAN necessita de **Estados**, **Transições**, **Eventos** e **Taxas**. Entretanto, nem todos estes parâmetros estão relacionados diretamente com a aplicação. Alguns deles estão ligados a variáveis provenientes do ambiente de execução (podendo também ser chamado de *middleware*, por ser um mediador entre a aplicação e a infra-estrutura). Esta seção tem por objetivo deixar clara a diferença entre os parâmetros que vêm da aplicação e os parâmetros oriundos do *middleware*, relacionando-os com suas respectivas funções nos modelos SAN que serão construídos.

O primeiro nível a ser discutido trata-se do nível de aplicação. Quando o usuário depara-se com uma situação na qual necessita modelar determinada aplicação, este é o primeiro nível a ser analisado. Nele, os principais pontos a serem observados estão relacionados ao comportamento da aplicação, que será útil para definir: em quais subsistemas a aplicação pode ser abstraída; as interações entre estes subsistemas; quando a aplicação tem seu estado global alterado; e qual o evento causador desta alteração de estado. Note que, com estas observações, já é possível realizar a criação dos autômatos, a definição dos estados de cada um deles e os eventos de transição entre os estados.

No outro nível (*middleware*) encontram-se as variáveis de ambiente. Na modelagem de sistemas para computação distribuída, comumente as transições entre os estados do modelo acontecem devido à ocorrência de eventos de comunicação entre os processos ou de término de processamento. As taxas dos eventos normalmente estão relacionadas a fatores externos à aplicação. Estes fatores são as variáveis de ambiente, que podem ser, por exemplo, o poder de processamento dos nodos, o tempo de realização de uma determinada tarefa, o tempo de transmissão de dados entre os processos pela rede, a latência dos links, dentre outros. Tais variáveis estão relacionadas à infra-estrutura do ambiente, e são utilizadas em conjunto para o

cálculo das taxas dos eventos do modelo (podendo ser utilizadas inclusive em conjunto com variáveis da aplicação).

Com isto, tem-se uma separação clara dos aspectos que influenciam o modelo provenientes da aplicação daqueles obtidos através das características do ambiente (*middleware*). Existem, ainda, os parâmetros de entrada da aplicação. Estes parâmetros não podem ser enquadrados genericamente em nenhum dos dois níveis supracitados, pois dependendo do tipo de parâmetro que a aplicação recebe como entrada, sua consequência pode tanto refletir na alteração de algum autômato no sistema quanto na alteração de alguma(s) taxa(s) de determinado(s) evento(s). Por exemplo, caso um parâmetro de entrada seja o número de tarefas a serem executadas, tal informação pode tanto alterar um autômato que representa uma fila de tarefas como também pode alterar a taxa de algum evento que eventualmente seja calculado em função deste número. Em um possível caso, estas duas alterações podem ocorrer simultaneamente no mesmo modelo.

4.2 SimGrid x SAN: Definições

Nas execuções realizadas no SIMGRID, os parâmetros das aplicações (definidos nos arquivos XML de entrada) podem ser descritos pela quíntupla **SG** = (φ, τ, β, θ, γ), na qual

- φ são os parâmetros da aplicação, tratando-se de um conjunto de entradas;
- τ é a topologia, que representa a configuração de máquinas e links na grade, modelada e a maneira como estão interligados;
- $\beta(lk)$ é a largura de banda de cada *link lk* presente na topologia τ;
- $\theta(lk)$ é a latência de cada *link lk* presente na topologia τ;
- $\gamma(p)$ é o poder computacional de cada máquina *p* presente na topologia τ.

Os dois arquivos XML criados pelo usuário são responsáveis pela definição destes parâmetros no SimGrid: *deployment.xml* e *platform.xml*. O arquivo de *deployment* descreve os parâmetros da aplicação (φ), definindo as funções de cada máquina (mestre ou escravo, qual máquina cria o *token*, etc.) e os valores de entrada (quando houver). Os demais termos (τ, β, θ e γ) são descritos no arquivo *platform.xml*.

A modelagem SAN das aplicações, por sua vez, pode ser definida pela tripla **SN** = (*G, E, T*), na qual

- *G* é um conjunto de grafos dirigidos, cada um deles composto por estados e arestas (transições);
- *E* é o conjunto de eventos que permite as transições de estados;

- *T* é o conjunto de taxas e probabilidades associadas a cada evento.

Todos os termos de SN são definidos pelo usuário no arquivo de entrada do PEPS (com extensão *.san*). Os termos *G* e *E* tratam-se da representação da topologia τ (da quíntupla SG) de acordo com as funcionalidades e a maneira pela qual a aplicação foi implementada. O termo *T* é obtido através de cálculos realizados sobre os termos φ, β, θ e γ de SG, de acordo com a necessidade do modelo SAN da aplicação.

4.3 Definindo Tempos com o SIMGRID

A comunicação entre os nodos denota um certo tempo, e este tempo depende da maneira pela qual os nodos estão interligados. Para parametrizar os modelos SAN de acordo com os parâmetros do SIMGRID, então, é necessário saber como o simulador calcula estes tempos de comunicação. Assim sendo, a fórmula da Equação 4.1 apresenta o cálculo do tempo de comunicação realizado pelo SIMGRID:

$$\textbf{\textit{TempoComunicacao}} = \theta + (\textit{TamC}/\textit{LBE}) \tag{4.1}$$

onde θ é a latência, *TamC* é o tamanho (em bytes) da tarefa e *LBE* é a *Largura de Banda Efetiva*. A fórmula da *LBE* é ilustrada na Equação 4.2:

$$\textbf{\textit{LBE}} = min(\beta, (\textit{W}/2 * \theta)) \tag{4.2}$$

onde *W* é a janela de congestionamento do protocolo TCP. Ou seja, *LBE* e o mínimo entre a largura de banda (β) e *W/2*θ*. O parâmetro *W* não se encontra na quíntupla SG, mas na implementação atual do SIMGRID este valor é fixado em 20000 *bytes*. Assim sendo, obtêm-se a *LBE* através da seguinte fórmula:

$$\textbf{\textit{LBE}} = min(\beta, (20000/2 * \theta)) \tag{4.3}$$

Perceba que, agora, a combinação dos termos da quíntupla permitem o cálculo do tempo de comunicação entre dois nodos.

Em alguns casos pode ser necessário conhecer o tempo de processamento dos nodos. Nestes casos, utiliza-se a fórmula:

$$\textbf{\textit{TempoProcessamento}} = \textit{TCT}/\gamma \tag{4.4}$$

O parâmetro γ é conhecido e o *Tamanho Computacional da Tarefa (TCT)* (em *flops*/s) deve ser definido pelo usuário.

5 Estudos de Caso

Para realizar experimentações sobre o mapeamento de parâmetros e realizar comparações e análises sobre a Simulação e a Modelagem Analítica, algumas aplicações foram escolhidas e modeladas nas duas abordagens. Nesta seção, três destas aplicações são descritas, detalhando seu funcionamento e a maneira de obtenção dos parâmetros.

5.1 Estudo de Caso 1: Token Ring

A primeira aplicação apresentada trata-se da aplicação *Token Ring*. Este tipo de aplicação é bastante utilizado em situações nas quais necessita-se garantir acesso único a certos recursos disponíveis (por este fato, também é conhecida como *Exclusão Mútua*). A idéia básica é de que os recursos ditos de acesso exclusivo só podem ser acessados pelo nodo que obtiver permissão para tanto. Esta permissão trata-se do *token*, que nada mais é do que uma mensagem que avisa o nodo que ele pode acessar a região crítica. Quando o nodo termina seu acesso, repassa o *token* para a próxima máquina da topologia, formando uma espécie de anel. O funcionamento da aplicação é simples: um nodo da grade (topologia) inicia a execução e cria o *token*. Este nodo repassa o *token* para seu sucessor, e assim sucessivamente, até que o *token* chegue novamente ao primeiro nodo da topologia. O critério de parada da aplicação é definido pelo usuário, e é o número de voltas que o *token* realiza na topologia inteira.

5.1.1 Modelagem no SimGrid

Conforme dito anteriormente, os parâmetros do simulador são definidos pela quíntupla SG. Logo, têm-se que para a aplicação *Token Ring*:

- φ: o parâmetro que o usuário deve informar é unicamente o critério de parada, ou seja, quantas passadas o *token* deve realizar por toda a topologia.
- τ: a topologia da grade é ilustrada na Figura 6, e é composta por um determinado número de nodos interligados um a um por um *link* unidirecional, ou seja, o Nodo 1 comunica-se através do *Link* 2 com o Nodo 2, mas o contrário não é verdade.
- $\beta(lk)$, $\theta(lk)$ e $\gamma(p)$, serão definidos especificamente para cada experimento realizado no

Capítulo 6.

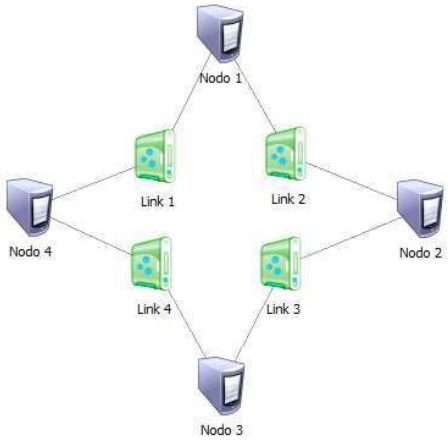

Figura 6 – Topologia exemplo para a aplicação Token Ring.

Um exemplo dos arquivos de *platform* e *deployment* para esta aplicação são apresentados nas Figuras 7 e 8.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <host id="Tremblay" power="137333000"/>
  <host id="Fafard" power="98095000"/>

  <link id="1" bandwidth="3430125" latency="1.5"/>

  <route src="Tremblay" dst="Fafard"><link:ctn id="1"/></route>
  <route src="Fafard" dst="Tremblay"><link:ctn id="1"/></route>
</platform>
```

Figura 7 – Exemplo de arquivo *platform.xml* para a aplicação Token Ring.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">

  <process host="Tremblay" function="node">
    <argument value="4000"/>     <!-- port to listen -->
    <argument value="Fafard"/>   <!-- peer (successor) host -->
    <argument value="4000"/>     <!-- peer port -->
  </process>

  <process host="Fafard" function="node">
    <argument value="4000"/>     <!-- port to listen -->
    <argument value="Tremblay"/> <!-- peer (successor) host -->
    <argument value="4000"/>     <!-- peer port -->
    <argument value="--create-token"/>   <!-- I'm first client -->
  </process>

</platform>
```

Figura 8 – Exemplo de arquivo *deployment.xml* para a aplicação Token Ring.

5.1.2 Modelagem SAN

A Figura 9 mostra o modelo SAN da aplicação *Token Ring*. O grafo dirigido representa o termo *G* da tripla SN, e os eventos e suas respectivas taxas completam a definição de SN.

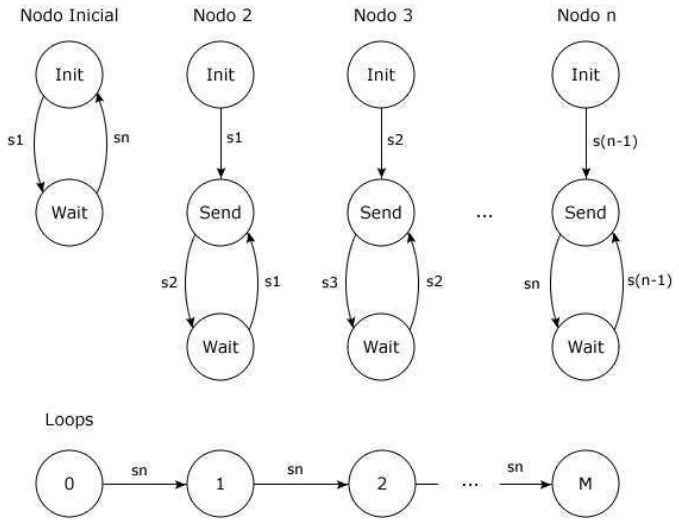

Figura 9 – Modelo SAN para a aplicação Token Ring.

A aplicação foi modelada com 3 diferentes tipos de autômatos. O primeiro deles trata-se do autômato **NodoInicial**, que possui os estados *Init* e *Wait*. Este autômato é aquele nodo que criará o *token*. O evento sincronizante *s1* representa o envio do *token* para seu sucessor, alterando seu estado para o estado *Wait*, que representa que o nodo está aguardando o seu predecessor (o último nodo da topologia) lhe devolver o *token*. Quando isto acontece, o autômato volta para o estado *Init* através do evento sincronizante *sn*, concretizando uma volta realizada. O critério de parada é controlado pelo autômato **Loops**, que a cada volta executada pelo *token* incrementa seu estado, até que este autômato alcance o estado *M* (*M* = número de voltas a serem realizadas). O terceiro tipo de autômato modela os nodos intermediários da topologia (autômato **Nodo**). Possui 3 estados (*Init*, *Send* e *Wait*), e a idéia é similar àquela do autômato NodoInicial. Os eventos *s(n-1)* representam a chegada do *token* e os eventos *sn* representam o envio do *token* para o seu sucessor.

Para continuar a modelagem, existe ainda a definição dos estados inicial e final. O **estado inicial** é quando todos os autômatos encontram-se no estado *Init* e o autômato *Loops* no estado *0*. O **estado final** é alcançado quando o autômato *Loops* está em *M*, o autômato NodoInicial está em *Init* e os demais autômatos encontram-se em *Wait*. Isto pode ser descrito como segue (onde N é o número de nodos intermediários):

- **estado inicial:** *(st NodoInicial == Init) && (nb[Nodo0..Nodo(N-1)] Init == N) && (st Loops == 0)*

- **estado final:** *(st NodoInicial == Init) && ((nb[Nodo0..Nodo(N-1)] Wait == N)) && (st Loops == M)*

5.1.3 Mapeamento dos Parâmetros

A seguir, serão apresentadas as maneiras como as taxas do modelo SAN foram calculadas através do mapeamento das informações obtidas através do SimGrid.

Como o tamanho da mensagem é 1 (pois trata-se apenas de um *byte* indicando o *token*), têm-se que

$$\textbf{\textit{TempoComunicacao}} = \theta + (\textbf{\textit{1}}/(min(\beta, 20000/2*\theta))) \quad (5.1)$$

e os eventos *s[2..n]* são calculados com a utilização desta fórmula. O evento *s1* também, porém, este evento possui uma taxa funcional que depende do autômato *Loops*, ou seja, o evento *s1* só ocorre se o critério de parada (autômato *Loops* estar no estado *M*) ainda não foi atingido. Com tais informações, a Tabela 3 apresenta os eventos, com suas respectivas taxas associadas.

Tabela 3 – Eventos do modelo SAN da aplicação Token Ring.

Evento	**Taxa**
s1	(st Loops != M) * (1/*TempoComunicacao*)
s[2..n]	1/*TempoComunicação*

5.2 Estudo de Caso 2: Exclusão Mútua Com Coordenador Centralizado (EMCC)

Esta aplicação representa um sistema no qual existe um recurso compartilhado e alguns clientes querendo acessá-lo simultaneamente. Trata-se de um recurso crítico, logo, os acessos a ele devem ser realizados de maneira exclusiva. Para realizar o controle desta região crítica, garantindo o acesso exclusivo ao recurso, existe um processo que é o processo coordenador. Todos os clientes que desejam acessar o recurso devem, primeiramente, enviar uma requisição para o coordenador, que verificará se o acesso é possível ou não. Caso o acesso seja possível, o cliente é liberado para utilizar o recurso. Caso contrário, o coordenador insere o cliente em uma fila de espera, até que o recurso esteja liberado novamente, assim garantindo o acesso exclusivo. Para esta aplicação, o critério de parada é o número de requisições realizadas por cliente.

5.2.1 Modelagem no SimGrid

Através da utilização da quíntupla SG, os parâmetros da aplicação EMCC são dados por:

- φ: o parâmetro que o usuário deve informar é unicamente o critério de parada, ou seja, quantas requisições os clientes desejam realizar ao recurso exclusivo.
- τ: a topologia da grade é ilustrada na Figura 10, e é composta por um conjunto de processos clientes, que se encontram interconectados com o coordenador através de um *link* bi-direcional.
- $\beta(lk)$, $\theta(lk)$ e $\gamma(p)$, serão definidos especificamente para cada experimento realizado no Capítulo 6.

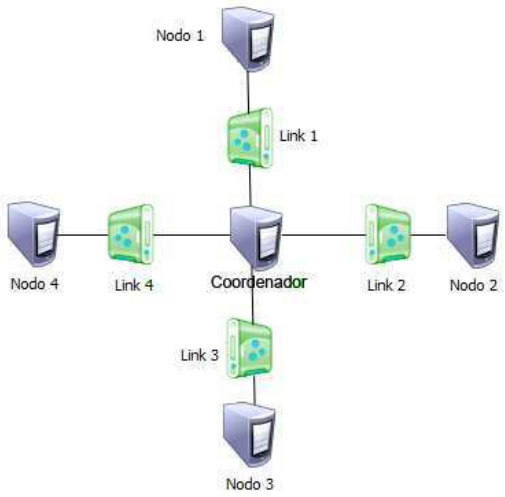

Figura 10 – Topologia exemplo para a aplicação EMCC.

Ainda relacionado aos parâmetros e a modelagem no SimGrid, as Figuras 11 a 12 exemplificam os arquivos de configuração *deployment.xml* e *platform.xml* para a aplicação Exclusão Mútua com Coordenador Centralizado.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">

  <process host="Tremblay" function="server">
    <argument value="4000"/>     <!-- port -->
  </process>

  <process host="Fafard" function="client">
    <argument value="Tremblay"/>   <!-- peer (successor) host -->
    <argument value="4000"/>     <!-- server port -->
  </process>

</platform>
```

Figura 11 – Exemplo de arquivo *deployment.xml* para a aplicação EMCC.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <host id="Tremblay" power="137333000"/>
  <host id="Fafard" power="98095000"/>

  <link id="1" bandwidth="3430125" latency="1.5"/>

  <route src="Tremblay" dst="Fafard"><link:ctn id="1"/></route>
  <route src="Fafard" dst="Tremblay"><link:ctn id="1"/></route>
</platform>
```

Figura 12 – Exemplo de arquivo *platform.xml* para a aplicação EMCC.

5.2.2 Modelagem SAN

Para a aplicação Exclusão Mútua com Coordenador Centralizado, o modelo SAN criado possui 4 autômatos: Coordenador, Cliente, Recurso e Requisições. O referido modelo SAN pode ser visualizado na Figura 13.

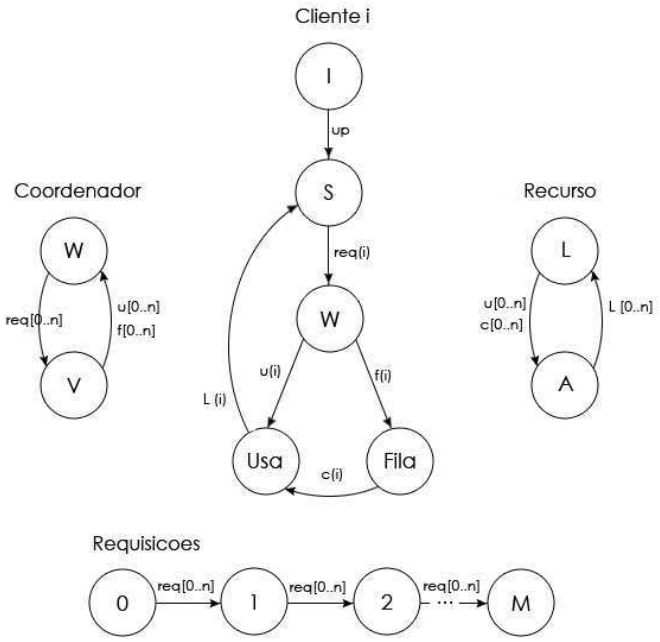

Figura 13 – Modelo SAN para a aplicação EMCC.

O primeiro autômato (**Coordenador**) representa exatamente o coordenador do sistema, que permite ou não os acessos à seção crítica. Este autômato possui 2 estados, *W* e *V*. Estar no estado *W* representa que o Coordenador está esperando alguma requisição de um Cliente, enquanto estar no estado *V* significa que esta entidade está verificando uma requisição enviada, se esta deve ser encaminhada para a fila ou poderá ter sua liberação para utilização do recurso.

No autômato **Cliente**, o estado *I* representa que ele ainda não foi inicializado. Desta maneira, quando o evento sincronizante *up* ocorrer, os clientes passarão para o estado *S*, que representa um estado em que o Cliente encontra-se ocioso. A partir deste estado, o Cliente pode

realizar uma requisição para o coordenador, que é acionada através do evento sincronizante *req*. Como existem *n* eventos *req* (onde *n* é o número de Clientes), somente um Cliente poderá ter sua requisição verificada por vez. Caso a requisição seja aceita pelo Coordenador, ocorrerá então o evento *u*, que representa a liberação do Coordenador para aquele Cliente. Este Cliente, então, terá seu estado alterado para *USA* e, ao término da utilização, ocorre o evento *l*, que libera o recurso do Cliente que estava utilizando. Caso contrário (Coordenador não libera utilização), acontece o evento *f*, que enfileira o pedido do Cliente, levando-o para o estado *FILA*. Neste estado, o cliente fará uso do recurso quando da ocorrência do evento *c*, que libera ele para utilizar o recurso. Tanto o evento *f* (que enfileira o cliente) como os eventos *u* e *c* (que liberam o acesso ao recurso), são relacionados com o estado do próximo autômato, o autômato Recurso.

No autômato **Recurso**, são 2 os estados possíveis: *A* (alocado) ou *L* (liberado). Quando um evento de acesso ao recurso ocorre (*u* ou *c*), o recurso passa automaticamente para o estado *A*. Deste estado, somente poder-se-á mudar para o estado *L* quando ocorrer o evento de liberação da seção crítica, ou seja, *l*.

Por fim, o autômato **Requisições** trata-se de uma fila de estados, variando de *0* até *M*, onde *M* é o número de clientes multiplicado pelo número de requisições que cada Cliente realiza (parâmetro de entrada da aplicação). Assim, este autômato é responsável por controlar o critério de parada, que é alcançado quando o número de requisições for atendido. A cada nova requisição (evento *req*), o autômato Requisições aumenta em 1 o seu estado.

A definição dos estados inicial e final é dada pela seguinte estrutura:

- **estado inicial:** *(st Recurso == L) && (st Requisições == 0) && (nb[Cliente0..Cliente(N-1)] I == N) && (st Servidor == S)*
- **estado final:** *(st Recurso == L) && (st Requisições == M)*

5.2.3 Mapeamento dos Parâmetros

Para o cálculo das taxas, a fórmula abaixo representa o tempo de comunicação entre os processos, uma vez que neste exemplo também a mensagem tem tamanho 1, por ser somente uma mensagem de controle de acesso liberado ou negado.

$$\textbf{\textit{TempoComunicacao}} = \theta + (\textbf{1}/(min(\beta, 20000/2*\theta))) \tag{5.2}$$

Como pôde ser percebido no modelo SAN anteriormente detalhado, 6 são os eventos presentes na modelagem. A Tabela 4 apresenta estes eventos, descrevendo suas respectivas taxas.

Tabela 4 – Eventos do modelo SAN da aplicação EMCC.

Evento	Taxa
up	(nb[Cliente0..Cliente(N-1)] I == N)
req	(st Requisicoes != M) * (1/*TempoComunicacao*)
u	(st Recurso == L) * (1/*TempoComunicacao*)
f	(st Recurso == A) * (1/*TempoComunicacao*)
l	1/*TempoComunicacao*
c	(st Recurso == L) * (1/*TempoComunicacao*)

5.3 Estudo de Caso 3: Mestre/Escravo

A última aplicação trata-se de um clássico da programação paralela e distribuída. *Mestre/Escravo* é um programa no qual um nodo centralizador (chamado Mestre) envia tarefas para os demais nodos da topologia (chamados Escravos). O Mestre é quem controla a divisão das tarefas e a quantidade de tarefas disponíveis no *buffer*. Enquanto ainda houver tarefas, o Mestre envia para os Escravos, que processarão esta tarefa e aguardarão o envio de uma nova. A aplicação chega ao seu final quando todas as tarefas designadas pelo Mestre tiverem sido processadas.

5.3.1 Modelagem no SimGrid

Na definição dos parâmetros do SimGrid para a aplicação Mestre/Escravo, temos a seguinte configuração:

- φ: os parâmetros que o usuário define são 3: o número de tarefas, o tamanho computacional de cada tarefa e o tamanho de comunicação das tarefas.
- τ: a topologia da grade é ilustrada na Figura 14, e é composta por um conjunto de processos escravos que se conectam (em um *link* bidirecional) com um processo Mestre.
- $\beta(lk)$, $\theta(lk)$ e $\gamma(p)$, serão definidos especificamente para cada experimento realizado no Capítulo 6.

Para a execução da aplicação Mestre/Escravo no simulador, os arquivos de configuração *deployment.xml* e *platform.xml* são estruturados conforme mostram as Figuras 15 e 16.

5.3.2 Modelagem SAN

O modelo SAN para a aplicação Mestre/Escravo pode ser visto na Figura 17. O modelo criado possui 3 autômatos: um representando o Mestre e outro representando dois tipos diferentes de Escravos (*A* e *B*, que serão explicados em seguida). O autômato **Mestre** é responsável por

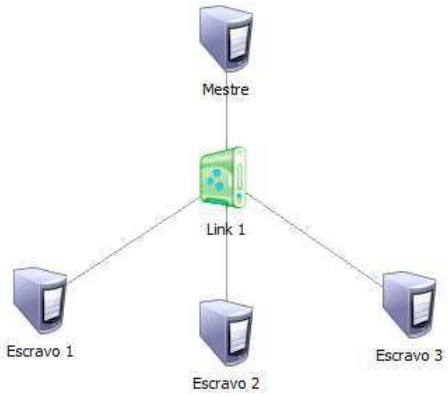

Figura 14 – Topologia exemplo para a aplicação Mestre/Escravo.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <!-- The master process (with some arguments) -->
  <process host="Mestre" function="master">
     <argument value="10"/>        <!-- Number of tasks -->
     <argument value="500000000"/> <!-- Computation size of tasks -->
     <argument value="100"/>       <!-- Communicat. size of tasks -->
     <argument value="Escravo1"/>  <!-- First slave -->
     <argument value="Escravo2"/>  <!-- Second slave -->
     <argument value="Escravo3"/>  <!-- Third slave -->
  </process>
  <!-- The slave process (with no argument) -->
  <process host="Escravo1" function="slave"/>
  <process host="Escravo2" function="slave"/>
  <process host="Escravo3" function="slave"/>

</platform>
```

Figura 15 – Exemplo de arquivo *deployment.xml* para a aplicação Mestre/Escravo.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <host id="Mestre" power="137333000"/>
  <host id="Escravo1" power="98095000"/>
  <host id="Escravo2" power="98095000"/>
  <host id="Escravo3" power="98095000"/>

  <link id="1" bandwidth="1000000" latency="0.1"/>

  <route src="Mestre" dst="Escravo1"><link:ctn id="1"/></route>
  <route src="Mestre" dst="Escravo2"><link:ctn id="1"/></route>
  <route src="Mestre" dst="Escravo3"><link:ctn id="1"/></route>

  <route src="Escravo1" dst="Mestre"><link:ctn id="1"/></route>
  <route src="Escravo2" dst="Mestre"><link:ctn id="1"/></route>
  <route src="Escravo3" dst="Mestre"><link:ctn id="1"/></route>

</platform>
```

Figura 16 – Exemplo de arquivo *platform.xml* para a aplicação Mestre/Escravo.

distribuir as tarefas para os escravos. Este autômato está organizado na forma de uma fila de estados, que variam entre 0 e o número total de tarefas que ele tem para distribuir (T). A cada envio do Mestre para os Escravos (ocorrência do evento t), o estado do autômato é alterado, até que quando todas as tarefas forem enviadas, o autômato não muda mais seu estado. O envio das tarefas, no entanto, é realizado de maneira organizada, pois como o tamanho de todas as tarefas é o mesmo, a aplicação foi implementada para realizar envios em ordem. Entretanto, o

importante neste caso é que a modelagem realizada consiga captar a essência da aplicação, o comportamento que ela apresenta.

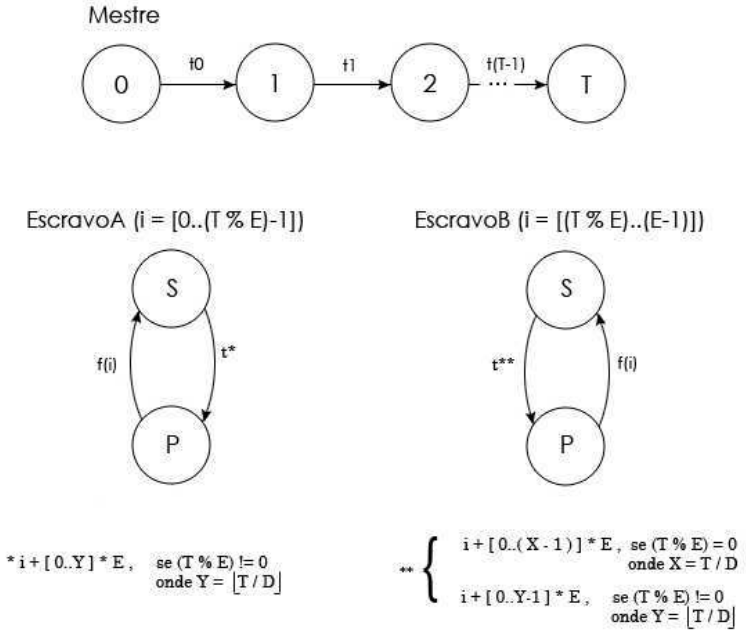

Figura 17 – Modelo SAN para a aplicação Mestre/Escravo.

Já no autômato **Escravo**, percebe-se a existência de 2 estados apenas: o estado *S*, no qual o Escravo aguarda um envio do Mestre, e o estado *P*, no qual o Escravo recebeu uma tarefa do Mestre e está processando-a. Quando o Mestre envia uma mensagem, o estado do Escravo que a recebe passa de *S* para *P* através do evento sincronizante *t*. Quando o Escravo termina o processamento, ocorre então o evento local *f*, representando o final do processamento por parte do Escravo, que torna a aguardar por mais tarefas caso existam.

Neste ponto, aparece a diferença entre os autômatos EscravoA e EscravoB. O que acontece neste modelo, é que o número de tarefas não necessariamente será divisível pelo número de Escravos. Porém, na aplicação (conforme dito anteriormente), o Mestre não entrega uma nova tarefa para o primeiro Escravo que terminar a sua execução, mas sim para o próximo Escravo na sua ordem de envio. Por exemplo, o envio de 5 tarefas é distribuído entre 3 Escravos na ordem Escravo 1, Escravo 2, Escravo 3. As tarefas 4 e 5 serão enviadas para os Escravos 1 e 2 respectivamente. Ou seja, os Escravos 1 e 2 receberam uma tarefa a mais do que o Escravo 3. Neste ponto, entra o caso da utilização dos dois diferentes tipos de autômatos EscravoA e EscravoB, para quando a quantidade de tarefas não for igual a quantidade de Escravos para processá-las.

Podemos fazer, então, a seguinte relação para este modelo (onde *T* é o número de tarefas e *E* o número de escravos):

- caso o número de tarefas seja divisível pelo número de Escravos (ou seja, $T \% E = 0$ - i.e.,

resto da divisão inteira for igual a 0), todos os Escravos serão modelados como autômatos do tipo EscravoB, pois conforme o modelo SAN apresenta, o índice *i* dos Escravos varia de *(T % E)* até *(E-1)*.

- caso o número de tarefas não seja divisível pelo número de Escravos (ou seja, $T \% E \neq 0$ - i.e., resto da divisão inteira não for igual a 0), os Escravos de índice variando no conjunto [*0..(T % E)-1*] serão modelados como autômatos do tipo EscravoA, enquanto os demais autômatos (*[(T % E)..(E-1)]*) serão modelados como autômatos do tipo EscravoB.

Tal situação ocorre porque quando o número de tarefas não for divisível exatamente por todos os Escravos, alguns deles (os do tipo EscravoA) receberão uma tarefa a mais do que os do tipo EscravoB. O número de escravos que recebem mensagens a mais é dado por $T \% E$.

Talvez esta diferenciação não seja trivial, mas a diferença encontra-se na quantidade de tarefas a serem recebidas. No caso dos Escravos do tipo EscravoA, o número de tarefas a serem recebidas é dado por $MRA = \lceil T / E \rceil$, enquanto para os escravos do tipo EscravoB este valor é dado por *MRA - 1*.

Definidas a distribuição dos autômatos e a quantidade de tarefas entre eles, basta apenas realizar a separação das tarefas. Conforme pode ser observado no modelo SAN da Figura 17, tem-se que:

- caso o número de tarefas seja divísivel exatamente entre os Escravos (ou seja, $T \% E = 0$), não utiliza-se o autômato do tipo EscravoA, e todos os eventos que farão parte do autômato EscravoB são dados (individualmente para cada EscravoB*(i)*) por:
  - *t(i + [0..(X-1)] * E), onde X = T / E*
- caso o número de tarefas não seja divisível exatamente pelo número de Escravos (ou seja, $T \% E \neq 0$), os eventos são dados por:
  - **autômato EscravoA:** *t(i + [0..Y] * E), onde* $Y = \lfloor T / E \rfloor$
  - **autômato EscravoB:** *t(i + [0..(Y-1)] * E), onde* $Y = \lfloor T / E \rfloor$

5.3.3 Mapeamento dos Parâmetros

Tabela 5 – Eventos do modelo SAN da aplicação Mestre/Escravo.

Evento	Taxa
t	1/*TempoComunicacao*
f	1/*TempoProcessamento*

Para o modelo SAN da aplicação Mestre/Escravo, dois cálculos são necessários: o tempo de comunicação entre os nodos e o tempo de processamento dos escravos. Os referidos cálculos,

que serão utilizados na sequência para o cálculo das taxas dos eventos, são realizados utilizando as fórmulas encontradas nas Equações 4.1 e 4.4, relativas ao tempo de comunicação e de processamento, respectivamente. A Tabela 5 apresenta os eventos para o modelo Mestre/Escravo e suas taxas.

6 Resultados Obtidos

Conforme descrito anteriormente, algumas aplicações foram utilizadas como exemplos neste estudo, objetivando a realização de análises e comparações entre uma abordagem utilizando Simulação e outra utilizando Modelagem Analítica. No Capítulo 5 tais aplicações foram descritas em detalhes, e neste capítulo serão abordados alguns experimentos realizados sobre elas.

Devido ao fato de cada aplicação possuir seus próprios parâmetros e características, as configurações a serem testadas são específicas para cada caso. Assim sendo, com o intuito de avaliar os pontos mais relevantes em cada situação, as aplicações foram submetidas a diferentes variações, que serão apresentadas no decorrer desta seção.

Para cada uma das aplicações, os testes que seguem estão assim organizados: primeiramente, são descritas as variações realizadas na aplicação; em seguida, no que tais variações afetam as modelagens (SimGrid e SAN) apresentadas anteriormente; e, finalmente, os resultados das execuções no SimGrid e do modelo SAN são apresentados.

6.1 Estudo de Caso 1: Token Ring

Dois testes principais são apresentados para a aplicação *Token Ring*. O primeiro deles trata-se da variação do número de voltas que o *token* realiza na topologia. O segundo ambiente de teste refere-se a variação da latência do *link* que interconecta os nodos. Para estes dois ambientes, o número de nodos da topologia foi variado entre 2 e 10.

6.1.1 Voltas do Token

Neste caso de teste, a variação ocorre no número de voltas que o *token* realiza no anel da topologia (ou seja, o número de passagens do *token* por cada um dos nós). Estas variações foram de 2, 4, 6, 8 e 10 voltas. A Tabela 6 apresenta a configuração do ambiente (presente nos arquivos de configuração do SimGrid) para este caso de teste, com os demais parâmetros da aplicação (que não foram alterados).

Com isto, conforme as equações apresentadas na Seção 4.3, os parâmetros do modelo SAN da aplicação Token Ring (Figura 9) sob estas configurações são apresentados na Tabela 7.

Baseado nestes parâmetros e com o *token* realizando 2 voltas pela topologia, obteve-se o

Tabela 6 – Configuração do SimGrid para Token Ring (voltas).

Tipo do Parâmetro	**Valor**
φ	2, 4, 6, 8 e 10 voltas do token
τ	de 2 até 10 nodos
$\beta(lk)$	3.430.125 bytes ($\cong$ 3,2Mb)
$\theta(lk)$	1 segundo
$\gamma(p)$	98.095.000 flops/s

Tabela 7 – Taxas dos eventos do modelo SAN para Token Ring (voltas).

Evento	**Taxa**
s1	(st Voltas != N) * (1/1), *onde N é o número de voltas o token*
si	1/1

gráfico da Figura 18. Os valores referentes a este gráfico encontram-se na seqüência, na Tabela 8.

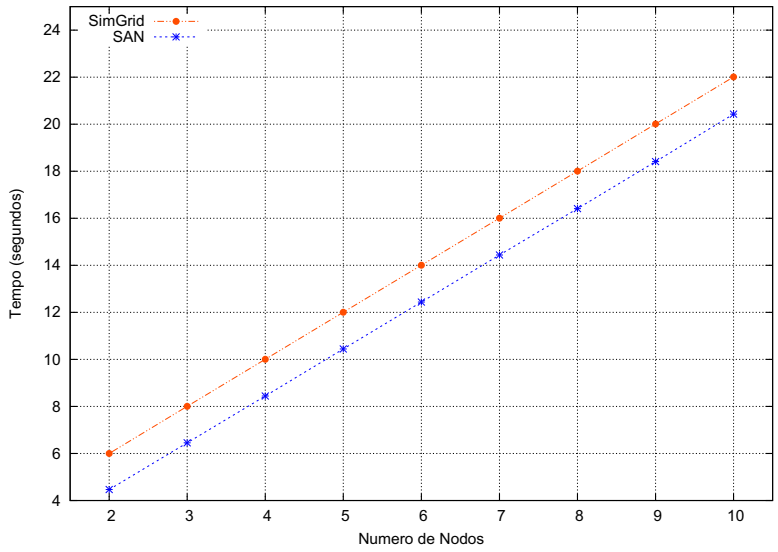

Figura 18 – Resultados para a aplicação Token Ring com 2 voltas.

Tabela 8 – Resultados para a aplicação Token Ring com 2 voltas.

		Número de Nodos								
		2	3	4	5	6	7	8	9	10
2 voltas	SimGrid (seg.)	6.0016	8.0024	10.0032	12.0040	14.0048	16.0056	18.0064	20.0072	22.0080
	SAN (seg.)	4.4681	6.4532	8.4430	10.4452	12.4402	14.4388	16.4083	18.4119	20.4268

Pode-se perceber no gráfico que o comportamento das duas curvas é similar, e mesmo quando são adicionados novos nodos na topologia ambas as curvas tendem a um crescimento contínuo. Comparando-se os tempos, nota-se que a diferença entre aqueles obtidos na execução no SimGrid e os obtidos na execução do modelo SAN é de aproximadamente 1,6 segundos.

Para dar continuidade na variação das voltas do *token* na topologia, o gráfico da Figura 19

apresenta as curvas para a execução com 4 voltas, enquanto a Tabela 9 apresenta os tempos relativos a esta execução.

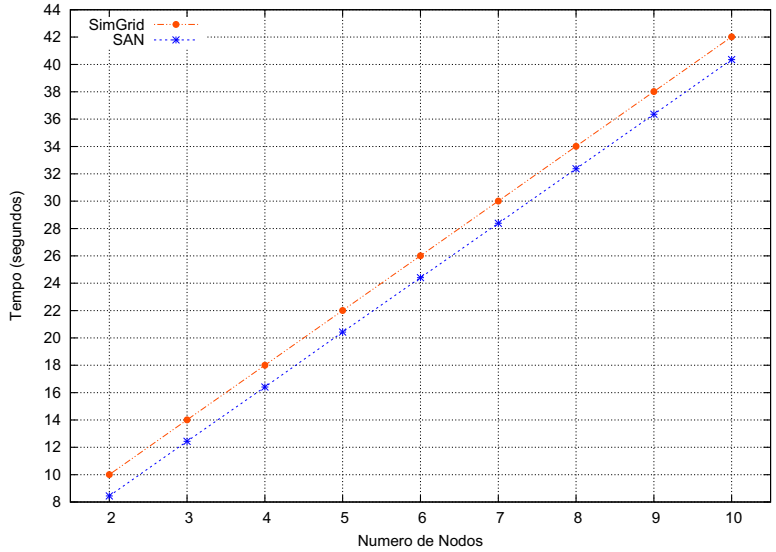

Figura 19 – Resultados para a aplicação Token Ring com 4 voltas.

Tabela 9 – Resultados para a aplicação Token Ring com 4 voltas.

		Número de Nodos								
		2	3	4	5	6	7	8	9	10
4 voltas	SimGrid (seg.)	10.0032	14.0048	18.0064	22.0080	26.0096	30.0112	34.0128	38.0144	42.0160
	SAN (seg.)	8.4430	12.4402	16.4083	20.4268	24.4064	28.3882	32.3730	36.3611	40.3523

Novamente, percebe-se a similaridade das curvas, com a mesma característica da execução anterior, na qual acrescentar nodos novos dá continuidade ao aumento do tempo. Para perceber este crescimento e esta continuidade nos valores de tempo, ainda foram executados testes para 6, 8 e 10 voltas. Os tempos obtidos com estas execuções pode ser verificado na Tabela 10.

Tabela 10 – Resultados para a aplicação Token Ring com 8 e 10 voltas.

		Número de Nodos								
		2	3	4	5	6	7	8	9	10
6 voltas	SimGrid (seg.)	14.0048	20.0072	26.0096	32.0120	38.0144	44.0168	50.0192	56.0216	62.0240
	SAN (seg.)	12.4402	18.4119	24.4064	30.3997	36.3611	42.3803	48.3515	54.3242	60.2991
8 voltas	SimGrid (seg.)	18.0064	26.0096	34.0128	42.0160	50.0192	58.0224	66.0256	74.0288	82.0320
	SAN (seg.)	16.4083	24.4064	32.3730	40.3523	48.3515	56.2964	64.3036	72.2573	80.2872
10 voltas	SimGrid (seg.)	22.0080	32.0120	42.0160	52.0200	62.0240	72.0280	82.0320	92.0360	102.0400
	SAN (seg.)	20.4268	30.3997	40.3523	50.3253	60.2991	70.2944	80.2872	90.2257	100.2447

Os resultados apresentados confirmam a tendência de crescimento e mantêm a diferença dos tempos entre aproximadamente 1,6 e 1,8 segundos, não importando quantos nodos ou quantas voltas o token realizará na topologia.

6.1.2 Latência do Link

Neste ambiente, a configuração dos parâmetros no SimGrid é a mesma apresentada anteriormente (Tabela 6), acrescentando-se o número de voltas do *token* na topologia, que neste exemplo foi fixado em 10. Além disto, a latência do *link* da topologia foi alterada, variando-a em 1, 1,2 e 1,5 segundos.

Logo, as taxas do modelo SAN para este exemplo são ilustradas na Tabela 11.

Tabela 11 – Taxas dos eventos do modelo SAN para Token Ring (latência).

Evento	**Taxa**
s1	(st Voltas != N) * (1/θ), *onde N é o número de voltas o token e* *θ = 1, 1,2 ou 1,5*
si	1/θ, *onde θ = 1, 1,2 ou 1,5*

Os valores para a execução com latência igual a 1 para 10 voltas já foi apresentado anteriormente, podendo ser observado na Tabela 10. Entretanto, a Tabela 12 mostra que para as latências com valores de 1,2 e 1,5 segundos, os tempos de execução da simulação e da modelagem analítica foram muito similares.

Tabela 12 – Resultados para a aplicação Token Ring com latência de 1,2 e 1,5 segundos.

		Número de Nodos								
		2	3	4	5	6	7	8	9	10
Latência de 1,2 segundos	SimGrid (seg.)	26	38.01	50.01	62.02	74.02	86.03	98.03	110.04	122.04
	SAN (seg.)	26.46	38.36	50.32	62.15	74.77	86.05	98.10	110.17	121.31
Latência de 1,5 segundos	SimGrid (seg.)	32.012	47.018	62.024	77.030	92.036	107.042	122.048	137.054	152.060
	SAN (seg.)	32.489	47.330	62.410	77.118	92.086	107.148	121.816	136.933	151.612

Nesta tabela, observa-se que em todos os casos a diferença entre os tempos de execução foi menor do que 1 segundo. Além disto, o comportamento crescente de ambas as abordagens continua sendo um fator perceptível e interessante, pois ilustra a compatibilidade da modelagem SAN criado com a execução no SimGrid.

6.1.3 Considerações Finais

O número de voltas da aplicação Token Ring e a latência do *link* na topologia são os parâmetros mais importantes desta aplicação. Isto ocorre por tratar-se de uma aplicação que não possui processamento sobre dados (pois o *token* é apenas uma mensagem que é recebida e repassada). Sendo assim, a variação na potência dos nodos, por exemplo, não afetaria o modelo, pois não alteraria nenhuma taxa.

As variações de tempo observadas nas execuções foram pequenas, mostrando que o modelo SAN criado conseguiu apresentar um bom resultado, assemelhando-se muito àquele obtido no

simulador. Além disso, o comportamento percebido em ambas as abordagens permite ao usuário uma idéia do comportamento da aplicação de acordo com as variações que serão realizadas. Acredita-se que as diferenças encontradas (apesar de mínimas) decorrem devido a diferente natureza das própria maneiras de calcular o tempo de execução, uma vez que na modelagem SAN são realizados cálculos e poderações probabilísticas, ocasionando erros de arredondamento.

6.2 Estudo de Caso 2: Exclusão Mútua Com Coordenador Centralizado

Para esta aplicação, foram executados testes que variam o número de clientes existentes na topologia, o número de requisições de acesso à seção crítica que estes clientes desejam realizar e a latência do *link* do sistema. A seguir, apresentam-se os resultados obtidos para os experimentos, seguidos por algumas considerações.

6.2.1 Clientes e Requisições

A Tabela 13 apresenta os valores para os parâmetros fixos deste experimento, constantes nos arquivos de configuração do SimGrid. Os parâmetros variáveis foram o número de clientes e o número de requisições (ambos variando entre 1 e 5).

Tabela 13 – Configuração do SimGrid para EMCC (clientes e requisições).

Tipo do Parâmetro	**Valor**
φ	de 1 até 5 requisições
τ	de 1 até 5 clientes
$\beta(lk)$	3.430.125 bytes ($\cong$ 3,2Mb)
$\theta(lk)$	1,5 segundos
$\gamma(p)$	98.095.000 flops/s

Já as taxas do modelo SAN referente a esta aplicação (Figura 17) podem ser calculadas com as fórmulas descritas anteriormente, e resultam nos valores encontrados na Tabela 14.

Tabela 14 – Taxas dos eventos do modelo SAN para EMCC (clientes e requisições).

Evento	**Taxa**
up	(nb [Cliente0..ClienteN] I == (N+1)), *onde N é o número de clientes - 1*
req	(st Requisicoes != R) * (1/1,5), *onde R é o número de requisições*
u	(st Recurso == L) * (1/1,5), *onde L indica que o recurso está livre*
f	(st Recurso == A) * (1/1,5), *onde A indica que o recurso está alocado*
l	1/1,5
c	(st Recurso == L) * (1/1,5), *onde L indica que o recurso está livre*

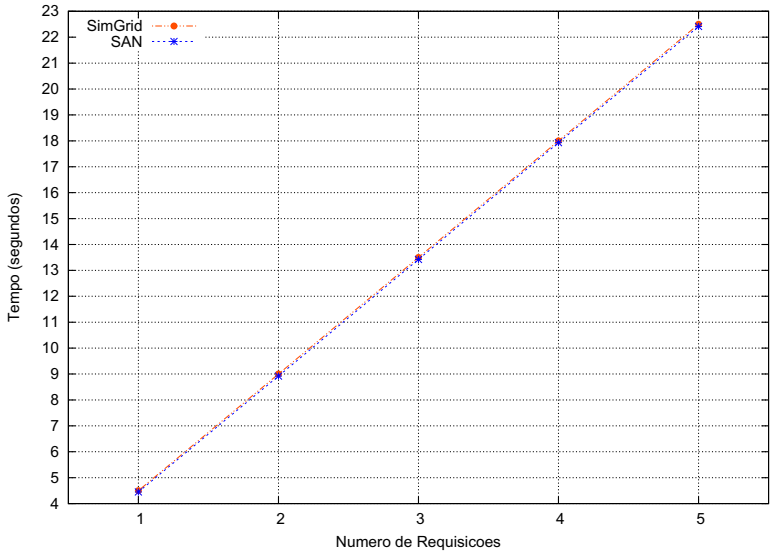

Figura 20 – Resultados para a aplicação Exclusão Mútua com 1 cliente.

A primeira execução realizada foi com 1 cliente, variando o número de requisições de 1 até 5. O gráfico da Figura 20 ilustra o comportamento das duas abordagens. Pode-se perceber no gráfico que o comportamento é muito semelhante, com valores tão próximos que as curvas se sobrepõem. Isto se deve ao fato de que com 1 cliente apenas não existe disputa pela seção crítica. Toda vez que o cliente deseja o acesso exclusivo, o recurso está disponível, e ele nunca vai para a fila. O fato de o cliente nunca ir para a fila está relacionado à taxa funcional do evento *f*. Esta taxa diz que um cliente aguarda na fila quando ele requisita o recurso e este encontra-se alocado, situação que nunca ocorre com apenas 1 cliente no sistema. A semelhança dos valores pode ser comprovada através da Tabela 15, que apresenta os valores referentes ao gráfico supracitado.

Tabela 15 – Tempos obtidos para a aplicação EMCC com 1 cliente.

		Requisições				
		1	2	3	4	5
1 cliente	SimGrid (seg.)	4.5	9	13,5	18	22,5
	SAN (seg.)	4.4498	8.9220	13.4219	17.9304	22.4132

Esta situação modifica-se quando mais clientes são adicionados. O gráfico da Figura 21 apresenta os resultados para a execução da aplicação com 2 clientes requisitando a seção crítica.

Tabela 16 – Tempos obtidos para a aplicação EMCC com 2 clientes.

		Requisições				
		1	2	3	4	5
2 clientes	SimGrid (seg.)	7.5	13,5	19,5	25,5	31,5
	SAN (seg.)	7.0742	13.3696	19.7129	26.1050	32.4555

Apesar de os valores continuarem muito próximos (Tabela 16), nota-se que as curvas não se sobrepõem mais como antes. A Figura 22 ilustra o afastamento das curvas quando mais um

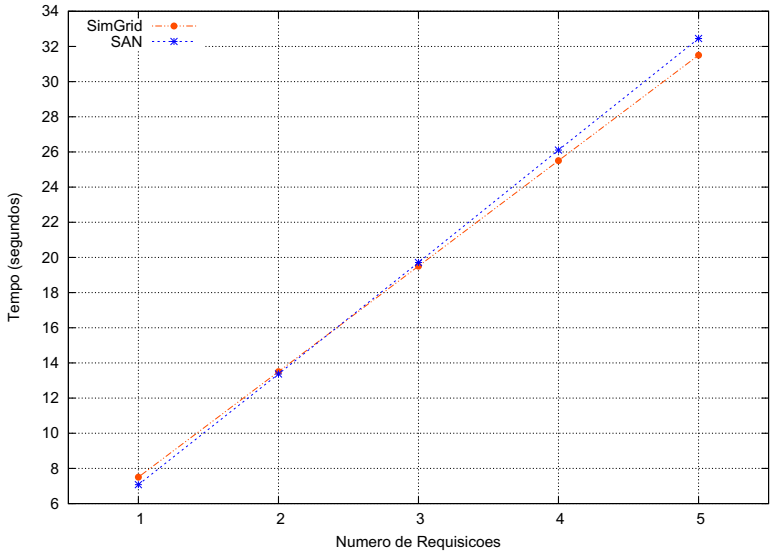

Figura 21 – Resultados para a aplicação Exclusão Mútua com 2 clientes.

cliente é inserido.

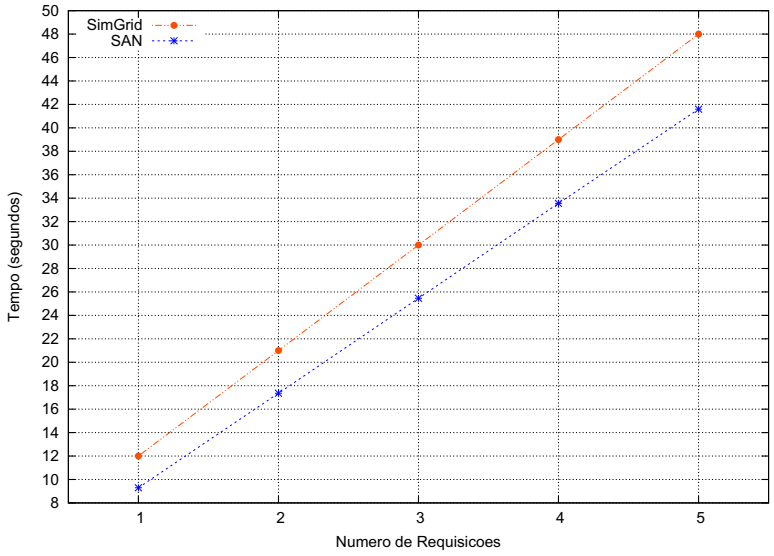

Figura 22 – Resultados para a aplicação Exclusão Mútua com 3 clientes.

Apesar de afastadas, as diferenças entre os tempos com 3 clientes ainda não são demasiadamente grandes. Entretanto, pelo comportamento observado na curva, pode-se acreditar que a diferença aumentará de acordo com o aumento do número de requisições dos clientes. A Tabela 17 apresenta os tempos de execução relativos à este gráfico, juntamente com os tempos de execução para uma topologia com 4 e 5 clientes.

Como pode-se perceber, os tempos de execução continuam se distanciando. Com 4 clientes, a diferença entre os tempos das duas abordagens passou de aproximadamente 5 segundos com 1 requisição para cerca de 14 segundos quando executado com 5 requisições, enquanto com 5

Tabela 17 – Tempos obtidos para a aplicação Exclusão Mútua com 3, 4 e 5 clientes.

		Requisições				
		1	2	3	4	5
3 clientes	SimGrid (seg.)	12	21	30	39	48
	SAN (seg.)	9.2943	17.3604	25.4669	33.5560	41.5924
4 clientes	SimGrid (seg.)	16,5	28,5	40,5	52,5	64,5
	SAN (seg.)	11,4409	21,2253	30,9522	40,6470	50,4061
5 clientes	SimGrid (seg.)	21	36	51	66	81
	SAN (seg.)	13,4967	24,9536	36,2652	47,6300	58,9517

clientes a diferença chegou a ser de mais de 20 segundos (com 5 requisições).

Isto quer dizer que além da diferença ocorrida devido à disputa da seção crítica entre os processos, há uma diferença associada ao número de requisições. Estas diferenças podem estar associadas ao fato de que, no modelo SAN criado, não importa qual dos clientes vai requisitar novamente a seção crítica. O que importa neste modelo, é que a seção crítica vai ser requisitada *N* vezes, o que aumenta a probabilidade de clientes que já utilizaram o recurso por "requisições"vezes utilizem novamente, fazendo com que os outros clientes não requisitem, ou, requisitem poucas vezes.

No SimGrid, por sua vez, a execução se dá na forma de que cada cliente utilizará a seção crítica "requisições"vezes. O que importa aqui, então, é que os *M* clientes utilizem a seção crítica "requisições"vezes.

Entretanto, pode-se ressaltar um fato interessante: tanto para uma abordagem quanto para outra, os tempos de execução para mais clientes podem ser encontrados empiricamente. Para predizer o tempo de execução da aplicação para 10 requisições, por exemplo, pode-se utilizar os valores dantes encontrados.

Por exemplo, nota-se através das execuções realizadas um crescimento constante dos valores nas execuções do SimGrid. De acordo com a Tabela 16, com 2 clientes e 1 requisição, o tempo é de 7,5 segundos. Para 2 requisições, o valor é de 13,5 segundos, ou seja, uma diferença de 6 segundos para o valor encontrado com 1 requisição. Ao verificar os demais valores, percebe-se que a diferença entre os tempos das requisições é sempre de 6 segundos neste caso (7,5s, 13,5s, 19,5s, 25,5s e 31,5s). Empiricamente, pode-se calcular que com 10 requisições, 2 clientes levariam 61,5 segundos.

Com os resultados do modelo SAN acontece a mesma situação, porém, não com valores exatos e constantes, por tratar-se de um método matemático, que pode, por exemplo, apresentar erros de arredondamento que se acumulam. Porém, as diferenças entre os tempos das requisições variam aproximadamente em 6,3 segundos. Desta forma, pode-se prever que para 10 requisições, 2 clientes utilizariam cerca de 63 segundos.

6.2.2 Latência dos Links

O próximo experimento realizado teve como variável a latência do *link*. Nestes experimentos, o número de clientes foi fixado em 2, e os valores da latência variaram entre 1,5, 3,5 e 5 segundos. Os demais parâmetros continuam conforme apresentados anteriormente na Tabela 13.

Com as alterações na latência, as taxas do modelo SAN tiveram seus valores re-calculados, resultando nos valores ilustrados na Tabela 18.

Tabela 18 – Taxas dos eventos do modelo SAN para EMCC (latência).

Evento	**Taxa**
up	(nb [Cliente0..ClienteN] I == (N+1)), *onde N é o número de clientes - 1*
req	(st Requisicoes != R) * (1/θ), *onde R é o número de requisições e* *θ = 3,5 ou 5*
u	(st Recurso == L) * (1/θ), *onde L indica que o recurso está livre* *θ = 3,5 ou 5*
f	(st Recurso == A) * (1/θ), *onde A indica que o recurso está alocado* *θ = 3,5 ou 5*
l	1/θ, *onde θ = 3,5 ou 5*
c	(st Recurso == L) * (1/θ), *onde L indica que o recurso está livre* *θ = 3,5 ou 5*

Os resultados das execuções utilizando a latência de 1,5 segundos com 2 clientes foi apresentada anteriormente, tratando-se do gráfico da Figura 21. Neste gráfico percebe-se que as curvas cruzam-se em determinado ponto (3 requisições), fazendo com que o tempo na modelagem SAN passe, a partir deste ponto, a ser maior do que o tempo do SimGrid.

Na Figura 23 encontram-se as curvas referentes às execuções realizadas com latência igual a 3,5 segundos. O fenômeno do cruzamento continua acontecendo, entretanto as curvas se cruzam somente a partir de 5 requisições, conforme pode ser observado na Tabela 19, que contém os tempos de execução.

Tabela 19 – Tempos obtidos para a aplicação EMCC com latência de 3,5 e 5 segundos.

		Requisições				
		1	2	3	4	5
Latência de 3,5 segundos	SimGrid (seg.)	17.5	31.5	45.5	59,5	73,5
	SAN (seg.)	14.5167	29.2138	44.0487	58.8814	73.7356
Latência de 5 segundos	SimGrid (seg.)	25	45	65	85	105
	SAN (seg.)	20.1033	41.0971	62.3103	83.5009	104.7055

Ainda na Tabela 19, é possível observar que o comportamento das curvas continua o mesmo, porém, quanto maior o valor da latência, mais longe as curvas se cruzam (em relação ao número de requisições). Mesmo assim, os tempos de execução obtidos são similares.

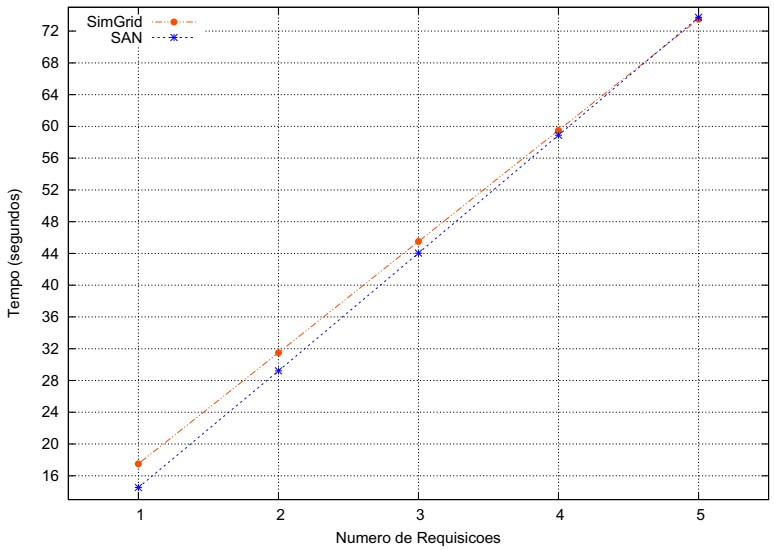

Figura 23 – Resultados para a aplicação Exclusão Mútua com latência de 3,5 segundos.

6.2.3 Considerações Finais

O modelo SAN criado para a aplicação Exclusão Mútua com Coordenador Centralizado apresentou resultados interessantes quando executado para poucos clientes. À medida que mais clientes são adicionados à topologia, os valores da sua execução distanciam-se dos valores obtidos através da simulação no SimGrid.

Este modelo, no entanto, mostrou-se interessante para ilustrar duas maneiras diferentes de modelagem de uma mesma aplicação. Enquanto a implementação no SimGrid foca no fato de que cada cliente deve realizar suas requisições, o modelo SAN criado foca na quantidade de requisições atendidas, não importando quais os clientes solicitaram a seção crítica.

6.3 Estudo de Caso 3: Mestre/Escravo

O último caso de teste foi executado com a finalidade de analisar alguns aspectos relevantes tais como o tamanho das tarefas e o poder computacional dos nodos. Nas outras aplicações estes testes não eram necessários, pois os nodos não realizavam nenhum processamento sobre as mensagens. Além destes aspectos, ainda são variados para este exemplo o número de tarefas a seram realizados e a quantidade de escravos presentes na topologia.

6.3.1 Número de Tarefas

Primeiramente, este experimento visa variar a quantidade de tarefas executada pelos escravos. Para tanto, variou-se entre 5, 10 e 20 este número, e para cada um destes casos realizaram-se execuções de 2 até 9 escravos. Os demais parâmetros para esta aplicação foram fixados conforme apresenta a Tabela 20.

Tabela 20 – Configuração do SimGrid para Mestre/Escravo (número de tarefas).

Tipo do Parâmetro	**Valor**
φ	número de tarefas: 5, 10 e 20 tamanho da tarefa: 500.000.000 flops (500 megaflops) tamanho de comunicação: 100 bytes
τ	de 1 até 10 escravos
$\beta(lk)$	1.000.000 bytes
$\theta(lk)$	0,1 segundos
$\gamma(p)$	98.095.000 flops/s

Com estas configurações, o mapeamento dos parâmetros do modelo SAN (Figura 17) para esta situação resulta nos valores constantes na Tabela 21. Cabe lembrar que a variação do número de tarefas altera a quantidade de estados do autômato Mestre.

Tabela 21 – Taxas dos eventos do modelo SAN para Mestre/Escravo (número de tarefas).

Evento	**Taxa**
t	1/0,1
f	1/5,1

A primeira execução realizada pode ser observada no gráfico da Figura 24. Trata-se da execução com 20 tarefas.

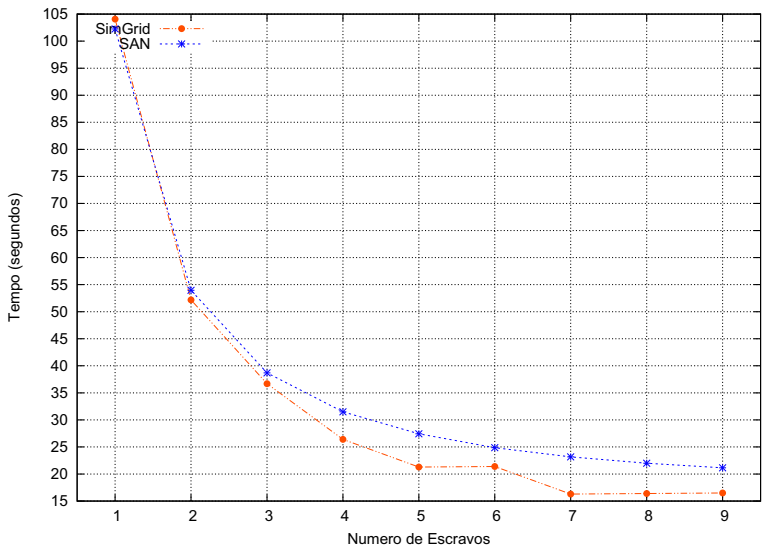

Figura 24 – Resultados para a aplicação Mestre/Escravo com 20 tarefas.

Através da análise do gráfico, percebe-se claramente que o comportamento da modelagem SAN e da simulação com o SimGrid foram semelhantes. À primeira vista, pode parecer estranho o ponto da curva do SIMGRID com 6 escravos. Porém, quando imagina-se o que deve acontecer na aplicação, percebe-se que tal fenômeno tem um bom embasamento. Nesta aplicação são enviadas 20 tarefas de mesmo tamanho. Pode-se dizer que, teoricamente:

- com 2 escravos, ocorrerão 10 envios de 2 tarefas;
- com 3 escravos, ocorrerão 6 envios de 3 tarefas e mais um envio das outras 2 (totalizando 7 envios);
- com 4 escravos, ocorrerão 5 envios de 4 tarefas;
- com 5 escravos, ocorrerão 4 envios de 5 tarefas;
- com 6 escravos, ocorrerão 3 envios de 6 tarefas e mais um envio de 2 tarefas (totalizando 4 envios);
- com 7 escravos, ocorrerão 2 envios de 7 tarefas e mais 1 envio de 6 tarefas (totalizando 3)

Assim, como o número de envios de tarefas quando a aplicação é executada com 5 ou com 6 escravos é o mesmo, a cada envio é realizada uma "bateria de execuções". Se com 5 e 6 escravos ocorrem o mesmo número de envios, logo, ocorrerá o mesmo número de "baterias de execução", resultando em um tempo de execução similar. Para validar tal fato, executou-se a aplicação com 20 tarefas e 20 escravos. Desta forma somente 1 envio seria realizado, resultando em apenas uma "bateria de execuções". Logo, o tempo total deve ser o tempo de enviar as 20 tarefas (20 * 0.1) mais o tempo de processar uma única tarefa, já que os 20 nodos estarão executando totalmente em paralelo cada um uma bateria (5.09 segundos). Assim (**teoricamente e desconsiderando tempos de inicialização e de finalização**): (20 * 0.1) + 5.09 = **7.09**

Executando o programa *Mestre/Escravo* com 20 escravos no SIMGRID, o tempo de execução encontrado foi **7.2171** segundos. Infelizmente, o modelo SAN não apresentou-se viável para a execução, pois teríamos o montante de 20971520 estados.

Outros testes foram realizados com os demais tamanhos de tarefa para verificar sua influência nos resultados. A Tabela 22 apresenta os valores para os tempos de execução dos experimentos com 5, 10 e 20 tarefas.

Observando os valores encontrados, nota-se que o comportamento das duas abordagens continua similar em todos os casos. Primeiramente, quando da execução com apenas 1 escravo, a diferença entre as execuções não é muito grande. Conforme o número de processos aumenta, a diferença dos tempos de execução também aumenta, mas ainda com valores próximos.

Um ponto interessante a ser ressaltado, trata-se do comportamento das curvas a partir de 5 processos. Deste número de escravos em diante, ambas as abordagens prevêem que não serão obtidos maiores ganhos de desempenho significativos com a inserção de novos escravos, o que

Tabela 22 – Resultados para a aplicação Mestre/Escravo com 5, 10 e 20 tarefas.

		Número de Escravos								
		1	2	3	4	5	6	7	8	9
5 tarefas	SimGrid (seg.)	26.0904	15.7942	10.6972	10.7962	5.7021	5.8021	5.9021	6.0021	6.1021
	SAN (seg.)	25.9380	15.7880	13.2674	12.4139	12.1384	12.1384	12.1384	12.1384	12.1384
10 tarefas	SimGrid (seg.)	52.0809	26.1914	21.0923	15.9952	10.9002	10.9992	11.0982	11.1972	11.2962
	SAN (seg.)	51.3675	28.5240	21.7293	18.7739	17.2399	16.3976	15.9269	15.6403	15.4987
20 tarefas	SimGrid (seg.)	104.0619	52.1819	36.6876	26.3934	21.2963	21.3933	16.2992	16.3972	16.4952
	SAN (seg.)	102.2600	53.9567	38.6982	31.4958	27.4264	24.8680	23.1736	22.0004	21.1465

serve como incentivo ao usuário para que analise o custo-benefício de fazer uso de mais nós para ganhos não tão significativos.

6.3.2 Tamanho das Tarefas

Outra variável que possivelmente afete os resultados das execuções trata-se do Tamanho Computacional das tarefas. No experimento que segue, os valores para estes tamanhos computacionais foram variados entre 100.000.000, 250.000.000 e 500.000.000 flops (100, 250 e 500 megaflops, respectivamente). Os demais valores foram fixados conforme a Tabela 20 apresentada anteriormente, e o número de escravos variou entre 2 e 9.

As taxas do modelo SAN para estas configurações são relatadas na Tabela 23.

Tabela 23 – Taxas dos eventos do modelo SAN para Mestre/Escravo (tamanho das tarefas).

Evento	Taxa para 100 megaflops	Taxa para 250 megaflops
t	1/0,1	1/0,1
f	1/1,02	1/2,55

A alteração no tamanho das tarefas resultou em tempos de execução diferentes, conforme teoricamente imagina-se. A Tabela 24 ilustra esta alteração dos tempos de execução para os três tamanhos computacionais executados sob as configurações supracitadas (no experimento anterior, a execução com 10 tarefas e tamanho de tarefa igual a 500 megaflops foi apresentada, mas seus valores são apresentados novamente para facilitar a comparação de seus valores).

Tabela 24 – Resultados para a aplicação Mestre/Escravo com 100, 250 e 500 megaflops.

		Número de Escravos								
		1	2	3	4	5	6	7	8	9
100 megaflops	SimGrid (seg.)	11.3042	5.8031	4.7816	3.7622	2.7448	2.8438	2.9428	3.041840	3.1408
	SAN (seg.)	10.7565	6.1605	4.8157	4.2203	3.9232	3.7680	3.6473	3.5931	3.5616
250 megaflops	SimGrid (seg.)	26.5954	13.4487	10.8982	8.3496	5.8031	5.9021	6.0011	6.1001	6.1991
	SAN (seg.)	25.9924	14.5436	11.1519	9.6710	8.9086	8.4999	8.2644	8.1077	8.0345
500 megaflops	SimGrid (seg.)	52.0809	26.1914	21.0923	15.9952	10.9002	10.9992	11.0982	11.1972	11.2962
	SAN (seg.)	51.3675	28.5240	21.7293	18.7739	17.2399	16.3976	15.9269	15.6403	15.4987

Analisando os valores encontrados, percebe-se que as diferenças de tempo nas execuções foram bem próximas do que teoricamente deveria acontecer. Em outras palavras, se o tamanho

computacional de 250 megaflops trata-se da metade de 500 megaflops, teoricamente, o tempo para a execução daquele deveria ser 50% menor do que o deste (teoricamente, pois sabe-se que existem variáveis que alteram esta teoria, tais como congestionamentos, perdas de pacotes, variáveis de controle, etc.). Ao verificar os valores obtidos e realizar tal cálculo, notar-se-á que esta relação está presente, não em valores exatos mas em valores aproximados.

Por exemplo, com 1 escravo e tamanho de tarefa de 500 megaflops, obteve-se com o modelo SAN o tempo de 51.37 segundos. Se 250 megaflops representa 50% de 500 megaflops, então o tempo teórico é de 51,37*0,5 = 25,685 segundos, enquanto o obtido foi 25.992 segundos. O mesmo vale para 100 megaflops. Se 100 megaflops trata-se de 20% de 500 megaflops, logo (teoricamente) teríamos 51,37*0,2 = 10,27 segundos, e o tempo obtido foi de 10,76 segundos aproximadamente. O mesmo fato também pode ser observado nos tempos do SimGrid.

6.3.3 Poder Computacional

A última variação desta aplicação diz respeito ao poder computacional dos nodos que realizarão a função de escravo. Nos exemplos anteriores, este poder computacional possuía o valor de 98.095.000 flops/s, e nestes novos experimentos, oa valores foram alterados para que os escravos executem com 50.000.000 e 200.000.000 flops/s de poder computacional. As demais variáveis permaneceram conforme a Tabela 20 apresentada anteriormente, e as execuções continuam sendo realizadas para número de escravos variando entre 2 e 9.

Com o novo poder computacional dos escravos, as taxas do modelo SAN foram novamente alteradas, e podem ser observadas na Tabela 25.

Tabela 25 – Taxas dos eventos do modelo SAN para Mestre/Escravo (poder computacional).

Evento	**Taxa para 50 megaflops/s**	**Taxa para 200 megaflops/s**
t	1/0,1	1/0,1
f	1/10	1/2,5

Os valores obtidos para as execuções com 50.000.000 e 200.000.000 flops/s podem ser vistas na Tabela 26. Através da análise destes tempos de execução, percebe-se que o poder computacional dos escravos altera bastante os valores, fazendo com que ambientes com nodos com maior poder de processamento de fato tenham seus desempenhos melhores do que ambientes com pouco poder de processamento.

Tabela 26 – Resultados para a aplicação Mestre/Escravo com 50 e 200 megaflops/s.

		Número de Escravos								
		1	2	3	4	5	6	7	8	9
50 megaflops	SimGrid (seg.)	101.1100	50.7060	40.7040	30.7040	20.7060	20.8050	20.9040	21.0030	21.1020
	SAN (seg.)	100.1781	55.3442	42.0543	36.2392	33.2445	31.5913	30.6392	29.9207	29.8241
200 megaflops	SimGrid (seg.)	26.1100	13.2060	10.7040	8.2040	5.7060	5.8050	5.9040	6.0030	6.1020
	SAN (seg.)	25.4693	14.2585	10.9593	9.5046	8.7568	8.3323	8.1029	7.9701	7.8979

Nos valores obtidos, nota-se que o comportamento destes continua semelhante, apresentando as mesmas características entre as duas curvas, conforme foi descrito anteriormente no primeiro e no segundo experimento.

6.3.4 Considerações Finais

Através da utilização da aplicação Mestre/Escravo, análises de comportamento das curvas, variação do número de tarefas e influência do tamanho computacional da tarefa e do poder de processamento dos escravos puderam ser realizadas. Um aspecto interessante trata-se do comportamento da curva, que não importando as variações que sejam realizadas, continua com a mesma similaridade.

Os experimentos realizados ressaltou alguns aspectos importantes da aplicação, no que se refere a custo-benefício de aquisição e/ou utilização de diversos nodos. Utilizando tanto a modelagem com SAN como a simulação com o SimGrid, obter-se-á uma boa noção do desempenho para usuários que possuem necessidades parecidas com as apresentadas pela aplicação Mestre/Escravo.

7 Conclusões e Trabalhos Futuros

Este trabalho comparou duas diferentes abordagens de avaliação de desempenho para aplicações de plataformas distribuídas: a Modelagem Analítica e a Simulação. Estas formas foram escolhidas por possibilitarem ao desenvolvedor (ou ao usuário que necessite avaliar o desempenho de seu sistema) a obtenção de resultados destes desempenhos antes mesmo da implementação do sistema, evitando assim gastos com re-construção de código, perda de tempo ao implementar algo que não apresentará um desempenho aceitável, auxiliando na procura de gargalos, etc.

Entretanto, uma das partes mais importantes na hora de criar um modelo de um sistema computacional trata-se da fase de coleta de parâmetros. De acordo com os parâmetros que forem obtidos e inseridos no modelo, os resultados sofrem grandes alterações, podendo iclusive invalidar o modelo. Neste contexto, neste estudo foi realizada uma comparação entre execuções do simulador de plataformas distribuídas SimGrid e a ferramenta de Modelagem Analítica SAN. Os parâmetros dos modelos SAN, então, puderam ser mapeados através de alguns cálculos e informações obtidas através dos arquivos de configuração do SimGrid. Desta maneira, a modelagem SAN realizada possuía parâmetros que estavam de acordo com as execuções do simulador, podendo assim serem realizados testes e análises comparativas.

Para facilitar o mapeamento dos parâmetros, algumas definições foram realizadas, incluindo algumas considerações sobre parâmetros dependentes do ambiente e parâmetros dependentes da aplicação. Para realizar alguns testes experimentais, foram descritas três aplicações, incluindo a modelagem no SimGrid, a modelagem SAN detalhada e a maneira pela qual os parâmetros foram calculados.

Na condução dos experimentos, percebeu-se que o mapeamento dos parâmetros de maneira geral foi bem sucedido. No primeiro exemplo, o modelo SAN da aplicação Token Ring apresentou diferenças muito pequenas, mesmo quando executado com diferentes configurações de ambiente e variáveis. O segundo exemplo fazia referência a uma aplicação chamada Exclusão Mútua com Coordenador Centralizado. Nos experimentos desta aplicação, o modelo SAN conseguiu captar o mesmo comportamento apresentado pelo simulador, apesar de ter havido um aumento na diferença dos tempos quando mais requisições eram adicionadas. No entanto, através dos resultados percebeu-se que as abordagens do modelo SAN e da aplicação no SimGrid eram um pouco diferentes, explicando a diferença nos tempos encontrados. Por fim, uma aplicação do tipo Mestre/Escravo foi executada. Nesta aplicação, outros fatores puderam ser testados, como variações no poder de processamento dos nodos e no tamanho das tarefas. Os resultados foram interessantes, mostrando que o modelo SAN e o SimGrid apresentaram o mesmo

comportamento, com pouca variação no tempo de execução.

Através do desenvolvimento deste trabalho, ficaram claras algumas diferenças importantes entre as duas abordagens utilizadas. Alguns exemplos podem ser citados, tais como a maior dificuldade de implementação e modelagem utilizando a abordagem de simulação. Isto se deve ao fato de que com a simulação necessita-se de um conhecimento concreto sobre aspectos específicos de programação. Muitas vezes, entretanto, o usuário que deseja avaliar o desempenho de algum sistema não é um desenvolvedor e não possui conhecimentos técnicos para tanto. Por outro lado, com a modelagem analítica, percebeu-se que há uma abstração maior, não necessitanto tais conhecimentos específicos, modelando apenas com os parâmetros obtidos.

Outra diferença que pôde ser observada durante a realização do trabalho, foi que o código utilizado para a simulação muitas vezes assemelha-se muito ao código do sistema em si, tornando a etapa de avaliação mais custosa quando não se possui o conhecimento necessário, fazendo com que a Modelagem Analítica fique em vantagem neste aspecto também. Porém, com simulações, os resultados são obtidos de maneira muito rápida, não necessitando a realização de cálculos e diversas execuções do modelo. Basta realizar alterações nos arquivos de configuração e executar novamente. Outra desvantagem percebida na Modelagem Analítica quando comparada à simulação, é que a explosão do espaço de estados ocorre rapidamente de acordo com o tamanho do modelo analítico criado, enquanto no simulador isto não acontece tão rapidamente.

Apesar de que o mapeamento dos parâmetros, as modelagens criadas e os testes realizados terem sido satisfatórios, acredita-se que para uma real validação destas comparações, análises estatísticas devem ser realizadas, assim como as execuções destas aplicações em ambientes reais. Desta maneira, validar-se-ia por completo o estudo, trazendo comprações e aspectos não apresentados neste trabalho. A utilização desta abordagem com a utilização de aplicações reais trata-se de um trabalho futuro a ser realizado.

Além disto, a comparação entre simulação e modelagem analítica apresentada neste trabalho com a utilização de um simulador para plataformas distribuídas abre novas idéias para uma modalagem de mais baixo nível e mais específica futuramente, não focando somente na aplicação. Esta nova leitura seria utilizando especificamente ambientes de grades computacionais, modelando a parte existente abaixo da aplicação, que é responsável por gerenciar toda a execução de tarefas neste tipo de ambiente.

Referências

[1] PLATEAU, B.; ATIF, K. Stochastic automata network of modeling parallel systems. *IEEE Transactions on Software Engineering*, v. 17, n. 10, p. 1093–1108, 1991.

[2] BOLCH, G. et al. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. New York, NY, USA: John Wiley & Sons Inc., 1998. 726 p.

[3] LEGRAND, A.; MARCHAL, L.; CASANOVA, H. Scheduling distributed applications: the simgrid simulation framework. In: *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. Tóquio, Japão: IEEE Computer Society, 2003. p. 138–145.

[4] MARCULESCU, R.; NANDI, A. Probabilistic application modeling for system-level perfromance analysis. In: *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Monique, Alemanha: IEEE Press, 2001. p. 572–579.

[5] BALDO, L. et al. Performance models for master/slave parallel programs. In: *1st International Workshop on Practical Applications of Stochastic Modelling*. London, UK: The Royal Society, 2005. p. 101–121.

[6] LHUILLIER, M. *Modélisation pour la synthse d'images partir d'images*. Tese (Doutorado) — Institut National Polytechnique de Grenoble, França, 2000.

[7] GUEROUI, A. Quality of service of a rerouting algorithm using stochastic automata networks. In: *ISCC '01: Proceedings of the Sixth IEEE Symposium on Computers and Communications*. Hammamet, Tunisia: IEEE Computer Society, 2001. p. 338–343.

[8] JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York, NY, USA: John Wiley & Sons Inc., 1991. 685 p.

[9] HU, L.; GORTON, I. *Performance Evaluation for Parallel Systems: A Survey*. Technical Report UNSW-CSE-TR-9707, Department of Computer Systems, University of NSW, Sydney, Australia, 1997. 56 p.

[10] MULLENDER, S. *Distributed Systems*. 2nd. ed. Reading, Massachusetts, USA: Addison Wesley, 1993. 595 p.

[11] OPTORSIM - A Grid Simulator for Studying Dynamic Data Replication Strategies. Disponível em: <mack.ittc.ku.edu/bell03optorsim.html>. Acesso em: 12 jul. 2008.

[12] BUYYA, R.; MURSHED, M. M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Journal of Concurrency and Computation: PracticExperience (CCPE)*, v. 14, p. 1175–1220, 2002.

[13] SONG, H. J. et al. The microgrid: A scientific tool for modeling computational grids. *Journal of Scientific Programming*, v. 8, n. 3, p. 127–141, 2000.

[14] CASANOVA, H. Simgrid: A toolkit for the simulation of application scheduling. In: *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. Brisbane, Australia: IEEE Computer Society, 2001. p. 430–437.

[15] GLOBUS: A Metacomputing Infrastructure Toolkit. Disponível em: <mack.ittc.ku.edu/article/foster96globus.html>. Acesso em: 23 jun. 2008.

[16] LUCAS, J. H. Performance evaluation and monitoring. *ACM Computing Surveys (CSUR)*, v. 3, n. 3, p. 79–91, 1971.

[17] MEIRA, J. W. *Modeling Performance of Parallel Programs*. Technical Report 589, Computer Science Department, University of Rochester, New York, USA, 1995. 42 p.

[18] MARSAN, M. A.; BALBO, G.; CONTE, G. *Performance models of multiprocessor systems*. Cambridge, MA, USA: MIT Press, 1987. 280 p.

[19] WILLIAMSON, H. *Xml: The Complete Reference*. Berkeley, CA, USA: Osborne/McGraw-Hill, 2001. 965 p.

[20] BENOIT, A. et al. The peps software tool. In: *Proceedings of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Urbana, Illinois, USA: Lecture Notes in Computer Science, 2003. p. 98–115.